1. Comparators
    a. See if two binary numbers are equal, output a 1 if true, 0 otherwise
    b. For an *n* bit comparison, use *n* 2-bit XNOR gates to compare if digits are equal
        i. XNOR outputs a 1 if both are equal
    c. AND the results of all XNORs together to get final answer
    d. Example: 2 bit comparator

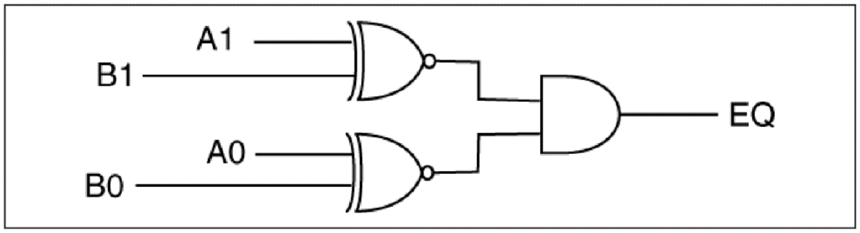
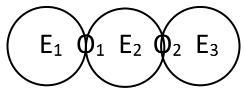


Figure 11.10: The final circuit for the 2-bit comparator as equation (e) in Figure 11.9.

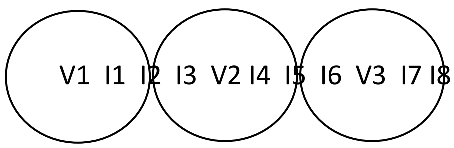
2. Arithmetic logic unit (ALU)
    a. One unit that can perform multiple operations
        i. Example: do ADD, NOT, AND, OR, XOR, equality check, so forth
    b. Each cell of the ALU has one of each type of gate in it
        i. Performs operation on corresponding bits
        ii. Use MUX to select which operation is chosen
        iii. Control/select bits determine what operation is done
        iv. Perform all operations at all time, discard results of operations not done
            1. Sounds wasteful, but it's cheaper to discard the results than attempt to disable/enable the unused pieces
3. Error detection and correction
    a. Error types
        i. Hard failure – permanent physical defect
        ii. Soft error – random non-destructive event that causes an error
            1. Example – voltage spike
            2. Mario 64 upwarp glitch – thought to be a soft error (bit flip)
        iii. More common now that components are smaller
    b. Will focus on errors that involve bits changing value
    c. Measure of size of error – Hamming distance between original and final values
    d. Three possible outcomes
        i. No errors detected
        ii. Error detected, and location specified, so we can correct it
        iii. Error detected, and location unspecified, so we can't correct it
    e. Will look at SECDED – Single Error Correction Double Error Detection
        i. If the number of bits changed in a set is large enough, *any* error detection system will fail
4. Parity
    a. Errors with a Hamming distance of 1 can be detected, but not located, with parity
    b. *Even parity* – count the number of 1s in the data
        i. Set or clear an additional bit so that the number of 1s is even (including the parity bit)
        ii. *Odd parity* – set or clear an additional bit so that the number of 1s is odd
            1. We'll be using even parity for the rest of these examples
        iii. One type of XOR gate produces a 1 whenever the number of 1s input is odd, so perfect for this use case
        iv. Example for even parity
            1. C denotes the position of the check bit
            2. C1001 -> 01001
            3. C1101 -> 11101

v. Even parity creates valid code words that have a Hamming distance of 2 between them
   1. Need valid and invalid code words so we know when something goes wrong
   2. Invalid code words in this case are the odd parity numbers



Circles have a radius of one Hamming distance.

c. Two bit errors
   i. To allow detection, need valid code words with at least a Hamming distance of 3 away



Circles have a radius of two Hamming distance.

   ii. All one Hamming distance errors are associated with exactly one valid code word
   iii. Errors with two bits will still be detected, but may be associated with another valid code word
      1. I2 above could either be associated with V1 or V2
   iv. What happens with three-bit errors?