COSC 065
Hardware Systems

Marquette University

Building Blocks

Logic and Math in Computers

- How do we get from silicon crystals to computers?

Logic and Math in Computers

- How do we get from silicon crystals to computers?
- Silicon crystals → switches called “transistors”

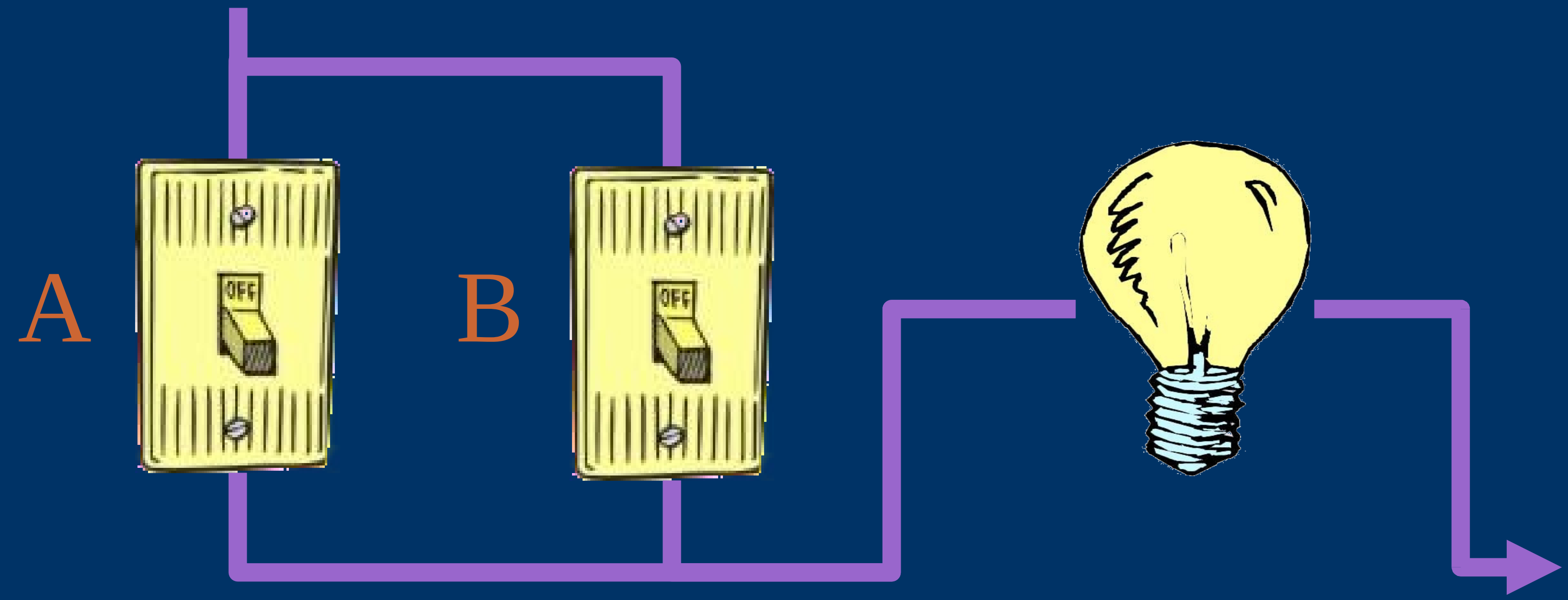
Logic and Math in Computers

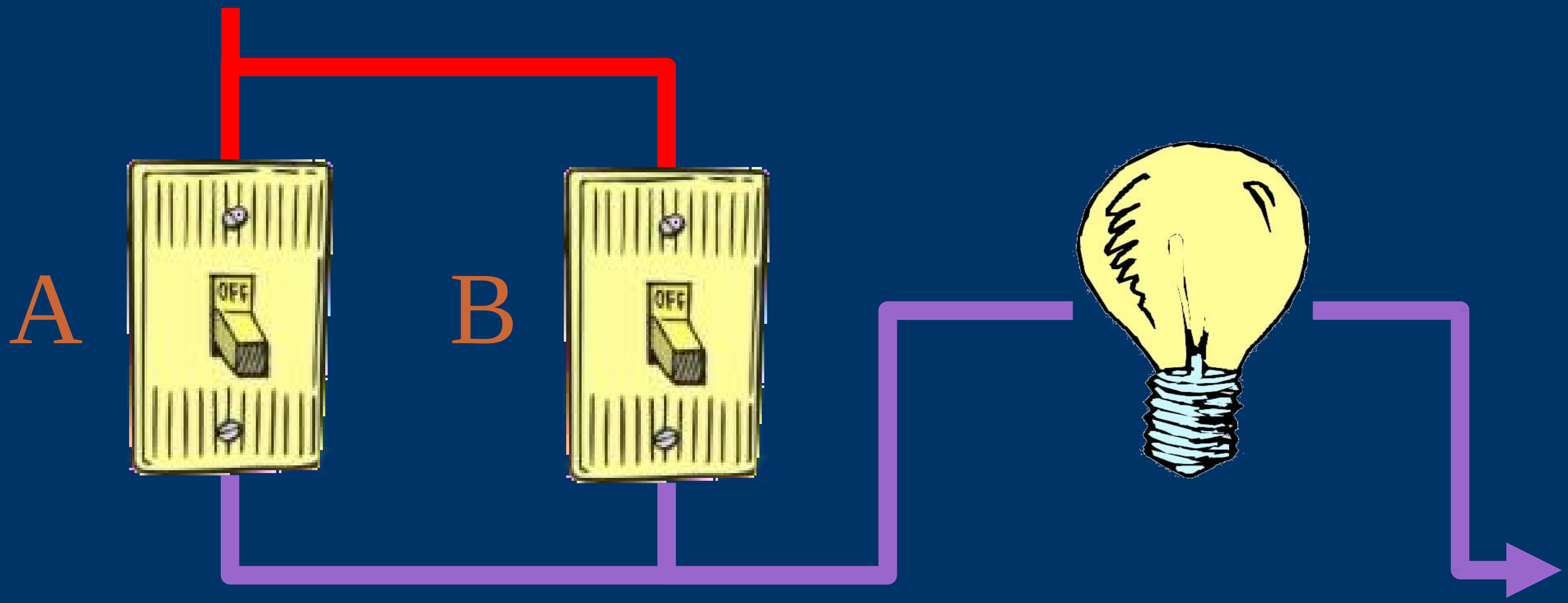
- How do we get from silicon crystals to computers?
- Silicon crystals → switches called “transistors”
- Transistors → Boolean logic blocks

Logic and Math in Computers

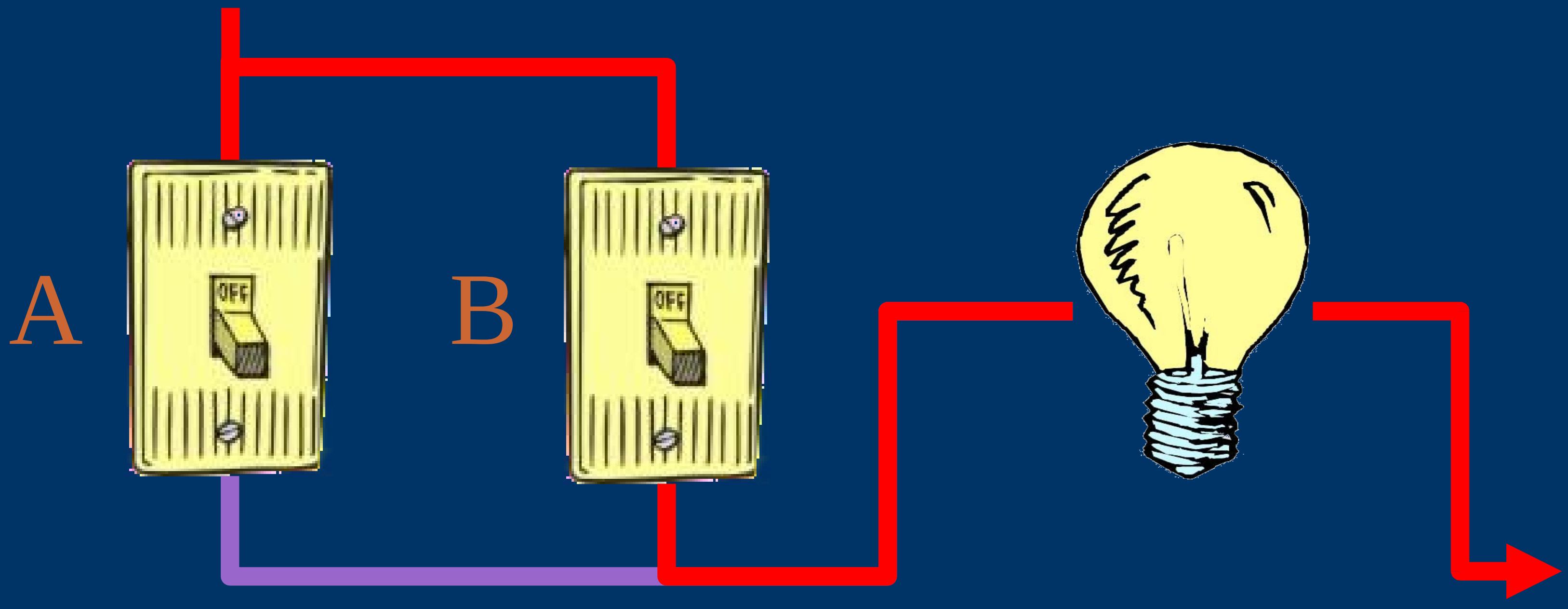
- How do we get from silicon crystals to computers?
- Silicon crystals → switches called “transistors”
- Transistors → Boolean logic blocks
- Modern computers are made up of millions of Boolean logic blocks.

The Logic of Simple Switches

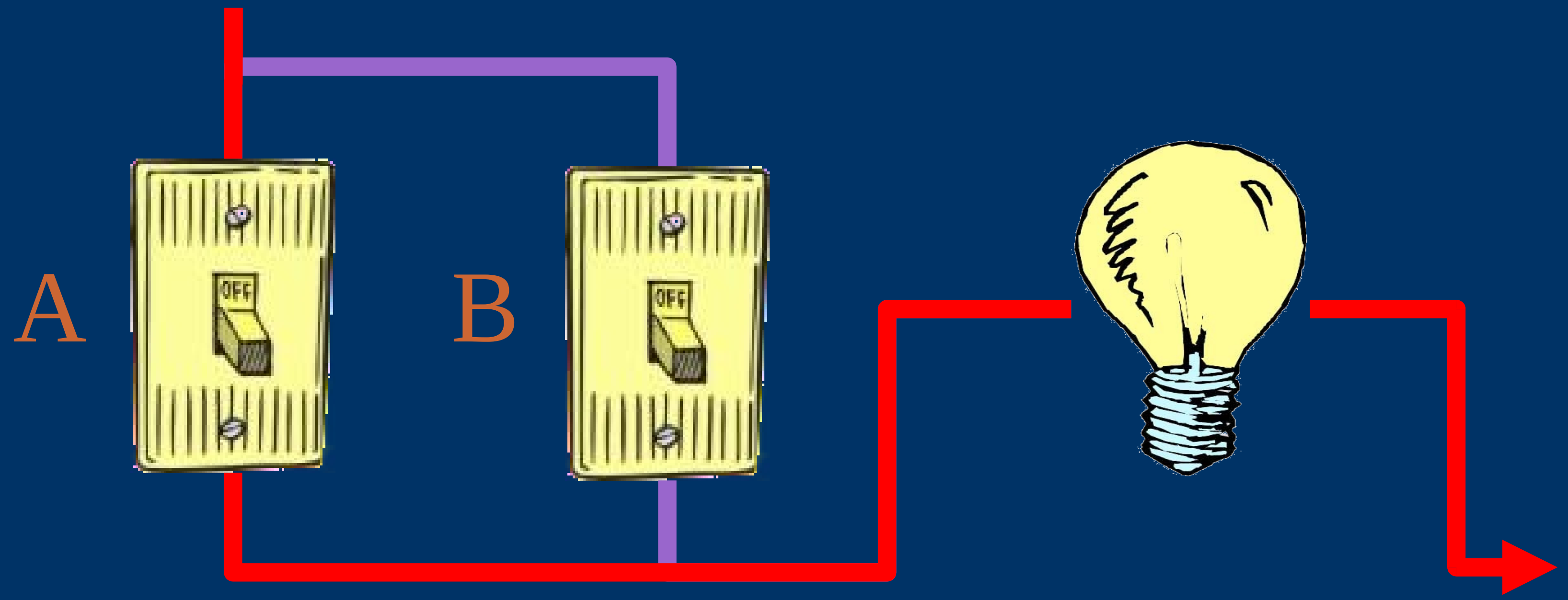




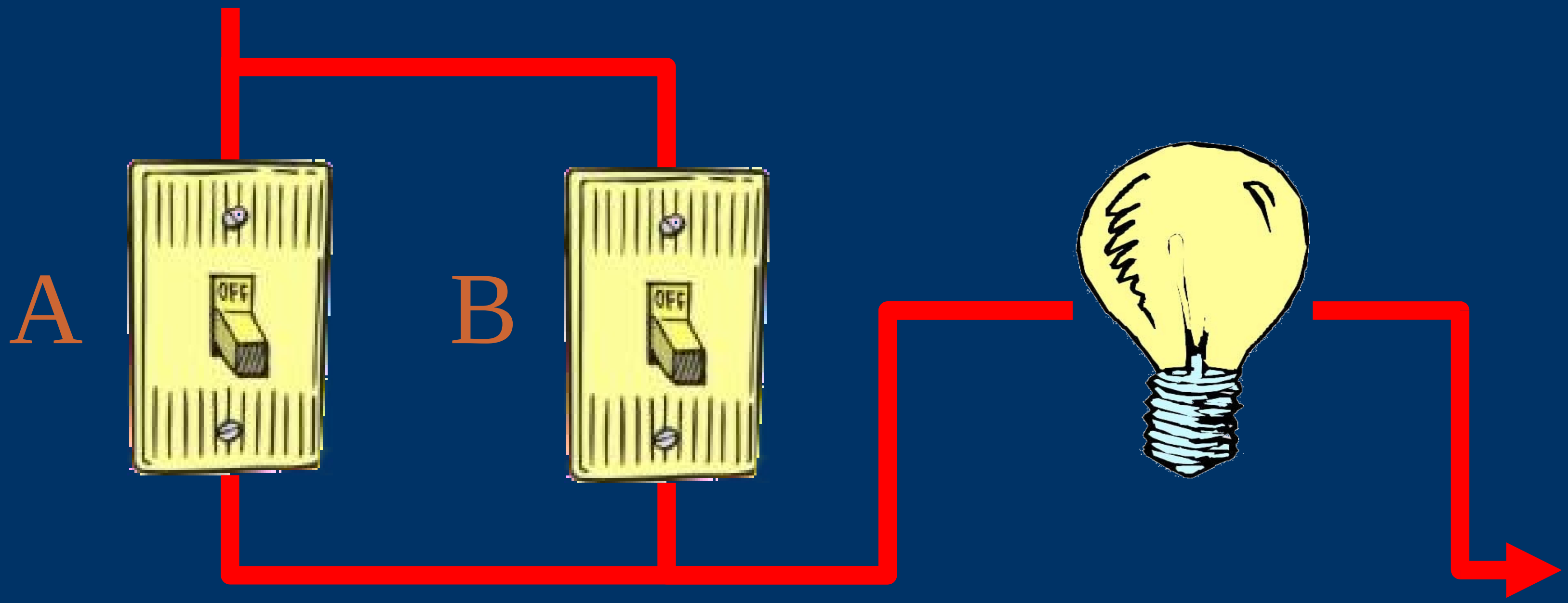
A	B	Light
off	off	off



A	B	Light
off	off	off
off	on	on

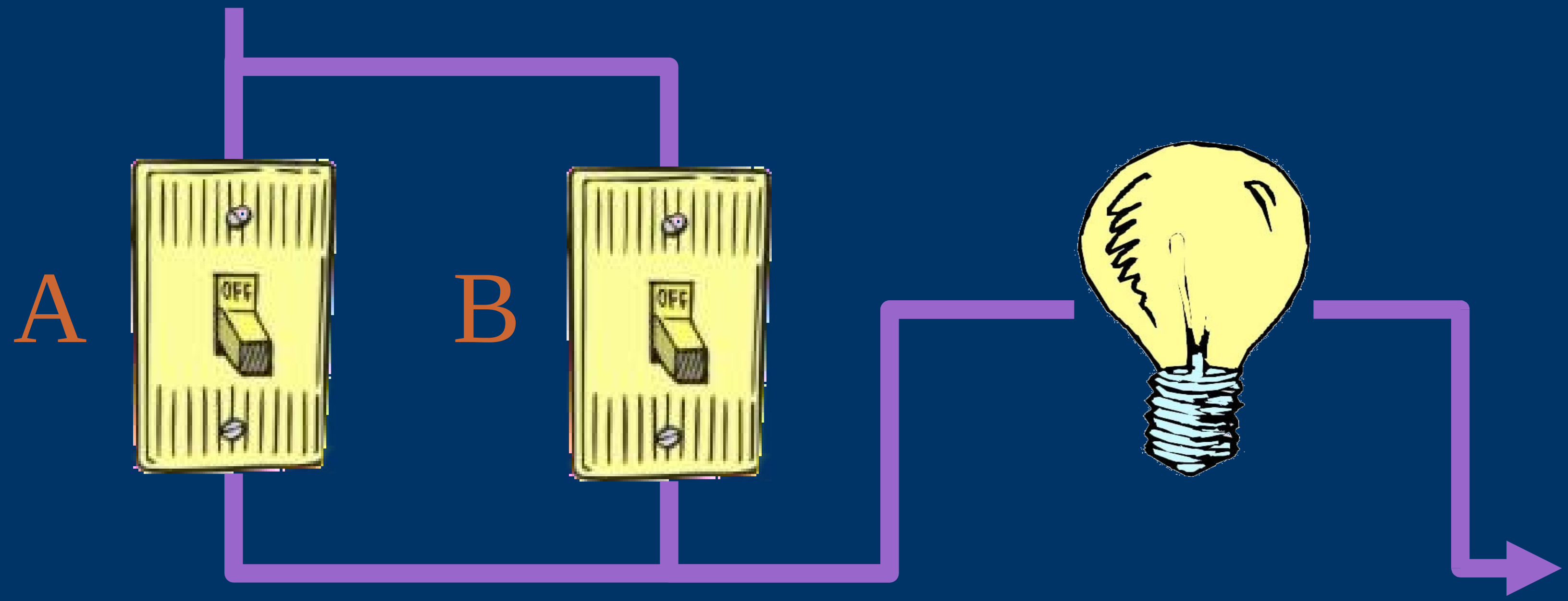


A	B	Light
off	off	off
off	on	on
on	off	on

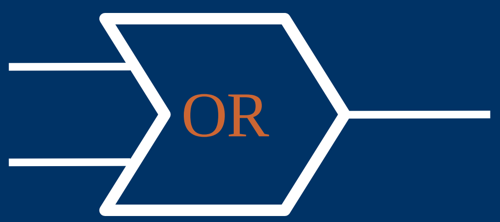


A	B	Light
off	off	off
off	on	on
on	off	on
on	on	on

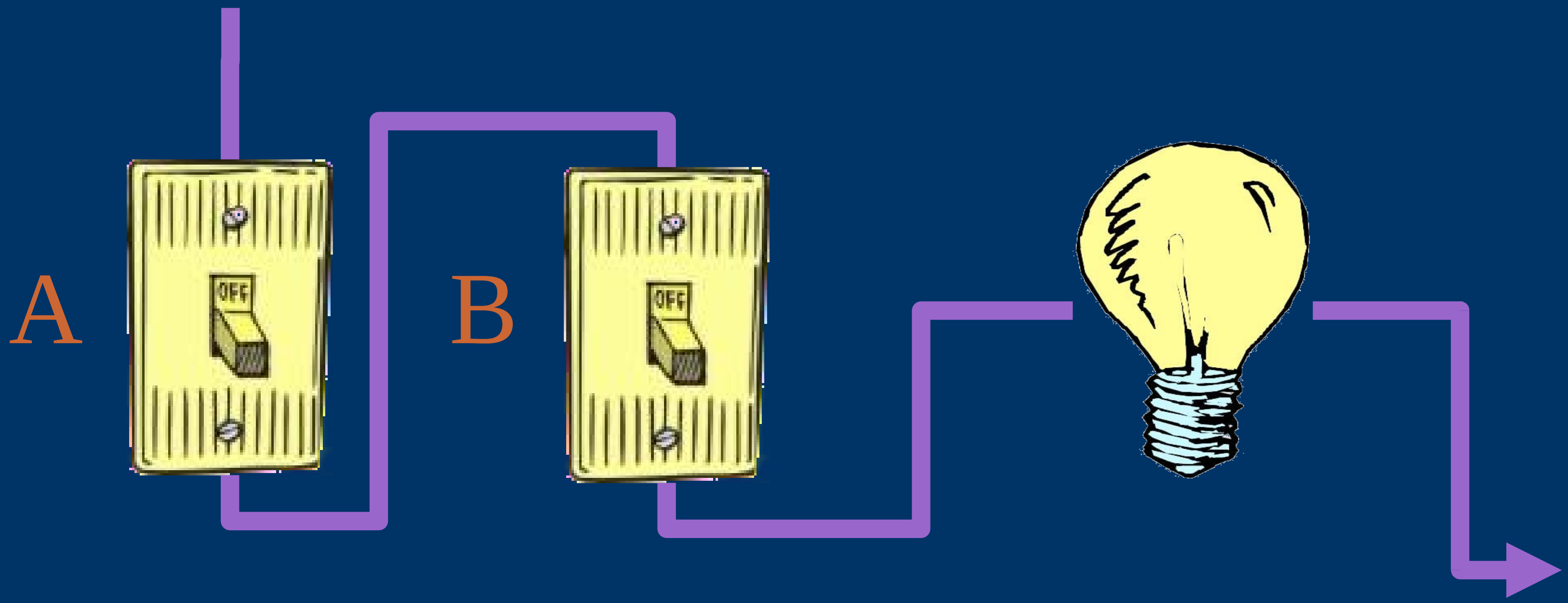
Light is on if A is on OR B is on



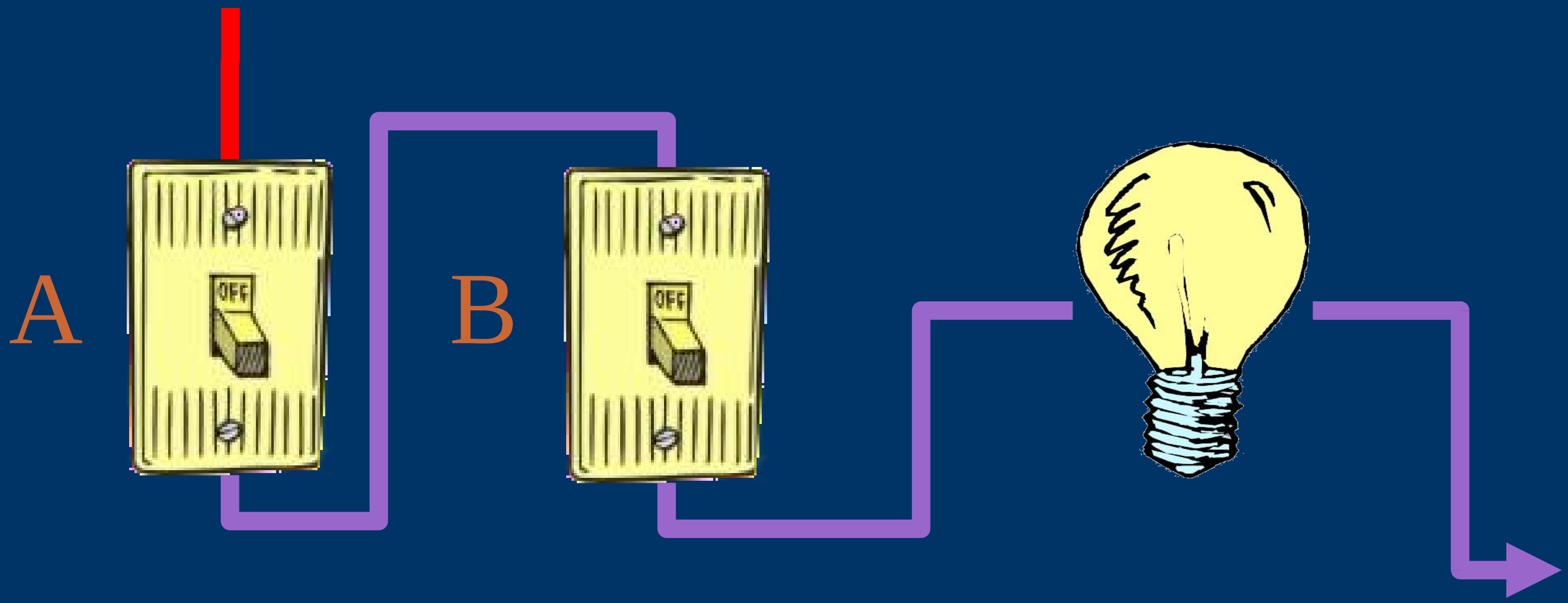
A	B	Light
off	off	off
off	on	on
on	off	on
on	on	on



Another Logic Block

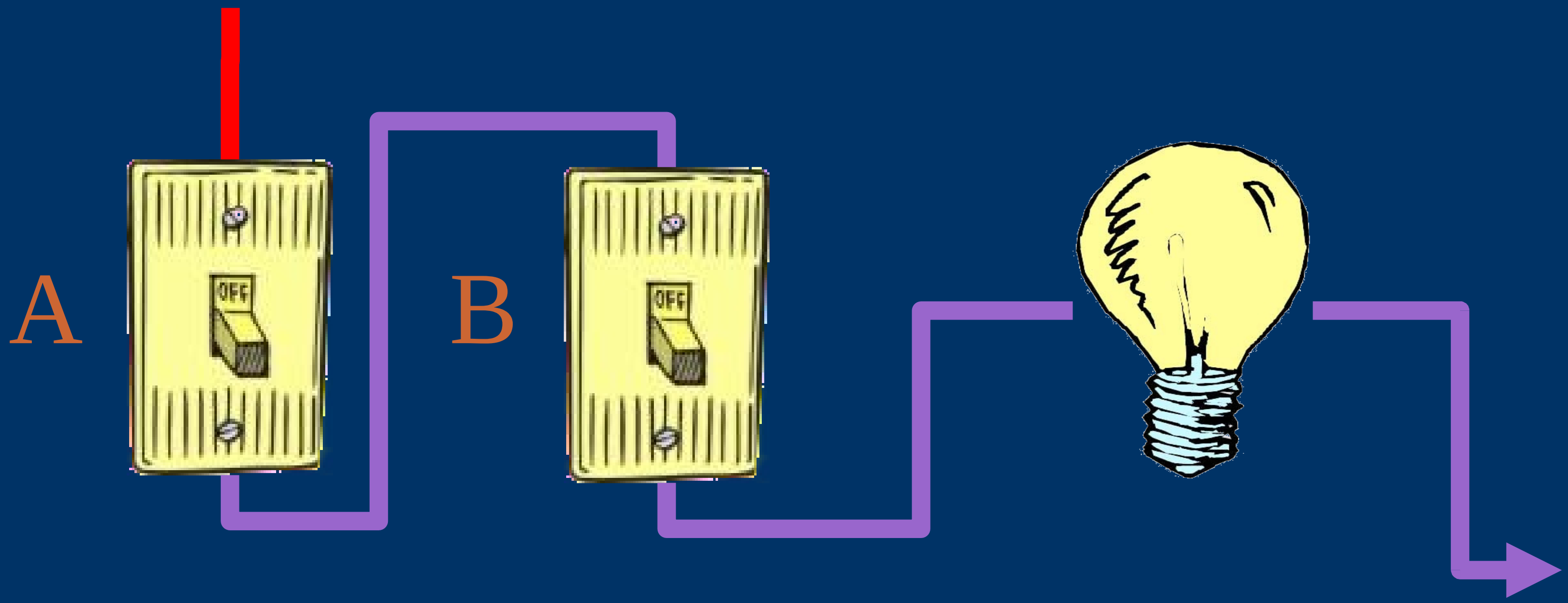


Another Logic Block



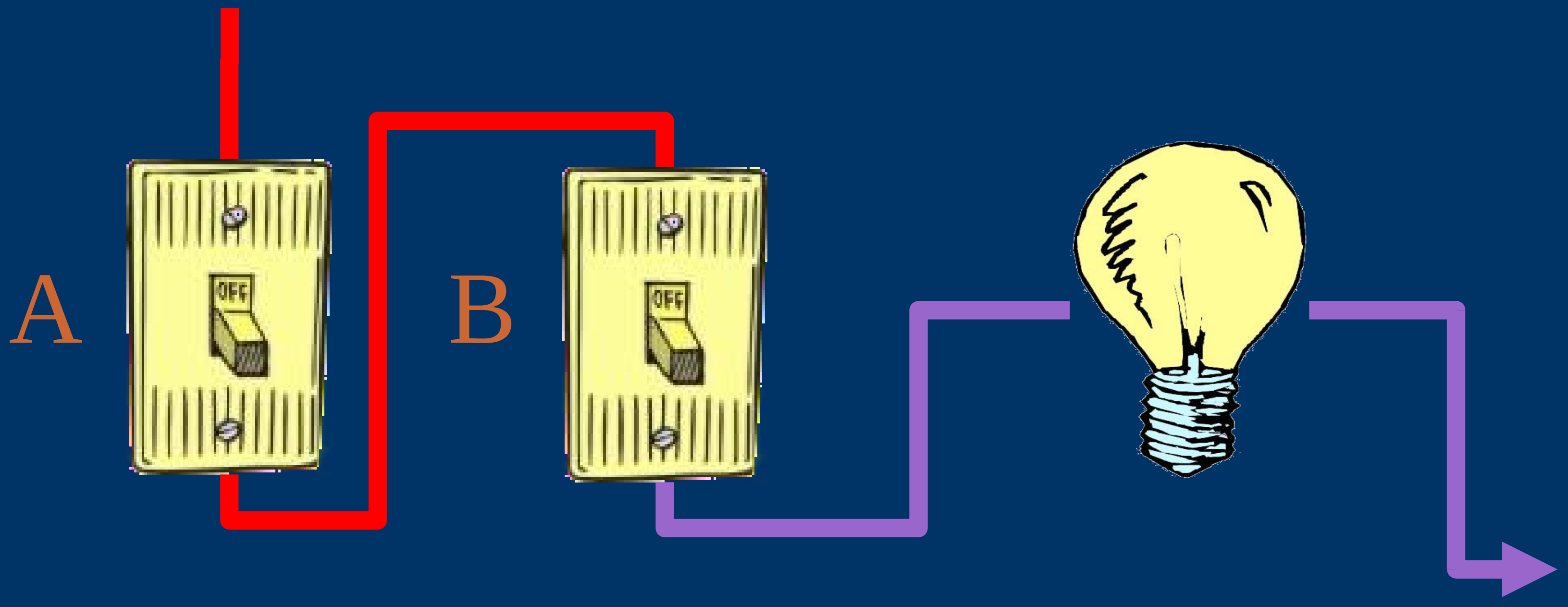
A	B	Light
off	off	off

Another Logic Block



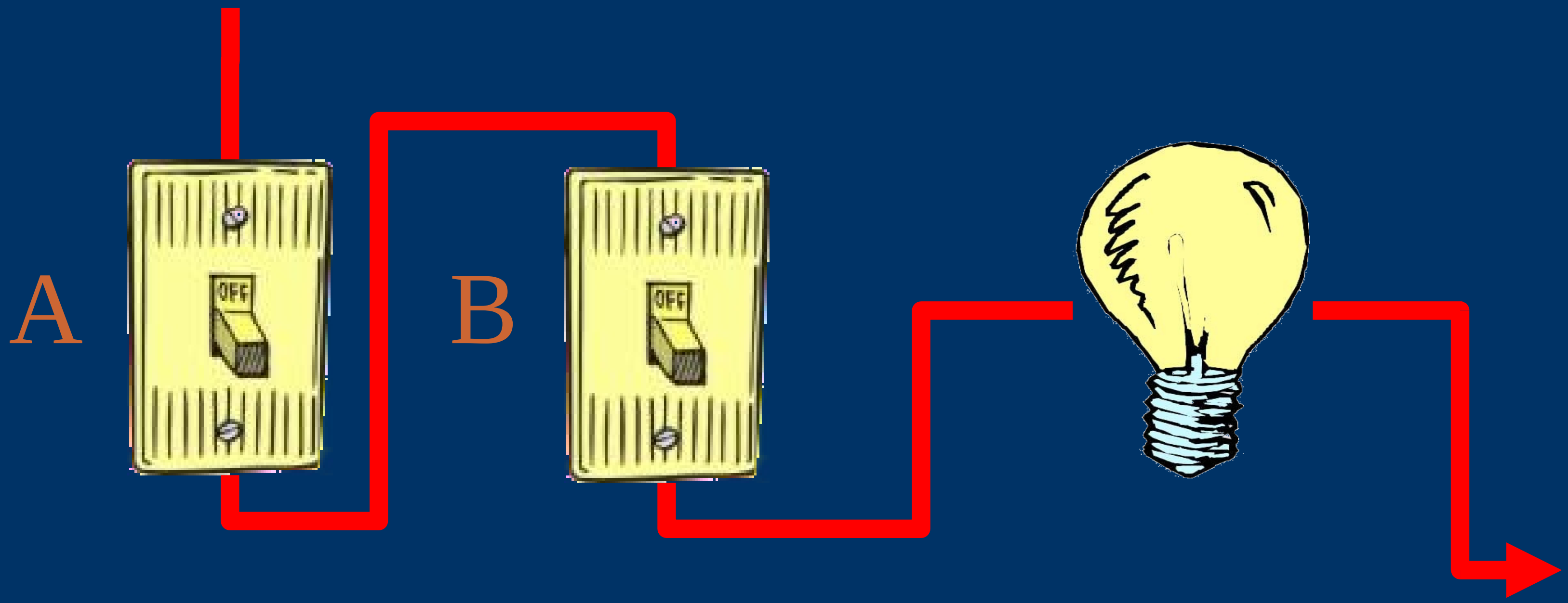
A	B	Light
off	off	off
off	on	off

Another Logic Block



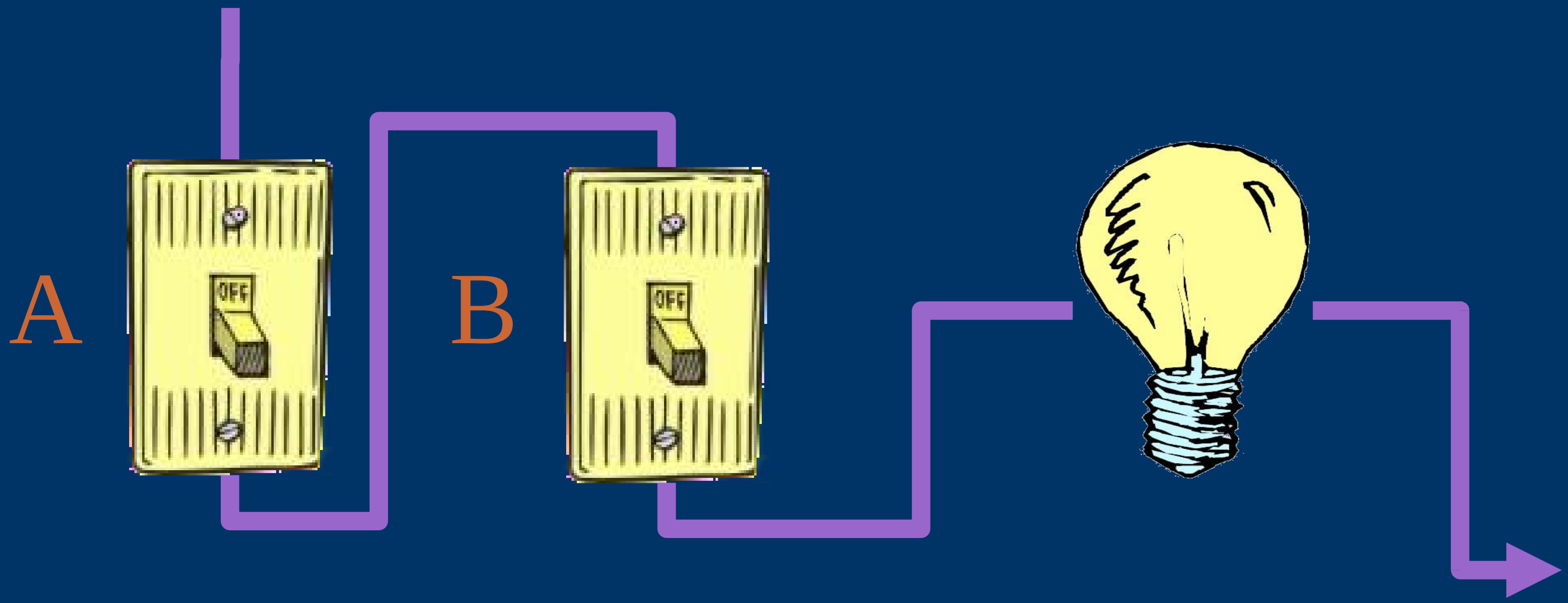
A	B	Light
off	off	off
off	on	off
on	off	off

Another Logic Block



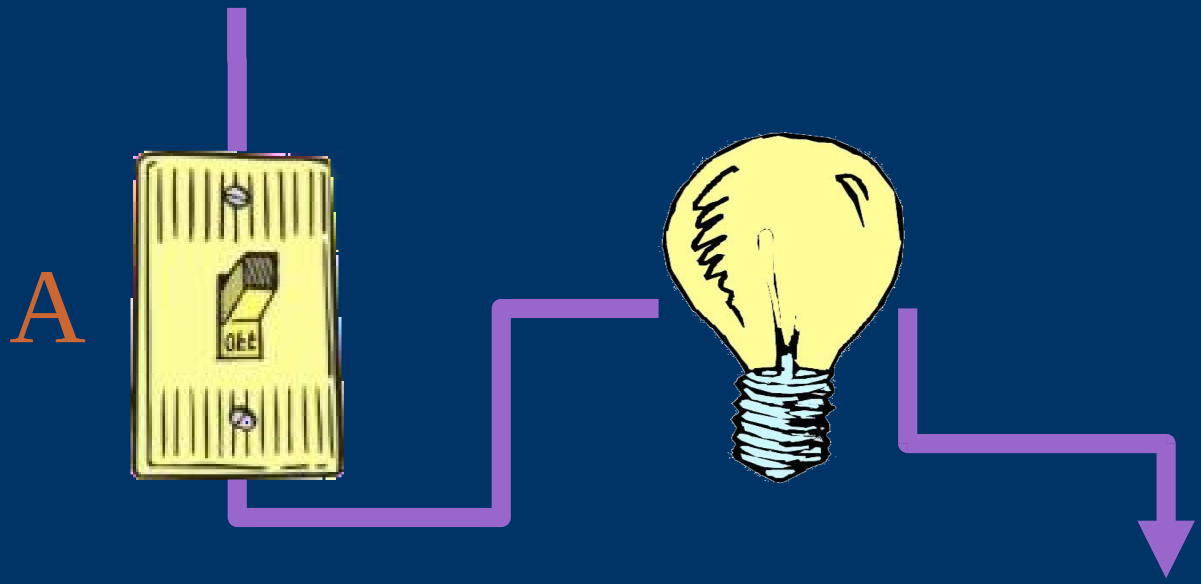
A	B	Light
off	off	off
off	on	off
on	off	off
on	on	on

Light on if A is on AND B is on



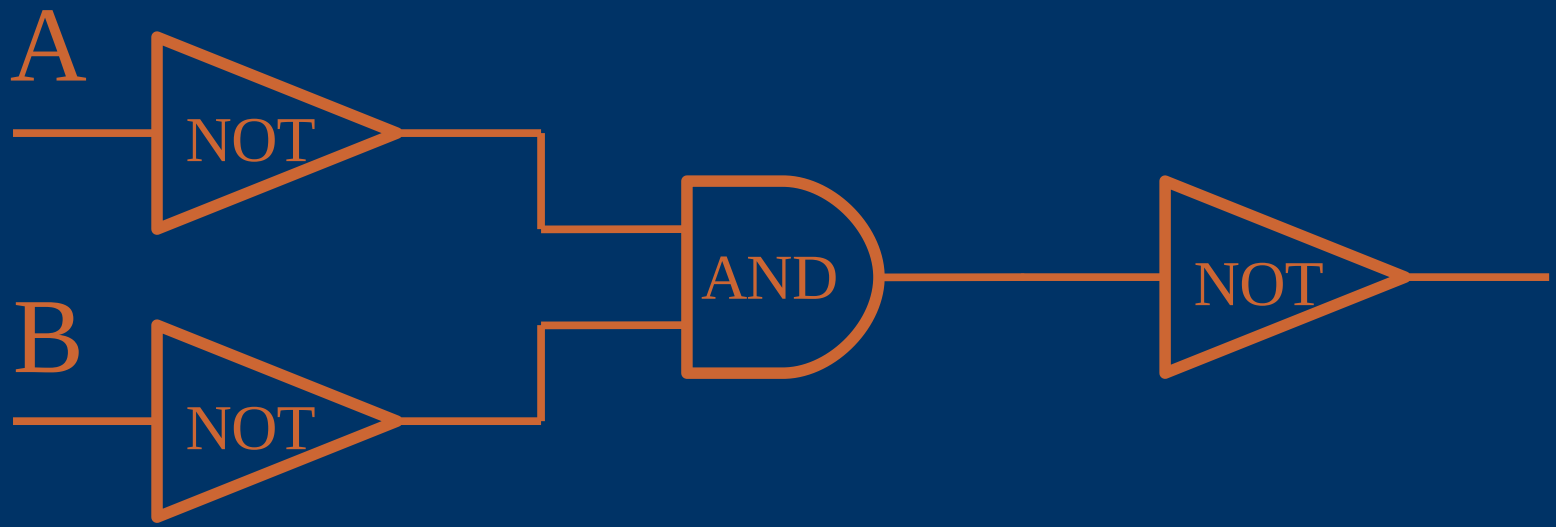
A	B	Light
off	off	off
off	on	off
on	off	off
on	on	on

The Inverter Block



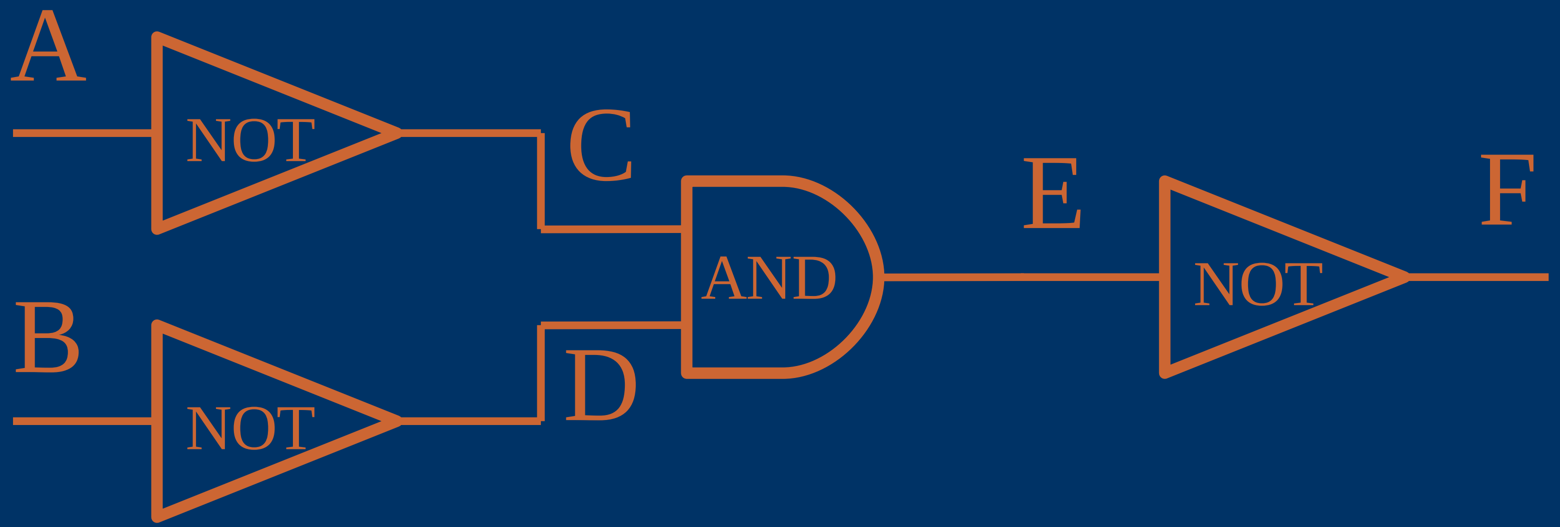
A	Light
off	on
on	off

Combining Logic Blocks



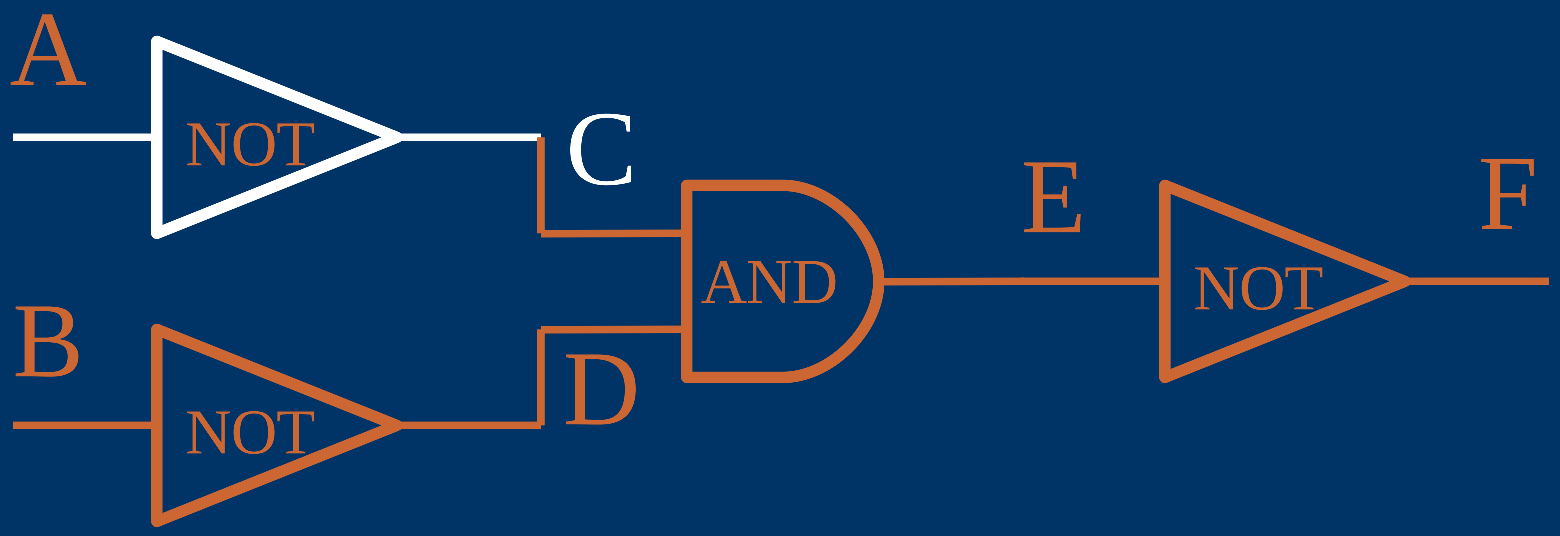
A	B	
off	off	
off	on	
on	off	
on	on	

Combinational Logic



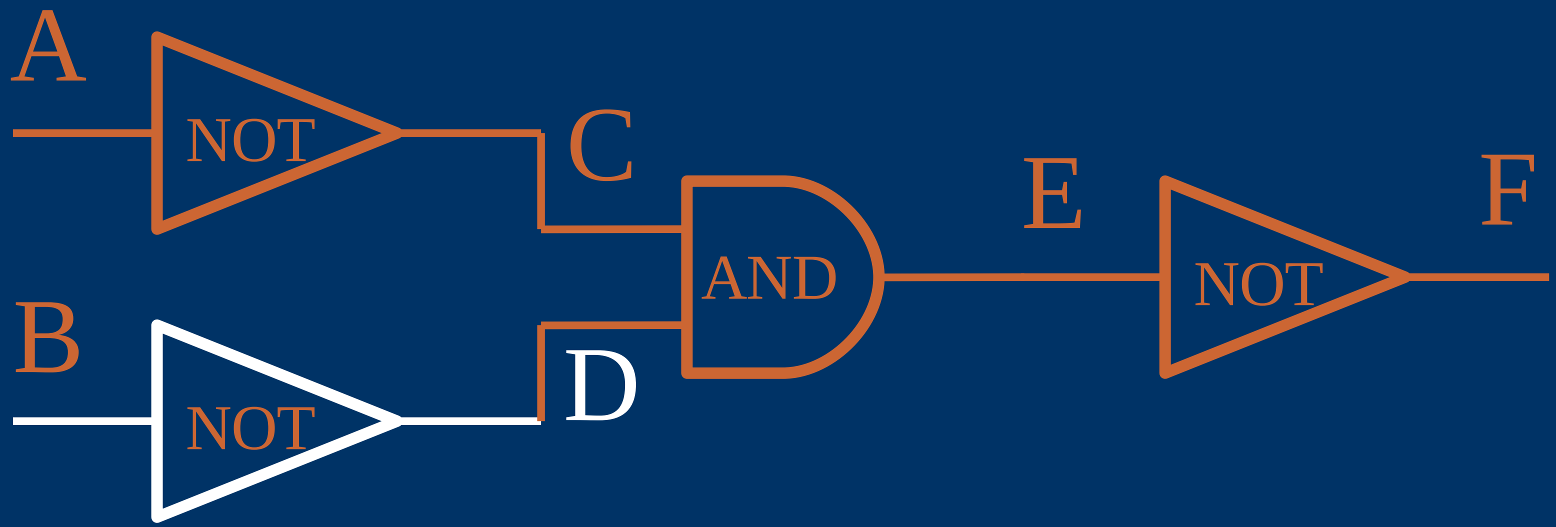
A	B	C	D	E	F output
off	off				
off	on				
on	off				
on	on				

Combinational Logic



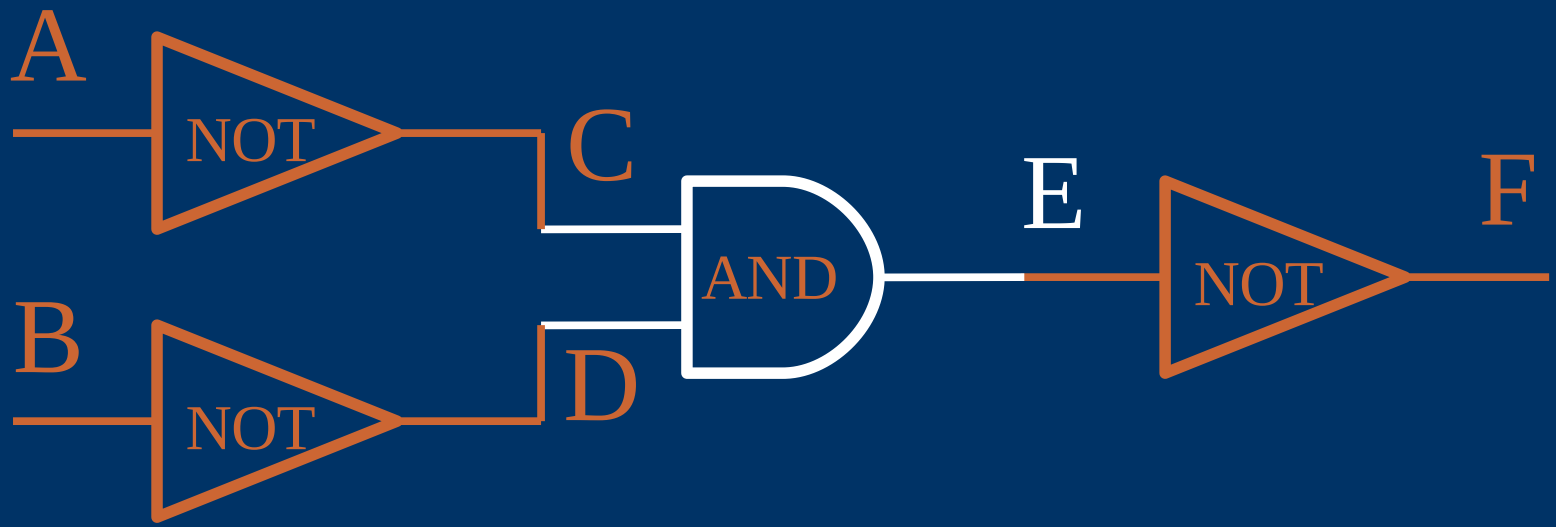
A	B	C NOT(A)	D	E	F output
off	off	on			
off	on	on			
on	off	off			
on	on	off			

Combinational Logic



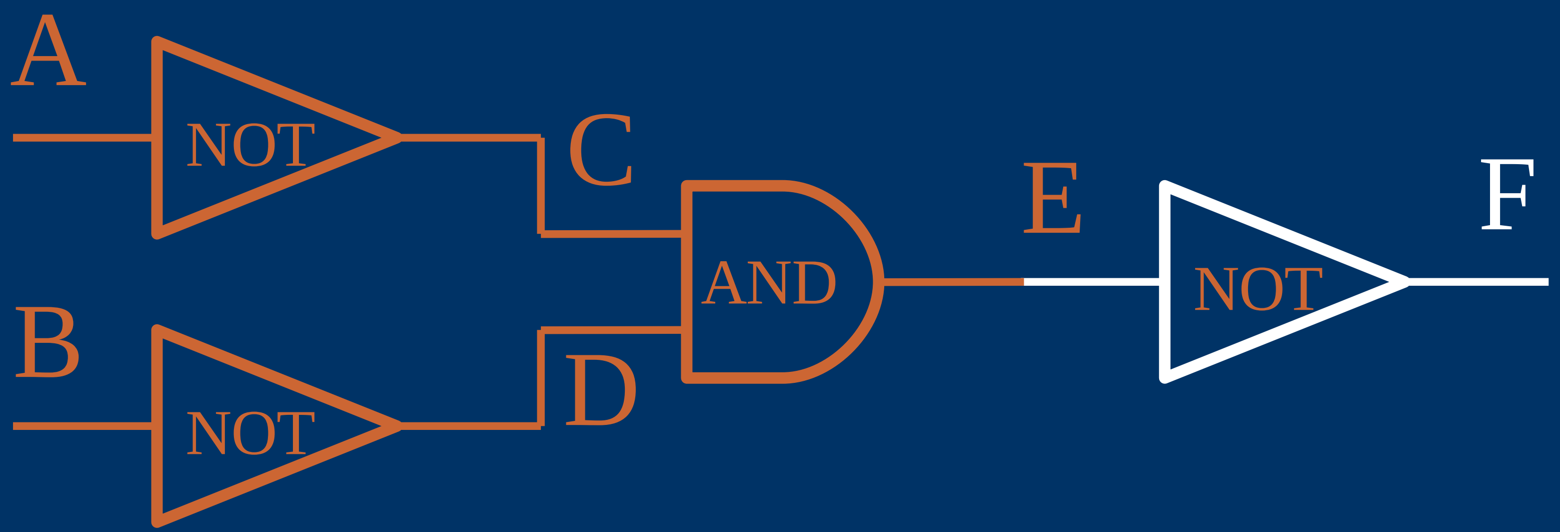
A	B	C NOT(A)	D NOT(B)	E	F output
off	off	on	on		
off	on	on	off		
on	off	off	on		
on	on	off	off		

Combinational Logic



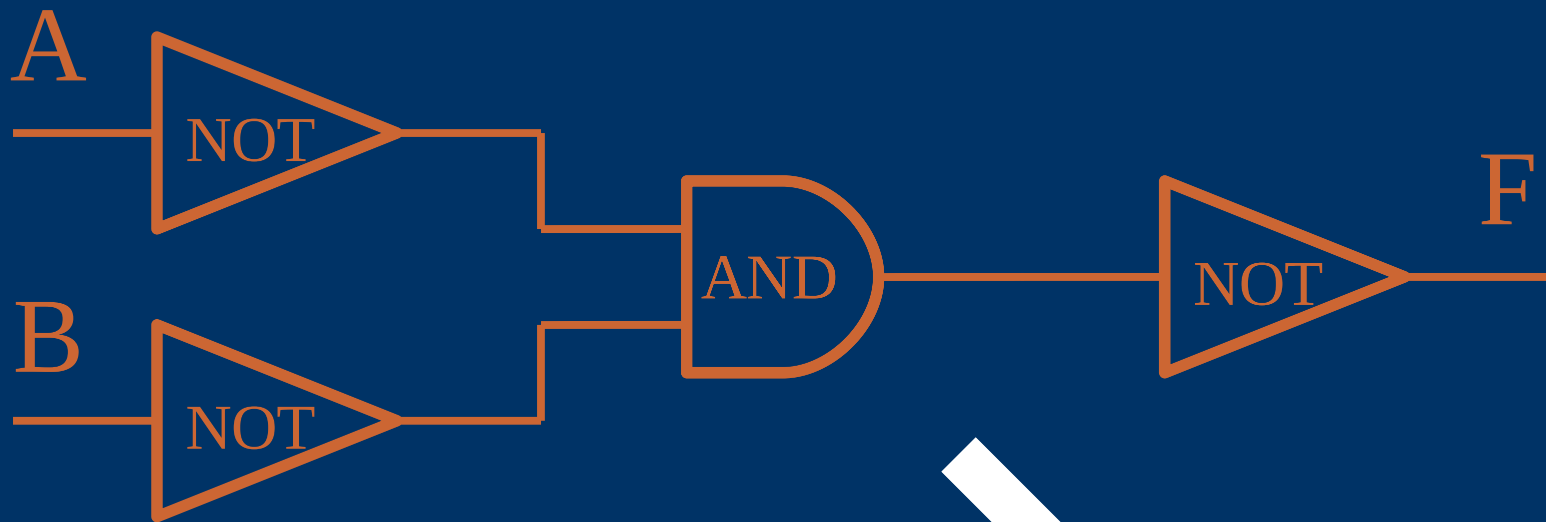
A	B	C NOT(A)	D NOT(B)	E AND(C,D)	F output
off	off	on	on	on	
off	on	on	off	off	
on	off	off	on	off	
on	on	off	off	off	

Combinational Logic

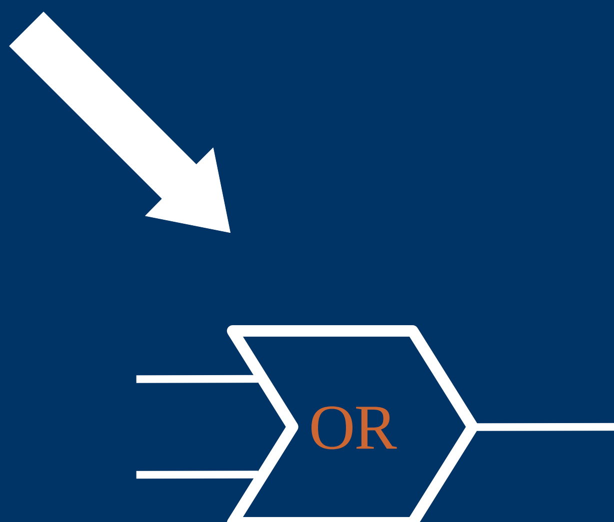


A	B	C NOT(A)	D NOT(B)	E AND(C,D)	F output NOT(E)
off	off	on	on	on	off
off	on	on	off	off	on
on	off	off	on	off	on
on	on	off	off	off	on

The Logic Gate Game



A	B	F output
off	off	off
off	on	on
on	off	on
on	on	on



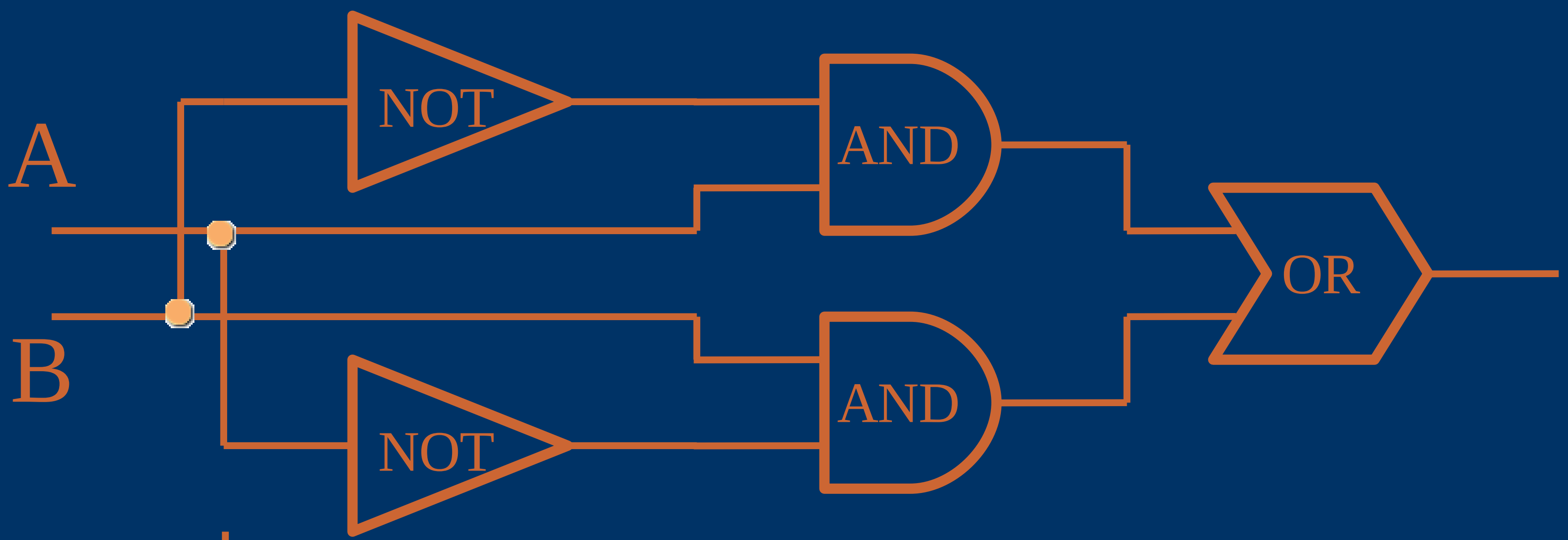
De Morgan's Law

- OR(A,B) == NOT(AND(NOT(A), NOT(B)))

De Morgan's Law

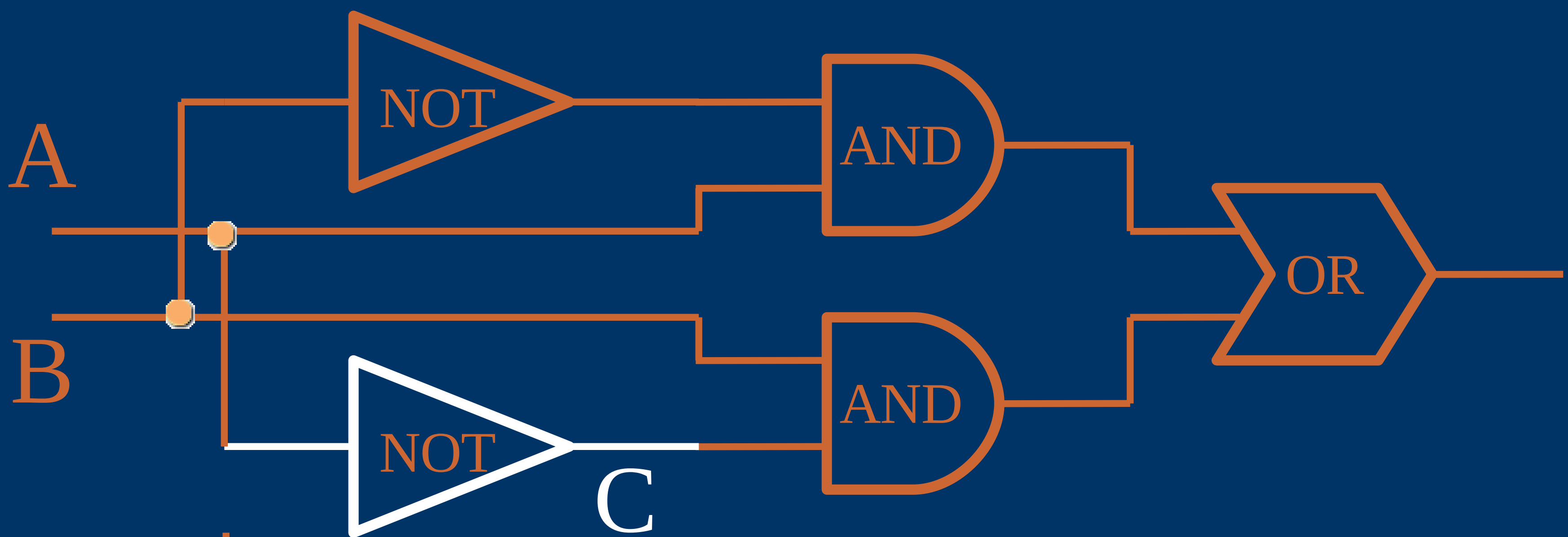
- OR(A,B) == NOT(AND(NOT(A), NOT(B)))
- NOT(OR(A, B)) == AND(NOT(A), NOT(B))
- NOT(OR(NOT(A), B)) == AND(A, NOT(B))
- NOT(OR(NOT(A), NOT(B))) == AND(A, B)
- These identities have many applications and parallels in logic, mathematics, and computer science.

One More Logic Block Puzzle...



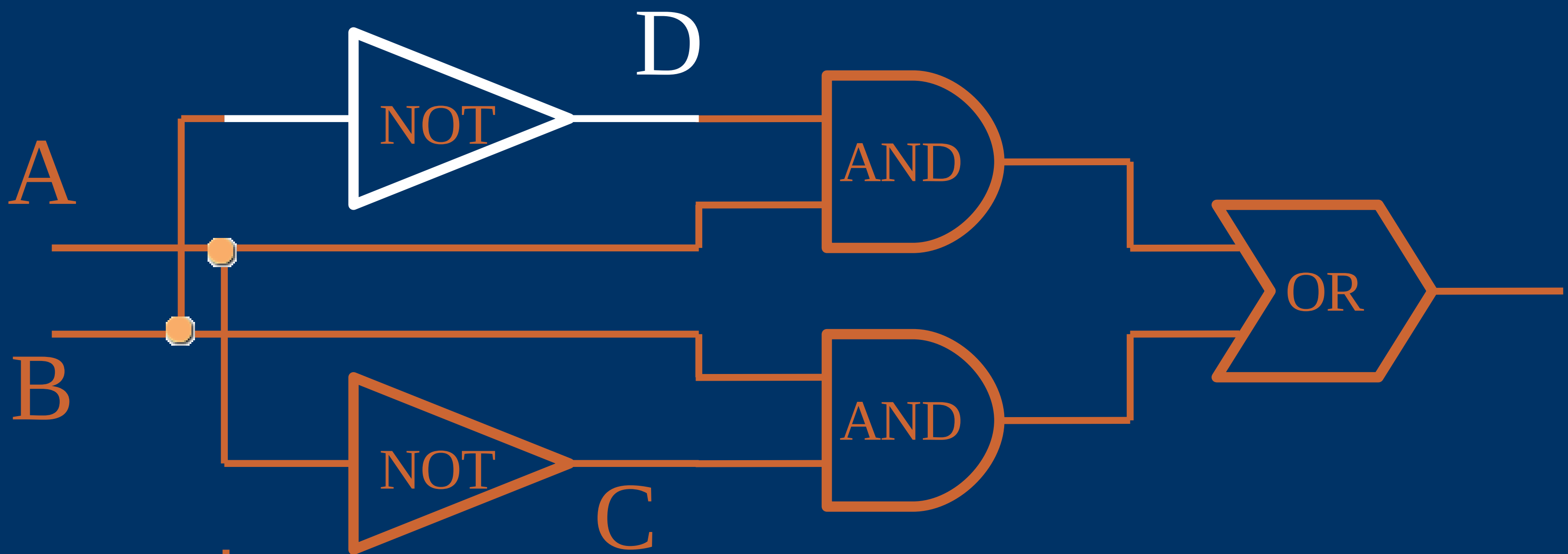
A	B	
off	off	
off	on	
on	off	
on	on	

One More Logic Block Puzzle...



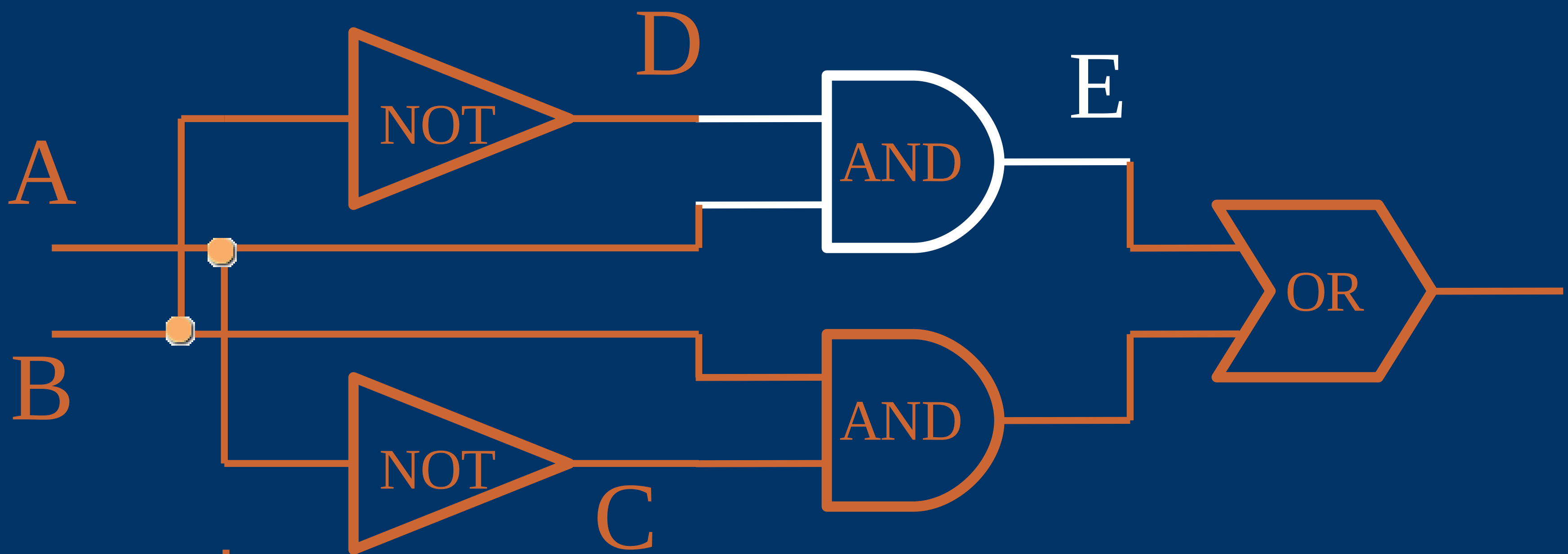
A	B	C NOT(A)
off	off	on
off	on	on
on	off	off
on	on	off

One More Logic Block Puzzle...



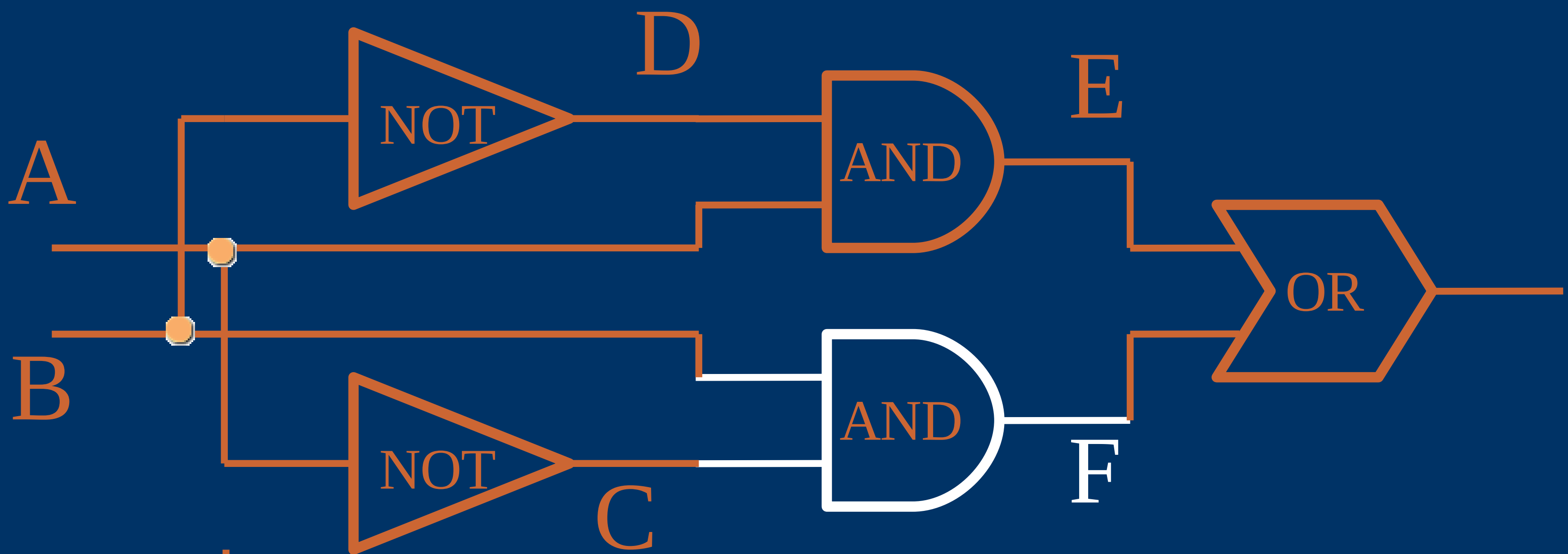
A	B	C NOT(A)	D NOT(B)
off	off	on	on
off	on	on	off
on	off	off	on
on	on	off	off

One More Logic Block Puzzle...



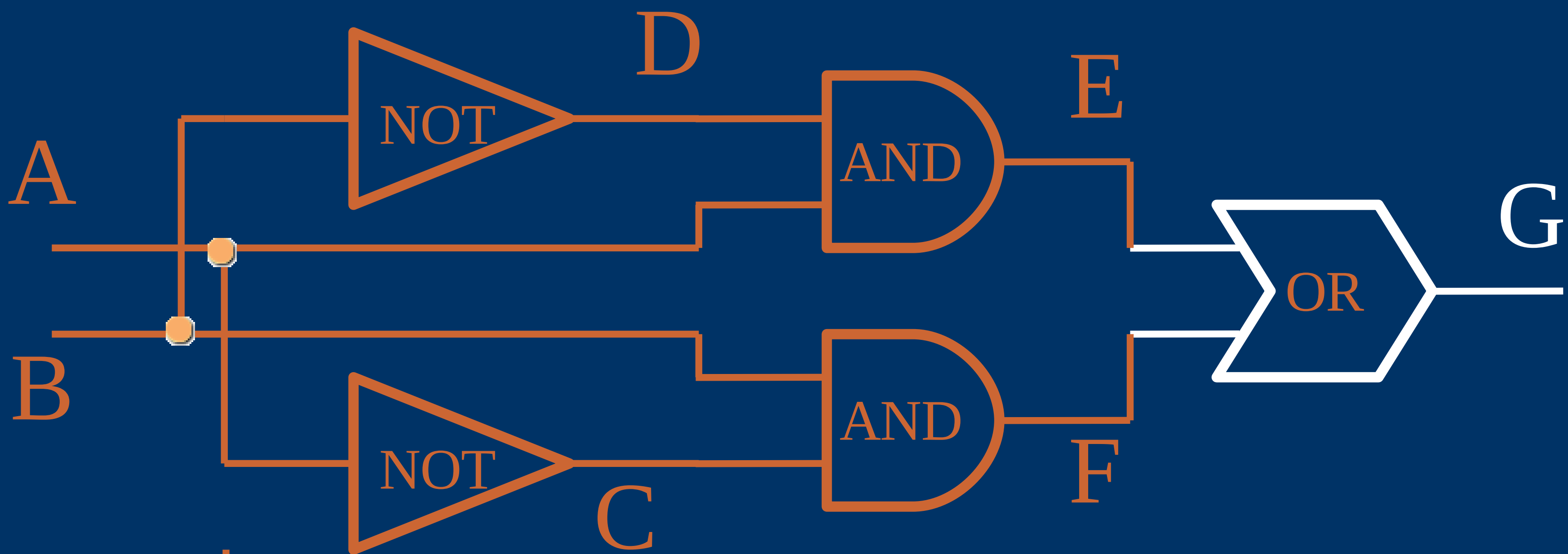
A	B	C NOT(A)	D NOT(B)	E AND(A,D)
off	off	on	on	off
off	on	on	off	off
on	off	off	on	on
on	on	off	off	off

One More Logic Block Puzzle...



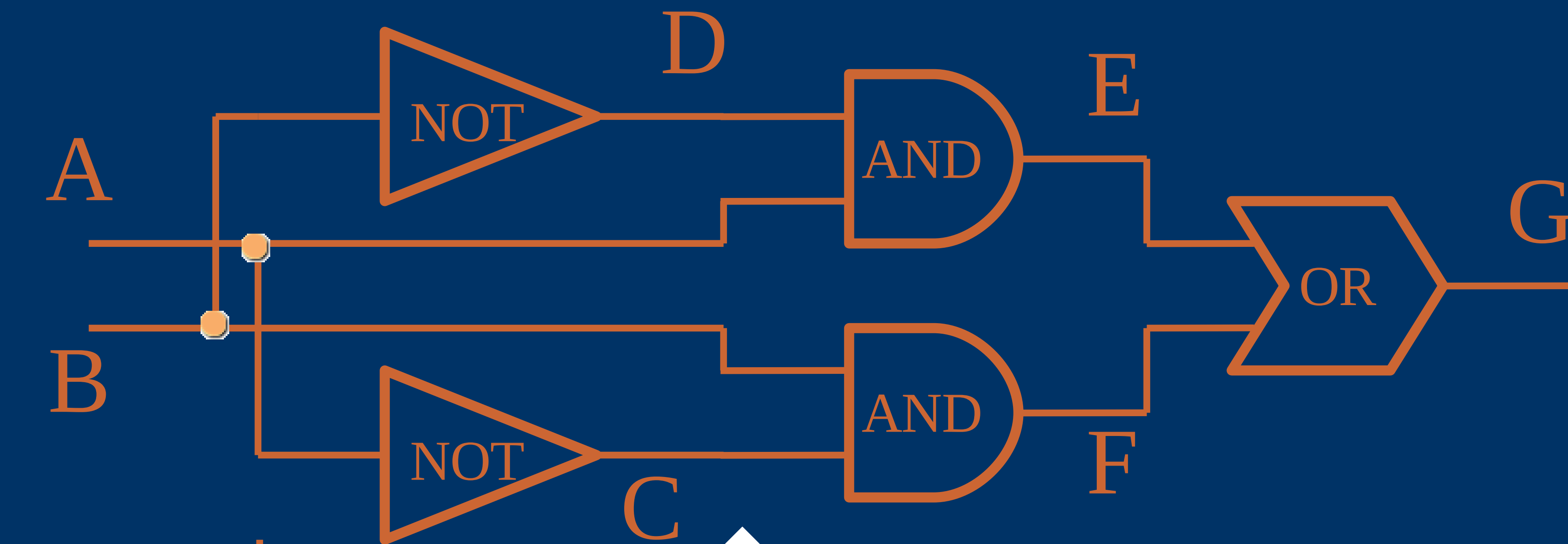
A	B	C NOT(A)	D NOT(B)	E AND(A,D)	F AND(B,C)
off	off	on	on	off	off
off	on	on	off	off	on
on	off	off	on	on	off
on	on	off	off	off	off

One More Logic Block Puzzle...



A	B	C NOT(A)	D NOT(B)	E AND(A,D)	F AND(B,C)	G OR(E,F)
off	off	on	on	off	off	off
off	on	on	off	off	on	on
on	off	off	on	on	off	on
on	on	off	off	off	off	off

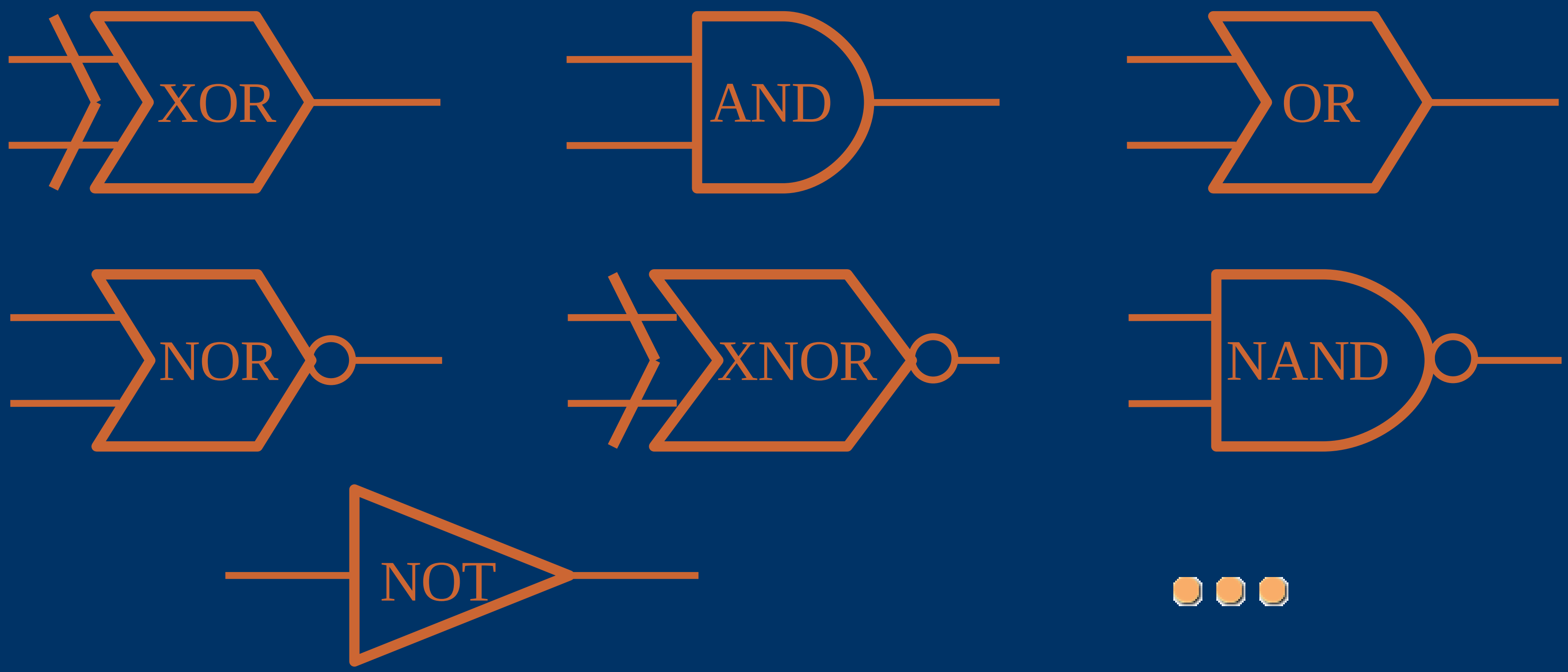
One More Logic Block Puzzle...



A	B	G
off	off	off
off	on	on
on	off	on
on	on	off

Logic Blocks – Collect Them All

- There are 16 (2^4) distinct Boolean logic functions (Logic Blocks) over two input variables.



Logical Completeness

- Any Boolean logic function can be built entirely out of AND and NOT blocks,

Logical Completeness

- Any Boolean logic function can be built entirely out of AND and NOT blocks,
- or out of OR and NOT blocks,
- or out of NAND blocks,
- or out of NOR blocks, etc., etc.

Logical Completeness

- Any Boolean logic function can be built entirely out of AND and NOT blocks,
- or out of OR and NOT blocks,
- or out of NAND blocks,
- or out of NOR blocks, etc., etc.

- SO, what does the Logic Block Game have to do with computer science?

Simple Addition of Binary Digits

$$0 + 0 = 0$$

Simple Addition of Binary Digits

$$0 + 0 = 0$$

$$0 + 1 = 1$$

Simple Addition of Binary Digits

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$

Simple Addition of Binary Digits

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

Simple Addition of Binary Digits

```
0 + 0 = 00
0 + 1 = 01
1 + 0 = 01
1 + 1 = 10
```

Simple Addition of Binary Digits

A	B	C	D
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

C = AND(A, B)

A	B	C	D
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

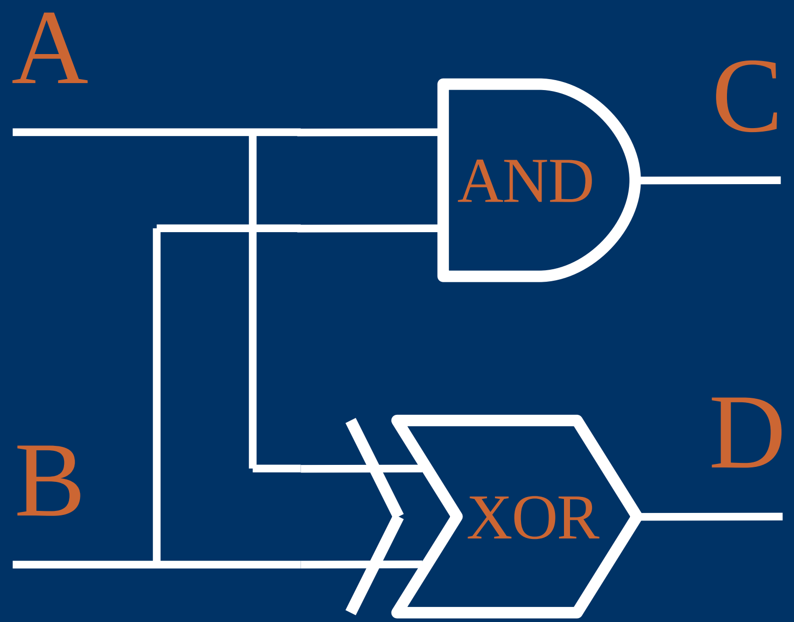
D = XOR(A, B)

A	B	C	D
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Two-Bit Logic Block Computer

```
A   B | C   D
------+------
0   0 | 0   0
0   1 | 0   1
1   0 | 0   1
1   1 | 1   0
```

C = AND(A,B)
D = XOR(A,B)

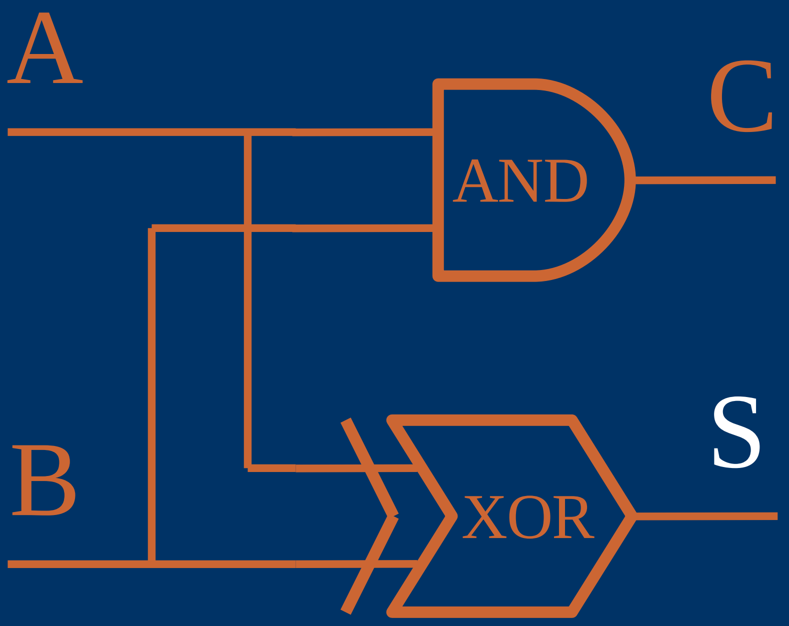


Two-Bit Logic Block Computer

```
A    B | C   S
------+-----
0    0 | 0   0
0    1 | 0   1
1    0 | 0   1
1    1 | 1   0
```

C = AND(A,B)
S = XOR(A,B)

S is for “Sum”
C is for “Carry”

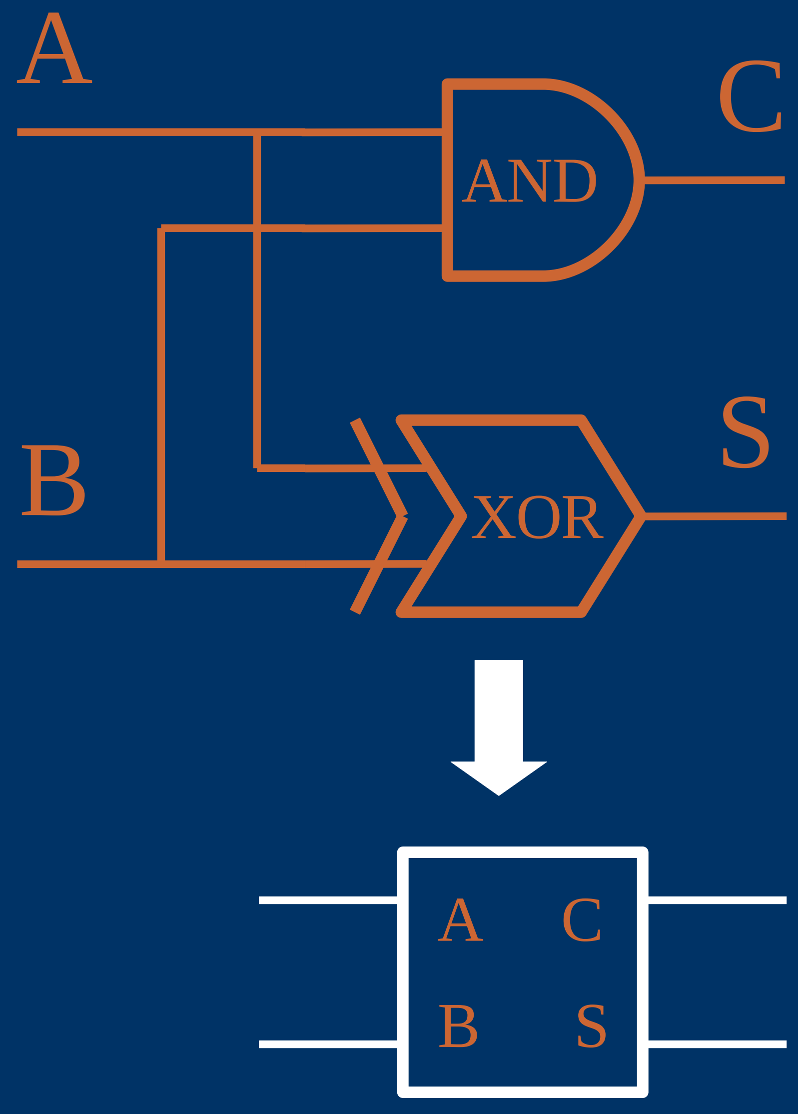


Bigger Blocks – The Half Adder

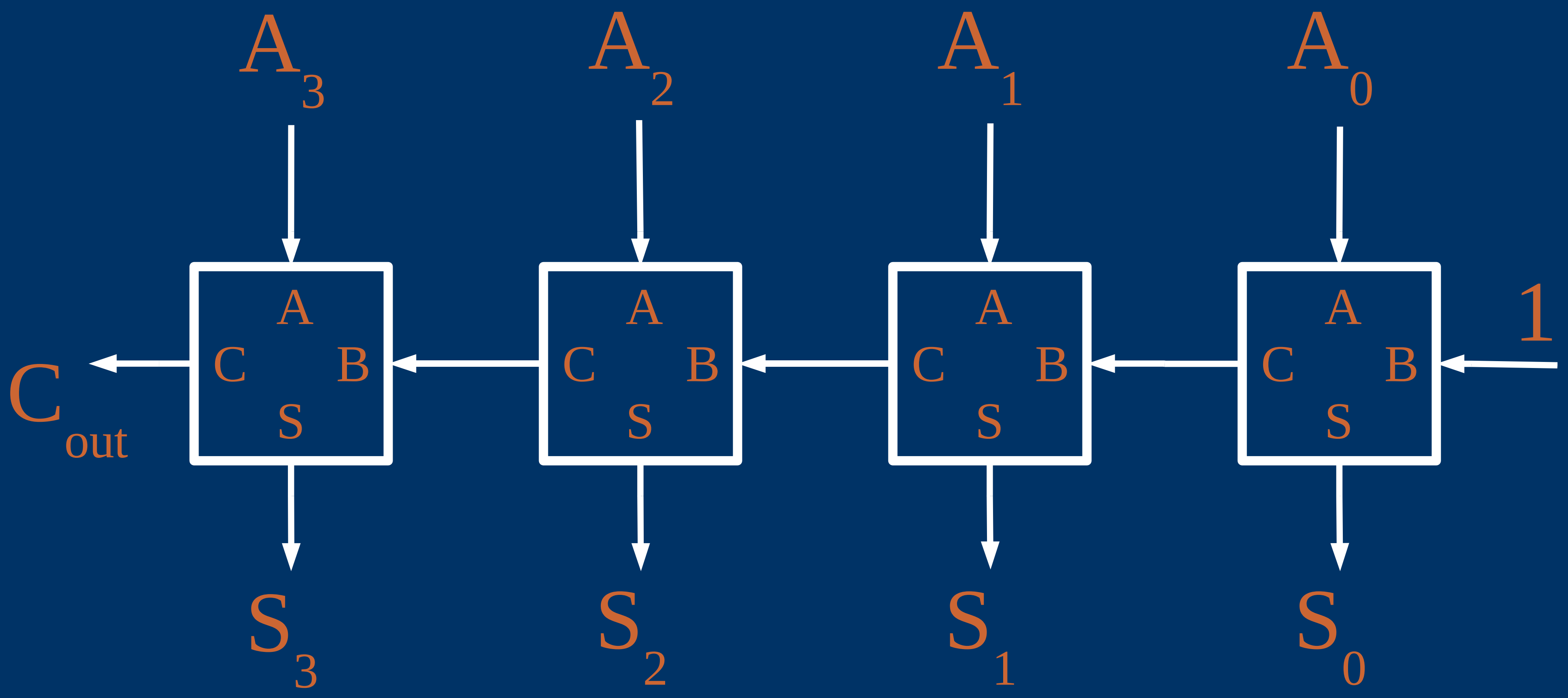
```
A   B | C   S
------+------
0   0 | 0   0
0   1 | 0   1
1   0 | 0   1
1   1 | 1   0
```

C = AND(A,B)
S = XOR(A,B)

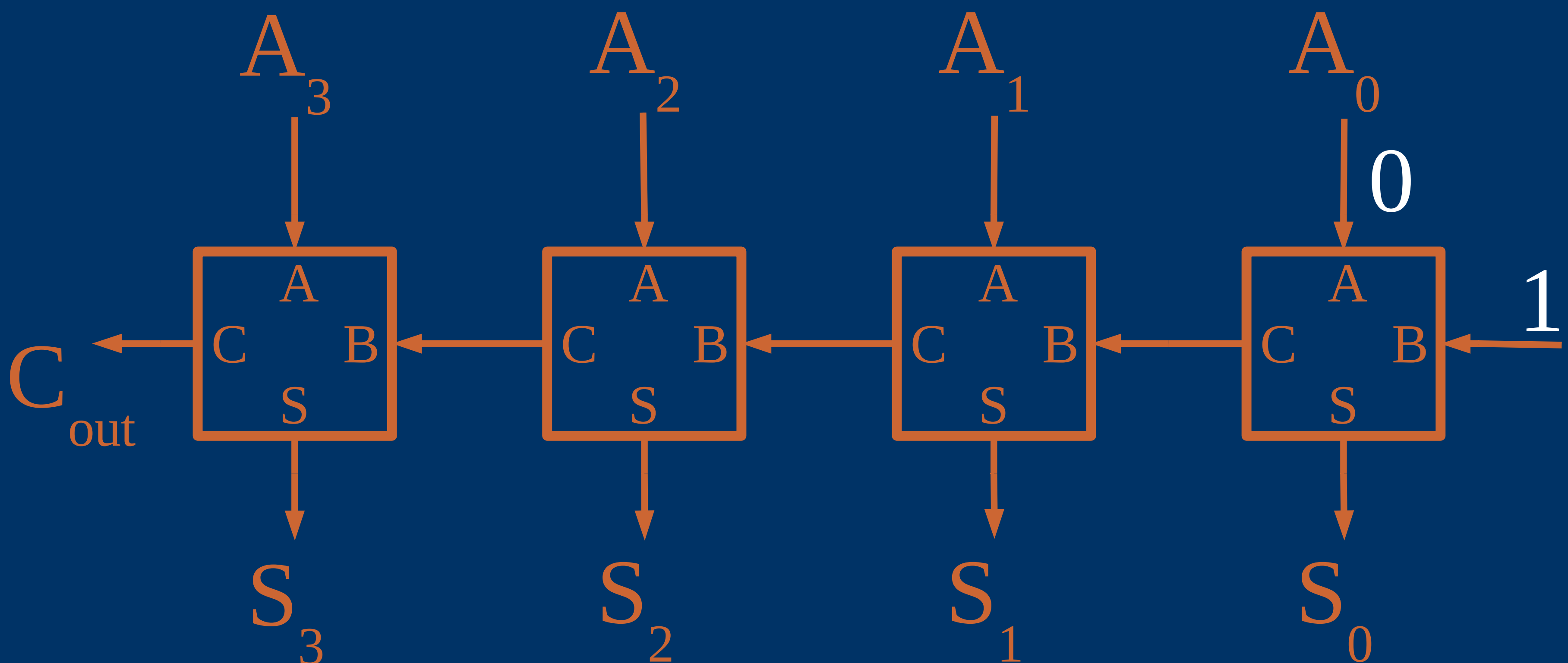
S is for “Sum”
C is for “Carry”



More Than Two Bits

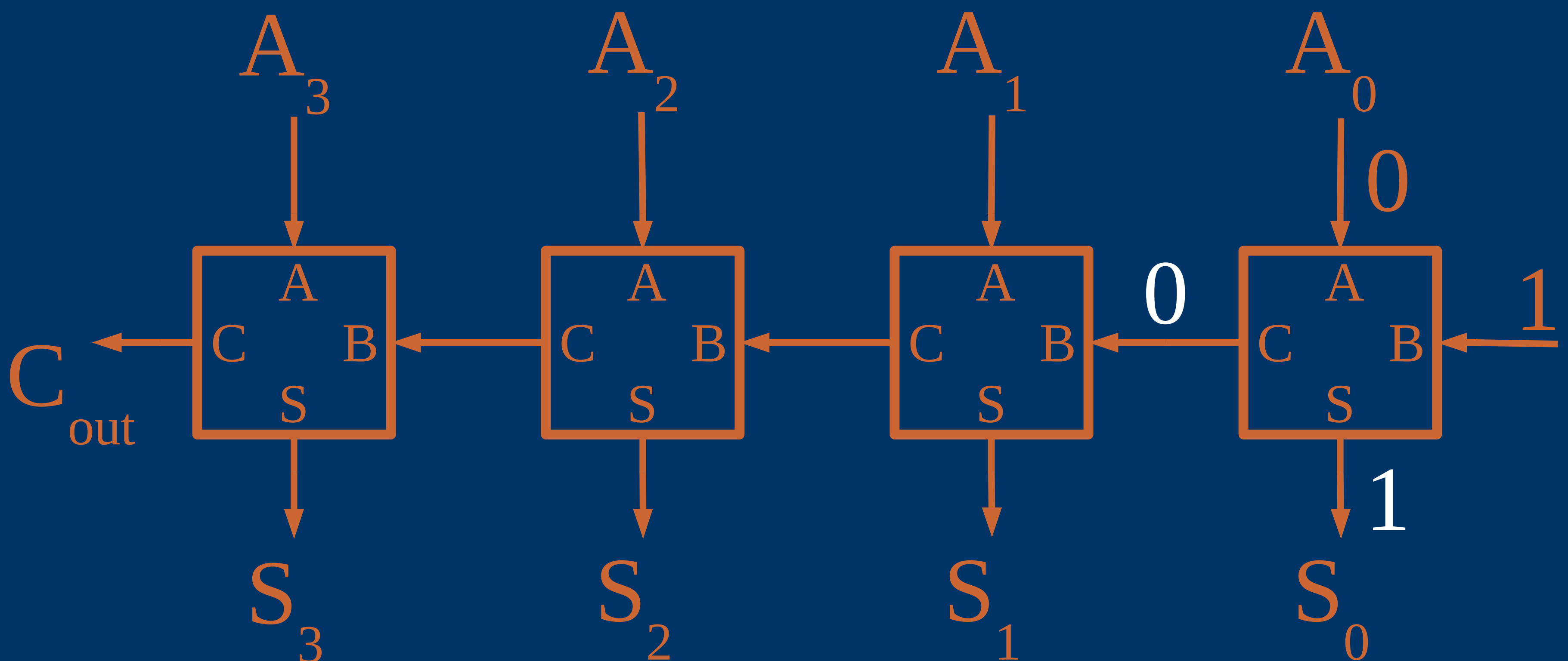


More Than Two Bits



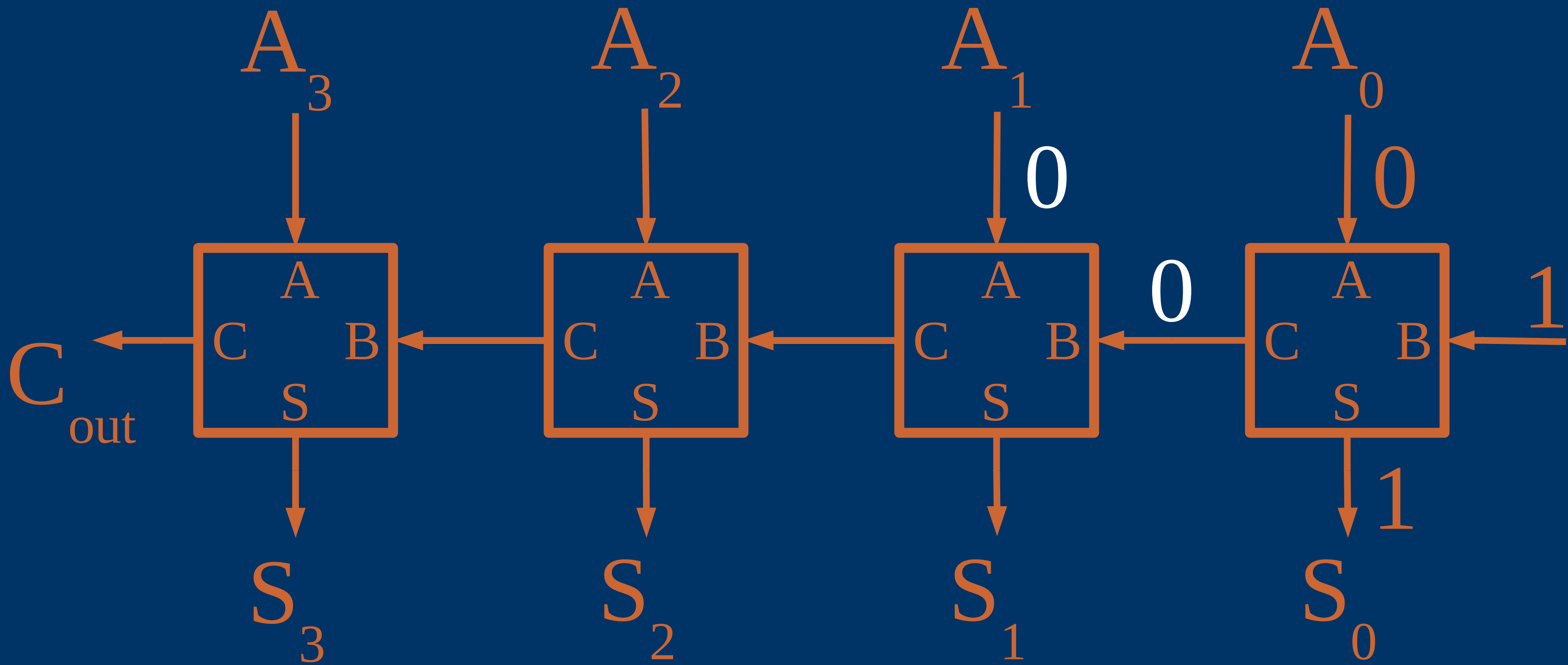
A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	0	0					

More Than Two Bits



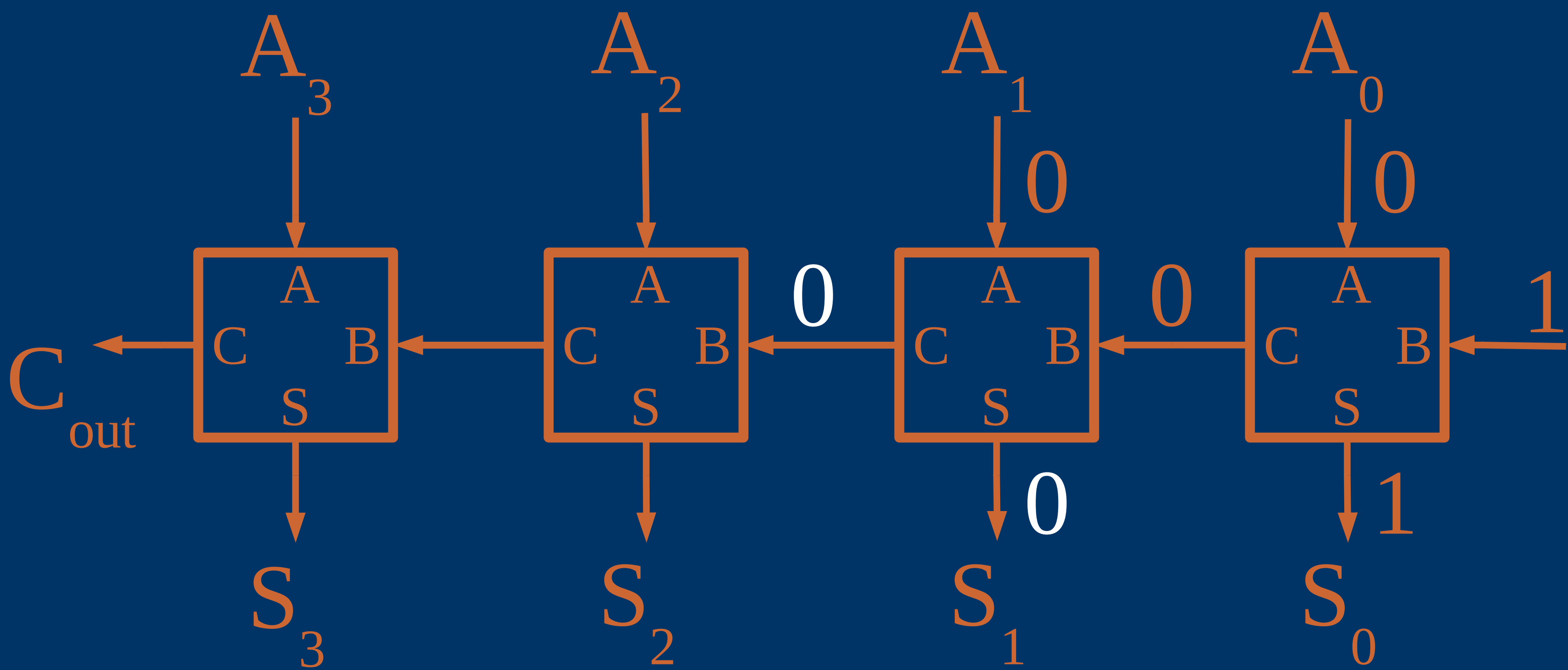
A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	0	0					1

More Than Two Bits



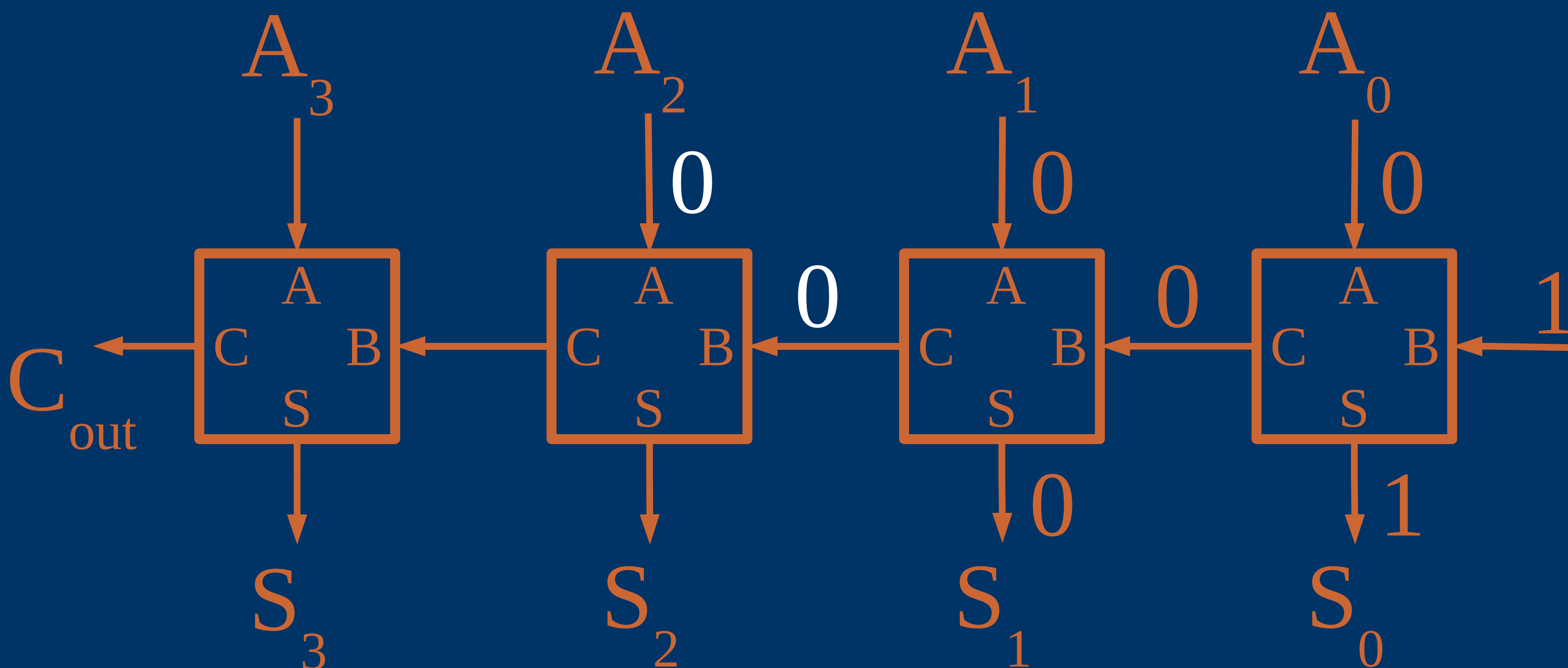
A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	0	0					1

More Than Two Bits



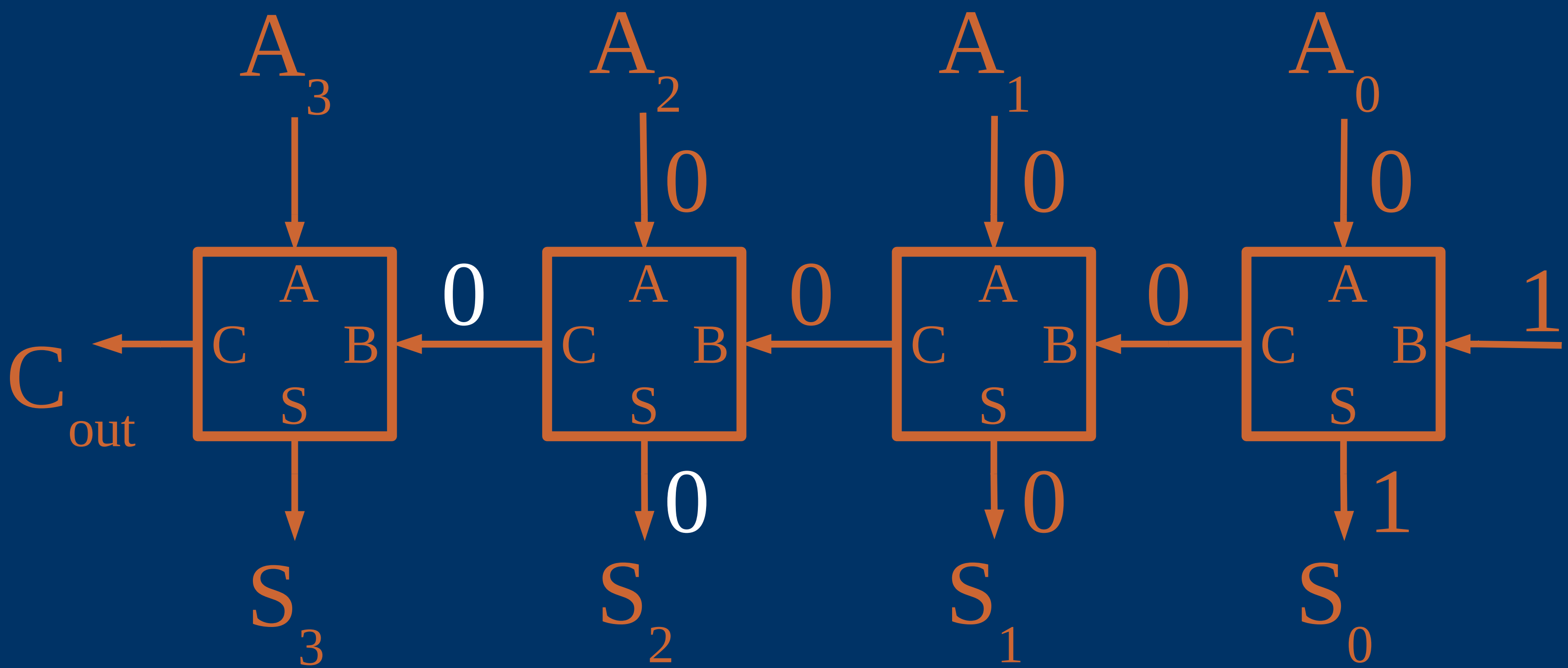
A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	0	0				0	1

More Than Two Bits



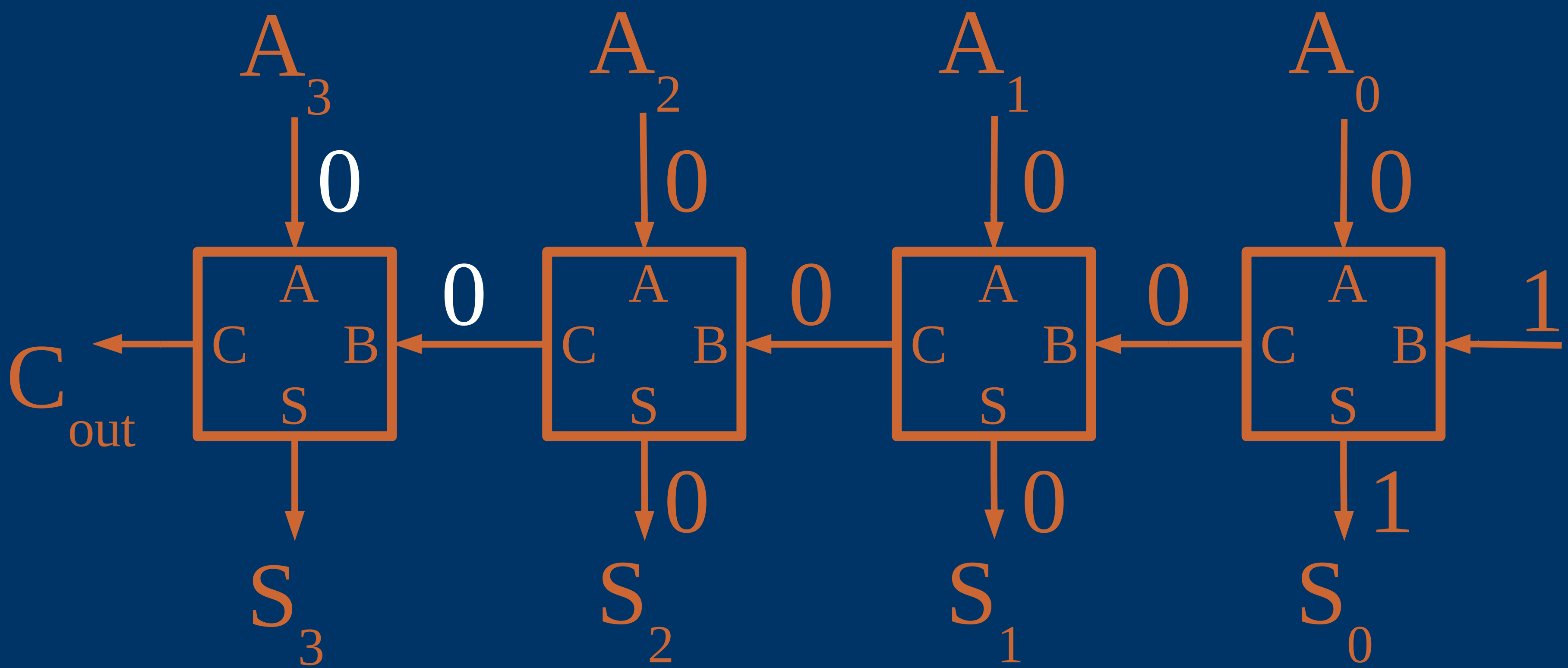
A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	0	0				0	1

More Than Two Bits



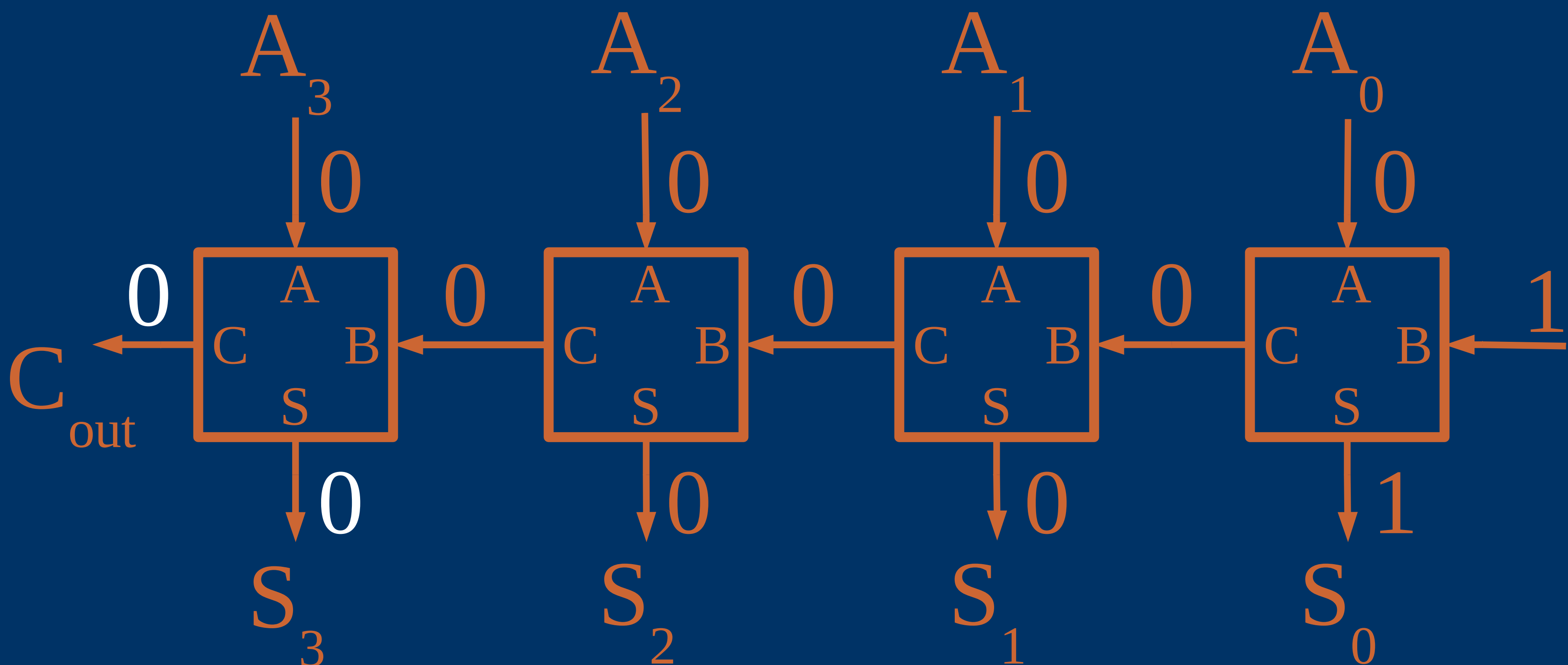
A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	0	0			0	0	1

More Than Two Bits



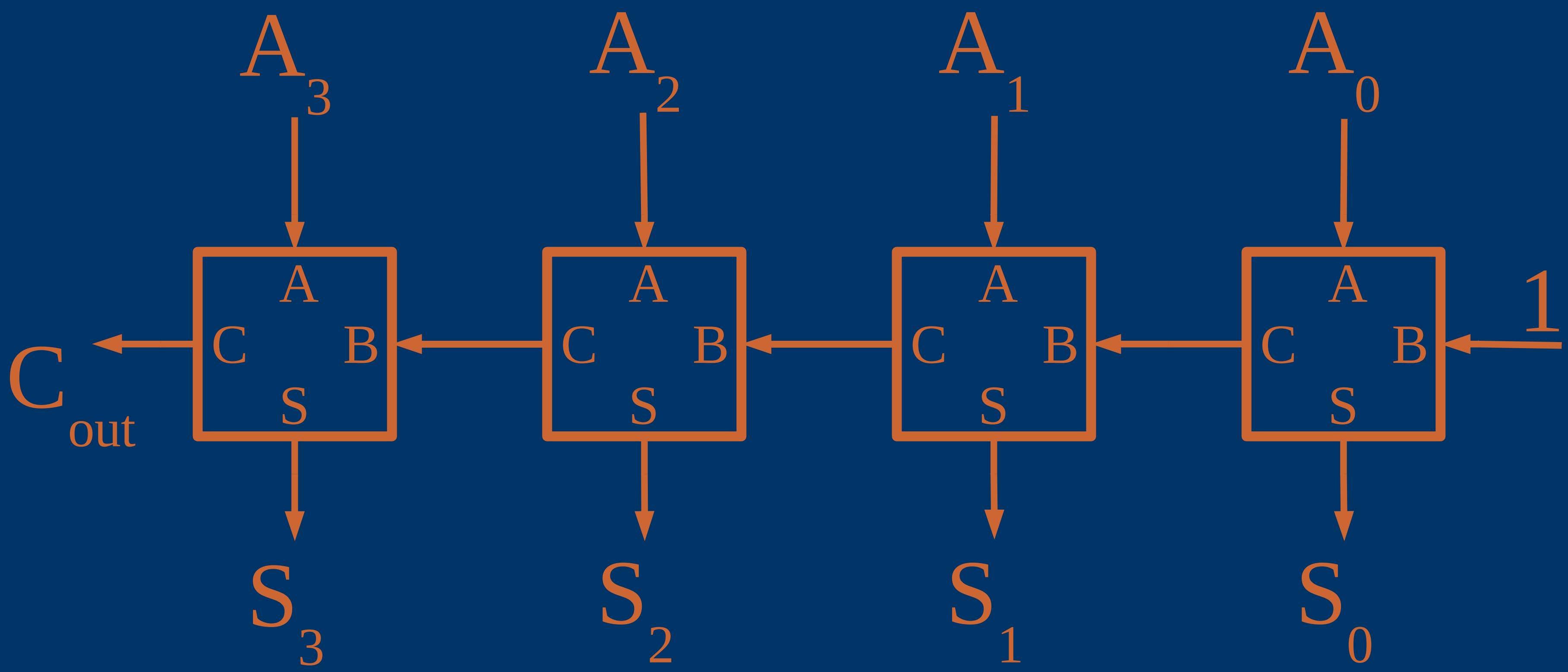
A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	0	0			0	0	1

More Than Two Bits



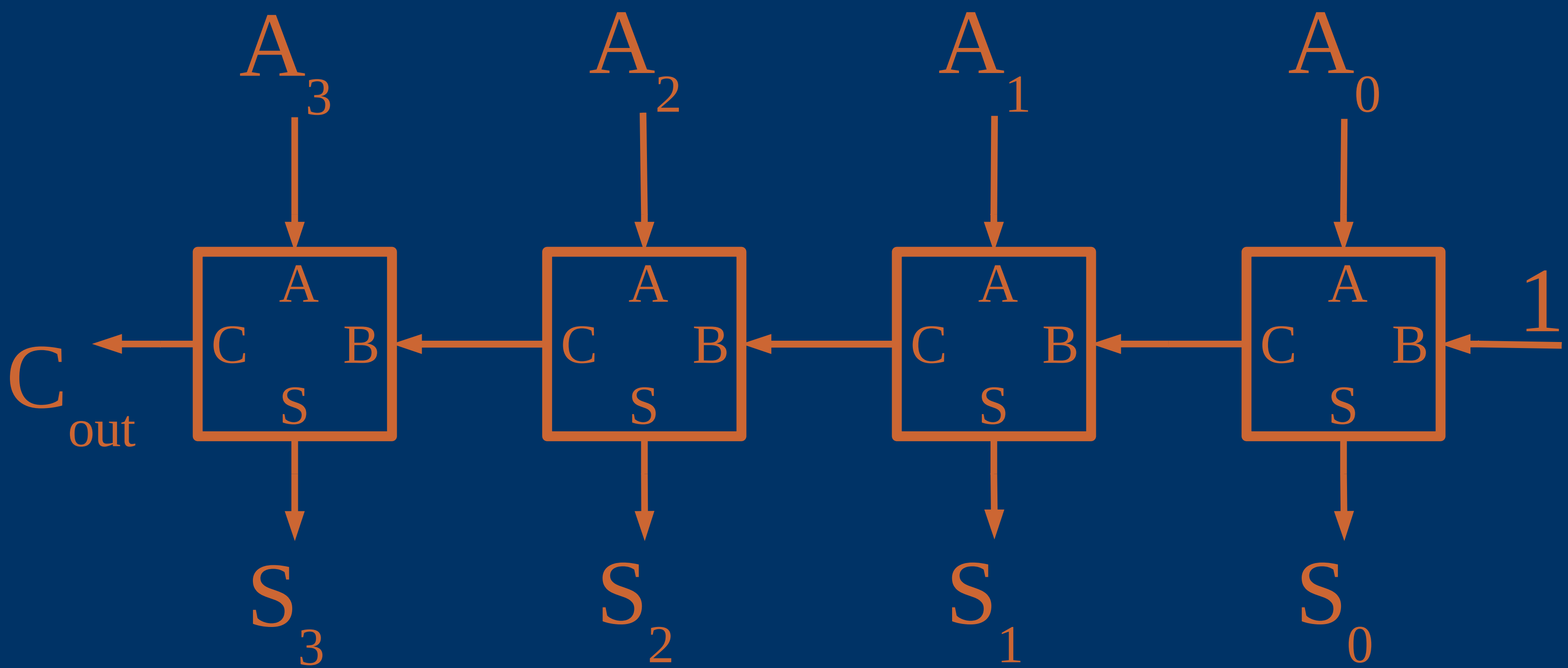
A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0	1

More Than Two Bits



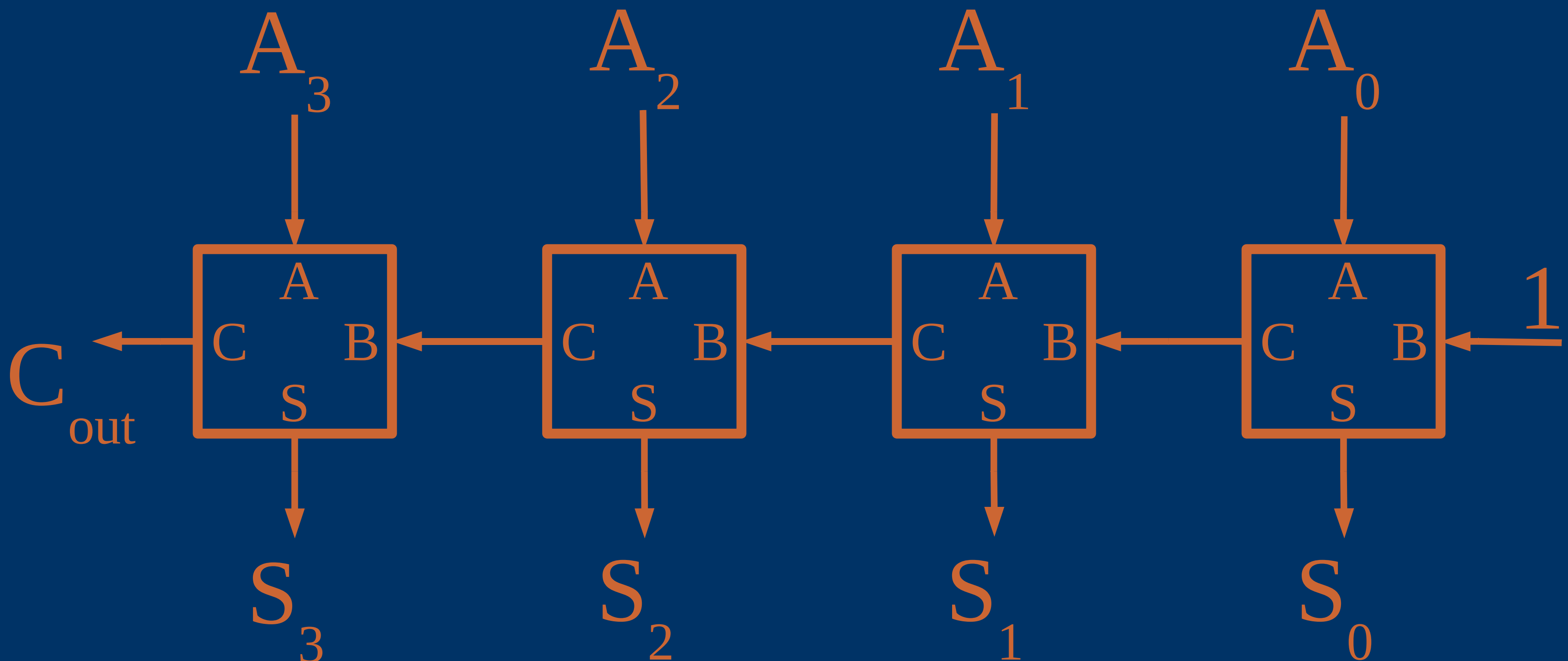
A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	0	1	0	0	0	1	0

More Than Two Bits



A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	1	0	0	0	0	1	1

More Than Two Bits



A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	0	1	1	0	0	1	0	0

More Than Two Bits

A_3	A_2	A_1	A_0	C_{out}	S_3	S_2	S_1	S_0
0	1	0	0	0	0	1	0	1
0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	1	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	0	1	0	0	1
...				...				
1	1	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0

All That Just To Do A+1?

- Any N-bit number can be *incremented* with N half-adder logic blocks.
- We can build a similar combination of logic blocks to *decrement* any N-bit number.
- We can build an N-bit *zero detector* with AND and NOT blocks.

- Where does that leave us?

Addition

```
int add(int x, int y)
{  if (isZero(x))
     {  answer = y;  }
     else
     {  answer = add(dec(x),
                     inc(y));  }
 }
```

Multiplication

```
int mult(int x, int y)
{  if (isZero(y)) {  answer = 0;  }
   else
      {  answer = add(x,
                      mult(x, dec(y))); }}
```

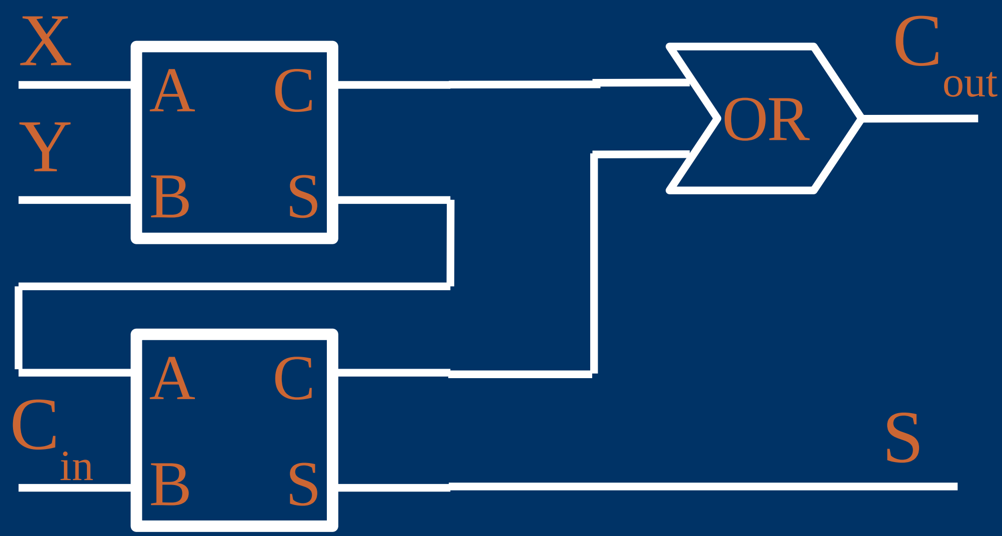
Multiplication

```
int mult(int x, int y)
{  if (isZero(y)) {  answer = 0;  }
   else
     {  answer = add(x,
                  mult(x, dec(y))); }}
```

- Yikes!
- Is this *really* what my computer / calculator is doing?

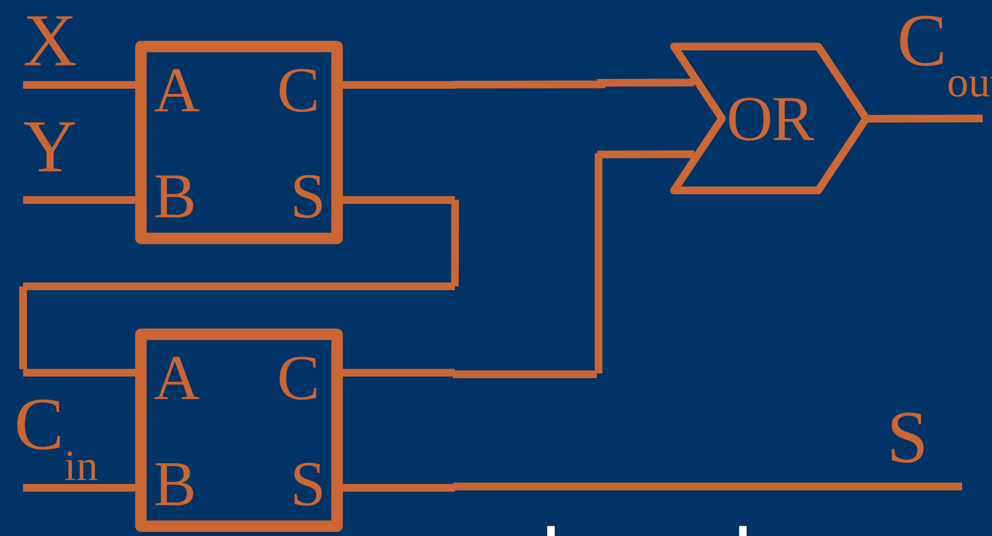
Addition – the Less Scenic Route

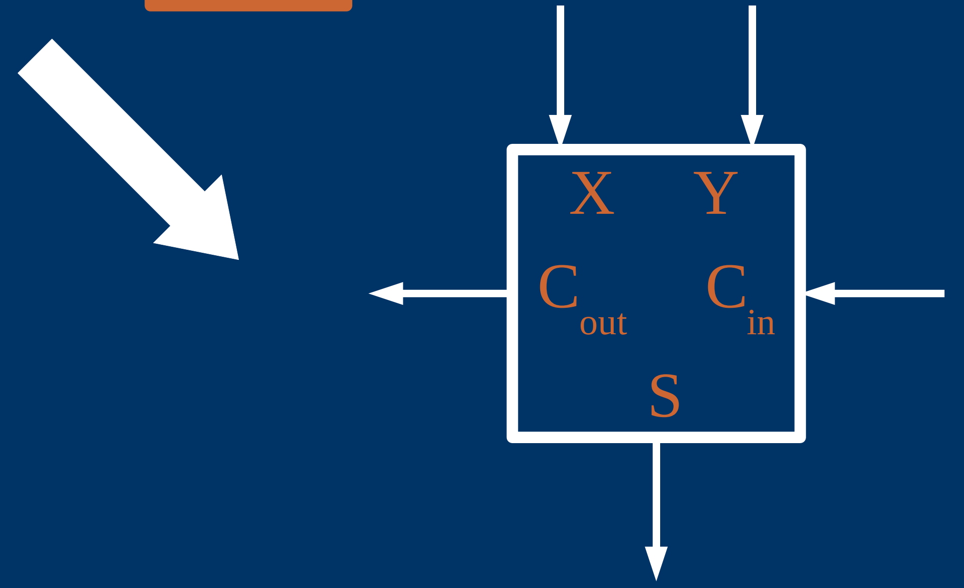
X	Y	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



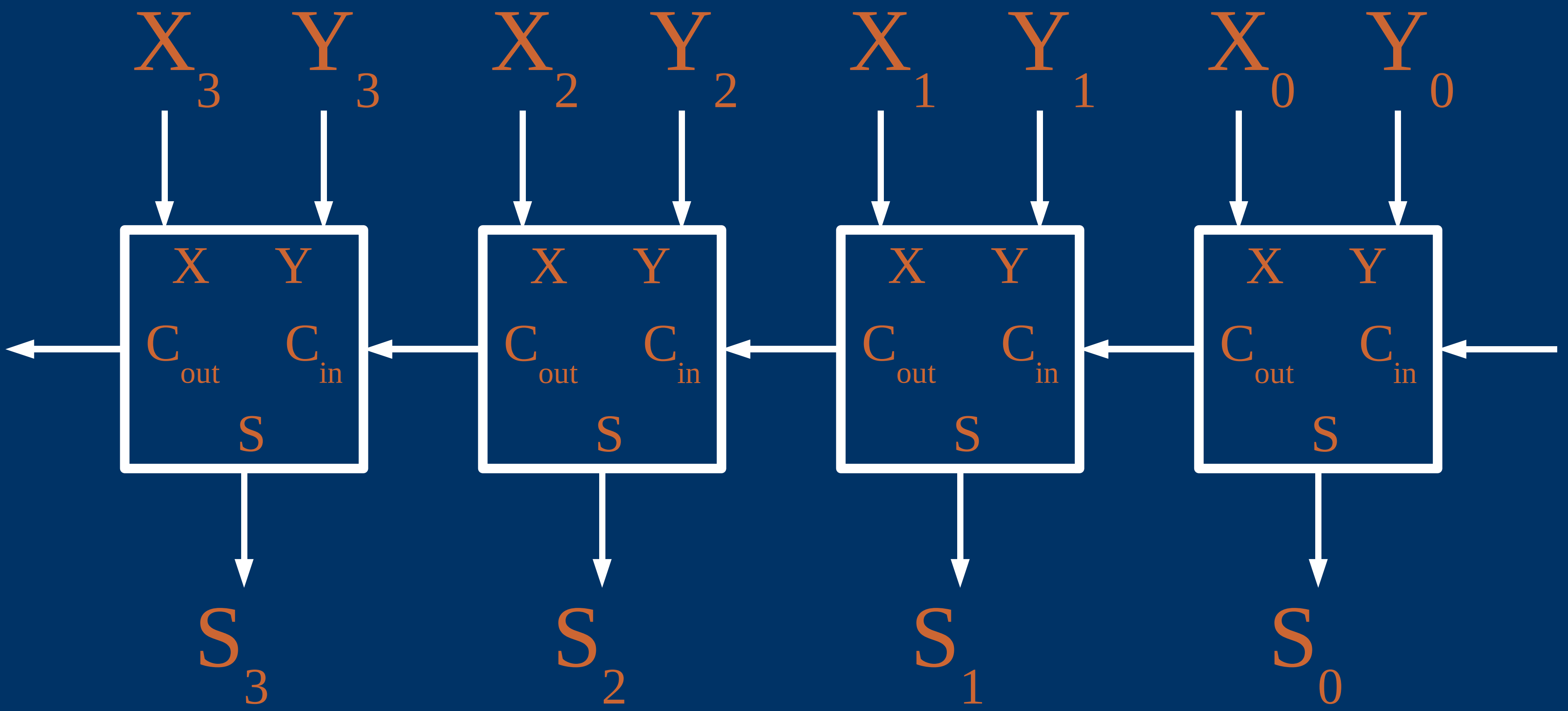
The “Full Adder” carries in and out

X	Y	C_{in}	C_{out}	S
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

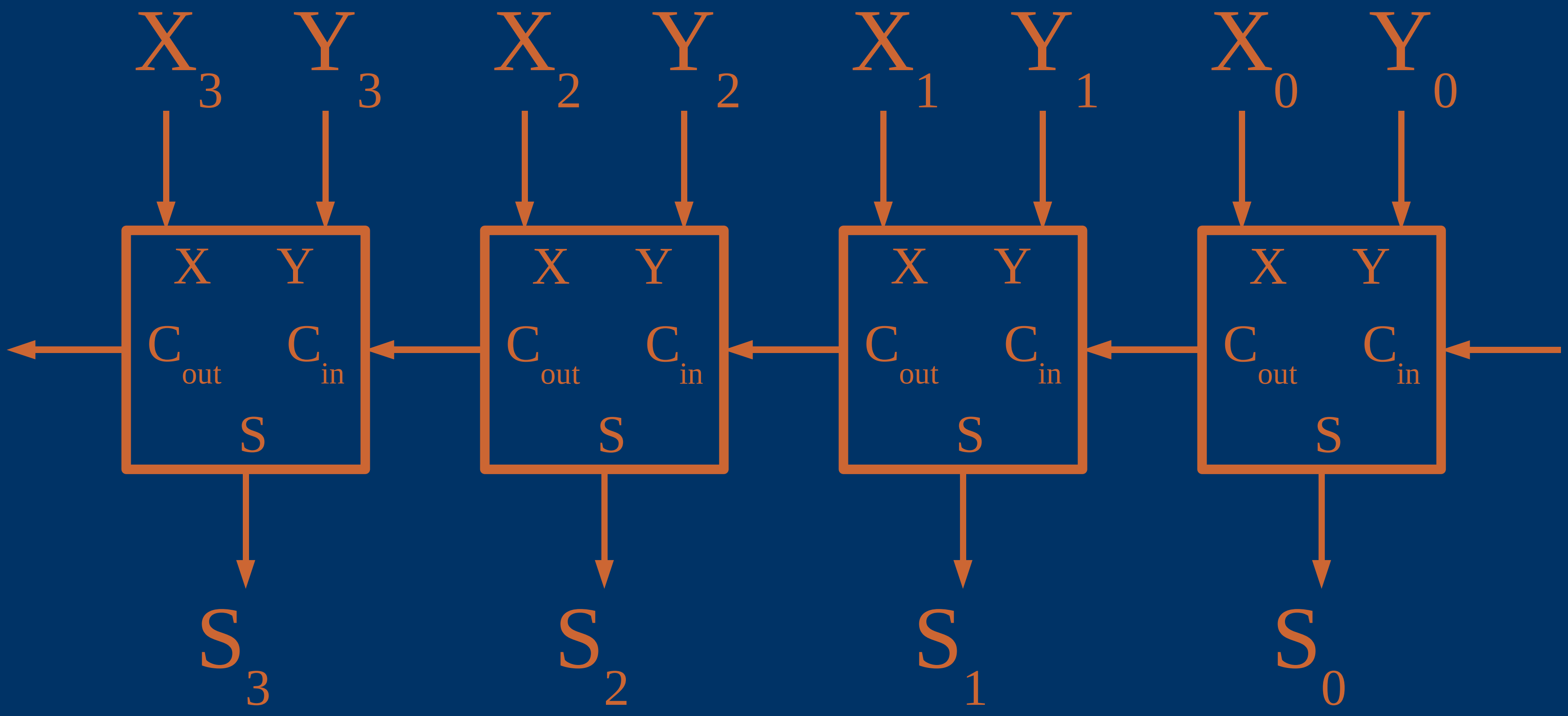




The Ripple Adder



The Ripple Adder



$$C_{out}S_3S_2S_1S_0 = X_3X_2X_1X_0 + Y_3Y_2Y_1Y_0 + C_{in}$$

Multiplication (Less Scenic Route)

- Before we dive into logic block multiplication, let's review “normal”, base-10 multiplication.

$$A_N A_{N-1} \ldots A_2 A_1 A_0 \times B_m B_{m-1} \ldots B_2 B_1 B_0$$

- Think algorithmically
- How do we learn long multiplication as grade school children?
- How is base-10 long multiplication the same as base-2 long multiplication?
- How is it different?

$$
\begin{array}{r}
A_N A_{N-1} \ldots A_2 A_1 A_0 \\
\times\ B_m B_{m-1} \ldots B_2 B_1 B_0 \\
\hline
\end{array}
$$

$$
\begin{array}{r}
A_N A_{N-1} \ldots A_2 A_1 A_0 \\
\times \; B_m B_{m-1} \ldots B_2 B_1 B_0 \\
\hline
B_0 \times A_0
\end{array}
$$

$$A_N A_{N-1} \ldots A_2 A_1 A_0$$

$$\times \; B_m B_{m-1} \ldots B_2 B_1 B_0$$

$$B_0 \times A_1 \times 10 + B_0 \times A_0$$

$$
\begin{array}{r}
A_N A_{N-1} \ldots A_2 A_1 A_0 \\
\times\ B_m B_{m-1} \ldots B_2 B_1 B_0 \\
\hline
B_0 \times A_2 \times 100 + B_0 \times A_1 \times 10 + B_0 \times A_0
\end{array}
$$

$$\begin{array}{r} A_N A_{N-1} \ldots A_2 A_1 A_0 \\ \times\ B_m B_{m-1} \ldots B_2 B_1 B_0 \\ \hline \ldots\ B_0 \times A_2 \times 100 + B_0 \times A_1 \times 10 + B_0 \times A_0 \end{array}$$

$$
\begin{array}{r}
A_N A_{N-1} \ldots A_2 A_1 A_0 \\
\times\ B_m B_{m-1} \ldots B_2 B_1 B_0 \\
\text{-------------------------------} \\
\ldots\ B_0 \times A_2 \times 100 + B_0 \times A_1 \times 10 + B_0 \times A_0 \\
+ \ldots\ B_1 \times A_1 \times 100 + B_1 \times A_0 \times 10
\end{array}
$$

$$
\begin{array}{r}
A_N A_{N-1} \ldots A_2 A_1 A_0 \\
\times\ B_m B_{m-1} \ldots B_2 B_1 B_0 \\
\text{--------------------------} \\
\ldots\ B_0 \times A_2 \times 100 + B_0 \times A_1 \times 10 + B_0 \times A_0 \\
+ \ldots\ B_1 \times A_1 \times 100 + B_1 \times A_0 \times 10 \\
+ \ldots\ B_2 \times A_1 \times 1000 + B_2 \times A_0 \times 100
\end{array}
$$

$$
\begin{array}{r}
A_N A_{N-1} \ldots A_2 A_1 A_0 \\
\times\ B_m B_{m-1} \ldots B_2 B_1 B_0 \\
\hline
\end{array}
$$

```
for i = 0..M
    for j = 0..N
        term = A_j x B_i x 10^j
        partProd = partProd + term
    product = product + partProd x 10^i
```

Peasant's Multiplication

37 x 5

Peasant's Multiplication

37	5
18	10

Peasant's Multiplication

37	5
18	10
9	20

Peasant's Multiplication

37	5
18	10
9	20
4	40

Peasant's Multiplication

37	5
18	10
9	20
4	40
2	80

Peasant's Multiplication

37	5
18	10
9	20
4	40
2	80
1	160

Peasant's Multiplication

37	5
~~18~~	~~10~~
9	20
~~4~~	~~40~~
~~2~~	~~80~~
1	160

Peasant's Multiplication

37	5
~~18~~	~~10~~
9	20
~~4~~	~~40~~
~~2~~	~~80~~
1	+ 160

Peasant's Multiplication

37	5
~~18~~	~~10~~
9	20
~~4~~	~~40~~
~~2~~	~~80~~
1	+ 160
	185

Peasant's Multiplication

37	5
~~18~~	~~10~~
9	20
~~4~~	~~40~~
~~2~~	~~80~~
1	+ 160
	185

Why does this work?

Peasant's Multiplication

37	5	5×2^0
~~18~~	~~10~~	$+ 5 \times 2^1$
9	20	$+ 5 \times 2^2$
~~4~~	~~40~~	$+ 5 \times 2^3$
~~2~~	~~80~~	$+ 5 \times 2^4$
1	+ 160	$+ 5 \times 2^5$
	185	

Peasant's Multiplication

37	5	$5 \times 2^{0} \times 1$
~~18~~	~~10~~	$+\ 5 \times 2^{1} \times 0$
9	20	$+\ 5 \times 2^{2} \times 1$
~~4~~	~~40~~	$+\ 5 \times 2^{3} \times 0$
~~2~~	~~80~~	$+\ 5 \times 2^{4} \times 0$
1	+ 160	$+\ 5 \times 2^{5} \times 1$
	185	

Peasant's Multiplication

37	5	$5 \times 2^0 \times 1$
~~18~~	~~10~~	$+ 5 \times 2^1 \times 0$
9	20	$+ 5 \times 2^2 \times 1$
~~4~~	~~40~~	$+ 5 \times 2^3 \times 0$
~~2~~	~~80~~	$+ 5 \times 2^4 \times 0$
1	+ 160	$+ 5 \times 2^5 \times 1$
	185	

"37_{10}" == "100101_2"

Binary Multiplication

```
   37          100101
x   5       x  000101
```

Binary Multiplication

```
   37          100101
x   5       x  000101
               100101
```

Binary Multiplication

```
   37          100101
x   5      x 000101
              100101
             000000
```

Binary Multiplication

```
   37          100101
x   5        x 000101
               100101
              000000
             100101
```

Binary Multiplication

```
   37              100101
 x  5           x  000101
 ----           ---------
                   100101
                  000000
                 100101
                000000
               000000
              000000
```

Binary Multiplication

```
   37
x   5
-----
```

```
          100101
        x 000101
        --------
          100101
         000000
        100101
       000000
      000000
 +   000000
 ---------------
     00010111001
```

Binary Multiplication

```
   37
x   5
  185
```

```
          100101
        x 000101
          100101
         000000
        100101
       000000
      000000
  + 000000
    00010111001 = 185
```

$00010111001 = 185_{10}$

Back to Logic Blocks

How do we multiply two bits with Logic Blocks?

Back to Logic Blocks

How do we multiply two bits with Logic Blocks?

```
X   Y | X*Y
-----+----
      |
      |
      |
      |
```

Back to Logic Blocks

How do we multiply two bits with Logic Blocks?

```
X   Y | X*Y
------+-----
0   0 |  0
      |
      |
      |
```

Back to Logic Blocks

How do we multiply two bits with Logic Blocks?

```
X   Y | X*Y
-----+----
0   0 |  0
0   1 |  0
      |
      |
```

Back to Logic Blocks

How do we multiply two bits with Logic Blocks?

```
X   Y | X*Y
------+-----
0   0 |  0
0   1 |  0
1   0 |  0
      |
```

Back to Logic Blocks

How do we multiply two bits with Logic Blocks?

```
X   Y | X*Y
-----+----
0   0 |  0
0   1 |  0
1   0 |  0
1   1 |  1
```

Back to Logic Blocks

How do we multiply two bits with Logic Blocks?

```
X   Y | X*Y
------+-----
0   0 |  0
0   1 |  0
1   0 |  0
1   1 |  1
```

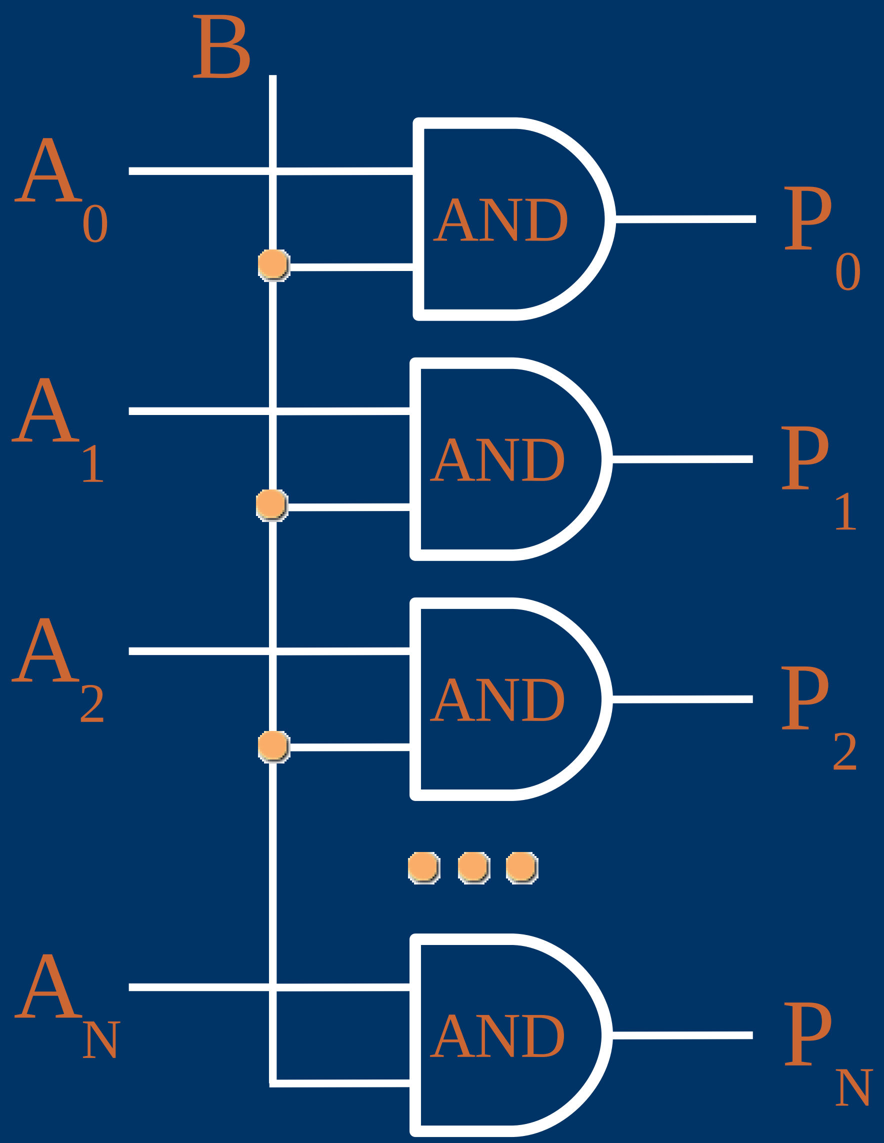
Back to Logic Blocks

How do we multiply two bits with Logic Blocks?

```
X  Y | X*Y
-----+-----
0  0 |  0
0  1 |  0
1  0 |  0
1  1 |  1
```

Multiplying two bits never carries into next column!

Bit Multiplier



A group of 'N' AND Blocks produces the partial product from a one-bit multiplier.

Logic Block Multiplication

- With a Bit Multiplier Logic Block and a Full Adder Logic Block for each bit of multiplier, we can construct a Logic Block for performing long multiplication of binary numbers.

Logic Block Multiplication

- With a Bit Multiplier Logic Block and a Full Adder Logic Block for each bit of multiplier, we can construct a Logic Block for performing long multiplication of binary numbers.

- The Logic Block Game has very simple rules.
- Only a small number of basic blocks types required.
- Surprisingly complex logic can be constructed from very little.