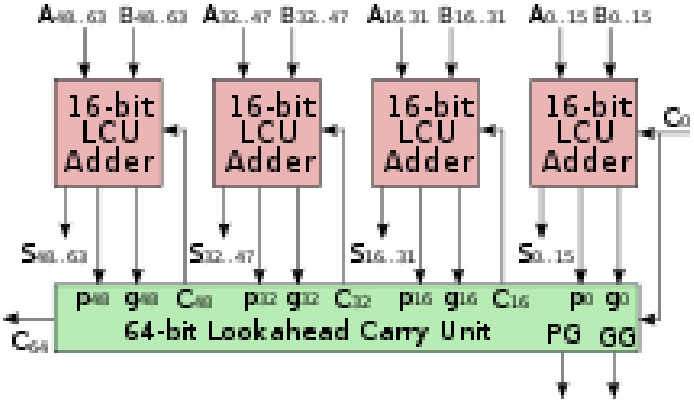
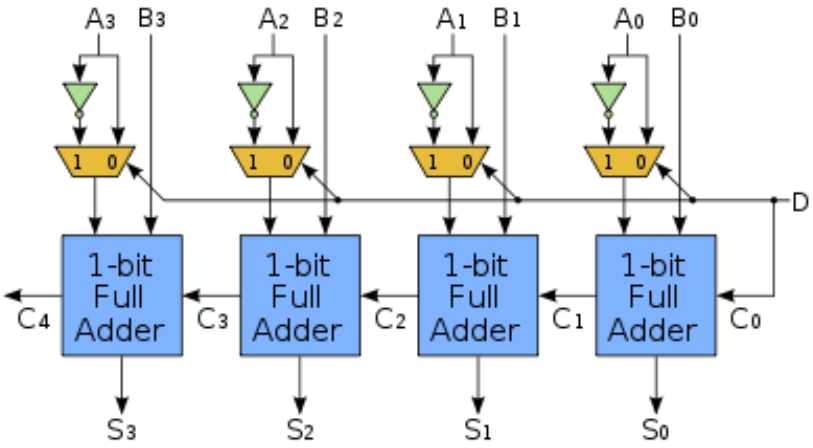
1. More on CLAs
    a. $C_{i+1} = G_i + P_iC_i$, where C is the carry
        i. $G_i = A_iB_i$, $P_i = A_i \oplus B_i$
        ii. Can expand this out
            1. $C_1 = G_0 + P_0 * C_0$
            2. $C_2 = G_1 + P_1 * C_1 = G_1 + P_1 * (G_0 + P_0 * C_0) = G_1 + P_1 * G_0 + P_1 * P_0 * C_0$
            3. $C_3 = G_2 + P_2 * C_2 = G_2 + P_2 * (G_1 + P_1 * C_1) = G_2 + P_2 * (G_1 + P_1 * (G_0 + P_0 * C_0))$
            $= G_2 + (P_2 * G_1) + (P_2 * P_1 * G_0) + (P_2 * P_1 * P_0 * C_0)$
            4. $C_4 = G_3 + P_3 * C_3 = G_3 + P_3 * G_2 + (P_3 * P_2 * G_1) + (P_3 * P_2 * P_1 * G_0) + (P_3 * P_2 * P_1 * P_0 * C_0)$
        iii. C_0 is the only carry that must be known for all these calculations
        iv. All of these expressions can be implemented with two levels of gates
            1. Longest path could always be two gates
            2. However, more inputs in an AND gate means it takes longer for the gate to resolve, as there are more transistors in series
            3. For more than 7 inputs, binary tree of AND gates could be faster, even though path is longer
    b. Process
        i. Each bit adder i calculates its P_i and G_i
            1. Takes one gate delay
        ii. CLA unit simultaneously calculates all carry for its adders
            1. Takes two gate delays
            2. Also calculates the carry for the next group of adders (C_4 in the picture above)
        iii. With the calculated carries, each bit-adder in the group simultaneously calculates its sums
            1. Only takes one gate delay, as carry and the XOR was already calculated for $P = A_i \oplus B_i$
    c. Can expand this 4-bit adder to further levels, like a 64-bit unit



2. Subtractors
    a. Create the 2's complement of the number to be subtracted
        i. Then add to other number
    b. Circuit below allows for that
        i. When wanting to subtract, D is set to 1
        ii. Multiplexers invert the entire number
        iii. Add one by setting carry in C_0 to 1



3. 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

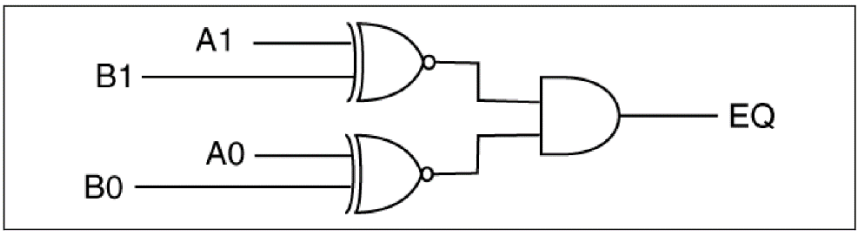


Figure 11.10: The final circuit for the 2-bit comparator as equation (e) in Figure 11.9.

4. Arithmetic logic unit (ALU)
    e. One unit that can perform multiple operations
        i. Example: do ADD, NOT, AND, OR, XOR, equality check, so forth
    f. 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

5. 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
        i. Measure of size of error – Hamming distance between original and final values
    c. Three possible outcomes
        i. No errors detected (may or may not be an error)
        ii. Error detected, and location specified, so we can correct it
        iii. Error detected, and location unspecified, so we can't correct it
    d. Will look at an error correction scheme that is SECSED
        i. SECSED – Single Error Correction Single Error Detection
        ii. Another scheme type – SECDED
            1. Single Error Correction Double Error Detection
    e. If the number of bits changed in a set is large enough, *any* error detection system will fail