Combinatorial Circuits

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Combinatorial logic circuits correspond to pure (state-free) calculations on booleans.

B. Outcomes

After this lecture, you should know

- Know what characterizes combinatorial logic circuits.
- Know some standard combinatorial logic circuits such as decoders and multiplexers (muxes), half- and full-adders, and programmable logic arrays.

C. Combinatorial Logic Circuits

- **Combinatorial Logic Circuits** correspond to pure calculations on boolean values (computations without an internal state).
  - Any circuit that corresponds to a propositional formula.
  - Built up from boolean inputs using boolean connectives (*AND*, *OR*, etc).
- **Sequential Logic Circuits** do have an internal state/memory that can affect the output and can change over time. (We'll see these later.)
- Examples of combinatorial circuits
  - Standard bitstring operations (*AND*, *OR*, etc).
  - Shift bitstring (e.g., 01101 to 11010 by left-shifting).
  - Add or subtract constant from bitstring.
  - Take 2's-complement negative of bitstring.
  - Add/subtract/multiply/divide/etc. two unsigned binary integers.
  - Convert 2's complement integer to/from floating-point.
  - Do arithmetic on floating-point values.

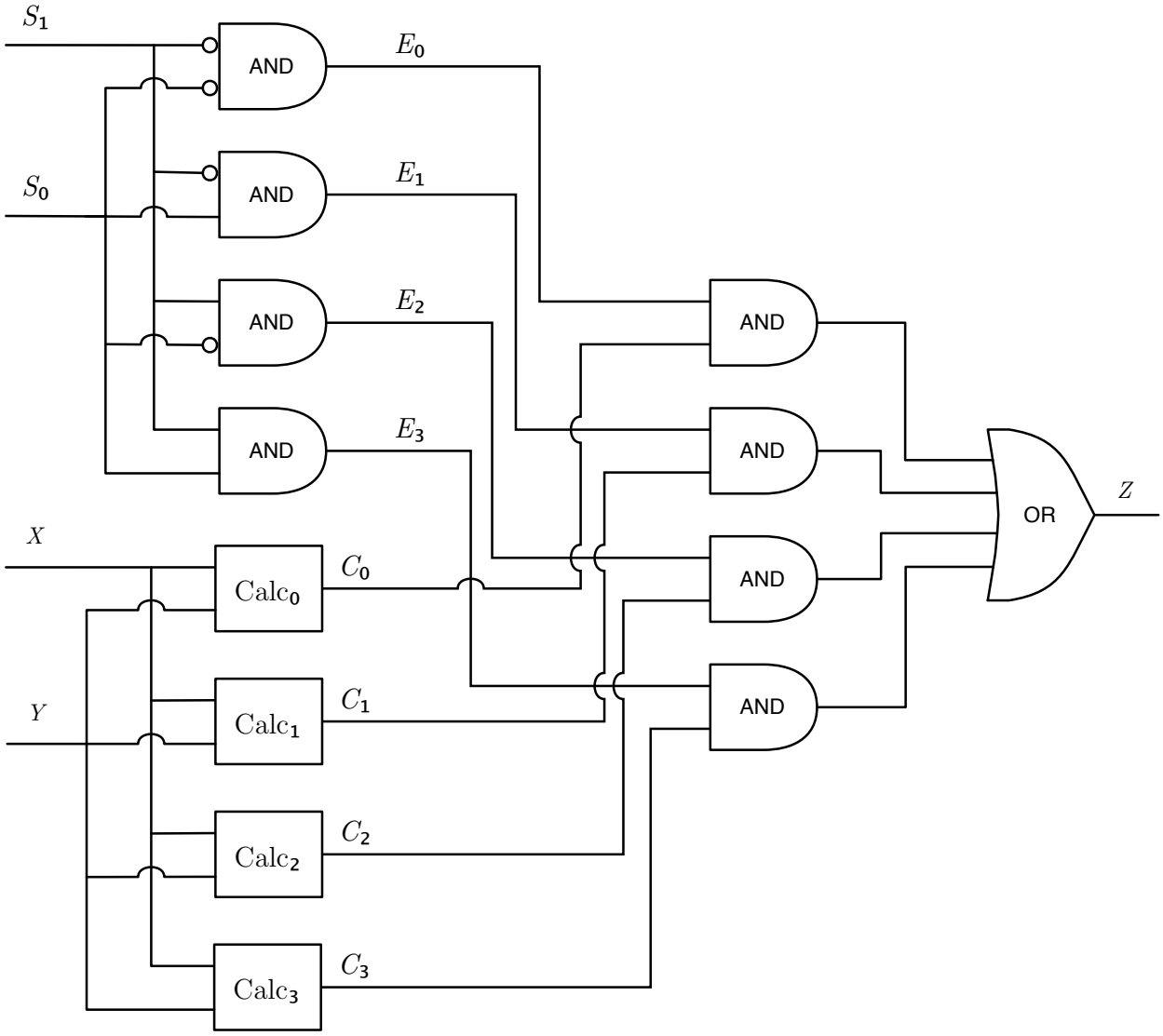
D. Four-Operation Calculator

- Let's look at designing a 4-operation calculator.
  - We'll have two data inputs X and Y and one operation selector S, a 2-bit string.
- To make things easier, let X and Y be one bit each.
  - The output Z is the *AND*, *OR*, *XOR*, or *XNOR* of X and Y.
- Which operation we do depends on S ($S = 00$ for *AND*, 01 for *OR*, 10 for *XOR*, and 11 for *XNOR*).
- For this example, let's write $\wedge$ for *AND*, $\vee$ for *OR*, $\oplus$ for *XOR*, and $\leftrightarrow$ for *XNOR*. Then
  - $S = 00 \rightarrow Z = X \wedge Y$
  - $S = 01 \rightarrow Z = X \vee Y$
  - $S = 10 \rightarrow Z = X \oplus Y$, and
  - $S = 11 \rightarrow Z = X \leftrightarrow Y$.

S	***Z***
00	$X \wedge Y$
01	$X \vee Y$
10	$X \oplus Y$
11	$X \leftrightarrow Y$

- Since $S = S_1 S_0$ (where juxtapositioning indicates a bitstring), we get
  - $Z = (\neg S_1 \wedge \neg S_0 \rightarrow X \wedge Y) \wedge (\neg S_1 \wedge S_0 \rightarrow X \vee Y) \wedge (S_1 \wedge \neg S_0 \rightarrow X \oplus Y) \wedge (S_1 \wedge S_0 \rightarrow (X \leftrightarrow Y))$
  - To implement this circuit, we can treat Z as a function of 4 variables, build up a 4-variable truth table, a 4-variable Karnaugh map, simplify the map, and get a minimal expression that calculates Z.
- Another way to calculate Z is to do all four calculations on X and Y in parallel but only select the one result we want. In the following circuit,
  - Each of the 4 "enable" lines E_0, E_1, E_2, and E_3 correspond to a different value of S. The line "selected" by S will be 1; the other 3 will be 0.
    - $E_0 = \neg S_1 \wedge \neg S_0$; $E_1 = \neg S_1 \wedge S_0$; $E_2 = S_1 \wedge \neg S_0$; $E_3 = S_1 \wedge S_0$.
  - The values of the four calculations $X \wedge Y$, etc. are given the names C_0, …, C_3.
    - $C_0 = X \wedge Y$; $C_1 = X \vee Y$; $C_2 = X \oplus Y$; $C_3 = X \leftrightarrow Y$.

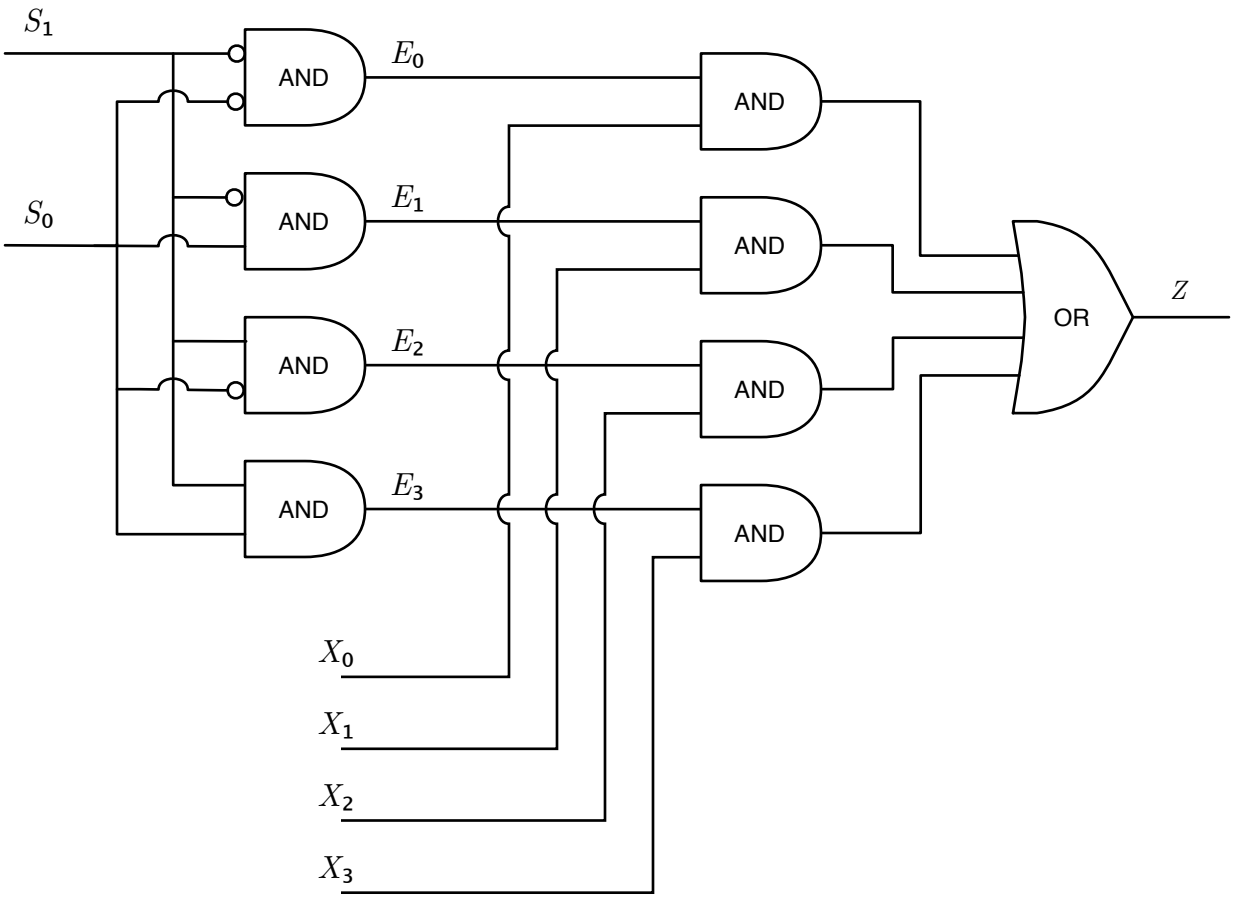
- So $Z = (E_0 \wedge C_0) \vee \ldots \vee (E_3 \wedge C_3)$
  - If you want the expansion, it's $Z = ((\neg S_1 \wedge \neg S_0) \wedge (X \wedge Y)) \vee ((\neg S_1 \wedge S_0) \wedge (X \vee Y)) \vee ((S_1 \wedge \neg S_0) \wedge (X \oplus Y)) \vee ((S_1 \wedge S_0) \wedge (X \leftrightarrow Y))$.
  - Each disjunct has the form "Do we want calculation number S?" ANDed with the value of calculation number S. (Think of S as 0, 1, 2, or 3.)
- This organization for Z uses two standard kinds of combinatorial circuits.
  - The circuit that takes S and calculates E_0, ..., E_3 is a **decoder**.
  - The circuit that takes S and C_0, ..., C_3 and calculates Z is a **multiplexer**.



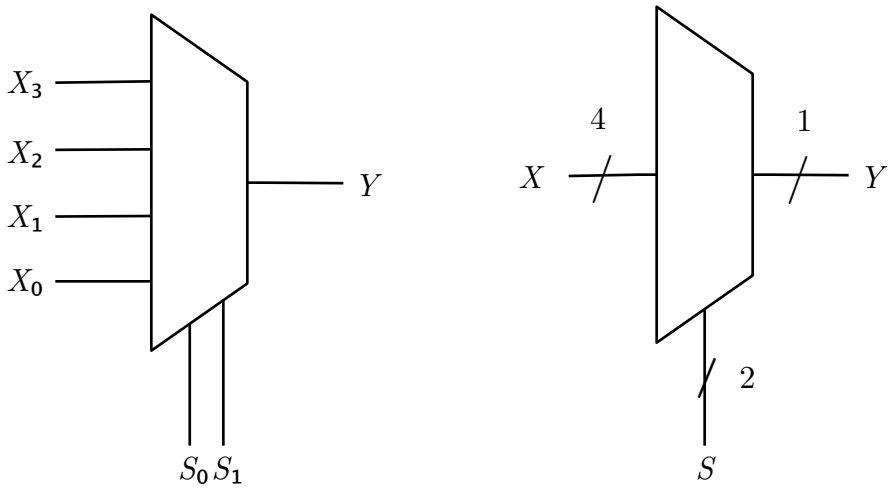
Four-op calculator: Inputs X and Y and operation selector $S = S_1S_0$

E. A 2^n-to-1 Multiplexer

- A **2^n-to-1 multiplexer** (or "**mux**") is a standard combinatorial circuit that has two kinds of input (data and selector) and one (data) output line. Exactly one input line gets connected to the output; which line that is depends on the selector.
- Let $N = 2^n$, then we have N data inputs (call them X_0, ..., X_{N-1}) and an n-bit selector input string S. The output $Y = X_S$ if we view S as an n-bit unsigned integer ≥ 0 and $< N$. The values of the other X_i are ignored.



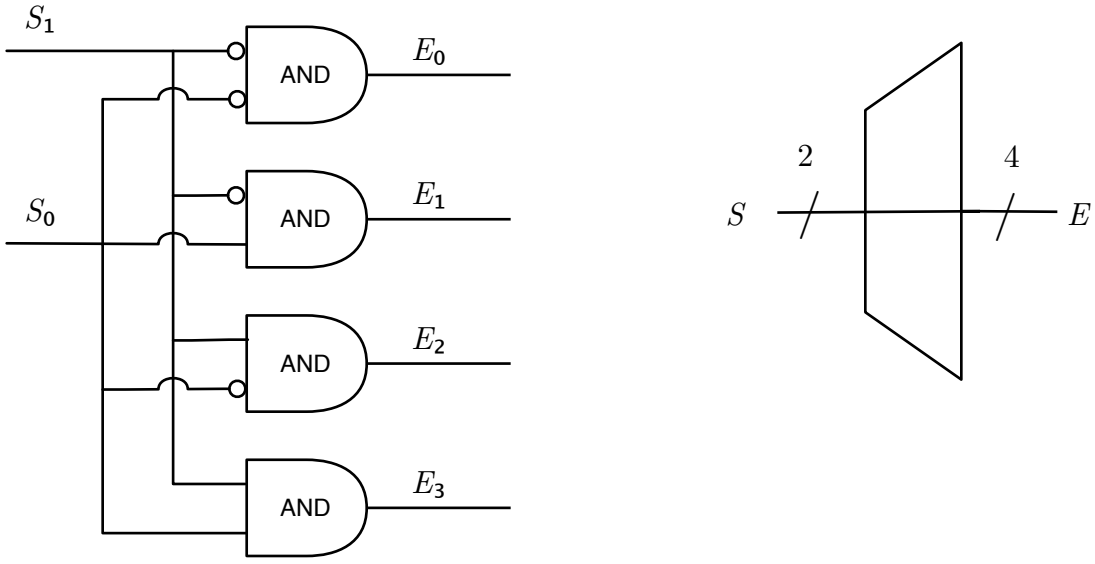
Circuit for 4-to-1 Multiplexer



Symbol for 4-to-1 Multiplexer

F. n-bit Decoder

- An ***n*-bit decoder** (also known as a **demultiplexer** or "**demux**") is a standard combinatorial circuit that takes an n-bit input (the **selector**) and has 2^n outputs, exactly one of which will be 1; the others will be 0.



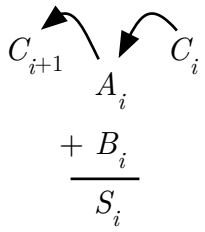
A 2-bit decoder

- If the input is S and the outputs are E_0, ..., E_{N-1} where $N = 2^n$, then the output $E_S = 1$ (where S is viewed as an unsigned n-bit integer ≥ 0 and $< N$)

and the other $E_i = 0$. E.g., for the 4-operator calculator example, we used a 2-bit decoder; the output line that was 1 specified the calculation we wanted.

G. Half Adders and Full Adders

- If we try to solve the general case of adding two n-bit unsigned integers without approaching the problem modularly, it's very complicated.
  - For each k, bit k of the result depends on the k–1 bits to its right. (Recall that bits are numbered right to left.)
- The modular approach mimics how we add numbers on paper: For each column, we add the two bits in that column plus a possible carry-in bit from the previous column. The result is the bit for this column plus a carry-out bit from this column.
- A full adder does the addition for one column: For column i, the 2-bit sequence $C_{i+1}\ S_i = A_i + B_i + C_i$ where A_i and B_i are the two bits being added, S_i is the sum bit, and C_i and C_{i+1} are the carry-in and carry-out bits



Addition of one column

- A **half-adder** doesn't have a carry-in bit but produces a sum and carry-out. (It acts like a full-adder with a carry-in of zero.)
- To add n bits, we use n–1 full adders and one half adder (for the rightmost column) and connect the carry out of each column to be the carry in of the next column.

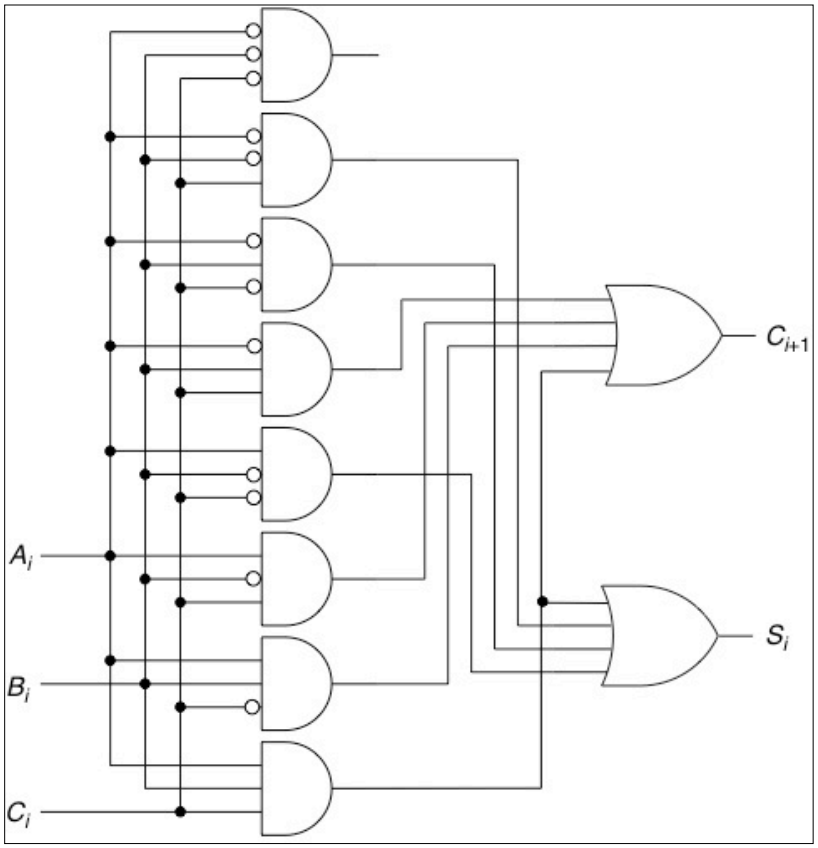


Fig 3.15: Full Adder

Combinatorial Circuits

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Combinatorial logic circuits correspond to pure (state-free) calculations on booleans.

B. Outcomes

After this activity, you should be able to:

- Implement some standard combinatorial logic circuits

C. Questions

1. A half subtractor implements one column of a subtraction, assuming there was no borrow out to the column to its right. We have inputs X_i and Y_i and outputs D_i and B_{i+1}; the borrow-in bit B_{i+1} and difference D_i is the result of subtracting X_i `-` Y_i. (Hint: we only borrow to calculate `0 - 1`.) Create a truth table for this operation with and give a full DNF representation for both D_i and B_{i+1}.
2. A 4-to-2 multiplexer takes 4 bits of data input $X[0{:}3]$ and uses 1 bit of selector input S to produce 2 bits of output $Y[0{:}1]$. If $S = 0$, then $Y_{\texttt{0}} = X_{\texttt{00}}$ and $Y_{\texttt{1}} = X_{\texttt{01}}$; if $S = 1$, then $Y_{\texttt{0}} = X_{\texttt{10}}$ and $Y_{\texttt{1}} = X_{\texttt{11}}$. Implement a 4-to-2 multiplexer using two 2-to-1 multiplexers.
3. Let $X[0{:}2]$ be a 3-bit unsigned number and let $Y[0{:}2]$ be the 3-bit result you get by incrementing X by 1, with variable $W = 1$ if overflow has occurred. (E.g. if $X = 011$ then $W = 0$ and $Y = 100$.) Write equations for outputs $Y_{\texttt{2}}$, $Y_{\texttt{1}}$, $Y_{\texttt{0}}$, and W from inputs $X_{\texttt{2}}$, $X_{\texttt{1}}$, and $X_{\texttt{0}}$; use logic gates to design a circuit for W, $Y_{\texttt{2}}$, $Y_{\texttt{1}}$, and $Y_{\texttt{0}}$.

Solution

1. (Half subtractor $X_i - Y_i = D_i$ with borrow-in B_{i+1}). We get $D_i = \overline{X}_i\ Y_i + X_i\ \overline{Y}_i$ and $B_{i+1} = \overline{X}_i\ Y_i$.

X_i	Y_i	D_i	B_{i+1}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

2. (Implement a 4-to-2 multiplexer with two 2-to-1 multiplexers): With one multiplexer, use S to select between X_{00} and X_{10} with output Y_0; with the other, use use S to select between X_{01} and X_{11} with output Y_1.

3. (Increment a 3-bit unsigned number $X[0{:}2]$ to get $Y[0{:}2]$ and overflow W.) $W = X_2\ X_1\ X_0$, $Y_0 = \overline{X}_0$, $Y_1 = X_1\ \overline{X}_0 + \overline{X}_1\ X_0$, and $Y_2 = X_2\ \overline{X}_1 + X_2\ \overline{X}_0 + \overline{X}_2\ X_1\ X_0$.

X_2	X_1	X_0	W	Y_2	Y_1	Y_0
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	0	0	1	1
0	1	1	0	1	0	0
1	0	0	0	1	0	1
1	0	1	0	1	1	0
1	1	0	0	1	1	1
1	1	1	1	0	0	0

