

ECE/CS 250

Computer Architecture

Summer 2024

Basics of Logic Design: Boolean Algebra, Logic Gates, and the ALU (Combinational Logic)

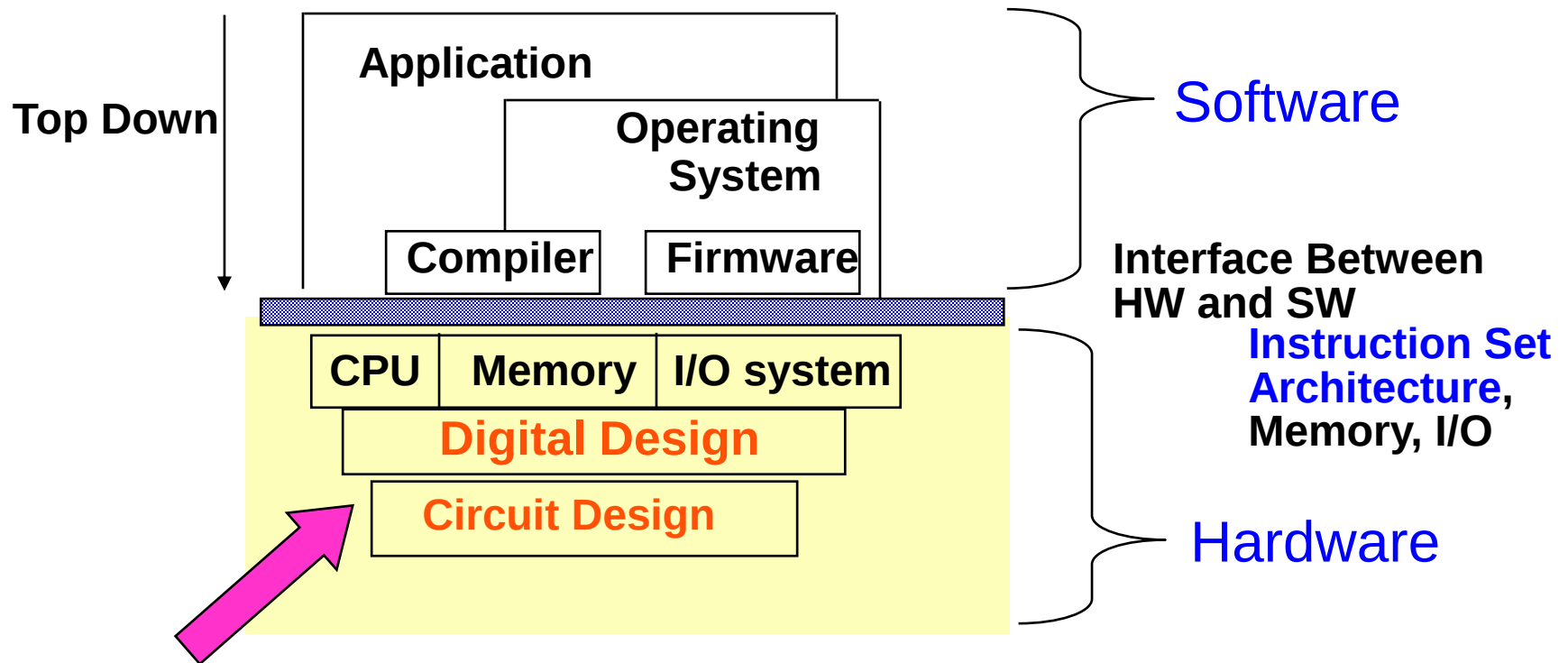
Tyler Bletsch
Duke University

Slides are derived from work by
Daniel J. Sorin (Duke), Alvy Lebeck (Duke), and Drew Hilton (Duke)

Reading

- Appendix B (parts 1,2,3,5,6,7,8,9,10)
- This material is covered in MUCH greater depth in ECE/CS 350 – please take ECE/CS 350 if you want to learn enough digital design to build your own processor

What We've Done, Where We're Going



(Almost) Bottom UP to CPU

Computer = Machine That Manipulates Bits

- Everything is in binary (bunches of 0s and 1s)
 - Instructions, numbers, memory locations, etc.
- Computer is a machine that operates on bits
 - Executing instructions → operating on bits
- Computers physically made of transistors
 - Electrically controlled switches
- We can use transistors to build logic
 - E.g., if this bit is a 0 and that bit is a 1, then set some other bit to be a 1
 - E.g., if the first 5 bits of the instruction are 10010 then set this other bit to 1 (to tell the adder to subtract instead of add)

How Many Transistors Are We Talking About?

Pentium III

- Processor Core 9.5 Million Transistors
- Total: 28 Million Transistors

Pentium 4

- Total: 42 Million Transistors

Core2 Duo (two processor cores)

- Total: 290 Million Transistors

Core2 Duo Extreme (4 processor cores, 8MB cache)

- Total: 590 Million Transistors

Core i7 with 6-cores

- Total: 2.27 Billion Transistors

How do they design such a thing? Carefully!

Abstraction!

- Use of **abstraction** (key to design of any large system)
 - Put a few (2-8) transistors into a **logic gate** (or, and, xor, ...)
 - **Combine gates into logical functions** (add, select,...)
 - Combine adders, shifters, etc., together into modules
 - Units with well-defined interfaces for large tasks: e.g., decode
 - Combine a dozen of those into a core...
 - Stick 4 cores on a chip...

Boolean Algebra

- First step to logic: Boolean Algebra
 - Manipulation of True / False (1/0)
 - After all: everything is just 1s and 0s
- Given inputs (variables): A, B, C, P, Q...
 - Compute outputs using logical operators, such as:
- NOT: $\neg A$ ($= \sim A = \bar{A}$)
- AND: $A \& B$ ($= A \cdot B = A * B = AB = A \wedge B$) = `A&&B` in C/C++
- OR: $A \mid B$ ($= A + B = A \vee B$) = `A || B` in C/C++
- XOR: $A \wedge B$ ($= A \oplus B$)
- NAND, NOR, XNOR, Etc.

Truth Tables

- Can represent as **truth table**: shows outputs for all inputs

a	NOT (a)
0	1
1	0

a	b	AND (a , b)
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR (a , b)
0	0	0
0	1	1
1	0	1
1	1	1

a	b	XOR (a , b)
0	0	0
0	1	1
1	0	1
1	1	0

a	b	XNOR (a , b)
0	0	1
0	1	0
1	0	0
1	1	1

a	b	NOR (a , b)
0	0	1
0	1	0
1	0	0
1	1	0

Any Inputs, Any Outputs

- Can have any # of inputs, any # of outputs
- Can have arbitrary functions:

a	b	c	f_1	f_2
0	0	0	0	1
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

Let's Write a Truth Table for a Function...

- Example:
(A & B) | !C

Start with Empty TT

Column Per Input

Column Per Output

A	B	C	Output

Let's write a Truth Table for a function...

- Example:
(A & B) | !C

Start with Empty TT

Column Per Input

Column Per Output

Fill in Inputs

Counting in Binary

A	B	C	Output
0	0	0	

Let's write a Truth Table for a function...

- Example:
(A & B) | !C

Start with Empty TT

Column Per Input

Column Per Output

Fill in Inputs

Counting in Binary

A	B	C	Output
0	0	0	
0	0	1	

Let's write a Truth Table for a function...

- Example:
(A & B) | !C

Start with Empty TT

Column Per Input

Column Per Output

Fill in Inputs

Counting in Binary

A	B	C	Output
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Let's write a Truth Table for a function...

- Example:
 $(A \& B) \mid !C$

Start with Empty TT

Column Per Input

Column Per Output

Fill in Inputs

Counting in Binary

Compute Output

$$(0 \& 0) \mid !0 = 0 \mid 1 = 1$$

A	B	C	Output
0	0	0	1
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Let's write a Truth Table for a function...

- Example:
(A & B) | !C

Start with Empty TT

Column Per Input

Column Per Output

Fill in Inputs

Counting in Binary

Compute Output

$(0 \& 0) \mid !1 = 0 \mid 0 = 0$

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Let's write a Truth Table for a function...

- Example:
(A & B) | !C

Start with Empty TT

Column Per Input

Column Per Output

Fill in Inputs

Counting in Binary

Compute Output

$$(0 \& 1) \mid !0 = 0 \mid 1 = 1$$

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Let's write a Truth Table for a function...

- Example:
(A & B) | !C

Start with Empty TT

Column Per Input

Column Per Output

Fill in Inputs

Counting in Binary

Compute Output

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



Logisim example

basic_logic.circ : example1

Suppose I turn it around...

- Given a Truth Table, find the formula?

Hmmm..

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Suppose I turn it around...

- Given a Truth Table, find the formula?

Hmmm ...

Could write down every "true" case

Then OR together:

$(\neg A \ \& \ \neg B \ \& \ \neg C) \mid$

$(\neg A \ \& \ \neg B \ \& \ C) \mid$

$(\neg A \ \& \ B \ \& \ \neg C) \mid$

$(A \ \& \ B \ \& \ \neg C) \mid$

$(A \ \& \ B \ \& \ C)$

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Suppose I turn it around...

- Given a Truth Table, find the formula?

Hmmm..

Could write down every “true” case

Then OR together:

$(\neg A \ \& \ \neg B \ \& \ \neg C) \mid$

$(\neg A \ \& \ \neg B \ \& \ C) \mid$

$(\neg A \ \& \ B \ \& \ \neg C) \mid$

$(A \ \& \ B \ \& \ \neg C) \mid$

$(A \ \& \ B \ \& \ C)$

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Suppose I turn it around...

- Given a Truth Table, find the formula?

Hmmm..

Could write down every "true" case

Then OR together:

$(\neg A \ \& \ \neg B \ \& \ \neg C) \mid$

$(\neg A \ \& \ \neg B \ \& \ C) \mid$

$(\neg A \ \& \ B \ \& \ \neg C) \mid$

$(A \ \& \ B \ \& \ \neg C) \mid$

$(A \ \& \ B \ \& \ C)$

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Suppose I turn it around...

- This approach: “sum of products”
 - Works every time
 - Result is right...
 - But really ugly

$(\neg A \ \& \ \neg B \ \& \ \neg C) \quad |$

$(\neg A \ \& \ \neg B \ \& \ C) \quad |$

$(\neg A \ \& \ B \ \& \ \neg C) \quad |$

$(A \ \& \ B \ \& \ \neg C) \quad |$

$(A \ \& \ B \ \& \ C)$

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Suppose I turn it around...

- This approach: “sum of products”
 - Works every time
 - Result is right...
 - But really ugly

(!A & !B & !C) |

(!A & !B & C) |

(!A & B & !C) |

(A & B & !C) |

(A & B & C)

Could just be (A & B) here ?

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Suppose I turn it around...

- This approach: “sum of products”
 - Works every time
 - Result is right...
 - But really ugly

(!A & !B & !C) |
(!A & !B & C) |
(!A & B & !C) |
(A&B)

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Suppose I turn it around...

- This approach: “sum of products”
 - Works every time
 - Result is right...
 - But really ugly

(!A & !B & !C) |

(!A & !B & C) |

(!A & B & !C) |

(A&B)

Could just be (!A & !B) here

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Suppose I turn it around...

- This approach: “sum of products”
 - Works every time
 - Result is right...
 - But really ugly

$(\neg A \ \& \ \neg B) \mid$

$(\neg A \ \& \ B \ \& \ \neg C) \mid$

$(A \ \& \ B)$

Could just be $(\neg A \ \& \ \neg B)$ here

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Suppose I turn it around...

- This approach: “sum of products”
 - Works every time
 - Result is right...
 - But really ugly

$(\neg A \ \& \ \neg B) \mid$
 $(\neg A \ \& \ B \ \& \ \neg C) \mid$
 $(A \ \& \ B)$

Looks nicer...

Can we do better?

A	B	C	Output
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Just did some of these by intuition.. but

- Somewhat intuitive approach to simplifying
- This is **math**, so there are formal rules
 - Just like “regular” algebra

Boolean Function Simplification

- Boolean expressions can be simplified by using the following rules (bitwise logical):

- $A \& A = A$

- $A \& 0 = 0$

- $A \& 1 = A$

- $A \& !A = 0$

$$A \mid A = A$$

$$A \mid 0 = A$$

$$A \mid 1 = 1$$

$$A \mid !A = 1$$

- $!!A = A$

- $\&$ and $\mid$ are both commutative and associative

- $\&$ and $\mid$ can be distributed: $A \& (B \mid C) = (A \& B) \mid (A \& C)$

- $\&$ and $\mid$ can be subsumed: $A \mid (A \& B) = A$

DeMorgan's Laws

- Two (less obvious) Laws of Boolean Algebra:
 - Let's push negations inside, flipping & and |

$$\neg (A \ \& \ B) = (\neg A) \ | \ (\neg B)$$

$$\neg (A \ | \ B) = (\neg A) \ \& \ (\neg B)$$

- You should try this at home – build truth tables for both the left and right sides and see that they're the same

Summary of all Boolean axioms



Name	AND form	OR form
Identity law	$1 \& A = A$	$0 \mid A = A$
Null law	$0 \& A = 0$	$1 \mid A = 1$
Idempotent law	$A \& A = A$	$A \mid A = A$
Inverse law	$A \& !A = 0$	$A \mid !A = 1$
Commutative law	$A \& B = B \& A$	$A \mid B = B \mid A$
Associative law	$(A \& B) \& C = A \& (B \& C)$	$(A \mid B) \mid C = A \mid (B \mid C)$
Distributive law	$A \mid (B \& C) = (A \mid B) \& (A \mid C)$	$A \& (B \mid C) = (A \& B) \mid (A \& C)$
Absorption law	$A \& (A \mid B) = A$	$A \mid (A \& B) = A$
De Morgan's law	$!(A \& B) = !A \mid !B$	$!(A \mid B) = !A \& !B$
Double negation law	$!!A = A$	

Simplification Example:

$\neg (\neg A \mid \neg (A \ \& \ (B \mid C)))$

DeMorgan's

$\neg \neg A \ \& \ \neg \neg (A \ \& \ (B \mid C))$

Double Negation Elimination

$A \ \& \ (A \ \& \ (B \mid C))$

Associativity of &

$(A \ \& \ A) \ \& \ (B \mid C)$

$A \ \& \ A = A$

$A \ \& \ (B \mid C)$

You try this:

Come up with a formula for this Truth Table

Simplify as much as possible

Sum of Products:

$(\neg A \ \& \ \neg B \ \& \ \neg C) \mid$

$(\neg A \ \& \ B \ \& \ \neg C) \mid$

$(A \ \& \ \neg B \ \& \ C) \mid$

$(A \ \& \ B \ \& \ C)$

Simplify this part

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

You try this:

Simplify:

$$(\neg A \ \& \ \neg B \ \& \ \neg C) \mid (\neg A \ \& \ B \ \& \ \neg C)$$

Regroup (associative/commutative):

$$((\neg A \ \& \ \neg C) \ \& \ \neg B) \mid ((\neg A \ \& \ \neg C) \ \& \ B)$$

Un-distribute (factor):

$$(\neg A \ \& \ \neg C) \ \& \ (\neg B \mid B)$$

OR identities:

$$(\neg A \ \& \ \neg C) \ \& \ \text{true} \quad = \quad (\neg A \ \& \ \neg C)$$

You try this:

Come up with a formula for this Truth Table
Simplify as much as possible

Sum of Products: Result of simplifying

$(\neg A \ \& \ \neg C)$ |

$(A \ \& \ \neg B \ \& \ C)$ |

$(A \ \& \ B \ \& \ C)$

You can simplify this
part in the same way...

A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

You try this:

Come up with a formula for this Truth Table

Simplify as much as possible

Sum of Products:

$(\neg A \ \& \ \neg C) \mid$
 $(A \ \& \ C)$

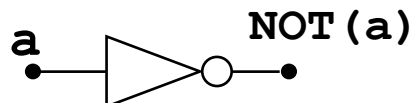
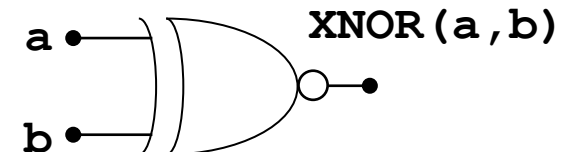
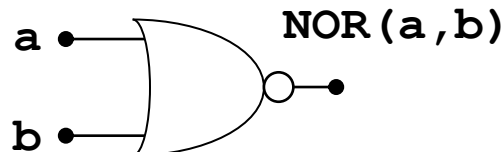
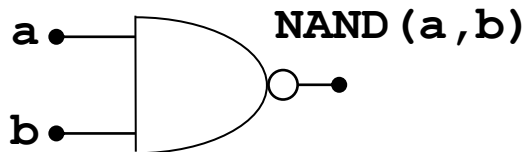
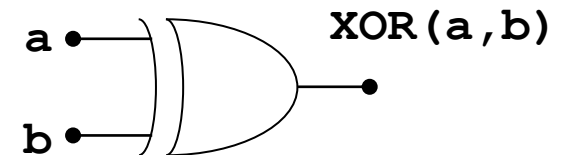
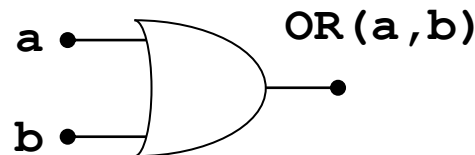
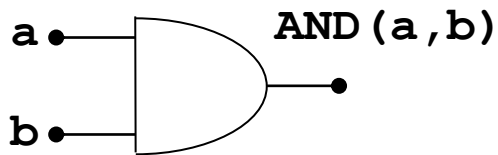
A	B	C	Output
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Applying the Theory

- Lots of good theory
- Can reason about complex Boolean expressions
- But why is this useful?

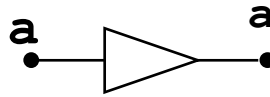
Boolean Gates

- **Gates** are electronic devices that implement simple Boolean functions (building blocks of hardware)

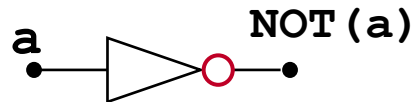


Guide to Remembering your Gates

- This one looks like it just points its input where to go
 - It just produces its input as its output
 - Called a buffer

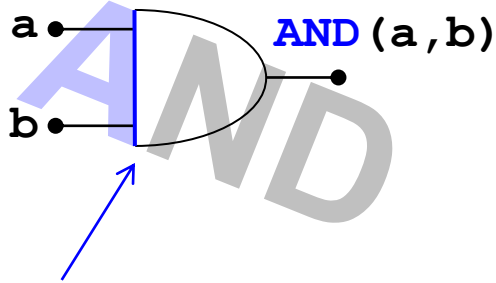


- A circle always means negate (invert)

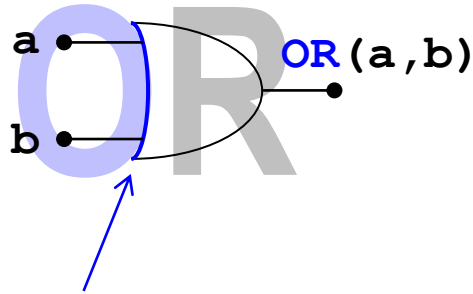


Circle = NOT

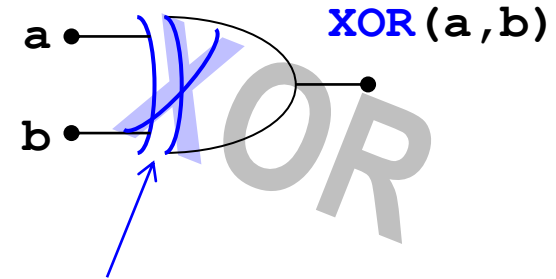
Guide to Remembering your Gates



Straight like an A

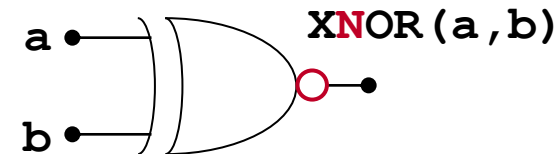
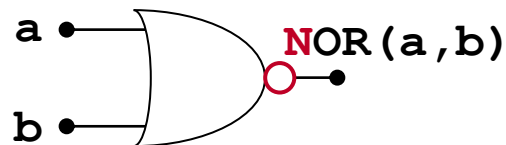
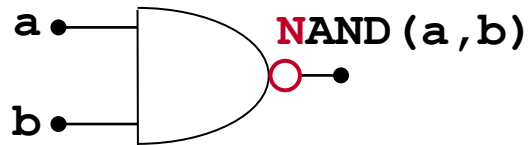


Curved, like an O

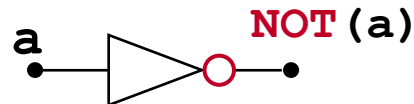


XOR looks like OR (curved line), but has two lines (like an X does)

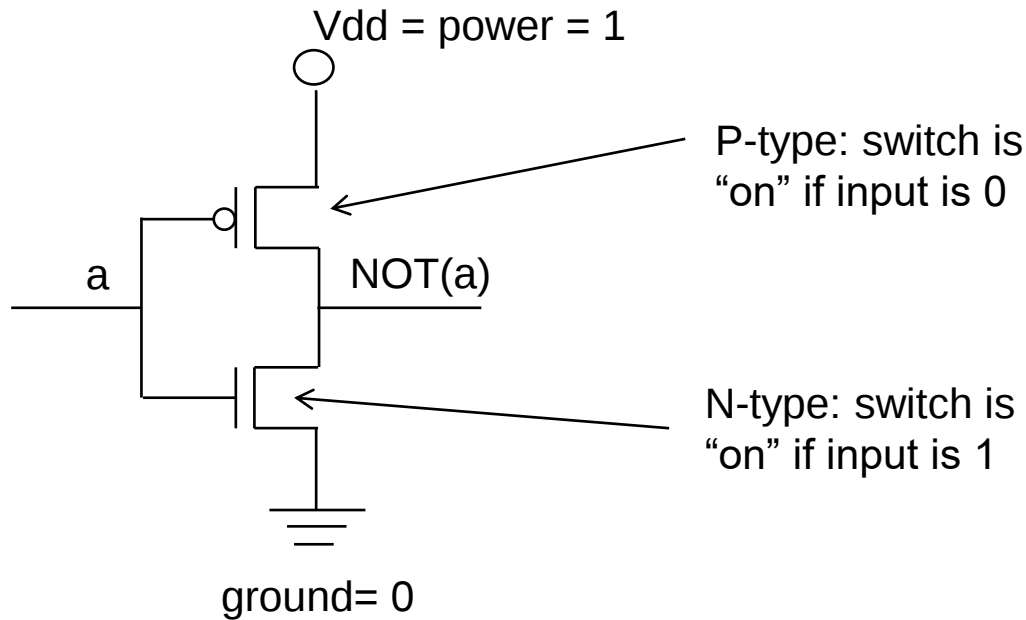
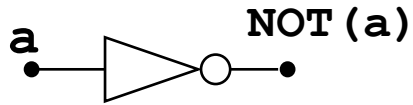
Circle means NOT



(XNOR is 1-bit "equals" by the way)



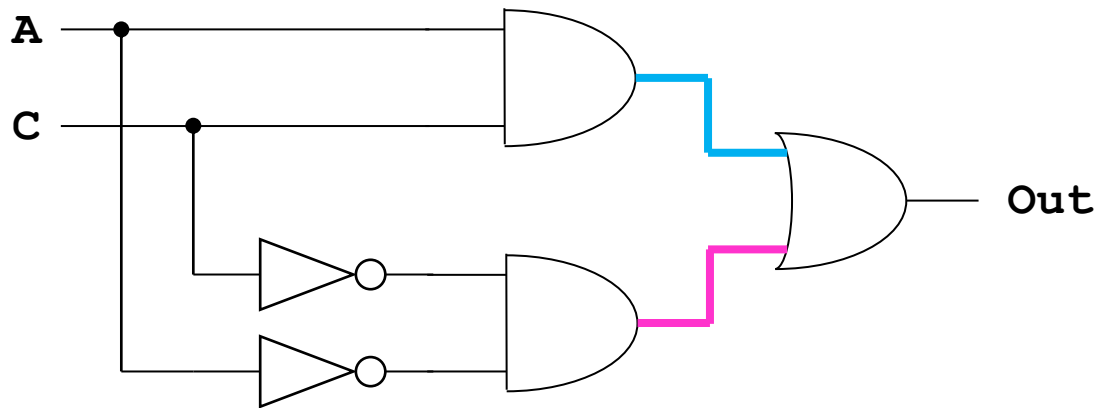
Brief Interlude: Building An Inverter



Boolean Functions, Gates and Circuits

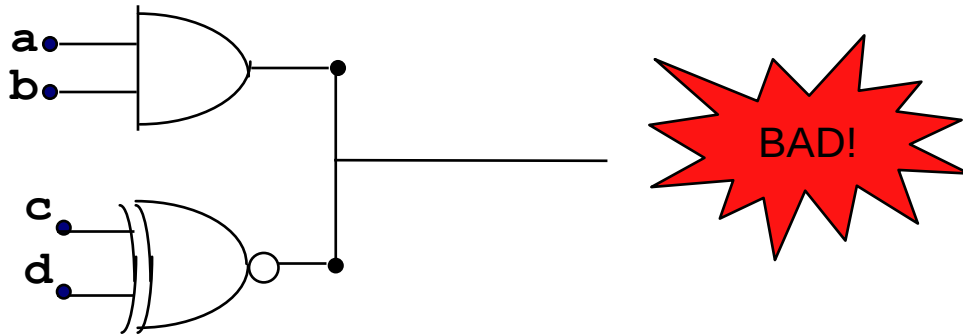
- Circuits are made from a network of gates.

$$(!A \ \& \ !C) \mid (A \ \& \ C)$$



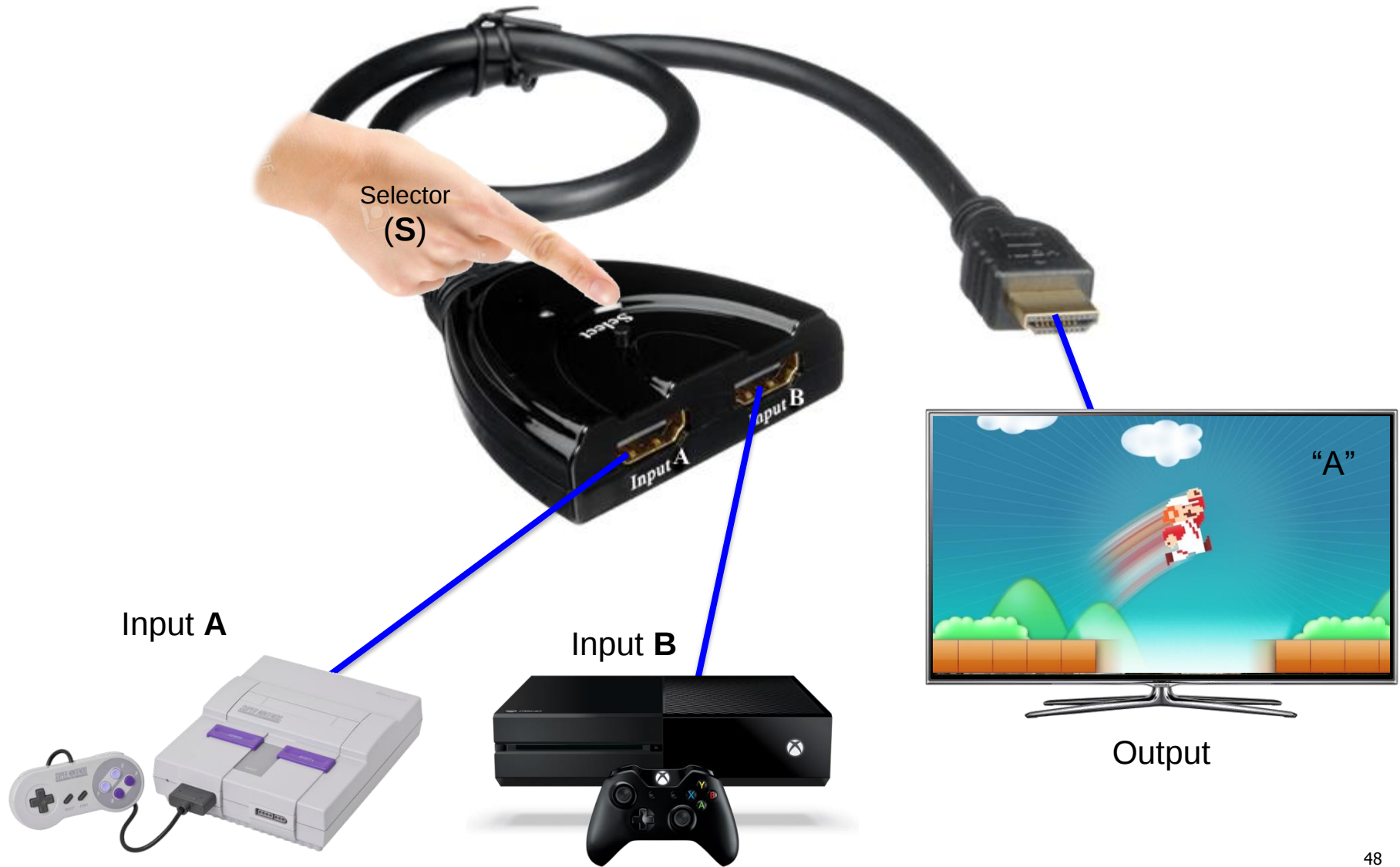
A few more words about gates

- Gates have inputs and outputs
 - If you try to hook up two outputs, bad things happen (your processor catches fire)

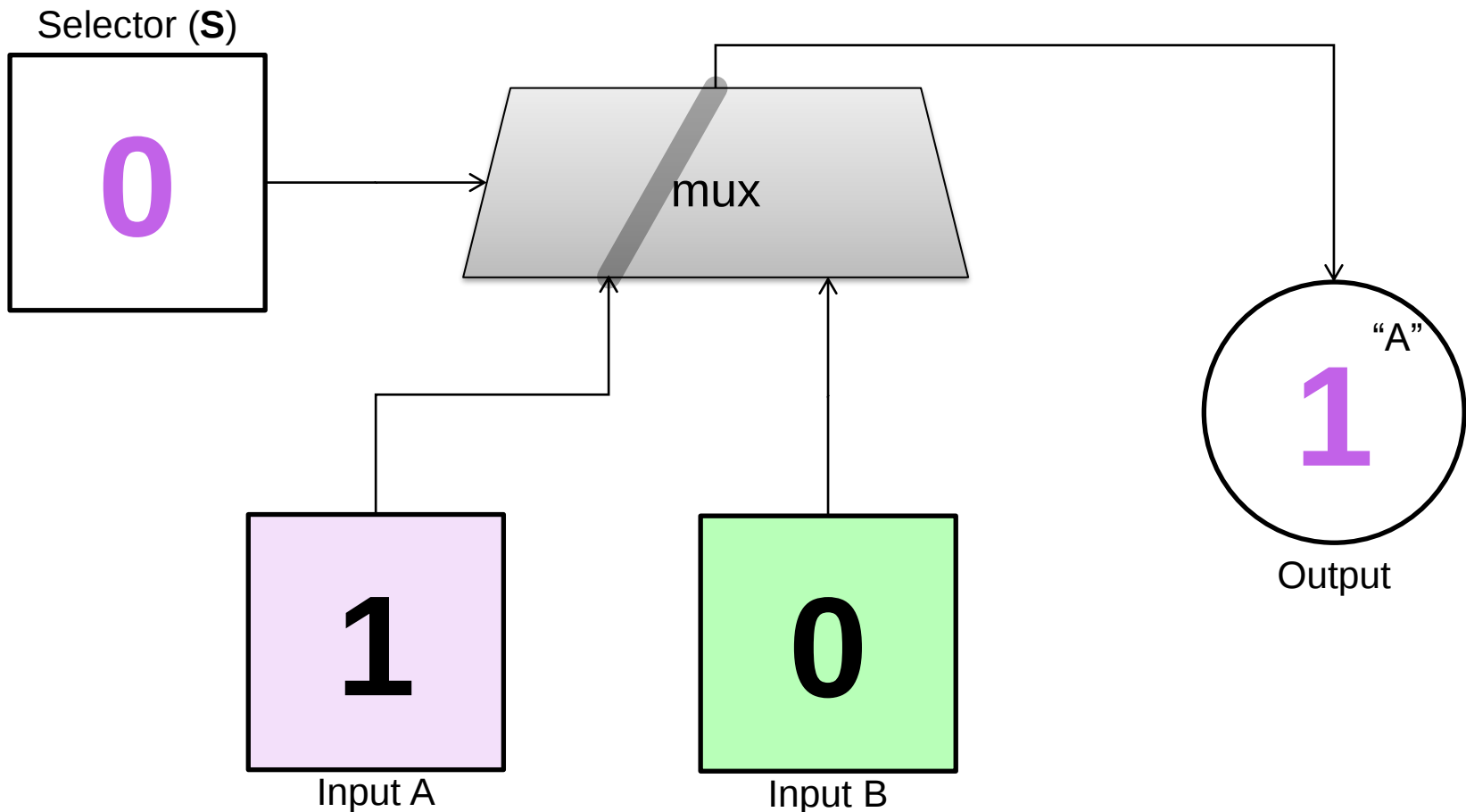


- If you don't hook up an input, it behaves kind of randomly (also not good, but not set-your-chip-on-fire bad)

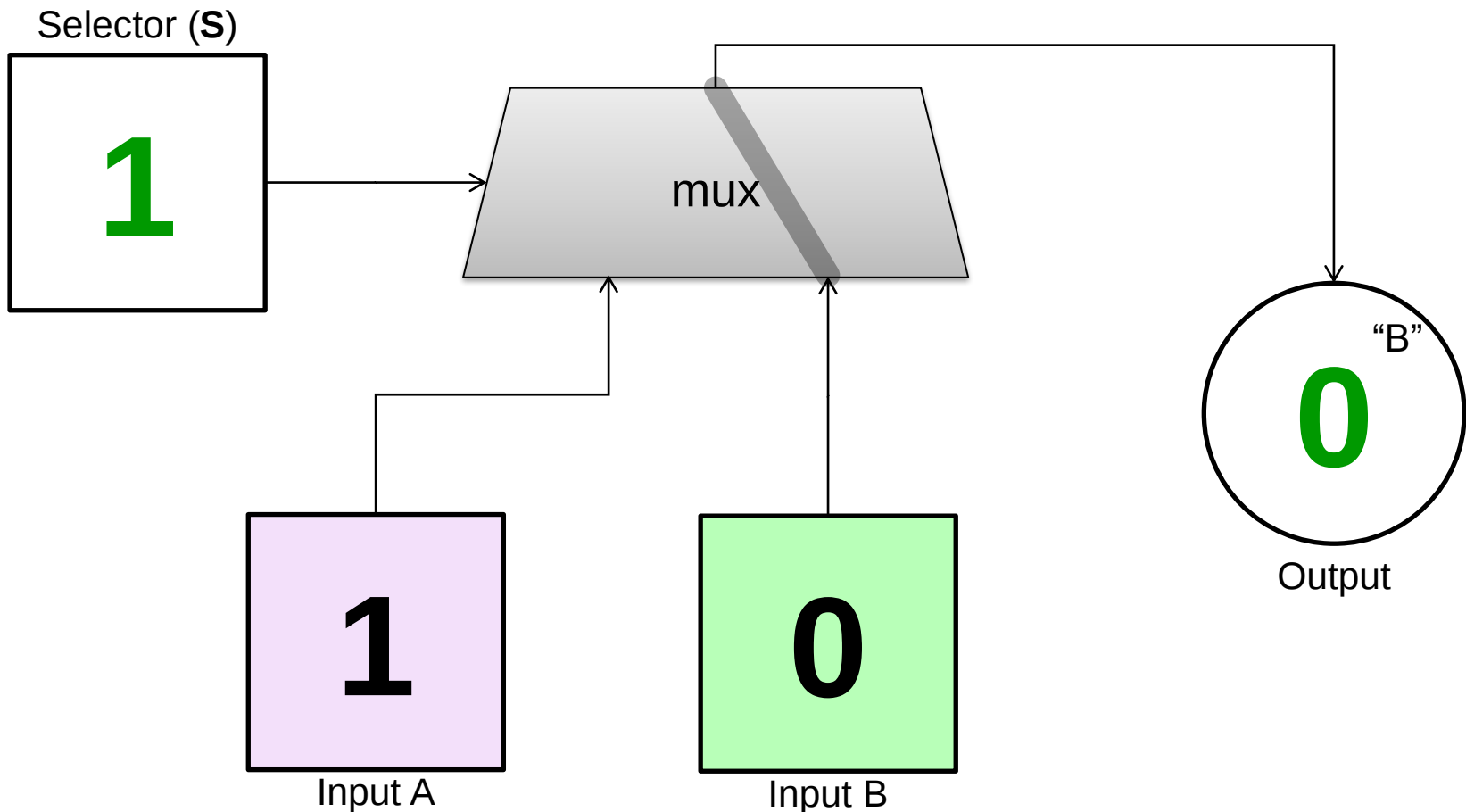
Introducing the Multiplexer (“mux”)



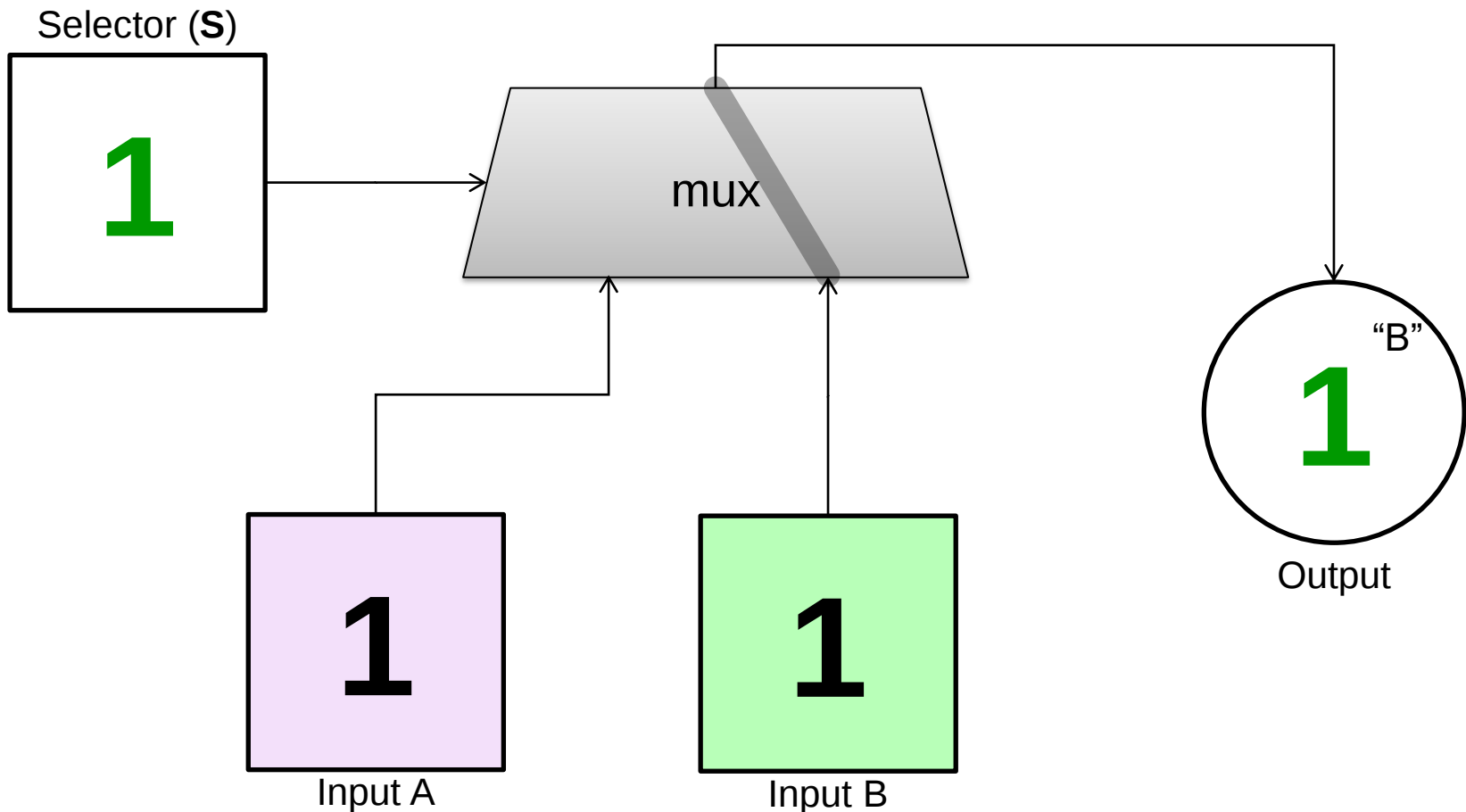
Introducing the Multiplexer (“mux”)



Introducing the Multiplexer (“mux”)



Introducing the Multiplexer (“mux”)



Let's Make a Useful Circuit

- Pick between 2 inputs (called 2-to-1 MUX)
 - Short for multiplexor

- What might we do first?

- Make a truth table?
 - S is selector:
 - S=0, pick A
 - S=1, pick B

- Next: sum-of-products

$(\neg A \ \& \ B \ \& \ S) \mid$
 $(A \ \& \ \neg B \ \& \ \neg S) \mid$
 $(A \ \& \ B \ \& \ \neg S) \mid$
 $(A \ \& \ B \ \& \ S)$

- Simplify

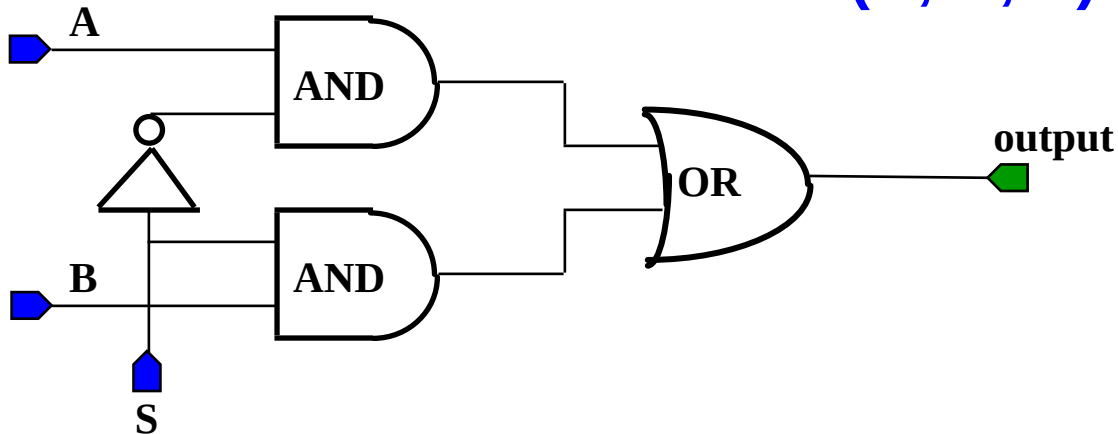
$(A \ \& \ \neg S) \mid (B \ \& \ S)$

A	B	S	Output
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

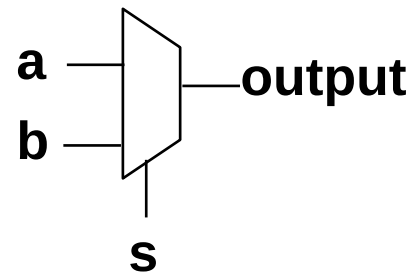
Circuit Example: 2x1 MUX

Draw it in gates:

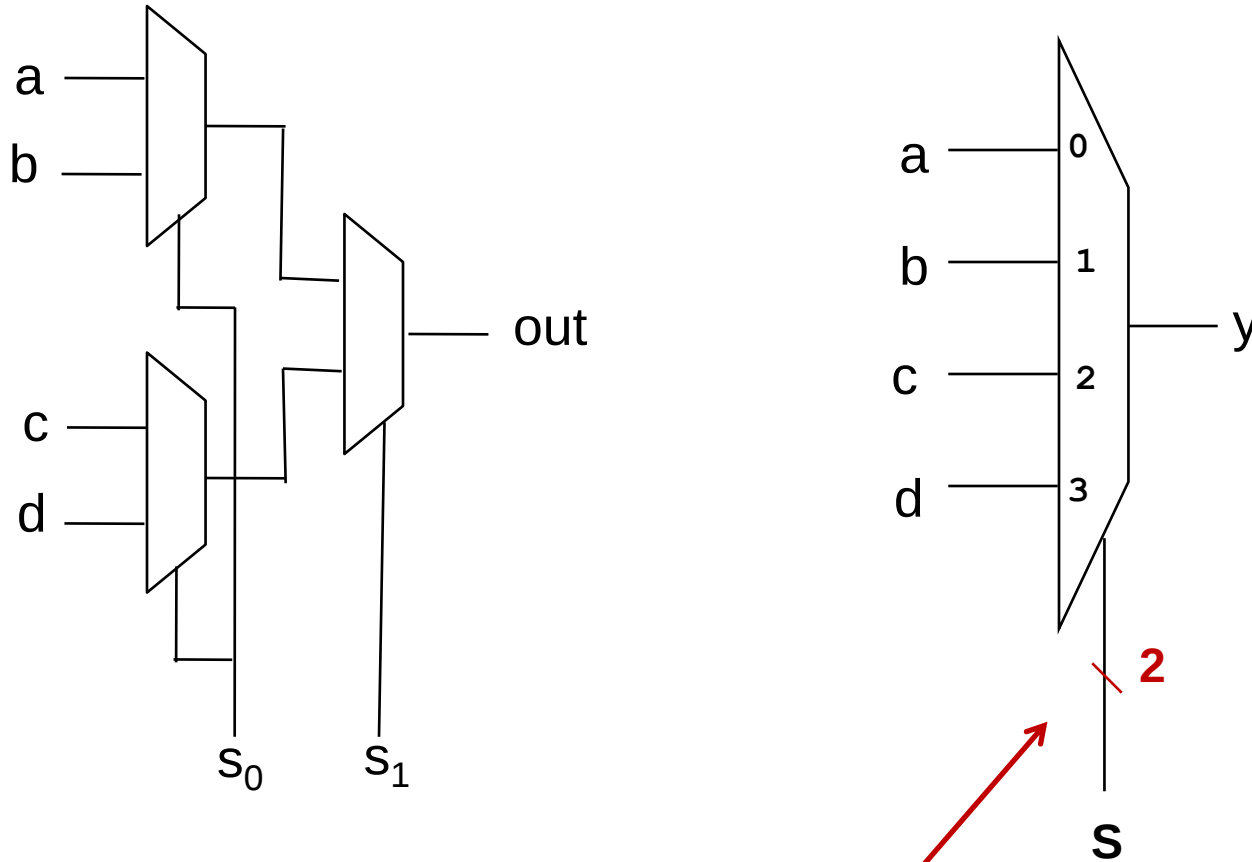
$$\text{MUX}(A, B, S) = (A \& !S) \mid (B \& S)$$



So common, we give it
its own symbol:

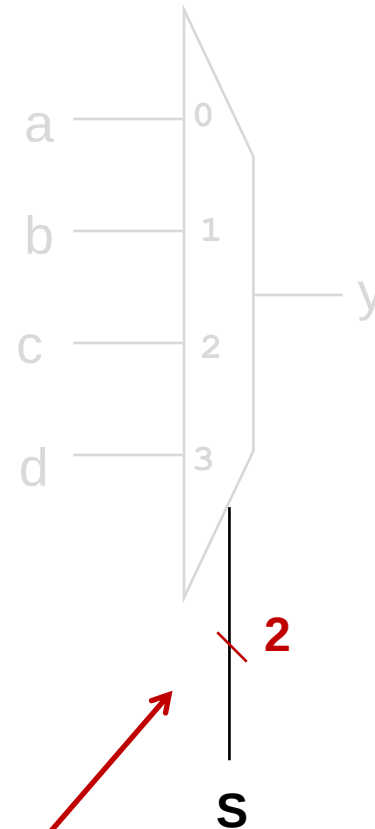
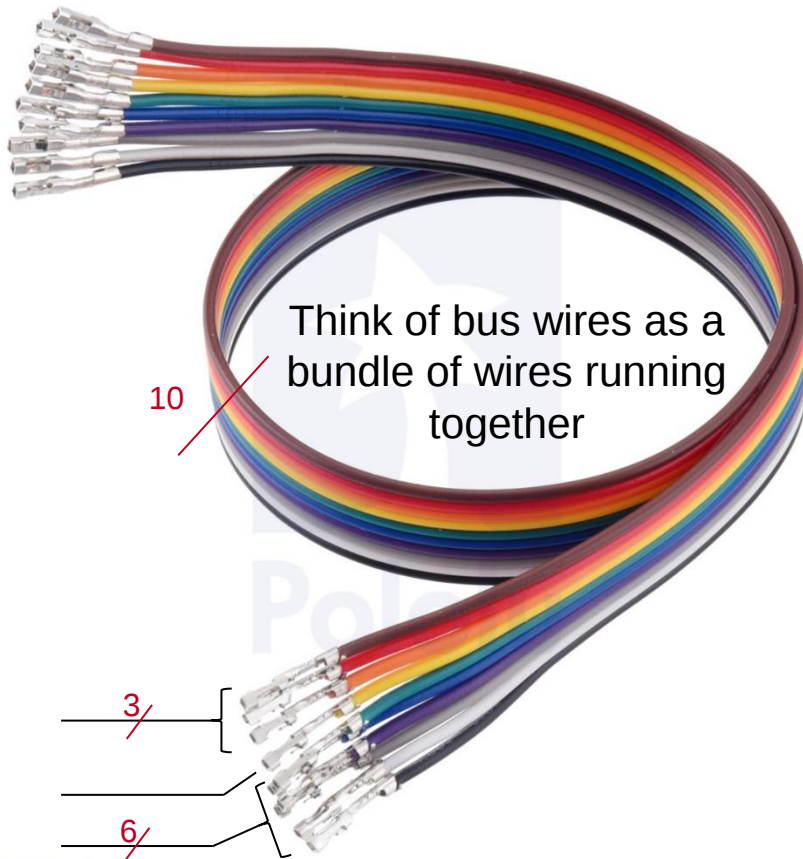


Example 4x1 MUX



The / 2 on the wire means "2 bits"

Quick side thing: bus wires



At the ends, we could even opt to “split” them up to different purposes (used later)

The / 2 on the wire means “2 bits”

This is called a **bus wire**

Arithmetic and Logical Operations in ISA

- What operations are there?
- How do we implement them?
 - Consider a 1-bit Adder

Designing a 1-bit adder

- What boolean function describes the low bit?
 - XOR
- What boolean function describes the high bit?
 - AND

$$0 + 0 = 00$$

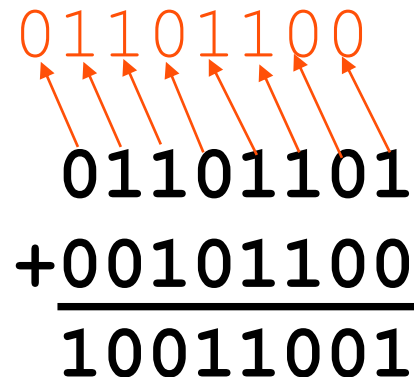
$$0 + 1 = 01$$

$$1 + 0 = 01$$

$$1 + 1 = 10$$

Designing a 1-bit adder

- Remember how we did binary addition:
 - Add the **two bits**
 - Do we have a **carry-in** for this bit?
 - Do we have to **carry-out** to the next bit?

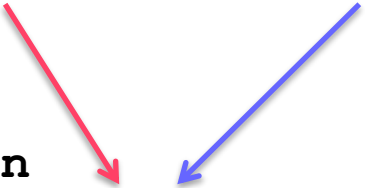


A binary addition diagram illustrating carry propagation. The first number, 01101101, is written in black. The second number, +00101100, is written in black below it, with a horizontal line underneath. The result, 10011001, is written in black below the line. Above the first number, the digits 01101100 are written in orange. Orange arrows point from each of these orange digits down to the corresponding digit of the first number (01101101), showing a carry-in of 1 for every bit position.

$$\begin{array}{r} 01101101 \\ +00101100 \\ \hline 10011001 \end{array}$$

Designing a 1-bit adder

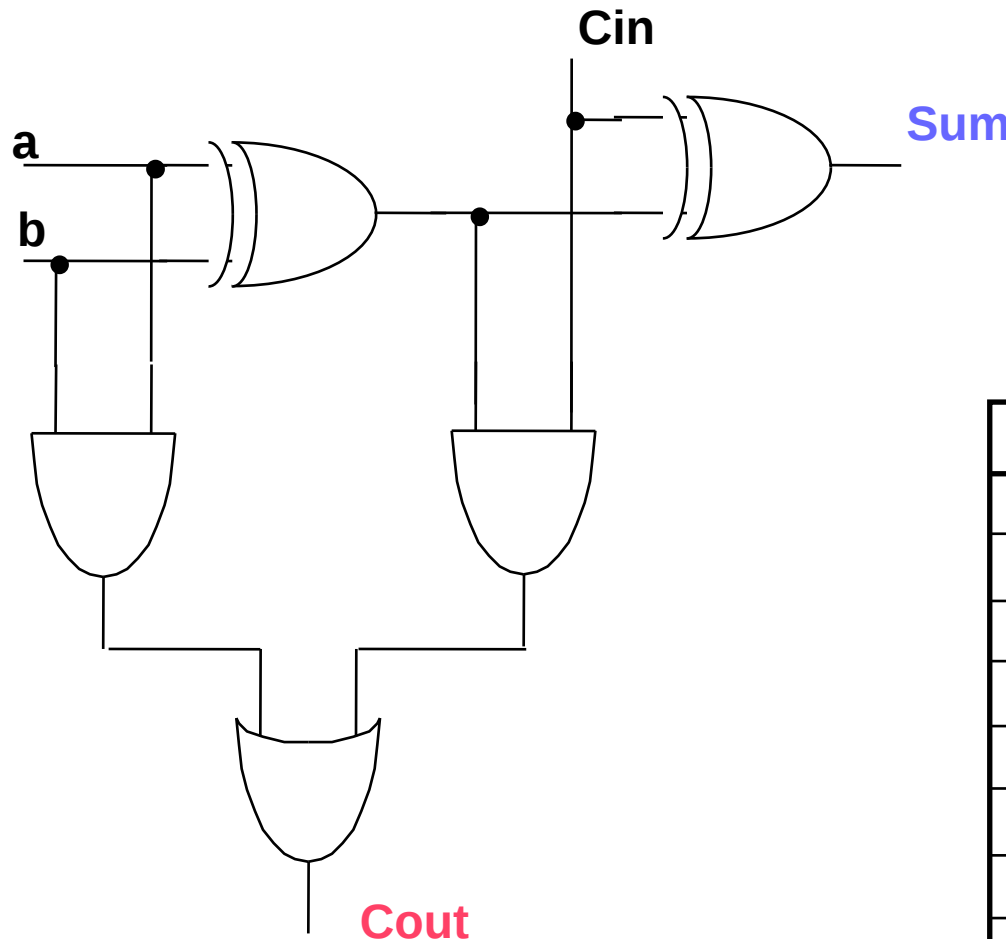
- So we'll need to add three bits (including carry-in)
- Two-bit output is the **carry-out** and the **sum**



a		b		C _{in}	=	
0	+	0	+	0	=	00
0	+	0	+	1	=	01
0	+	1	+	0	=	01
0	+	1	+	1	=	10
1	+	0	+	0	=	01
1	+	0	+	1	=	10
1	+	1	+	0	=	10
1	+	1	+	1	=	11

Turn into expression,
simplify,
circuit-ify,
yadda yadda yadda...

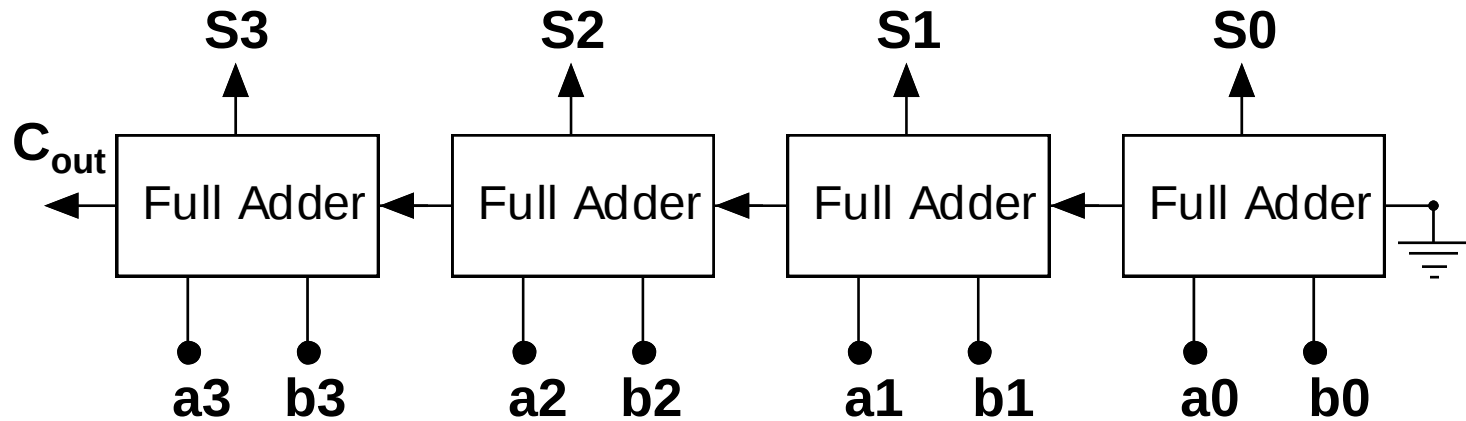
A 1-bit Full Adder



$$\begin{array}{r} 01101100 \\ 01101101 \\ +00101100 \\ \hline 10011001 \end{array}$$

a	b	C _{in}	Sum	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

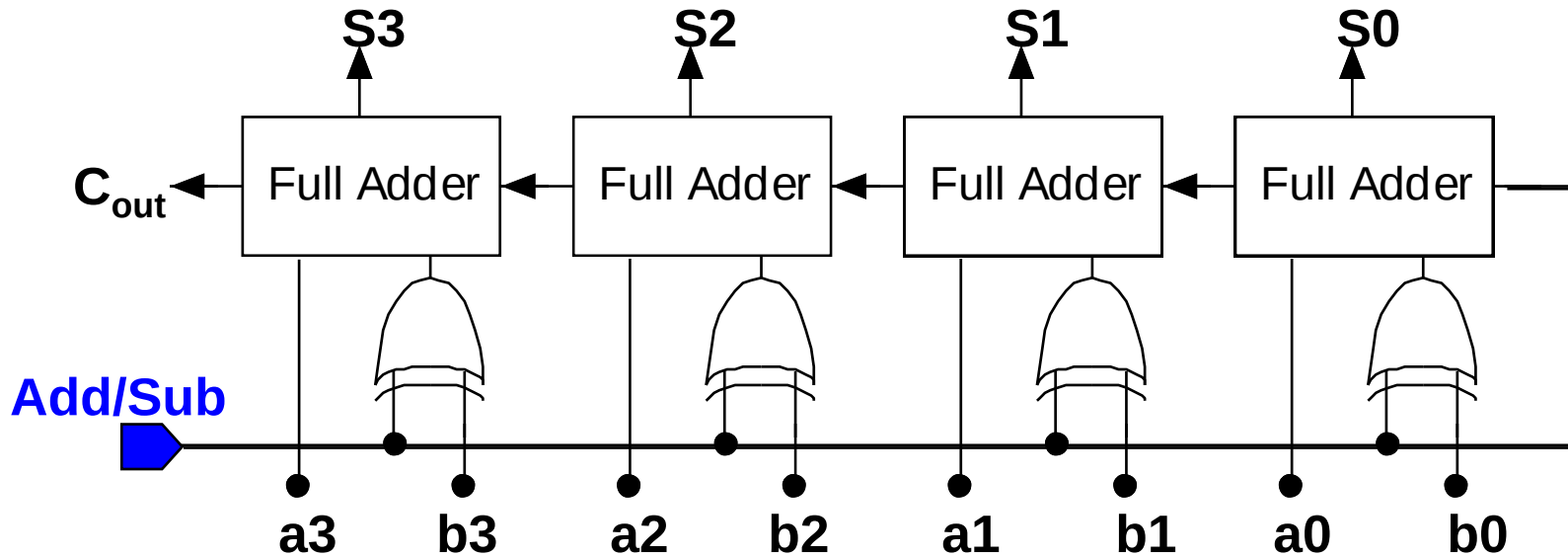
Example: 4-bit adder



Subtraction

- How do we perform integer subtraction?
- What is the hardware?
 - Recall: hardware was why 2's complement was good idea
- Remember: Subtraction is just addition
$$X - Y =$$
$$X + (-Y) =$$
$$X + (\sim Y + 1)$$

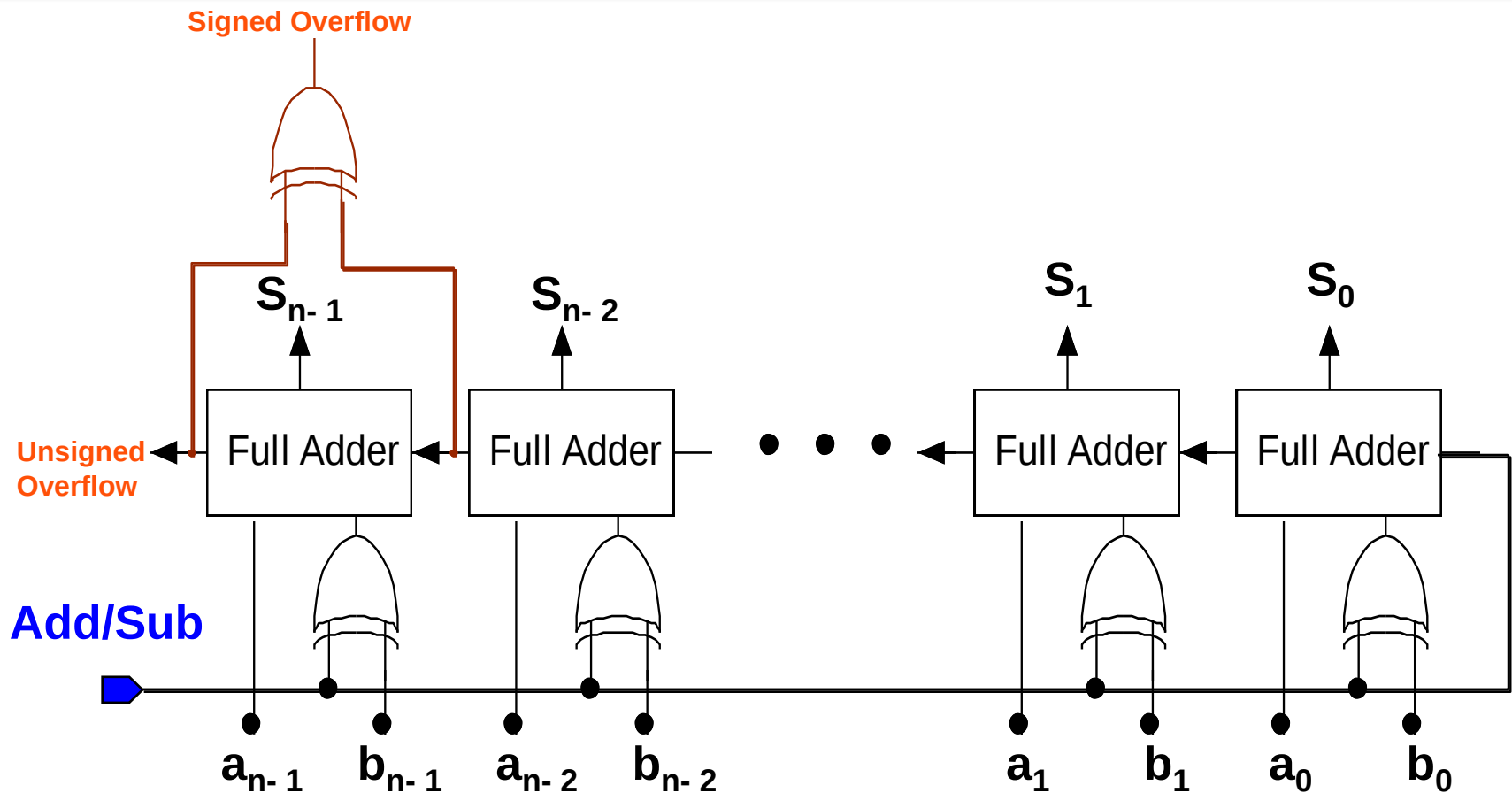
Example: Adder/Subtractor



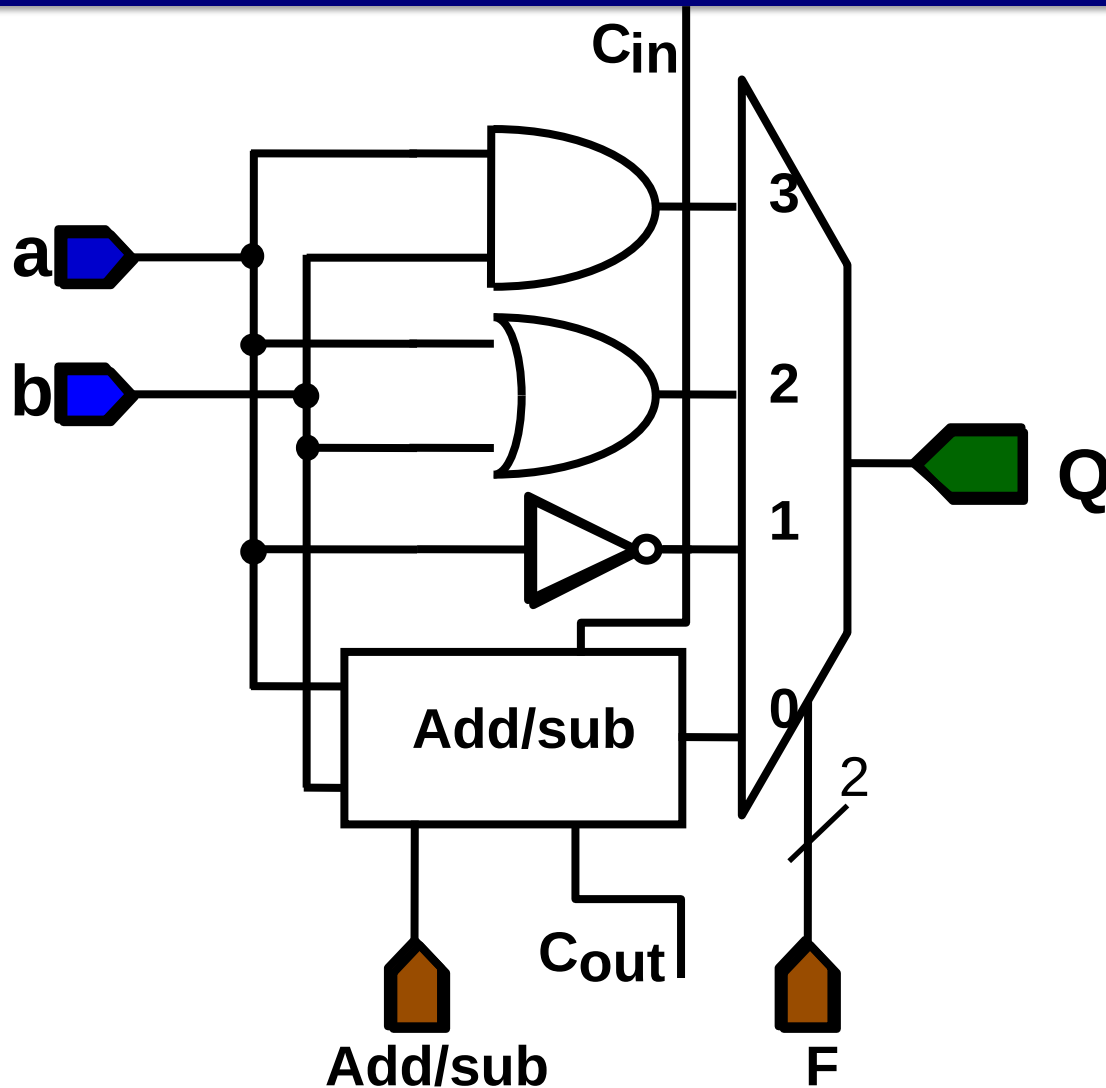
Overflow

- We can detect **unsigned overflow** by looking at CO
- How would we detect **signed overflow**?
 - If adding positive numbers and result "is" negative
 - If adding negative numbers and result "is" positive
 - At most significant bit of adder, check if $CI \neq CO$
 - Can check with XOR gate

Add/Subtract With Overflow Detection

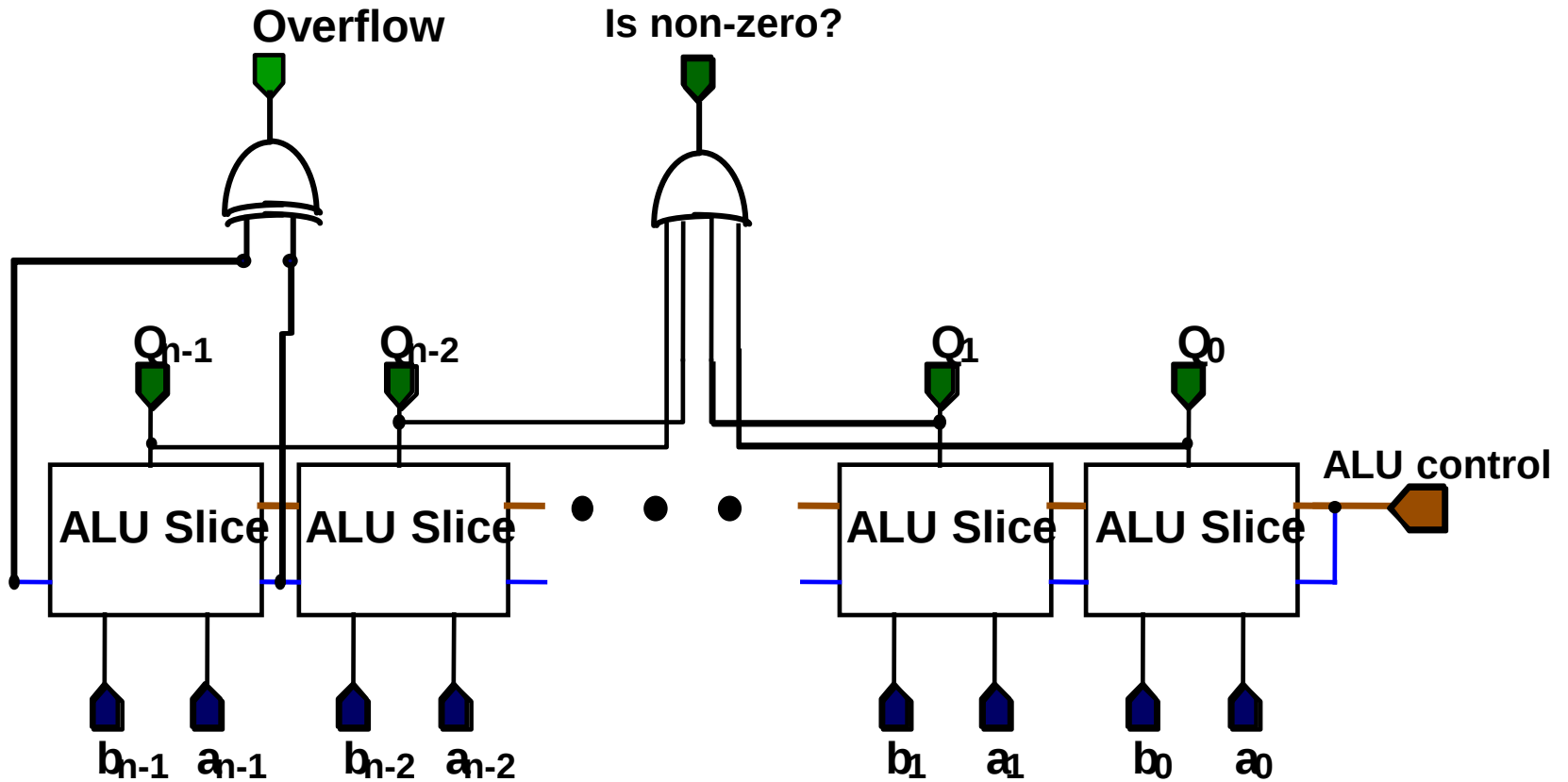


ALU Slice



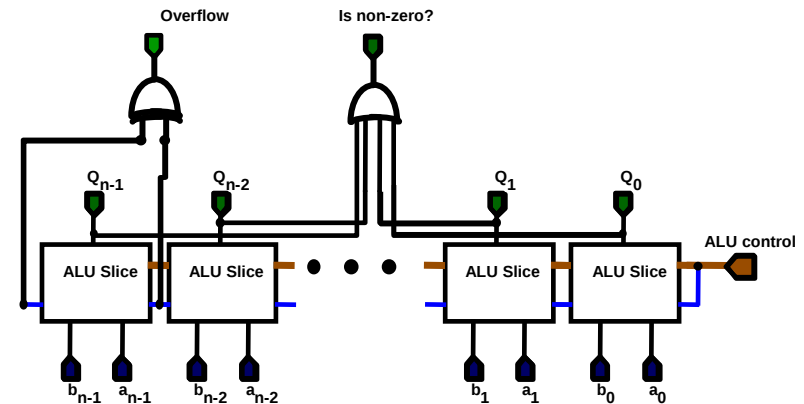
A	F	Q
0	0	$a + b$
1	0	$a - b$
-	1	NOT b
-	2	$a \text{ OR } b$
-	3	$a \text{ AND } b$

The ALU

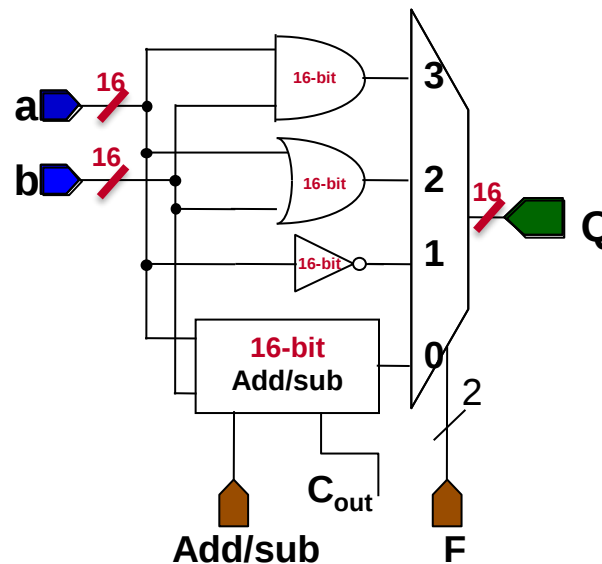


Alternate ALU design

- Previous design did ALU stuff for each bit, then chained them.

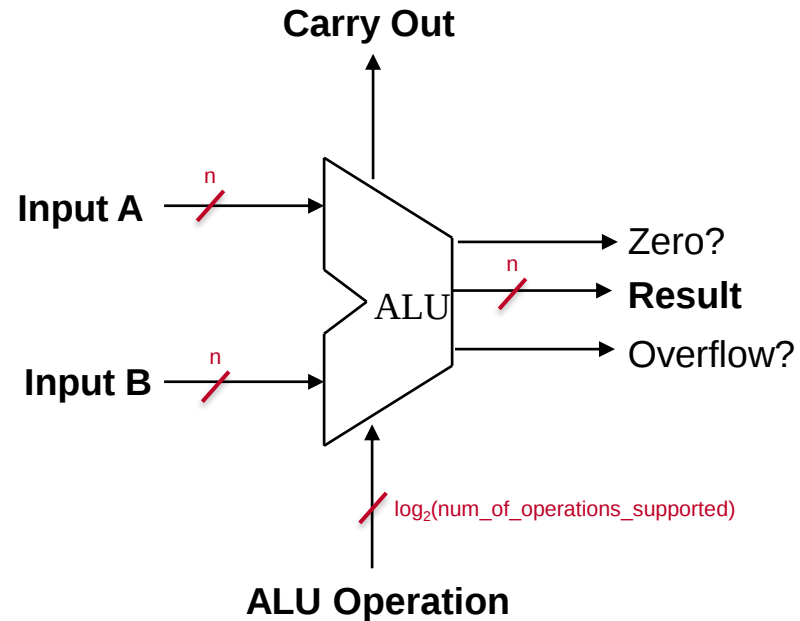


- Can also do each word-size operation and mux the resulting words.



Abstraction: The ALU

- General structure
- Two operand inputs
- Control inputs
- We can build circuits for
 - Multiplication
 - Division
 - They are more complex



Another Operations We Might Want: Shift

- Remember the << and >> operations?
 - Shift left/shift right?
 - How would we implement these?
- Suppose you have an 8-bit number
 $b_7b_6b_5b_4b_3b_2b_1b_0$
- And you can shift it left by a 3-bit number

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$$
$$S_2 S_1 S_0$$

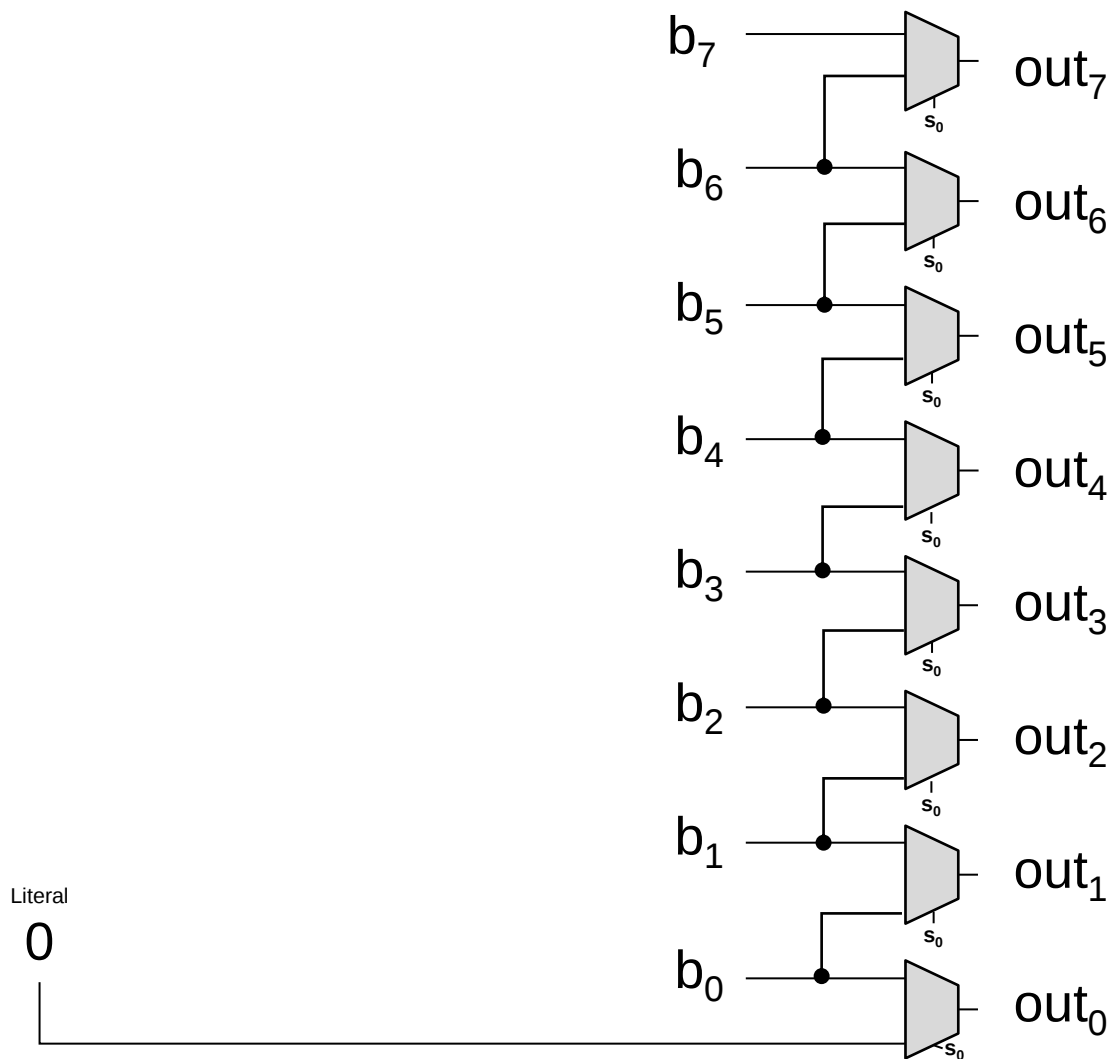
- Option 1: Truth Table?
 - $2^{11} = 2048$ rows? Yuck.

...but you can do it. Truth table gives this expression for output bit 0:

101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

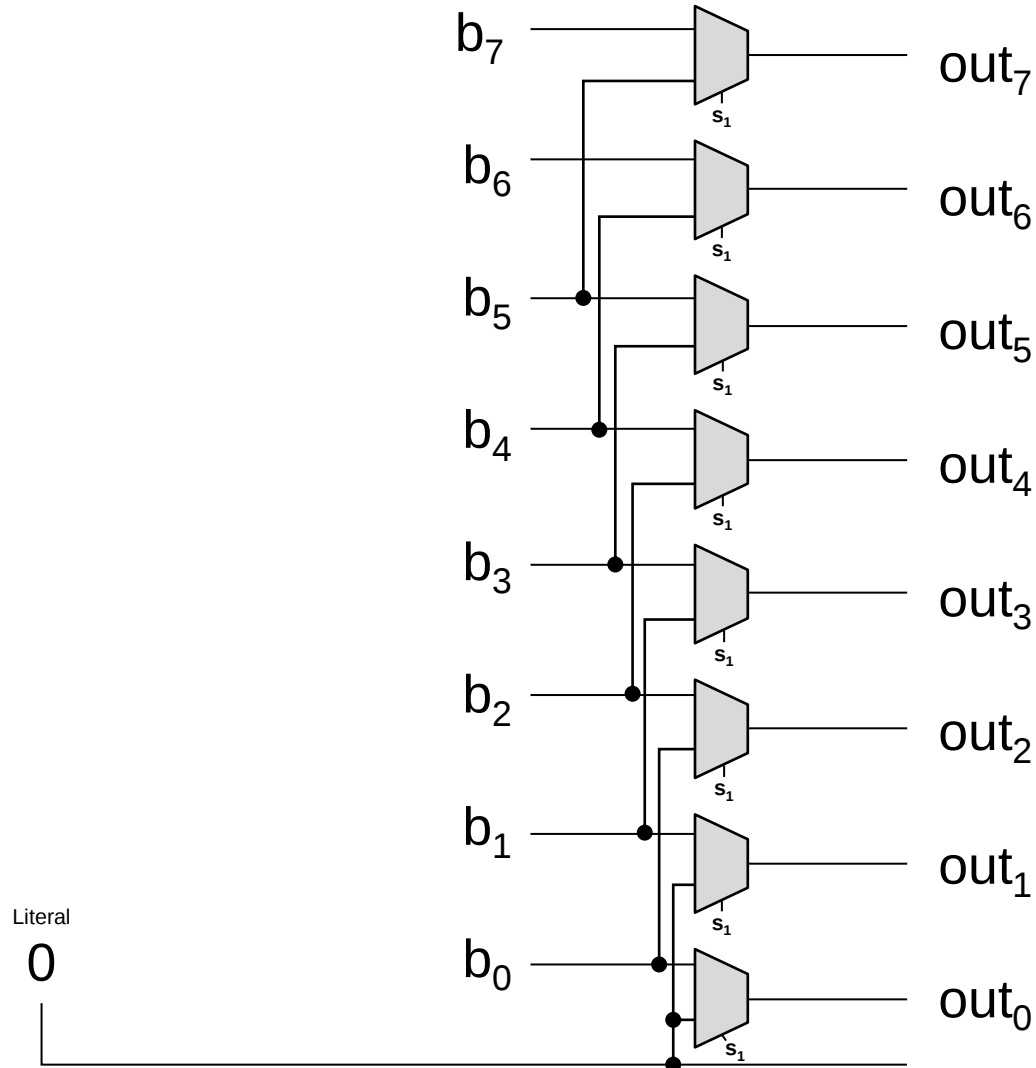
Building a bit shifter

- Simpler problem: A shift-by-**one** circuit, all controlled by the same 1 bit input (s_0)



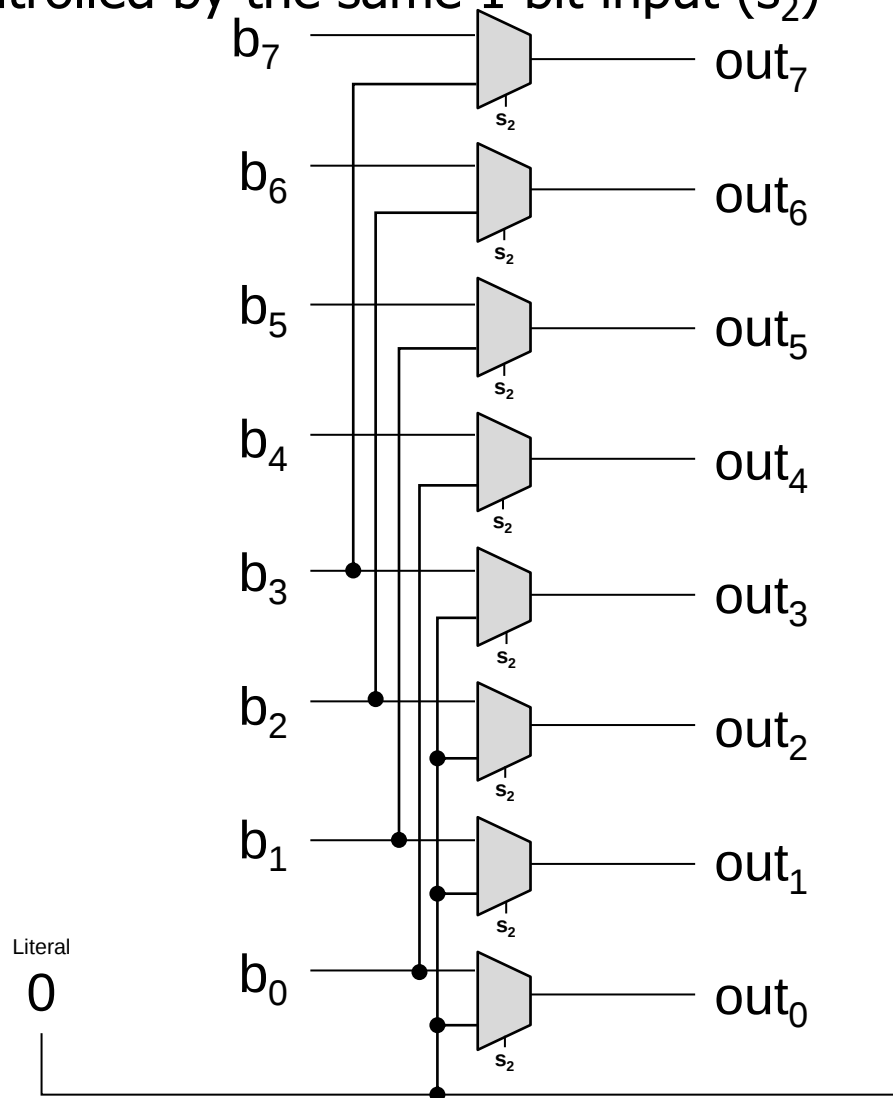
Building a bit shifter

- Simpler problem: A shift-by-**two** circuit, all controlled by the same 1 bit input (s_1)



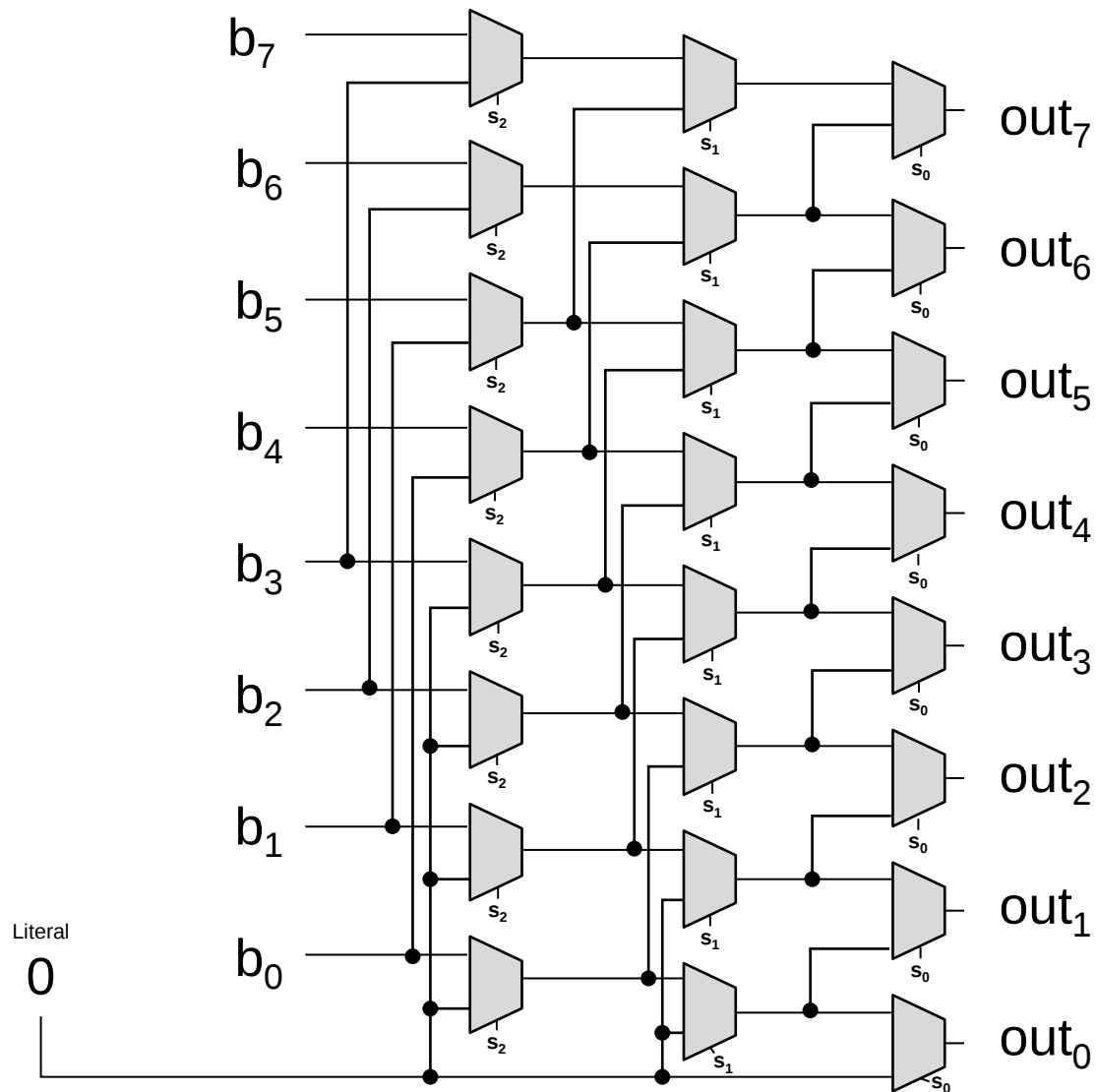
Building a bit shifter

- Simpler problem: A shift-by-**four** circuit, all controlled by the same 1 bit input (s_2)



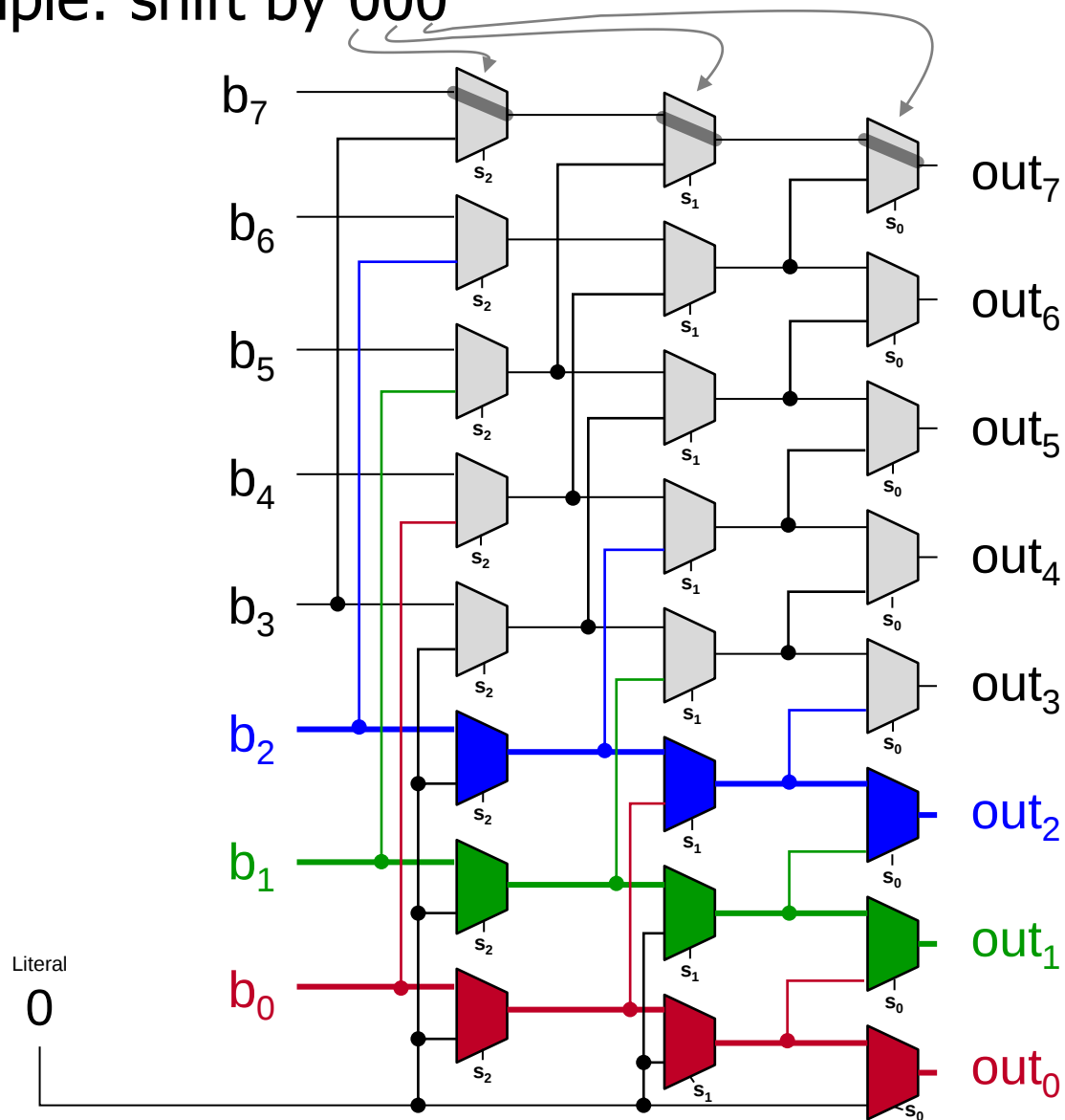
Now shifted by 3-bit number

- Full problem: stick them all together, controlled by 3-bit value $s_{2:0}$



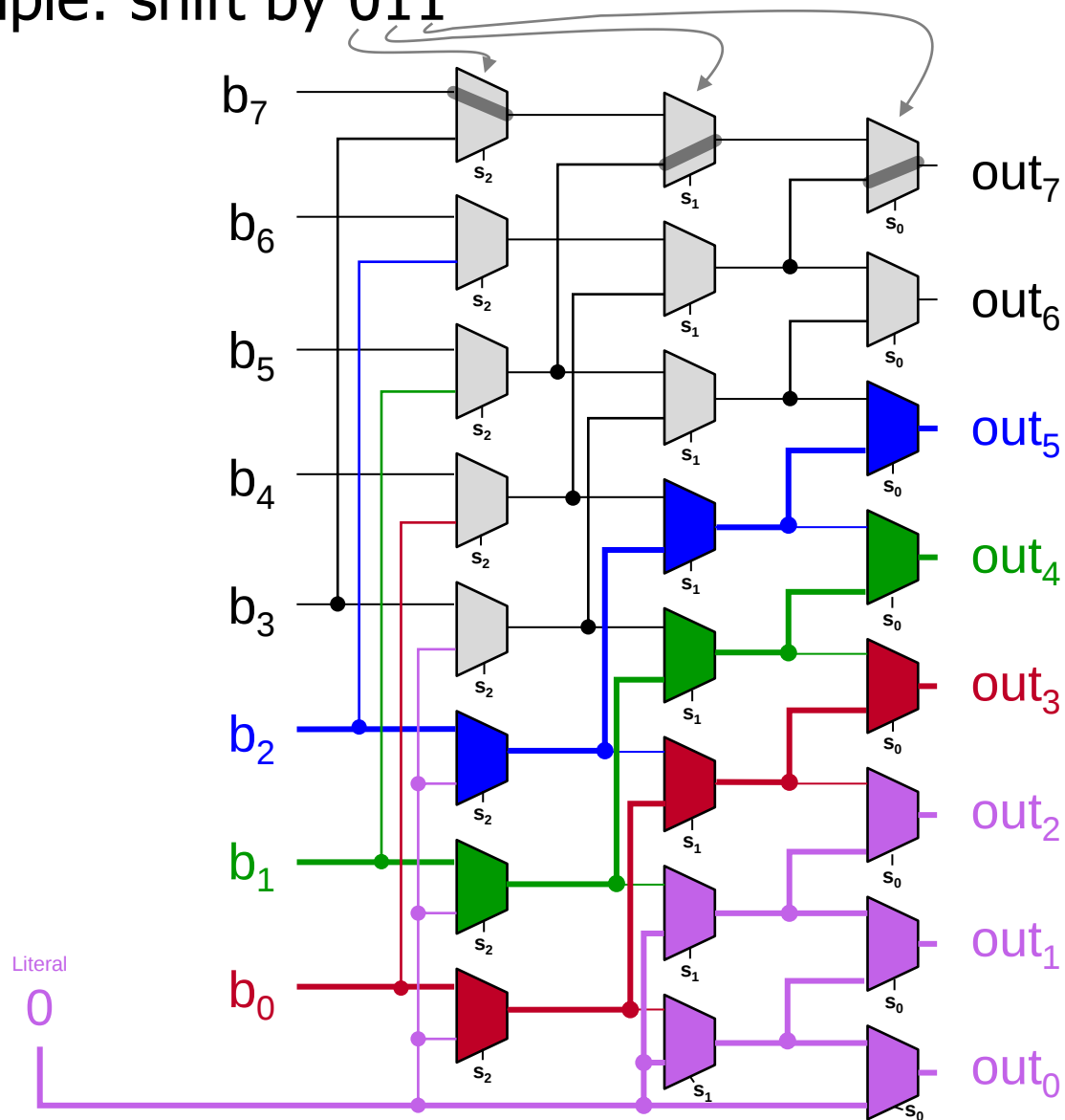
Now shifted by 3-bit number

- Example: shift by 000



Now shifted by 3-bit number

- Example: shift by 011



Summary

- Boolean Algebra & functions
- Logic gates (AND, OR, NOT, etc)
- Multiplexors
- Adder
- Arithmetic Logic Unit (ALU)
- Bit shifting