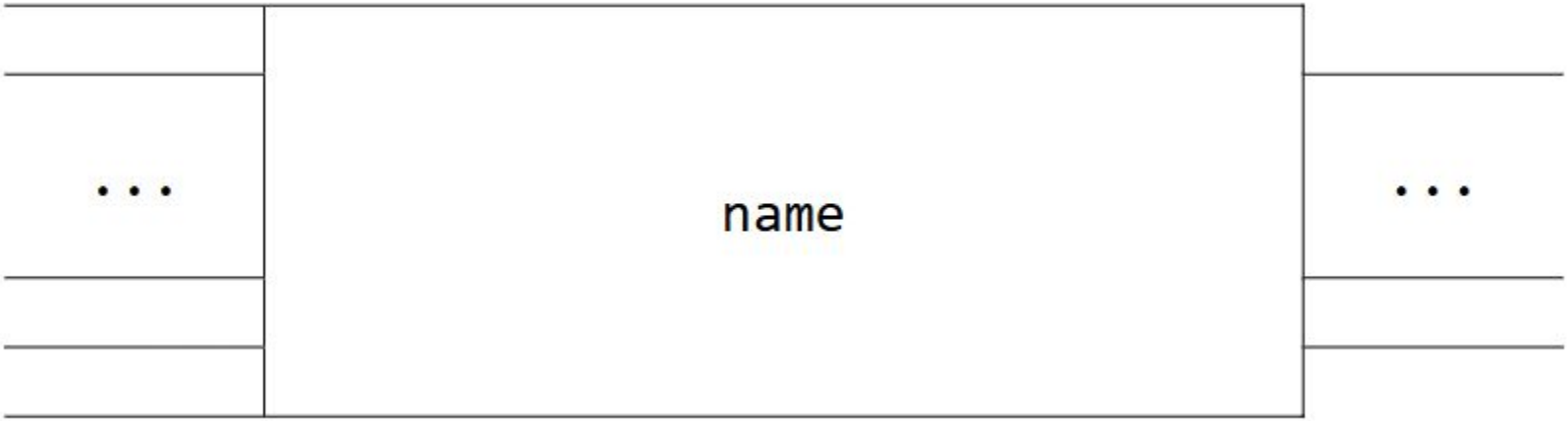
Combinatorial Circuits

Types of Circuits

- There are two types of circuits
  - Combinatorial Circuits
    - outputs are determined exclusively by its current inputs
  - Sequential Circuits
    - outputs may be determined with previous outputs

Combinatorial Circuit - Block Diagram

Diagram: block diagram of combinational circuit with m inputs and n outputs



Each output may be described by a Boolean function with *m* inputs.

Combinatorial Circuits

- Any circuit built by composing combinational components is also combinational
  - Composition must be without cycles
  - Omitted details for now: noise margin, timing specification
- An exhaustive study of combinational logic analysis and design is beyond the scope of this course
  - Analysis → From circuit to description
  - Design → From description to circuit

Types of Combinatorial Circuits

Luckily, in order to implement a basic computer, only a small number of fundamental combinational components are needed, including:

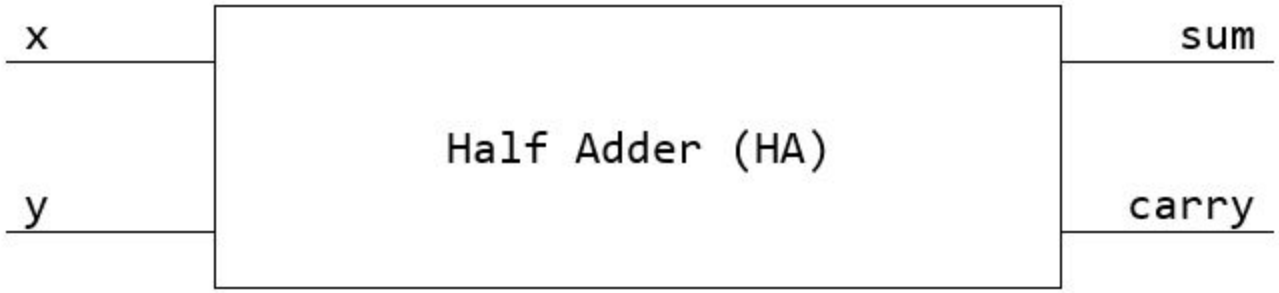
- adders (and related circuits)
- decoders
- encoders
- multiplexers

We will limit our study of combinational circuits to these (and possibly a few others).

Half Adder (HA)

A half-adder (HA) adds two bits (x, y) to produce sum and carry bits (s, c).

Diagram: HA block diagram



Half Adder (HA)

Determine the truth table for the Half-Adder

x	y	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

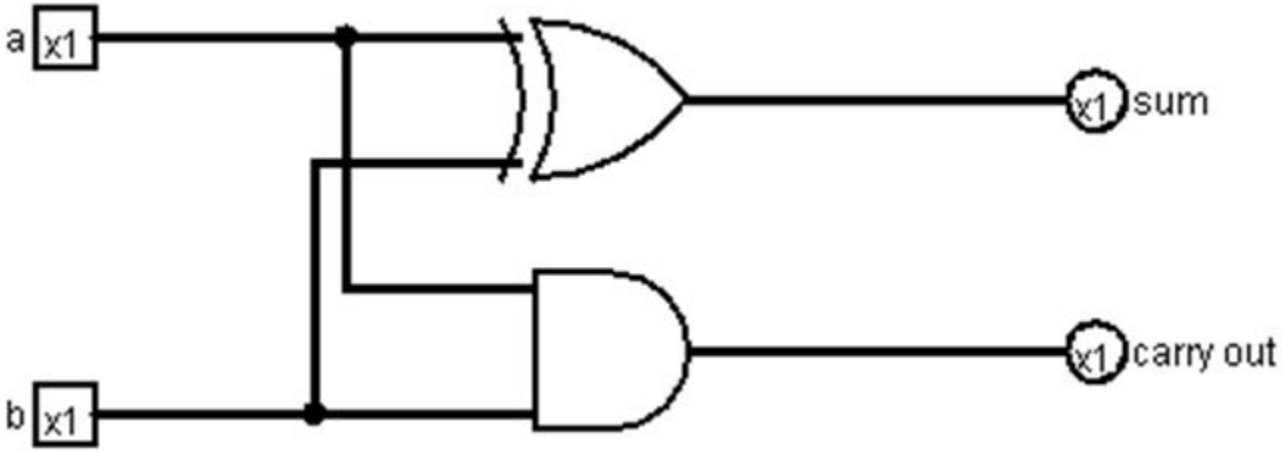
Half Adder (HA)

- Derive the boolean function(s) to generate the outputs
  - Thus, can look either for the SOP or POS based on the Truth Table
  - OR in this case since there are only 2 inputs does the output match a known gate?

- The sum column matches an XOR gate
- The carry column matches an AND gate.
- So $sum = x \oplus y$ and $carry = x \cdot y$

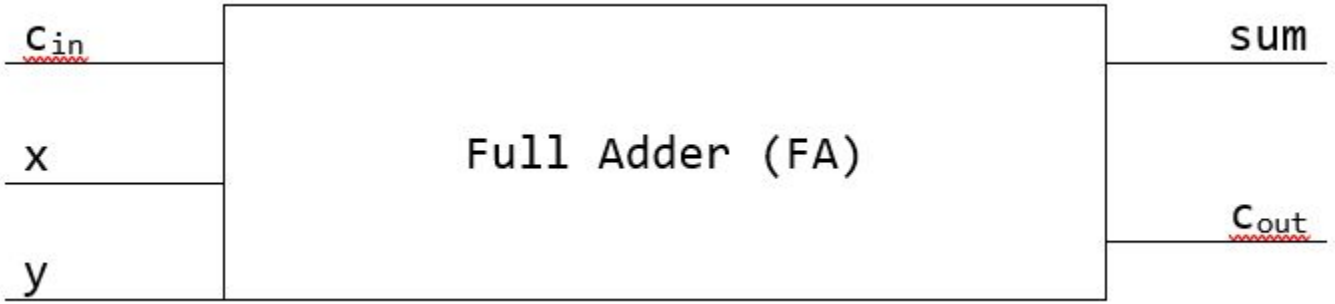
Half Adder (HA)

Create the corresponding logic diagram using gates but no other electronic components



Full Adder (FA)

- Full Adder does, so it adds three bits (x, y, c_{in}) to produce sum and carry bits (c_{out}, s)
- **Diagram:** FA block diagram



Full Adder (FA)

Determine the truth table for the Full-Adder

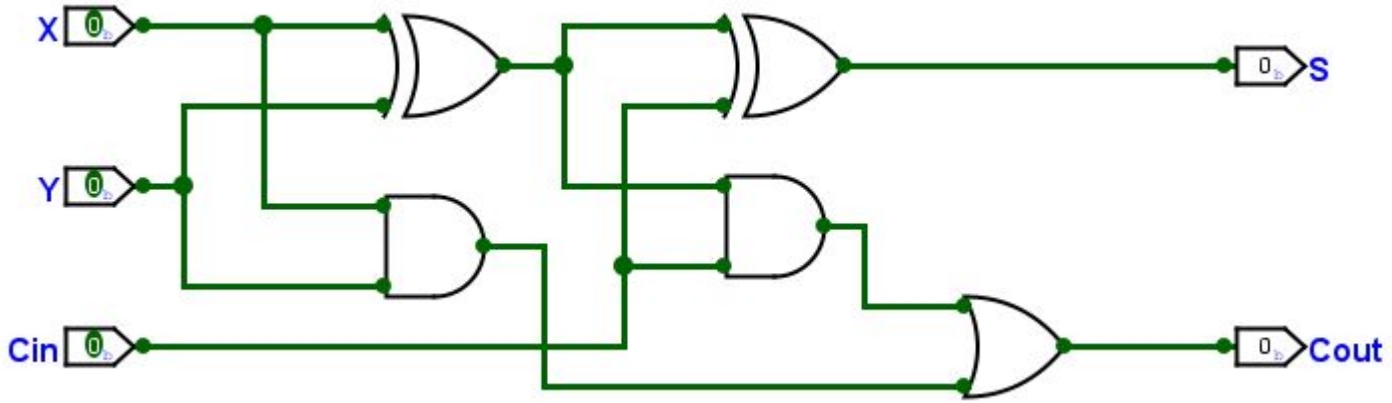
x	y	c_{in}	c_{out}	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full Adder (FA)

- Derive the boolean function(s) to generate the outputs
  - Thus, can look either for the SOP or POS based on the Truth Table
  - There are more than 2 inputs
    - Likely not a single gate for each output
- $s = x'y'c_{in} + x'yc_{in}' + xy'c_{in}' + xyc_{in}$
  $= x \oplus y \oplus c_{in}$ (remember operator precedence)
- $c_{out} = x' \cdot y \cdot cin + x \cdot y' \cdot cin + x \cdot y \cdot cin' + x \cdot y \cdot cin$
  $= cin \cdot (x' \cdot y + x \cdot y') + x \cdot y$
  $= cin \cdot (x \oplus y) + x \cdot y$

Full Adder (FA)

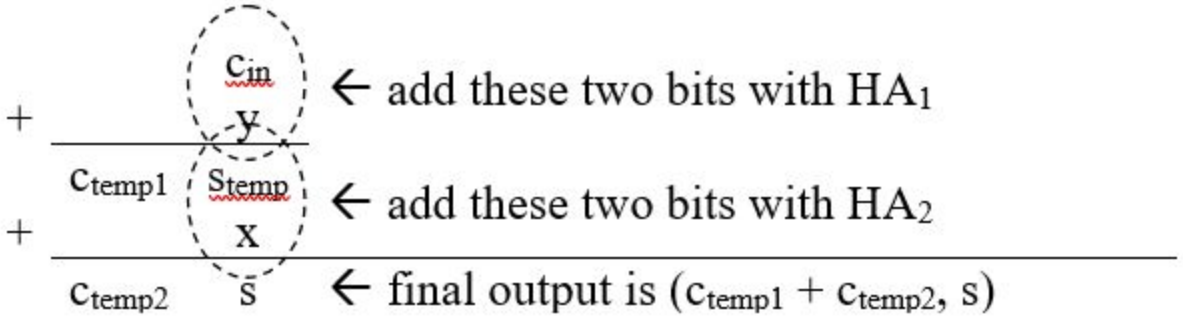
Design the corresponding circuit



Note: Both C_{out} and S have an $x \oplus y$ term, one gate used for both terms

Full Adder (FA)

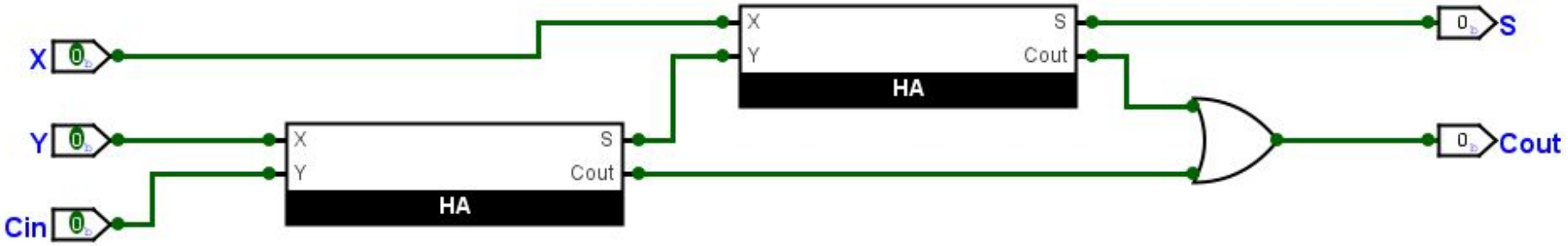
- Observe that the FA circuit can be implemented using HA circuits
  - Chain two HA circuits, with an OR of the two carry out bits
- Idea:



  - Prove this works by tracing through all cases for the two possible values of x
  - **Note:** The two temporary carries can't both be 1 simultaneously - why?

Full Adder (FA)

- Draw the logic diagram using HA circuits

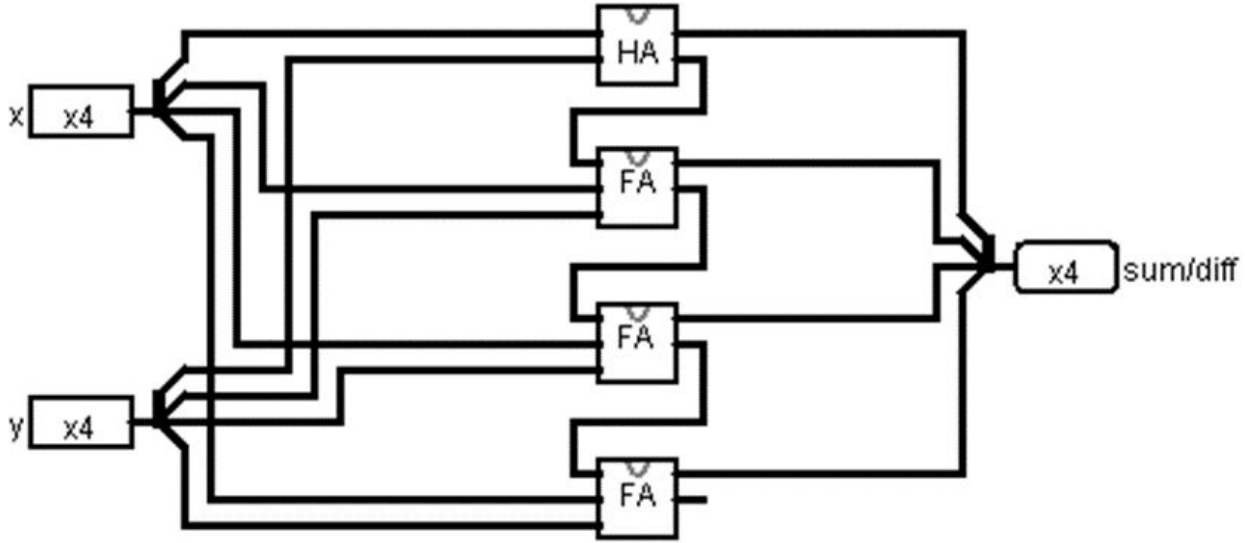


- The FA thus obtained is the simplest one

Complete Adder/Subtractor

- An *n*-bit adder can be obtained by combining one HA and *n*-1 FA circuits

Diagram: 4-bit adder - uses the “ripple carry” technique



Complete Adder/Subtractor

- This can be generalized to an n-bit adder/subtractor!
- **Recall:** to compute x – y, compute x + $NEG_{2comp}(y)$.
- **Recall:** to compute $NEG_{2comp}(y)$, flip all its bits and add 1.
- So what? How can the $NEG_{2comp}(y)$ be implemented?
- Stated in English we say flip the bits,
    - i.e. if the bit is 0 then it becomes a 1, but if it is a 1 it becomes a 0
- Is there a gate/circuit that does this?
    - Yes!! AN XOR

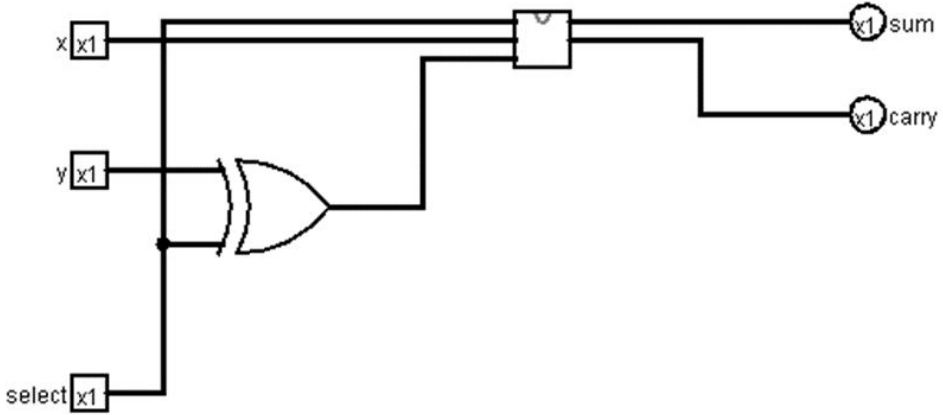
Complete Adder/Subtractor

- To use a single circuit, we need to add a selection bit
  - Selects between add(0) and subtract(1)
- In order to add 1 we can use a FA, instead of the HA circuit
  - c_{in} will be the selection

selection	input (y)	(x+y)(xy)’
0	0	0
0	1	1
1	0	1
1	1	0

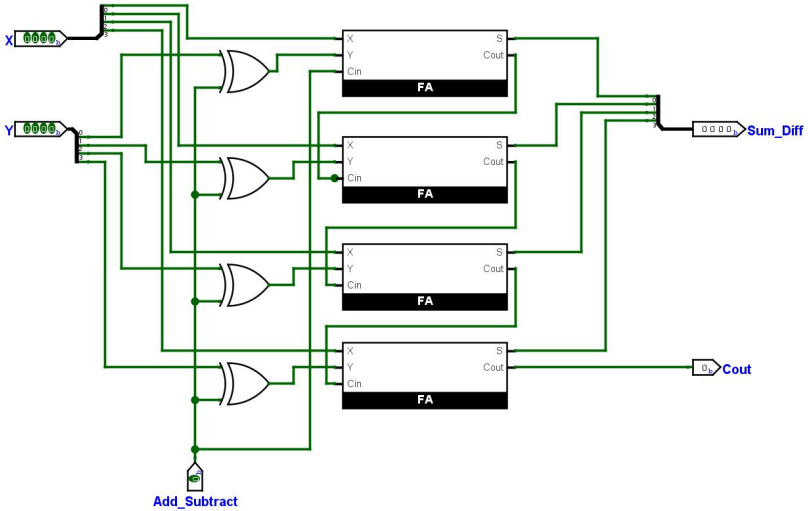
Complete Adder/Subtractor

- If select == 0 then doing a normal addition
- If select == 1 then doing a subtraction. In the case of x == y == 0 it may seem strange that the carry is set, but realize this is NOT going to the C condition code



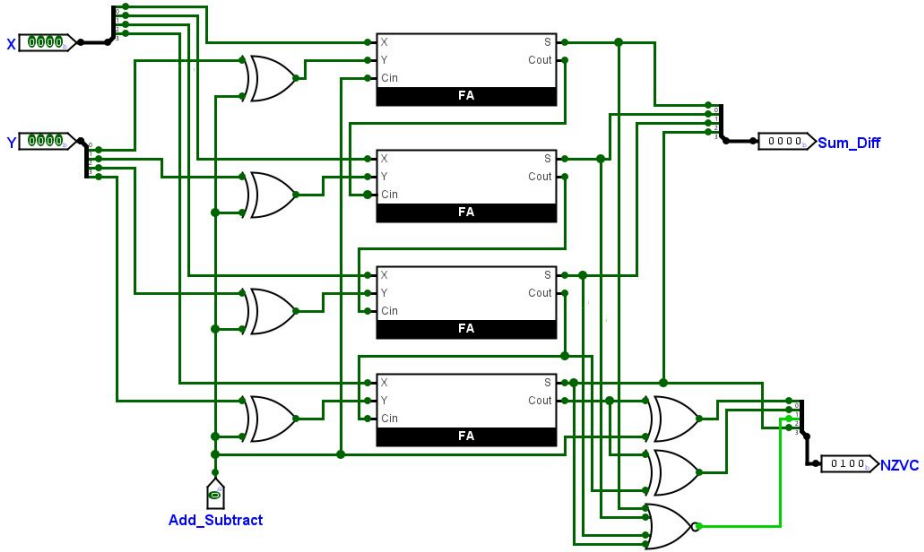
Complete Adder/Subtractor

- Instead of calling it select we will call it Add/Sub or A/S
- Exercise: generalize the 4-bit adder so that it has an additional A/S input. When A/S = 0, the circuit computes x + y. When A/S = 1, it computes x – y



Complete Adder/Subtractor

- It is useful for arithmetic circuits to generate N, Z, V and C condition codes (CCs)
- Exercise: add N, Z, V and C outputs to the circuit



Complete Adder/Subtractor

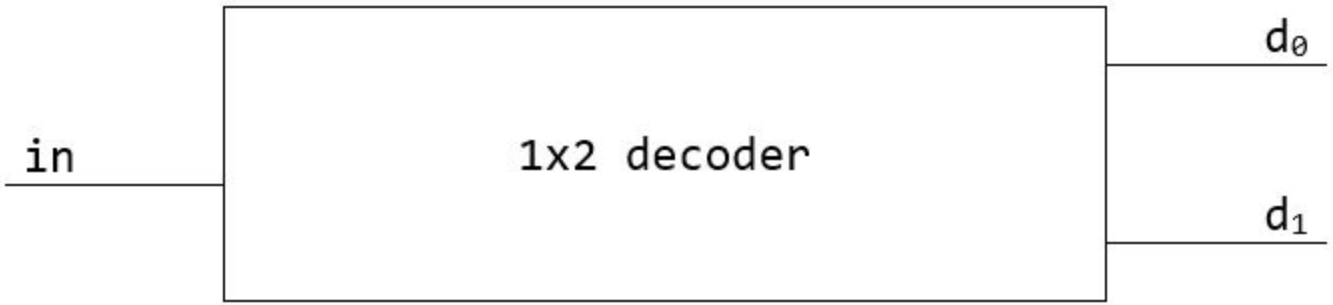
- **Note:** the carry out from the current add/sub propagates to the carry in to the next add/sub
- For larger word sizes, ripple carry is too slow.
- A faster alternative is carry-lookahead, which is beyond the scope of this lecture

Decoder

- **Recall:** an n-bit value represents 2^n distinct values.
- A decoder is a component which has n inputs and up to 2^n outputs, such that a binary coded value which is presented on the input produces an active signal on only ONE corresponding output
  - Not all all possible values need to be represented in the output!!
  - A maximized decoder is commonly used
    - Even if not all output lines are connected

Decoder

- E.g. a 1×2 decoder has one input and two outputs.
  - If the input is 0, then the first output is active
  - Otherwise, the second output is
- **Diagram:** 1x2 decoder block diagram



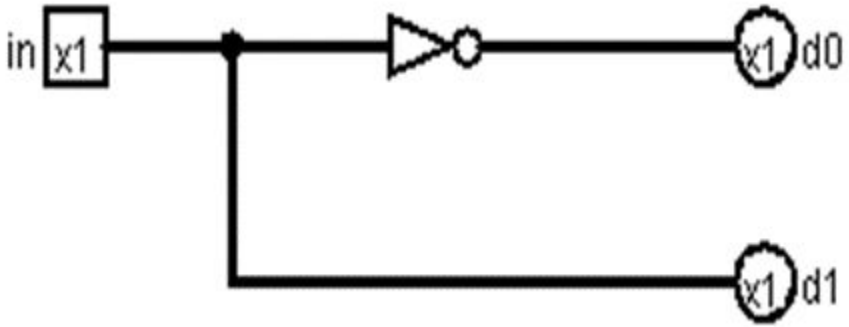
Decoder - How does it work

- To understand this examine the truth table for the 1x2 decoder
- Notice:
  - `d0 is the opposite of in`
  - `d1 is the same as in`

in	d0	d1
0	1	0
1	0	1

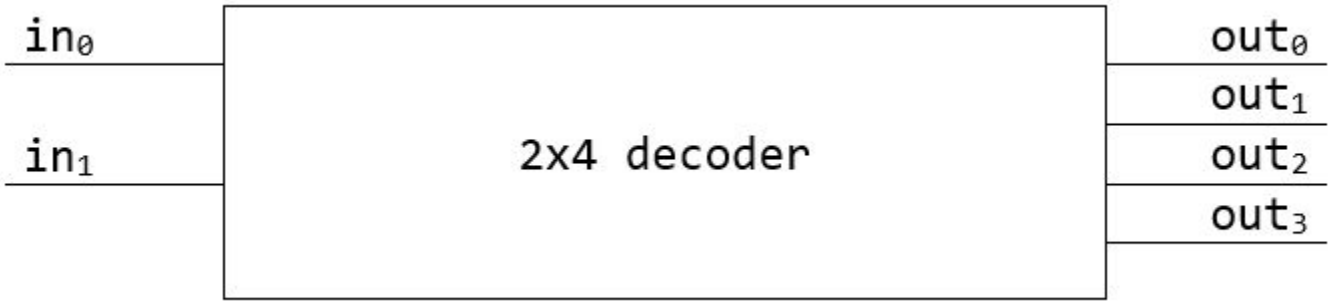
Decoder - How does it work

- **Diagram:** 1x2 decoder logic diagram



Decoder Exercise

- Draw the block diagram for a 2x4 decoder



Decoder Exercise

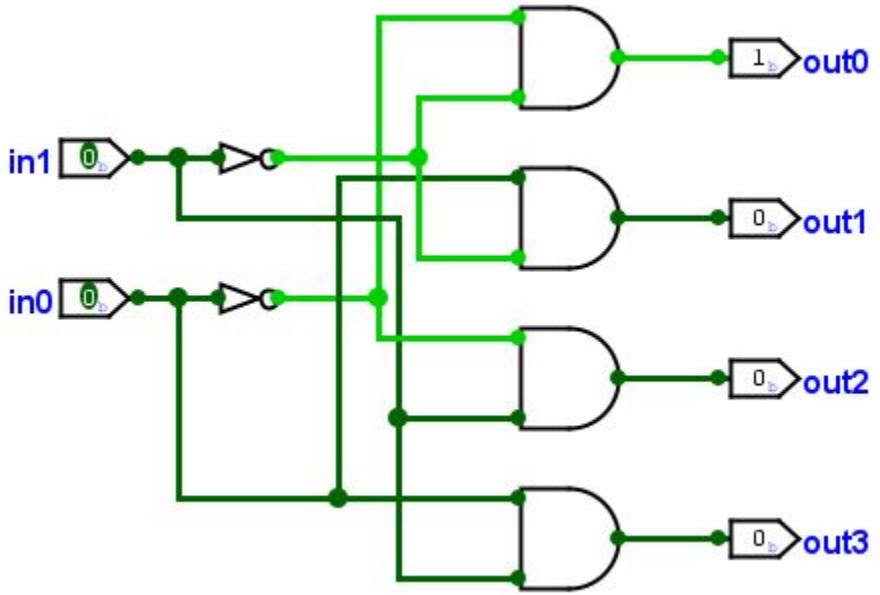
- Determine the truth table for the Determine the truth table for the 2x4 decoder2x4 decoder
- Determine the boolean equations for each output

in1	in0	out0	out1	out2	out3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

```
out0 = in1'·in0'
out1 = in1'·in0
out2 = in1·in0'
out3 = in1·in0
```

Decoder Exercise

- Determine the logic diagram for the 2x4 decoder



Decoder Summary

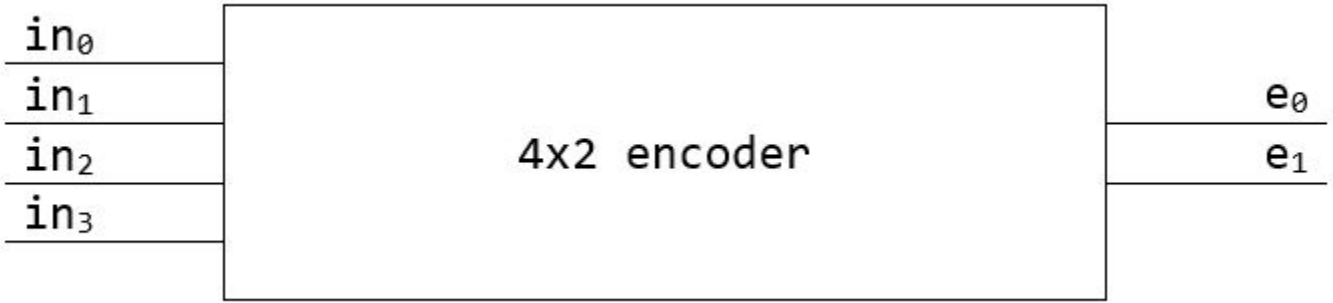
- At a basic level, decoders can be thought of as a binary to decimal converter OR a selector
- Decoders are useful in many instances
  - Example:
  - a machine language opcode contains encoded information (operation, addressing modes, etc)
  - decoders are used to extract the relevant information from each field of the opcode (i.e. size bits in 68000)

Encoders

- Performs the inverse operation of a decoder.
- Has 2^n inputs and up to n outputs.
- In a standard encoder,
  - Exactly one input is assumed to be active at any one time
  - Otherwise, the output is undefined.
  - Thus, the one input signal produces up to n active output signals that represent a binary coded value

Encoders Exercise

- Draw the block diagram for a 4x2 encoder



Encoder Exercise

- Determine the truth table for a 4x2 encoder

in3	in2	in1	in0	e1	e0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Encoder Exercise

- Convert to boolean functions for e0 and e1

in3	in2	in1	in0	e1	e0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

- For each of these pairs, notice for the red boxes that if in1 is 0 or 1 then e0 is also either 0 or 1. Thus, in this case e0 follows in1.
- This is the same for the green boxes, in this case e0 follows in3

Encoder Exercise

- Determine the truth table for a 4x2 encoder

in3	in2	in1	in0	e1	e0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

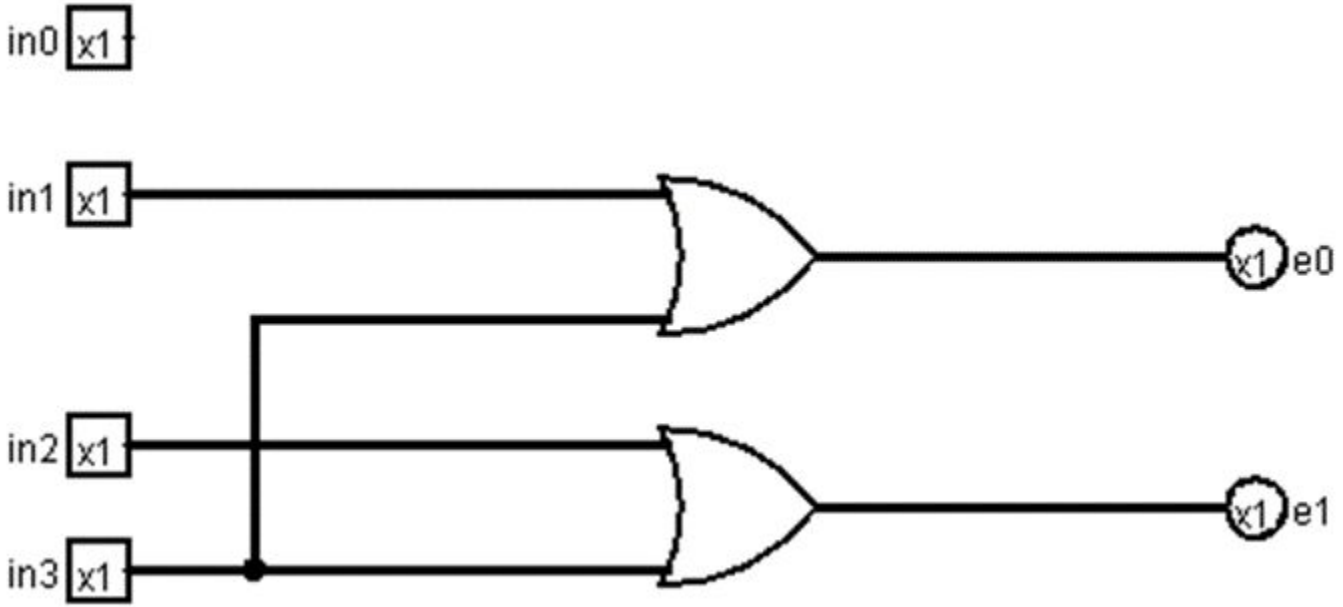
- For each of these pairs, notice for the red boxes that if in1 is 0 or 1 then e1 is also either 0 or 1. Thus, in this case e1 follows in3.
- This is the same for the green boxes, in this case e1 follows in2

Encoder Exercise

- The previous two slides give us our boolean formula for the two outputs
  - `e0 = in1 + in3`
  - `e1 = in2 + in3`
- This should make sense given the values between 0 and 3
  - in binary which of these numbers require setting bit 0
    - 1 and 3.
- Similarly which of these require setting bit 1
    - 2 and 3.

Encoder Example

- Determine the logic diagram for the 4x2 encoder

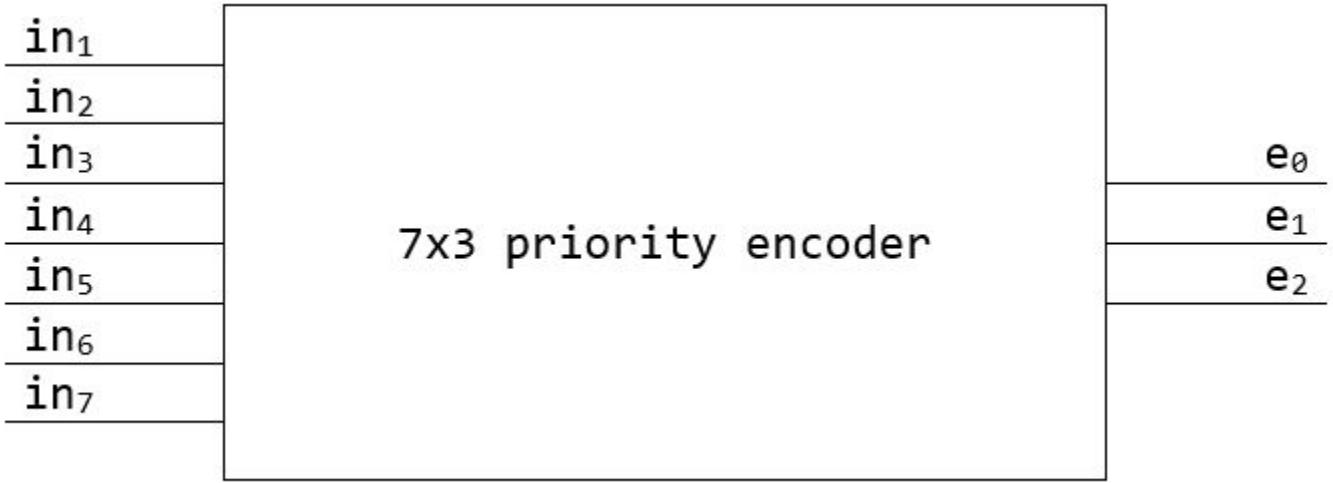


Priority Encoder

- Multiple inputs may be active simultaneously
- The output corresponds to the highest numbered input which is active
- It is common for the “zero” input to be omitted,
    - The priority encoder outputs a zero if no inputs are active

Priority Encoder

- **Diagram:** 7x3 priority encoder block diagram
  - Note: the in_0 input is not included



Priority Encoder

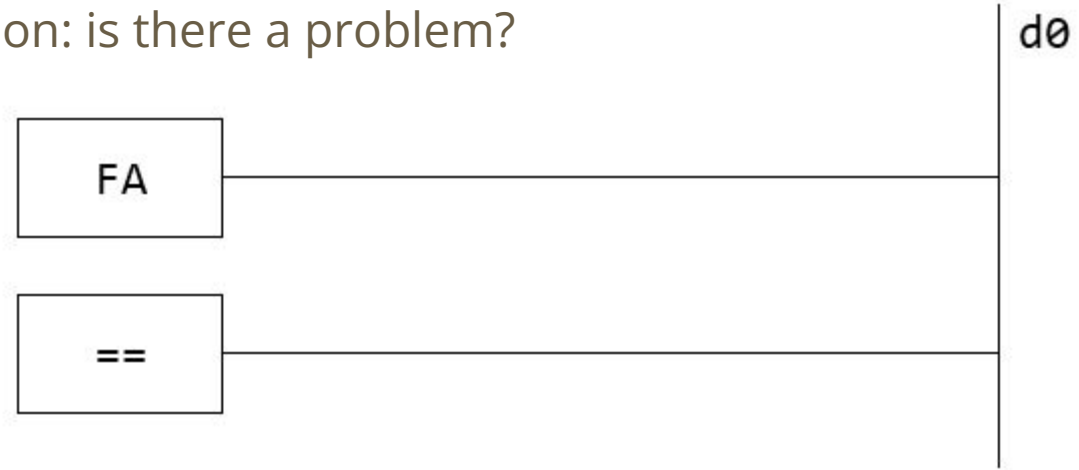
- For a normal 7x3 encoder with only one input allowed the following are the Boolean functions:
  - `e2 = in4 + in5 + in6 + in7`
  - `e1 = in2 + in3 + in6 + in7`
  - `e0 = in1 + in3 + in5 + in7`
- In a priority encoder multiple inputs are allowed but only the highest input is allowed
  - The above functions don’t work
  - If in4 and in3 are both on the above would generate 111 not the required 100.

Priority Encoder

- Priority encoders are determined using a different method than truth tables, called Karnaugh maps
- Karnaugh maps are well beyond the scope of this course and will not be covered, so deriving a priority encoder will NOT be tested
- One way to generate the appropriate function is to use the above functions, but the negation all of the higher inputs are added to all lower input terms
- `For example: e2 = in7 + in7'in6 + in7'in6'in5 + in7'in6'in5'in4`
- While this does generate a Boolean function it is not the most efficient way of doing this, nor the most efficient function.

Multiplexers

- Recall: a bus interconnects ≥ 2 functional units.
- For a given bus line, there may be several potential writers.
- **Diagram:** two units with outputs connected directly to the same line
  - Question: is there a problem?



Multiplexer

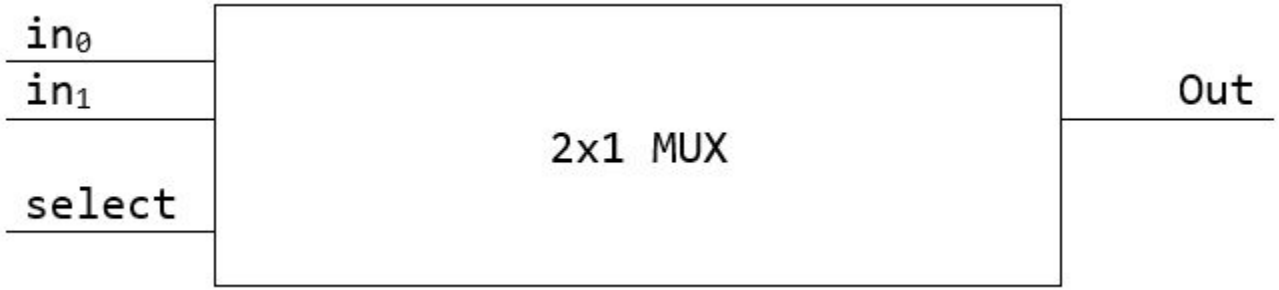
- **YES!**
- If both items try to write to the bus at the same time
  - One will get to this line first.
- This means that the voltage will be flowing on the wire to the second item in the wrong direction. The output from a gate is setup to send power, it is not setup to receive power. This will have an adverse effect on the gate – destroy it!!

Multiplexer

- The outputs of two gates must not be connected!
  - This generates short circuits.
- A multiplexer (MUX) is a component which allows multiple writers to share the same bus by allowing one-at-a-time writing.
- **A MUX is simply a selector**

Example

- a 2×1 MUX has 3 inputs (a, b, sel) and one output. If sel = 0 then a is placed on the output, otherwise sel = 1 and then b is placed on the output
- Diagram: 2×1 MUX block diagram



Example

- Let's examine the truth table for the 2x1 MUX
  - If select=0 choose in0
  - If select=1 choose in1
- Normal use of min/max terms can be used to calculate the circuit

in0	in1	select	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

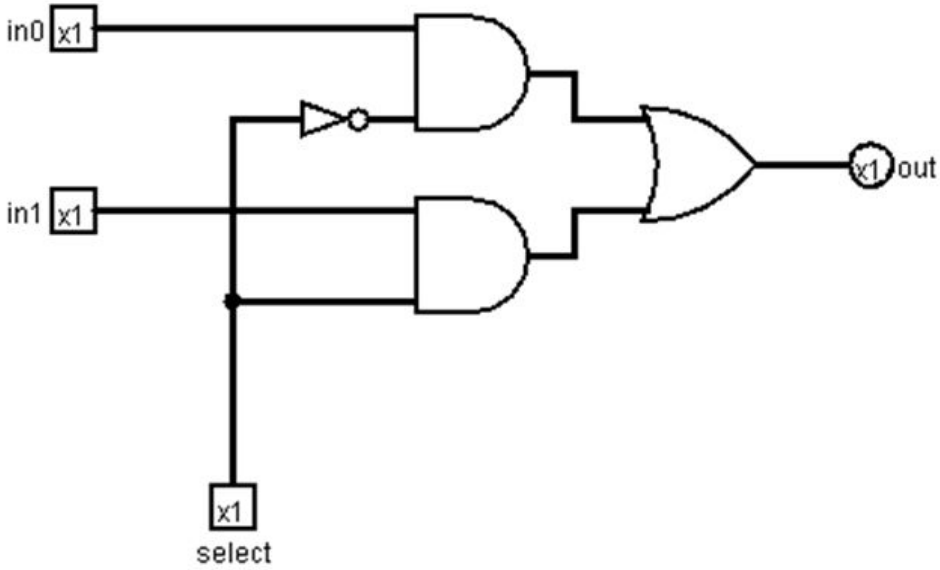
Example

- Or look at the highlighted lines
- The two orange lines have the common term `in1·select`
- The two green terms have the common term `in0·select'`
- Use distributivity to make those two terms the final ones
- `out = in0·select'+in1·select`

in0	in1	select	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

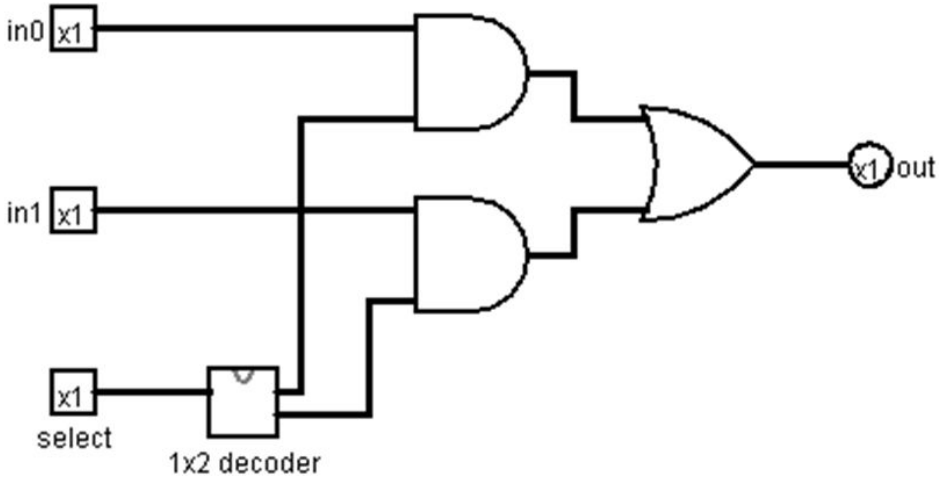
Example

- `out = in0·select'+in1·select`
- Gives the circuit to the right
- Look at the select portion of the circuit
    - Where was it used before?



Example

- The select portion of the circuit is a 2x1 decoder
- Thus the circuit can be rewritten

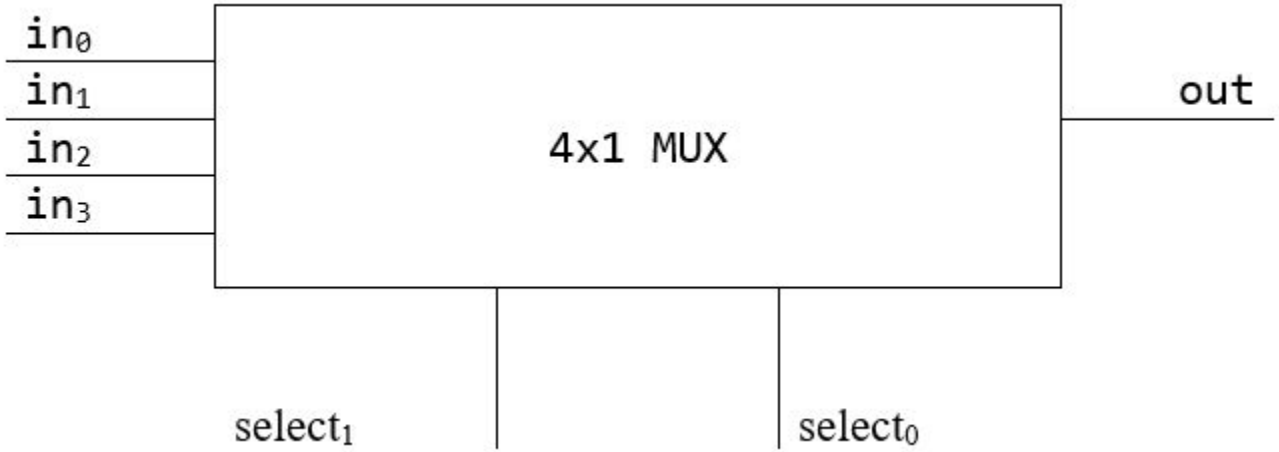


4x1 MUX

- How many inputs does a 4x1 MUX have?
- 6 inputs
- 4 sources & 2 selects
- # of selects = roundup($\log_2$ # of sources)
  - $2^{\text{\# of selects}}$ = # of sources

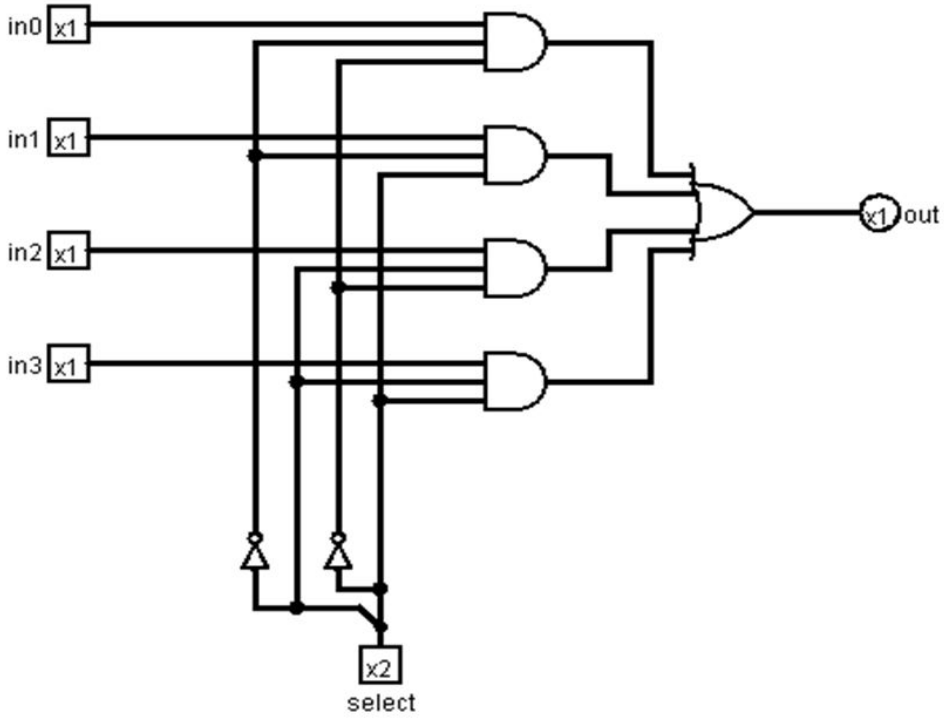
4x1 MUX

- The block diagram for the 4x1 MUX
- the two select lines can be viewed as a binary number and the value they represent will be the input line that is selected



4x1 MUX Logic Diagram

- As an exercise determine:
  - Truth table
  - Boolean equation
- Additional Exercise:
  - Convert the logic diagram to use a 1x4 decoder for the select lines



Additional uses for a MUX

- It is trivial to implement any n-input Boolean function using a $2^n \times 1$ MUX
  - If gate count is not an issue
- Type 1: n inputs, where 2^n is a reasonable sized MUX
  - all inputs are used as selection
  - the output, from the truth table, will be the input for that combination of input values in the selection

Additional uses for a MUX

- For any truth table the function results in either 0 or 1. As already done the Boolean function can be created via SOP or POS and simplification. Realize that the function and corresponding circuit generates either a 0 or 1 depending on the input
- It can also be done via a MUX, in this case there are 8 input combinations, which become the selection inputs
- The inputs to the MUX are the function results. Thus, the selection input will select the appropriate line from the truth table

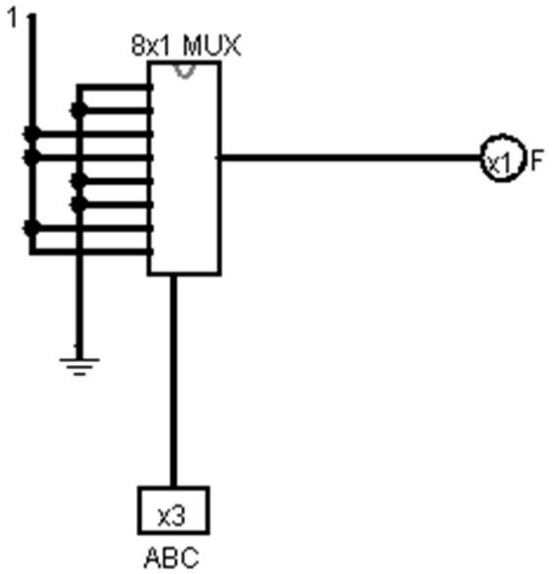
Additional uses for a MUX

- Use the following truth table:
  - $2^3 = 8$ which is reasonable

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Additional uses for a MUX

- Will generate the following logic diagram



A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Additional uses for a MUX

- Type 2 – removing 1 of the inputs to get to n inputs, where 2^n is reasonable
  - In the following case with 4 inputs, $2^4 = 16$ while not totally excessive can be used as an example
  - Truth table on the next slide
  - In the truth table
    - Term # is the Binary Coded Decimal (BCD) of terms ABCD
    - I is the BCD of terms BCD I_x is I with BCD of X

Additional uses for a MUX

A	B	C	D	F (output)	Term #	I
0	0	0	0	0	0	0
0	0	0	1	1	1	1
0	0	1	0	0	2	2
0	0	1	1	1	3	3
0	1	0	0	1	4	4
0	1	0	1	0	5	5
0	1	1	0	0	6	6
0	1	1	1	0	7	7
1	0	0	0	0	8	0
1	0	0	1	0	9	1
1	0	1	0	0	10	2
1	0	1	1	1	11	3
1	1	0	0	1	12	4
1	1	0	1	1	13	5
1	1	1	0	1	14	6
1	1	1	1	1	15	7

Additional uses for a MUX

- Implementation table:
  - Notice that I subscript refers to the BCD input value in decimal
- Draw a table with columns for I0 to I7 and a leading column with two rows for the A values 0 and 1
  - Notice there are eight Ix values

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A' (0)								
A (1)								

Additional uses for a MUX

- Under the I_x headings put 0-7 in the A' row and 8-15 in the A row
- Realize that these are the possible input values
  - $A'I_0$ really means A'B'C'D'
  - AI_0 really means AB'C'D'
- From the F column in the truth table circle the Term #s (0-15) that are 1.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A' (0)	0	(1)	2	(3)	(4)	5	6	7
A (1)	8	9	10	(11)	(12)	(13)	(14)	(15)

Additional uses for a MUX

- Now in the last row under the Ix headings use the following rules to put the correct value:
  - if both values are NOT circled, put a 0
  - if both values are circled, put a 1
  - if only the top value is circled, put A’
  - if only the bottom value is circled, put A

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A' (0)	0	(1)	2	(3)	(4)	5	6	7
A (1)	8	9	10	(11)	(12)	(13)	(14)	(15)
	0	A'	0	1	1	A	A	A

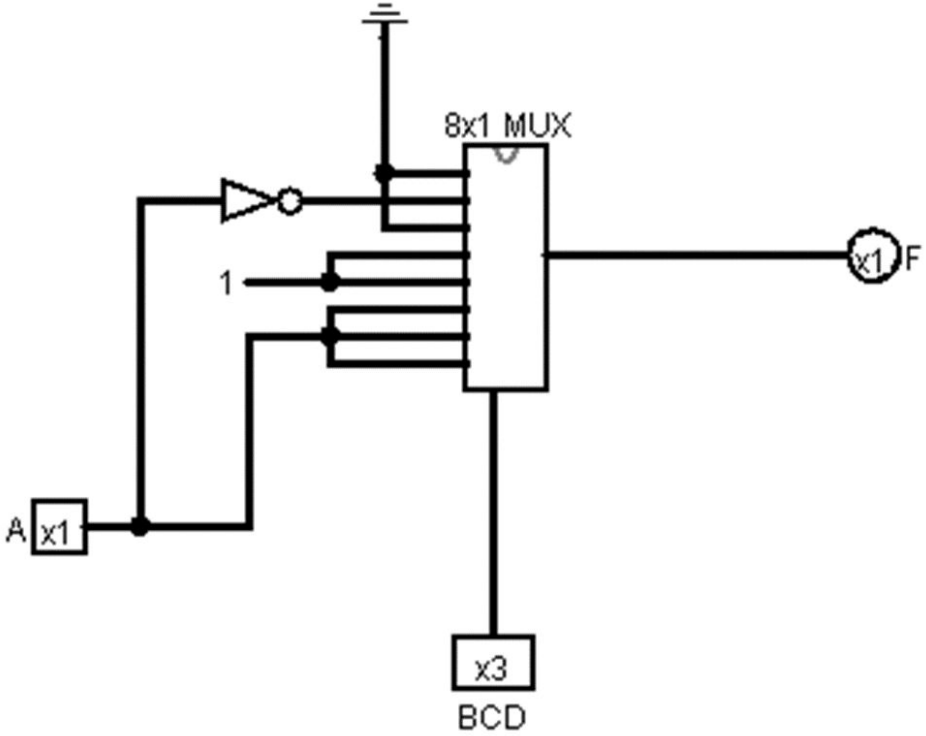
Additional uses for a MUX

- Now the values in the last row are the input value for the Ix input to the MUX
- From the four terms (A,B,C,D) that were there originally there are now only three (B,C,D) that will be generating the selection for the MUX
- From this the corresponding circuit can be created

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A' (0)	0	(1)	2	(3)	(4)	5	6	7
A (1)	8	9	10	(11)	(12)	(13)	(14)	(15)
	0	A'	0	1	1	A	A	A

Additional uses for a MUX

- Type 2 Logic Diagram



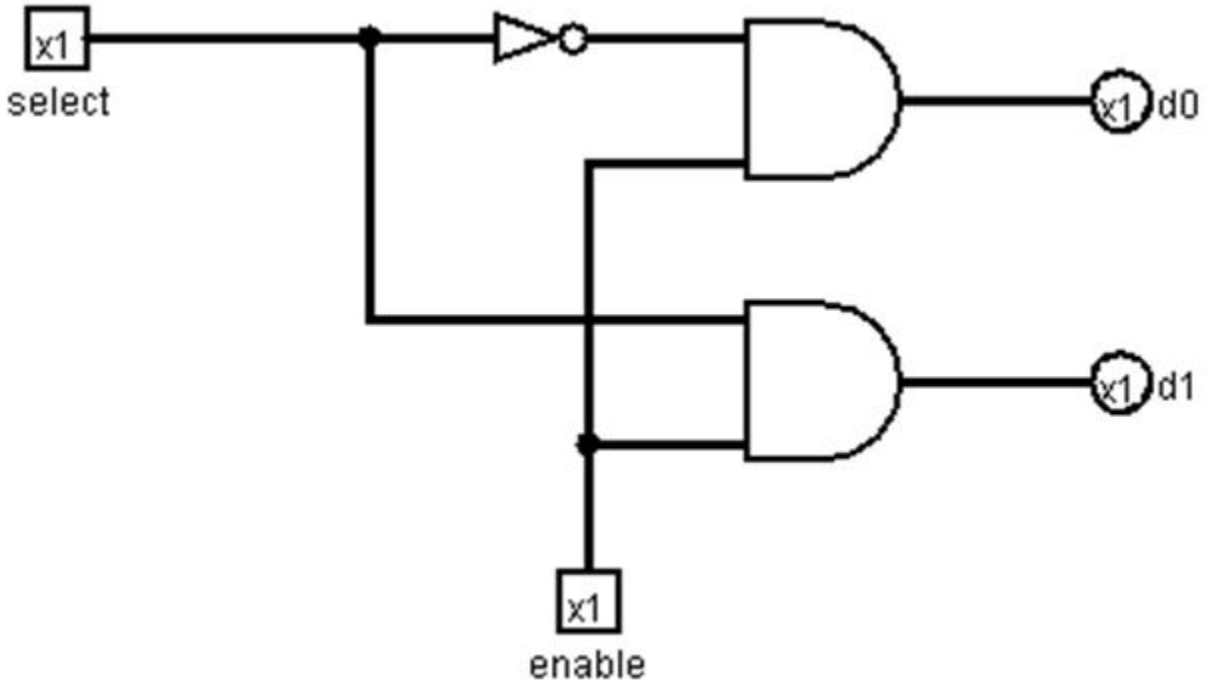
Additional uses for a MUX

- Multiplexers are often combined
    - Example: a 4-bit bus with 4 potential writers. A quadruple 4×1 MUX would allow each of the 4 sources to be able to place a 4-bit value on the bus
- MUXs (as well as decoders and encoders) commonly have an additional “enable” signal. The circuit performs as described if enable is active. Otherwise, it outputs all zeros

Enable Signal

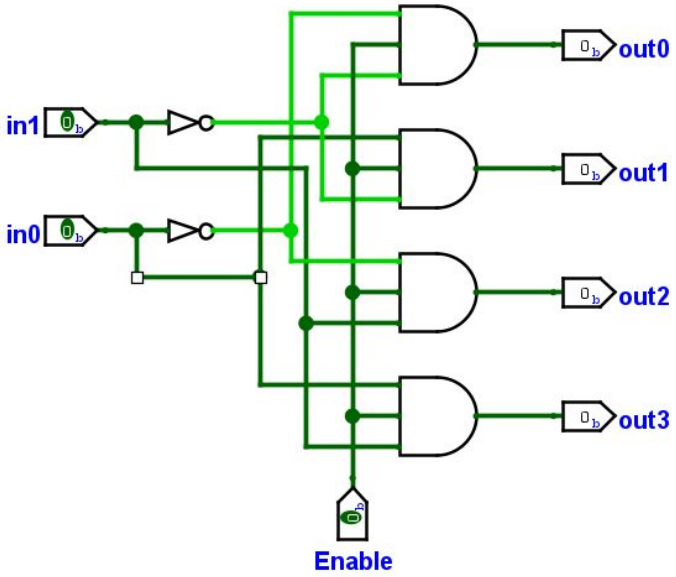
- Adding an enable signal is quite straightforward
  - Example: what would be needed to add an enable signal to our 1x2 decoder?
  - If the enable signal is on: the two d_i's get output
  - Otherwise if the enable signal is off nothing (zero) should go through
  - How can this be done?
- This is the definition of and AND gate combined with a 1-bit 'enable' signal

Enable Signal



Build bigger circuits from smaller ones

- Enable inputs can be used to build larger circuits from smaller ones.
- Exercise: Build a 3×8 decoder out of two 2×4 decoders, each with enable inputs



Build bigger circuits from smaller ones

- The truth table for a 3x8 decoder is:

in2	in1	in0	out7	out6	out5	out4	out3	out2	out1	out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

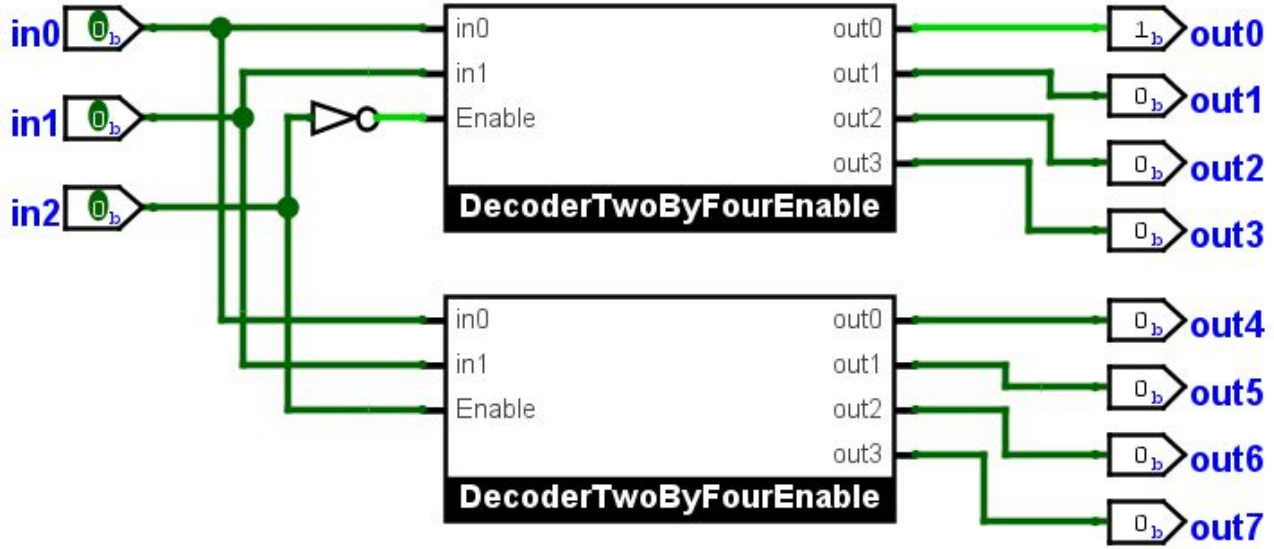
Build bigger circuits from smaller ones

- Notice that the pattern of in0 and in1 for out0 – out3 and out4 – out7 is identical. Each of these is a 2x4 decoder. The only difference is whether in2 is on or off, i.e. an enable!

in2	in1	in0	out7	out6	out5	out4	out3	out2	out1	out0
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0
0	1	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	1	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0

Build bigger circuits from smaller ones

- So if in2 is off, in0 and in1 go to the low order 2x4 decoder and the high order 2x4 decoder is disabled, but if in2 is on it is exactly the opposite

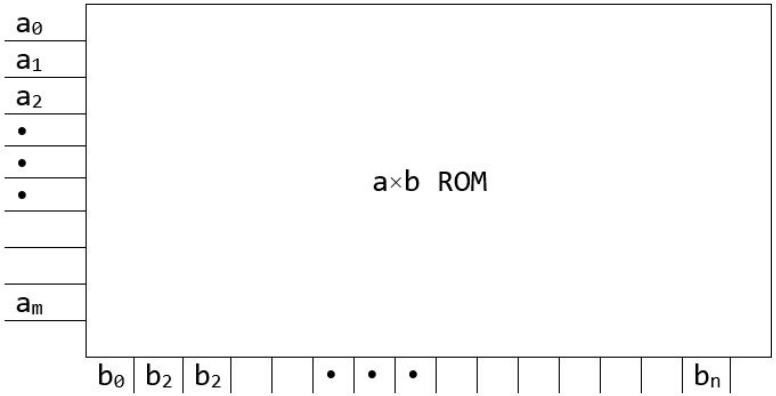


ROMs

- A ROM (Read Only Memory) is a combinational component:
  - a×b means “a addresses” with a b-bit value stored at each location
  - $\lceil \log_2 a \rceil$ inputs → the address of a memory location
  - b outputs → the value stored at that location
  - each output bit is a Boolean function of the address bit

ROMs

- The address size and data size are unrelated
  - i.e. the Atari has 4Mb of 8 bit memory
- A read-only memory (ROM) is a logic circuit that can generate all of the possible minterms of its inputs, which are used as input to a series of circuits to generate each bit of the output value.



4x2 ROM

- This means there are 4 addresses, where the addresses are 2 bits wide, and the data values are 2 bits wide
  - In this case the value at address 00 is 10, at address 01 is 01, etc
- The truth table for a 4x2 ROM could be:

a1	a0	d1	d0
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	1

4x2 ROM

- Boolean Equations:
- `d0 = a0'a1 + a0a1' + a0a1`
- `d1 = a0'a1' + a0a1'`
- Notice that the above addresses are the set of minterms for n bits. Thus, a minterm generator is required – decoders are also called minterm generators.

4x2 ROM

- The following are the truth table and equations for a 2x4 decoder:

```
out0 = in1′·in0′
out1 = in1′·in0
out2 = in1·in0′
out3 = in1·in0
```

- Remember that only one output from the decoder will be active

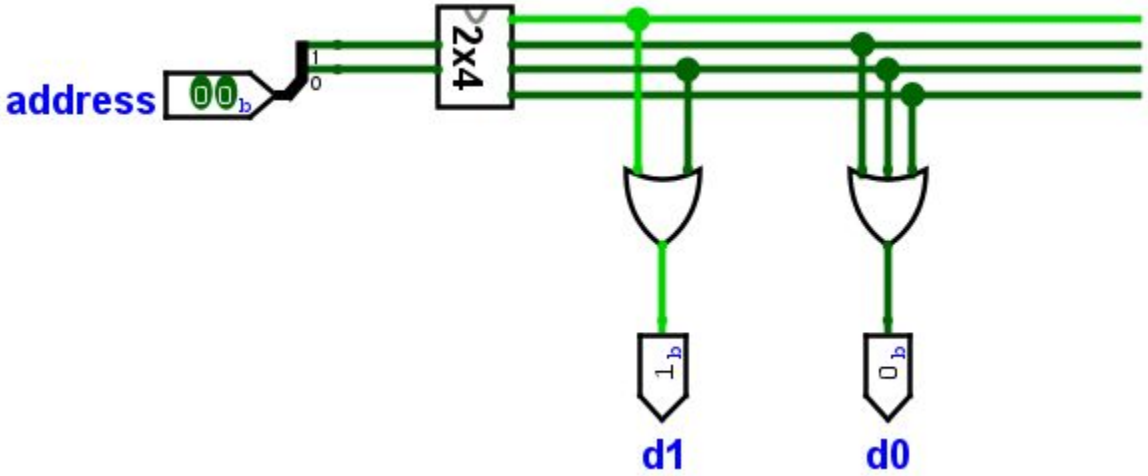
in_1	in_0	out_0	out_1	out_2	out_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

4x2 ROM

- The data is stored in an array of binary data that can be read only and not changed
- Each minterm output is either connected or not connected to each output line
- For each output bit:
  - If the minterm is connected then a 1 value is generated
  - Otherwise a 0 value is generated for this output bit

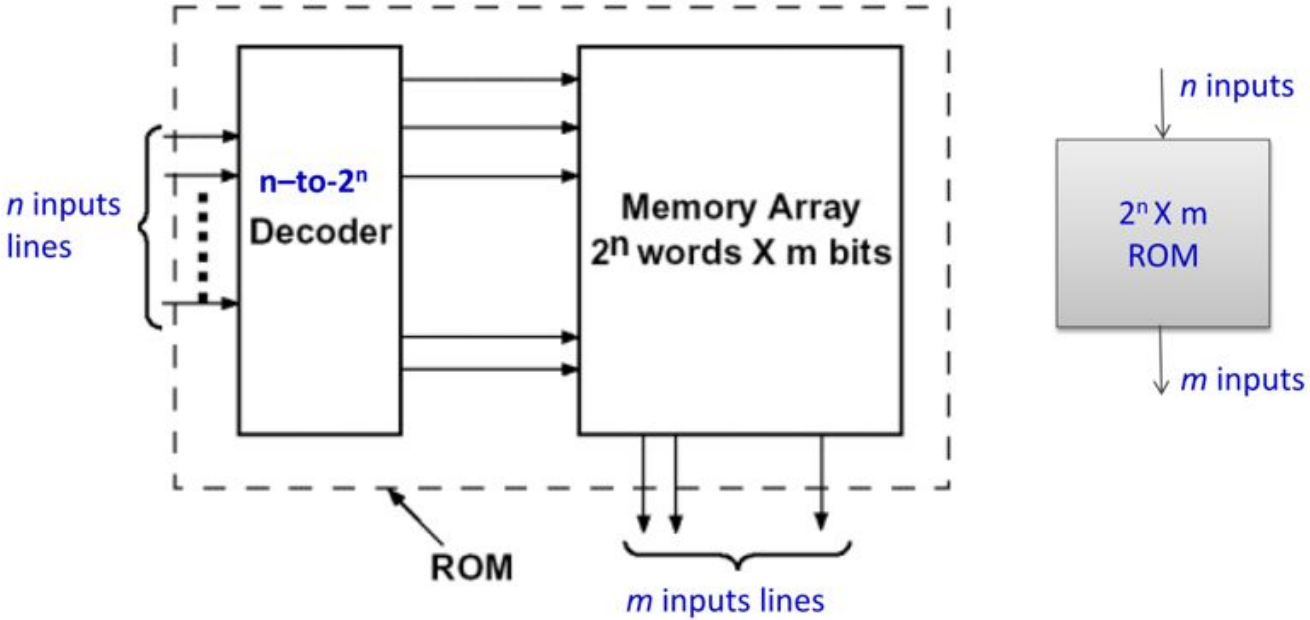
4x2 ROM

- Logic circuit to implement the 4x2 ROM



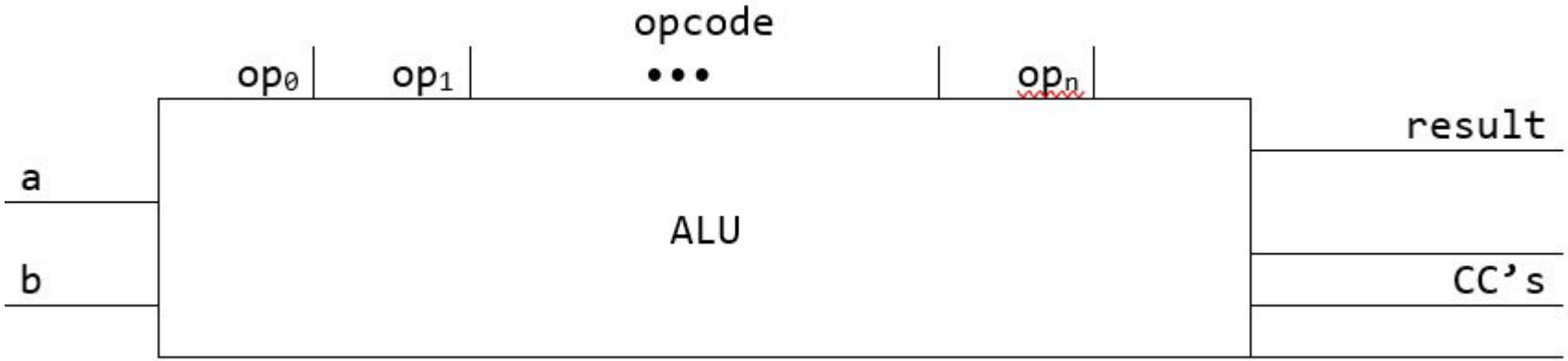
ROM Generalized

- Can be generalized to 2^n to m ROM



ALU

- Recall: we can build an n-bit adder/subtractor out of FAs.
- This can be generalized to an (n+1)-bit ALU.
- **Diagram:** generalization of block diagram



ALU

- Two input values allow binary and unary operations can be implemented
- For example:

	operation code	operation (C notation)		
	000	a + b	add/sub circuit	
	001	a – b		
*	010	a << 1	(arithmetic shift left)	from previous bit
*	011	a >> 1	(arithmetic shift right)	from next bit
	100	a & b		
	101	a | b	direct gate operations	
	110	a ^ b		
*	111	!a		

* unary operations ignore b

ALU

- Diagram: generalization of an FA stage to an ALU stage
- All operations are performed regardless of what the desired operation is!
    - Only the desired operation goes through the MUX
- A working 8-bit ALU with NZVC outputs is provided for you to study