CS 240
Foundations of Computer Systems

WELLESLEY

Combinational Logic

Karnaugh maps

Building blocks: encoders, decoders, multiplexers

Abstraction!

But first...

Recall: *sum of products*

logical sum (OR)
of products (AND)
of inputs or their complements (NOT).

A	B	C	M
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Construct with:
- 1 code detector per 1-valued output row
- 1 large OR of all code detector outputs

Is it minimal?

Gray Codes = reflected binary codes

Alternate binary encoding
designed for electromechanical switches and counting.

00	01	11	10
0	1	2	3

000	001	011	010	110	111	101	100
0	1	2	3	4	5	6	7

How many bits change when incrementing?

Karnaugh Maps: find (minimal) sums of products

ex

Truth table:

A	B	C	D	F(A, B, C, D)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

K-map:

gray code order

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	1	1	0	1
10	1	1	1	1

To build a k-map (best for functions of 2-4 inputs)
1. Split the inputs, half as the header row and half as the header column.
2. Put the input *values* as products in **gray code order.**
3. Fill in each cell based on the truth table.

Karnaugh Maps: find (minimal) sums of products

ex

Truth table:

A	B	C	D	F(A, B, C, D)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

K-map:

gray code order

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	1	1	0	1
10	1	1	1	1

To derive *a* minimal expression from a k-map

1. Cover **exactly** the **1s** by drawing a (minimum) number of (maximally sized) rectangles whose dimensions are **powers of 2**.
   - They may overlap or wrap around!
2. For each, make a ***product*** of the inputs (or complements) that are 1 for all cells in the rectangle. (*minterms*)
3. Take the ***sum*** of these products.

Karnaugh Maps: find (minimal) sums of products

ex

Truth table:

A	B	C	D	F(A, B, C, D)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

K-map:

gray code order

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	1	1	0	1
10	1	1	1	1

AC’ + AB’ + BCD’

To convert to algebra:

1. Any literals that *change* are excluded from the product.
2. A literal that is always 1 should be included as is.
3. A literal that is always 0 should be negated and included.
4. Take the ***sum*** of these products.

Karnaugh Maps and Wrapping

ex

Blocks of 1s in Karnaugh maps can wrap around sides and even 4 corners.

Give the minimal sum-of-products for the Karnaugh map to the left.

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	1	0	0	1
10	1	0	0	1

The grouping and ordering of variables in a Karnaugh map doesn’t matter, but the **AB/CD** ordering is easier to read from a truth table.

Convince yourself that the **AC/DB** table is equivalent to the **AB/CD** table and has the Same sum-of-products expression. In this particular AC/DB table, no wrapping is required for the rectangles!

AC \ DB	00	01	11	10
00	1			
01	1			
11	1	1		
10	1	1		

Karnaugh Maps and Ambiguity

ex

The minimal sum-of-products expression for a Karnaugh map may not be unique.

Ambiguity is introduced when an arbitrary choice needs to be made.

An example of ambiguity is this Karnaugh map. Give four different minimal sum-of-product expressions for this map.

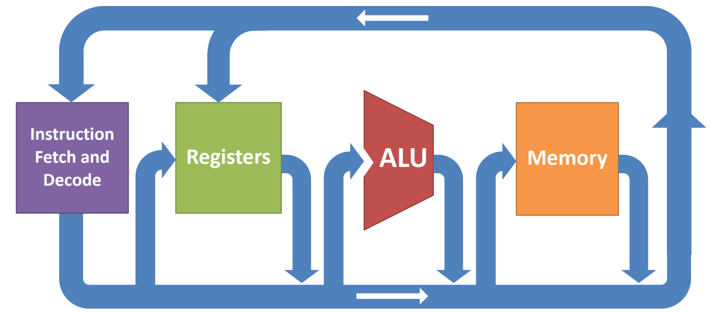
AB \ CD	00	01	11	10
00	1	1	1	1
01	1	1	0	1
11	1	1	1	1
10	0	0	0	0

Voting again with Karnaugh Maps

ex

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Goal for the next 3 weeks: **Simple Processor**



Toolbox: Building Blocks

Abstraction!

Microarchitecture

Processor datapath

Instruction Decoder

Arithmetic Logic Unit

Memory

Digital Logic

Adders
Multiplexers
Demultiplexers
Encoders
Decoders

Registers

Flip-Flops
Latches

Gates

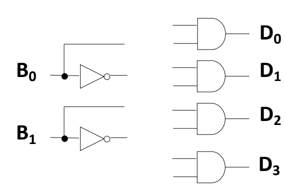
Devices (transistors, etc.)

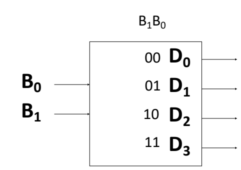
Decoders

ex

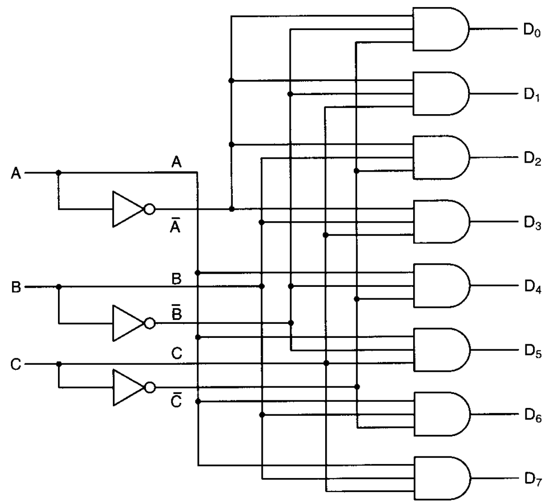
Decodes input number, asserts corresponding output.

- *n*-bit input (an unsigned number)
- 2^n outputs
- Built with code detectors.

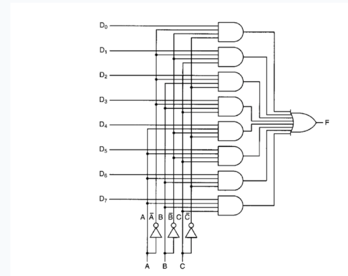




3-bit decoder



Warmup question: is the following a *decoder* or a *multiplexer*?



Decoder

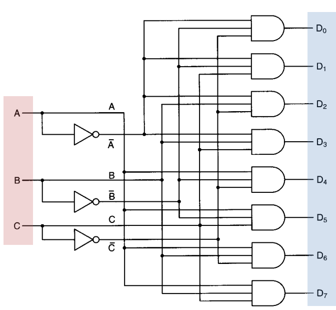
Multiplexer (mux)

None of the above

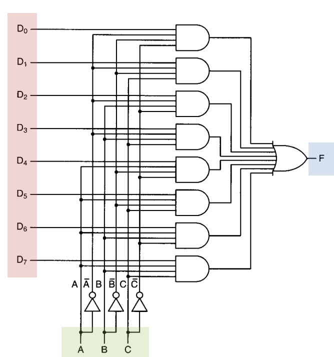
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Recall: decoders and multiplexers

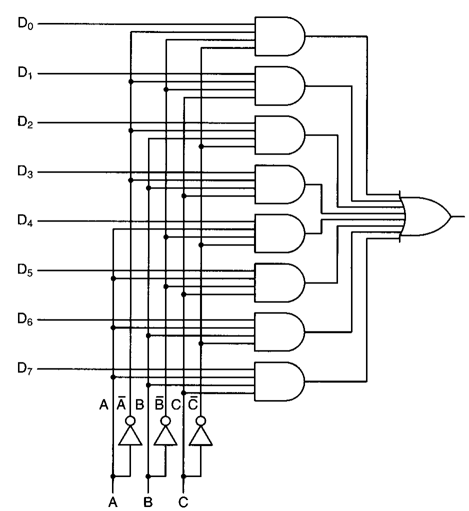
A decoder has an n-bit input and 2^n outputs. Only 1 output active at once.



A multiplexer has 2^n inputs, n selector wires, and 1 output.

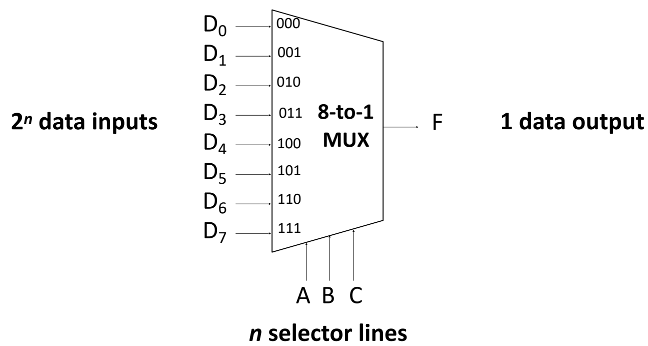


8-to-1 MUX



Multiplexers

Select one of several inputs as output.

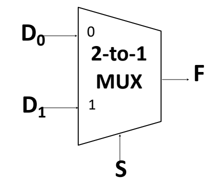


Build a 2-to-1 MUX from gates

ex

If S=0, then F=D_0.
If S=1, then F=D_1.

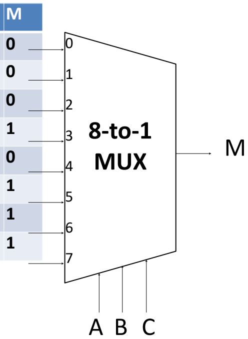
1. Construct the truth table.

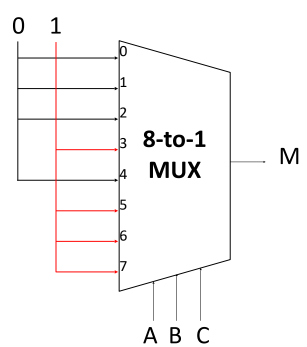


2. Build the circuit.

MUX + voltage source = truth table

A	B	C	M
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**





Buses and Logic Arrays

A bus is a collection of data lines treated as a single logical signal.
= *fixed-width value*

An array of logic elements (logical array) applies same operation to each bit in a bus.
= *bitwise operator*