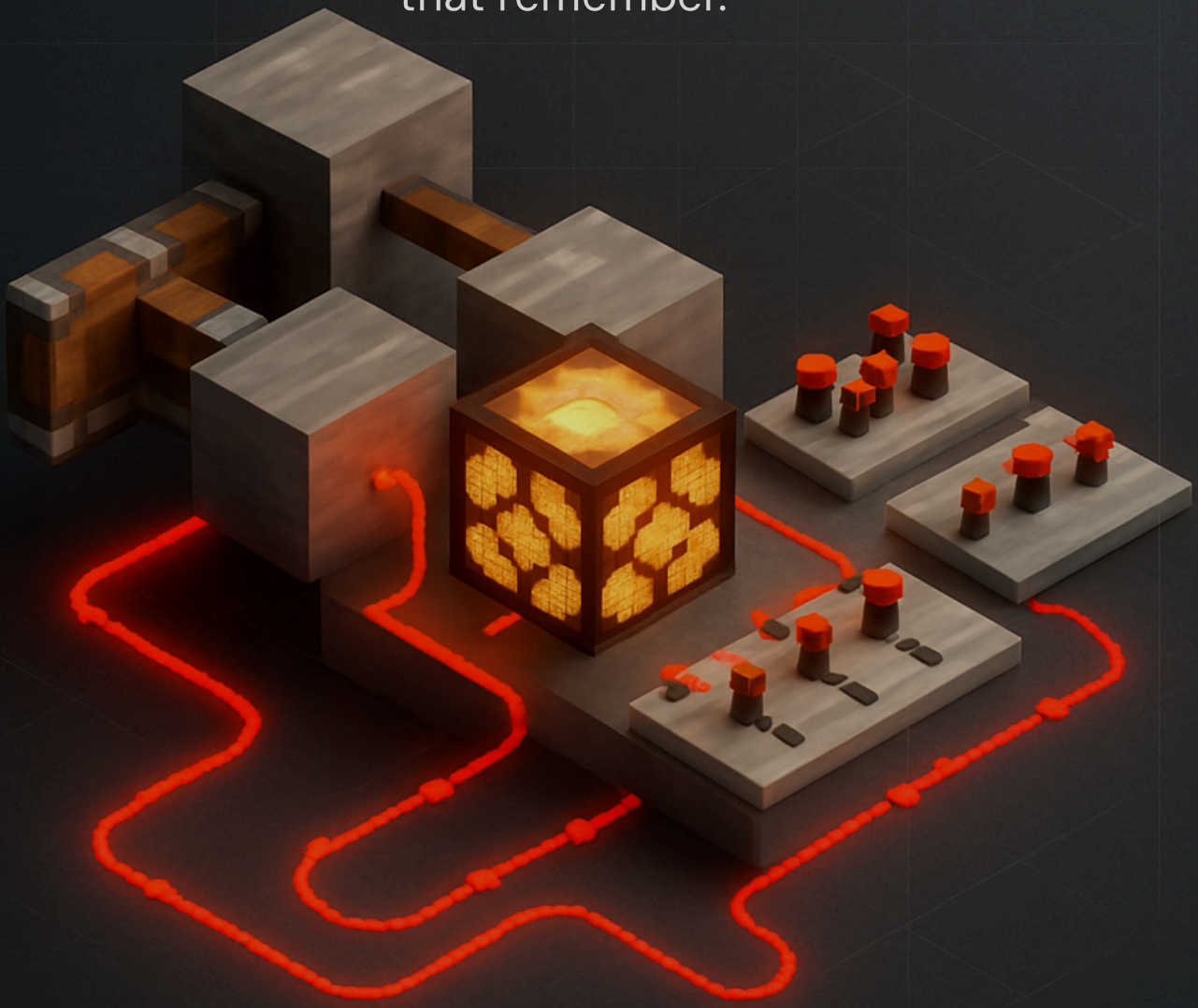
The Redstone Engineer's Handbook

From your first lever to logic gates, clocks, and machines that remember.



REDWIRE

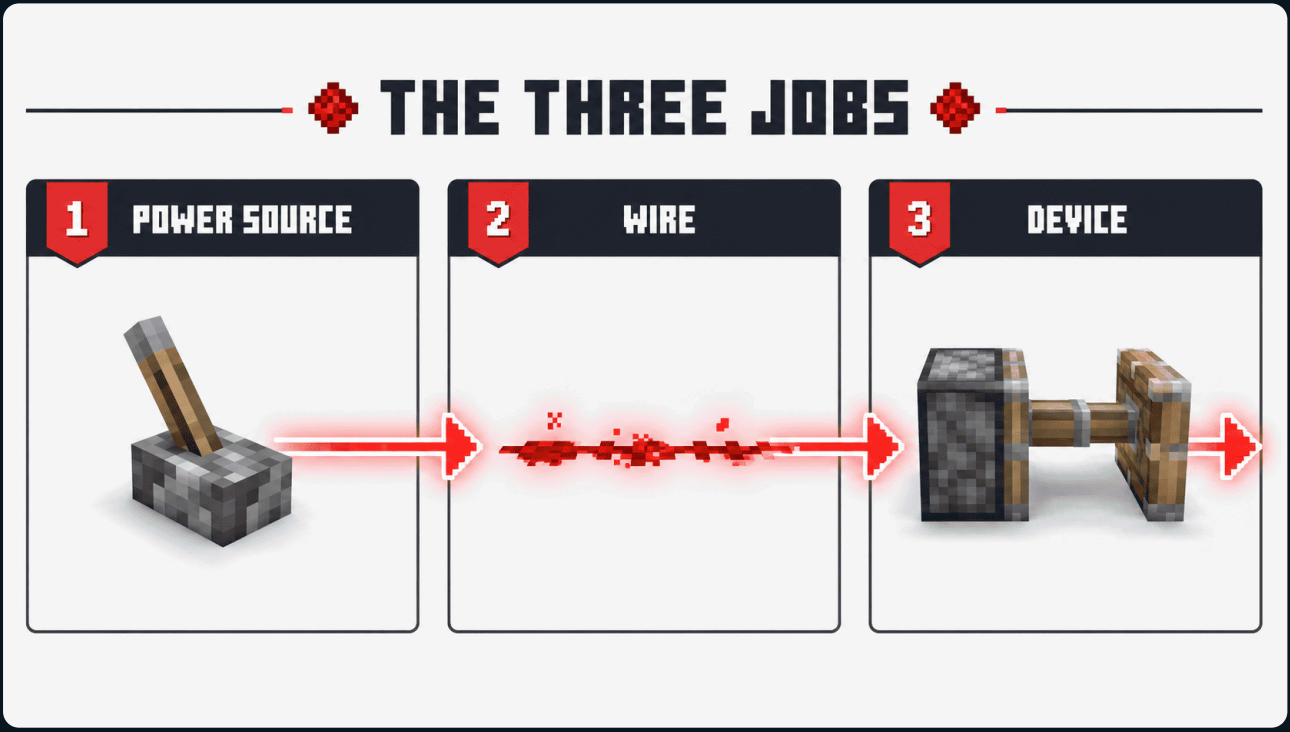
Contents

The Redstone Engineer’s Handbook

From your first lever to logic gates, clocks, and machines that remember. A builder’s guide to the hidden electronics of Minecraft.

Chapter One: What Redstone Actually Is

Redstone is Minecraft's electricity, and like electricity, it feels like magic until you learn the handful of rules underneath it. Then it stops being magic and becomes engineering, which is far more fun, because engineering you can design.



Every redstone build, from a lamp to a working calculator, is only ever these three jobs in combination: something that makes a signal, something that carries it, and something that does a job when it arrives.

Here is the whole subject in one sentence. Every redstone contraption is built from three kinds of thing: **power sources** that create a signal, **wires** that carry the signal from place to place, and **devices** that do something when the signal reaches them. A lever (source) sends power down a line of dust (wire) to a piston (device). That is the atom of redstone. Everything else is just this pattern, repeated and combined until it does something clever.

Signal strength: the one number that runs everything

A redstone signal is not simply "on" or "off." It has a **strength**, a number from 0 to 15. A power source at full blast outputs a signal strength of 15. And here is the rule that trips up every single beginner:

Redstone dust loses one point of signal strength for every block it travels.

Send a full strength 15 signal down a line of dust, and after one block it is 14, after two it is 13, and by the fifteenth block it has dropped to 0. Dead. Your signal has a maximum reach of fifteen blocks, and then it simply runs out, like water losing pressure in a long pipe.

> ***DEFINITION — Signal strength*** *is the "loudness" of a redstone signal, from 0 (off) to 15 (maximum). Power sources output 15. Plain redstone dust drops the strength by 1 for every block it crosses, so an unassisted signal dies after 15 blocks. Almost every mysterious "why won't my thing turn on" problem is really a signal strength problem.*

Most devices do not care about the exact number: a piston fires on any strength above 0, and does not fire harder at 15 than at 3. But some components read the number precisely, and those are where redstone gets genuinely powerful. We will meet them soon. For now, burn the number 15 into your memory. It is the speed limit, the range limit, and the source of half your future frustration.

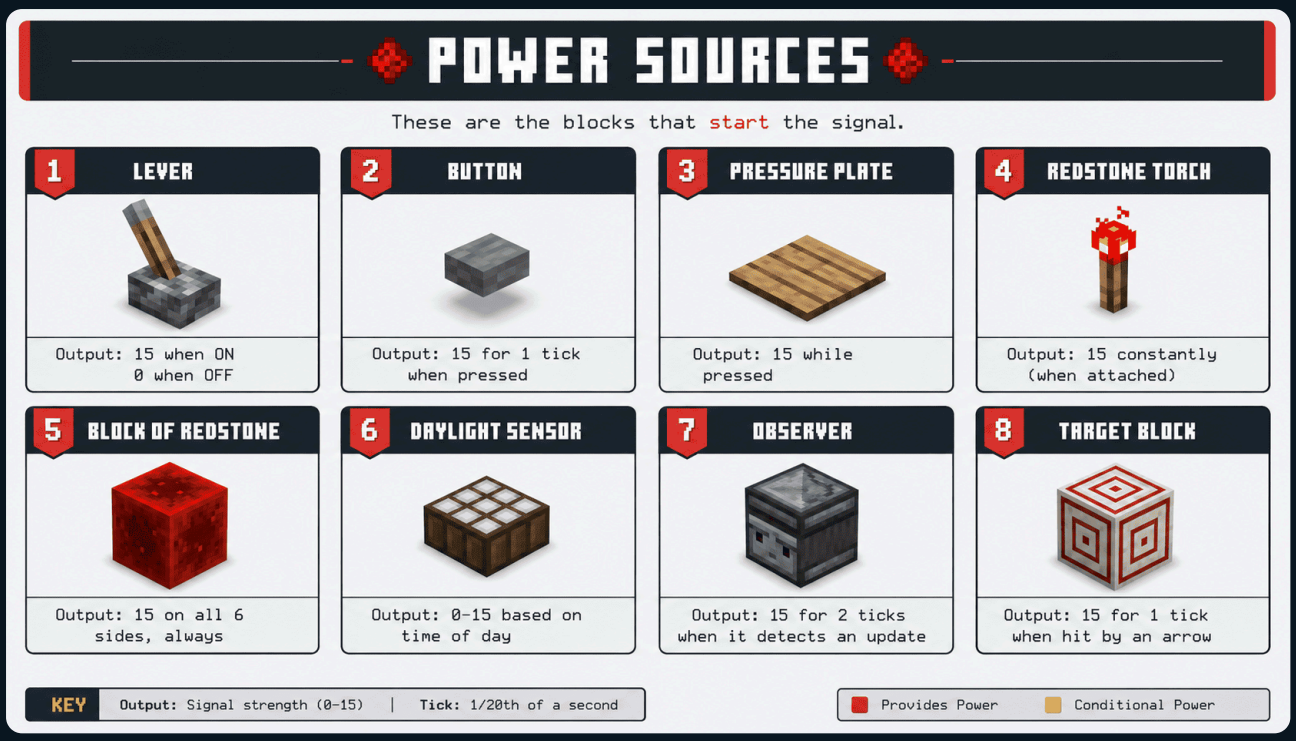
> *"Stop thinking of redstone as on and off. Start thinking of it as a signal with a strength that fades as it travels. Once that clicks, everything else clicks."*

Chapter Two: The Power Sources

A machine is only as good as its trigger. Pick the wrong power source and even a perfect circuit behaves wrong. The sources fall into three families.

Manual sources: you flip the switch

- **Lever.** The workhorse. Flip it on, it stays on; flip it off, it stays off. A steady, held signal. This is your best friend for testing, because it holds a state while you go and inspect what it did.
- **Button.** A momentary trigger. Press it and it sends a short pulse of power, then switches itself off. Stone buttons stay on for about a second; wooden buttons stay on a little longer and can also be triggered by an arrow. Use buttons where you want a "do it once" action: open a door, fire a dispenser, ring a bell.
- **Pressure plate.** Triggers when something stands on it. A stone plate reacts only to players and mobs; a wooden plate reacts to almost anything, including dropped items and arrows; **weighted** pressure plates output a variable signal that scales with how many entities are standing on them, which lets you count things.



The redstone toolbox of signal makers. Learn what each one outputs and when, and you can start any machine with the right trigger for the job.

Constant sources: always on

- **Block of redstone.** A solid block that outputs strength 15, forever, in all directions. Crucially, a piston can push it. A huge number of clever machines work by shoving a block of redstone next to a circuit to switch it on, then pulling it away to switch it off.
- **Redstone torch.** Also outputs 15, and also does something much stranger that earns it its own chapter. Hold that thought.

Sensing sources: the world flips the switch

- **Daylight sensor.** Outputs a signal that tracks the sky's light level, strong at noon, zero at midnight. Right click it to invert its behavior so it powers on at night instead, which is how automatic street lamps work.
- **Observer.** Stares at the block directly in front of its face and fires a short one tick pulse the instant that block changes in any way: a crop growing, a piston moving, a neighbor updating. The tireless watchman of automated farms.
- **Target block, tripwire, sculk sensor, lightning rod.** A family of situational detectors. The target block outputs a strength based on how close an arrow lands to its center. The sculk sensor detects vibrations (footsteps, breaking blocks) and is the basis of wireless and hidden triggers.

Source	Output	Behavior	Best used for
Lever	15, held	Stays until flipped	Testing, permanent toggles
Button	15, brief pulse	Auto resets	One shot actions
Pressure plate	15 (or scaled)	While stood on	Traps, doors, entry detection
Block of redstone	15, constant	Always on, pushable	Piston driven switching
Daylight sensor	0 to 15 by light	Tracks day or night	Automatic lighting

Observer	1 tick pulse	Fires on block change	Farms, automation
Target block	0 to 15 by accuracy	On arrow hit	Minigames, ranged triggers

> ***BEST PRACTICE*** *— When you are building and testing, wire everything to a lever first. Because a lever holds its state, you can switch the circuit on, walk around, and study exactly what it did. Swap in a button or a sensor only once the logic works. Debugging with a button is like trying to read a page that is only lit for one second at a time.*

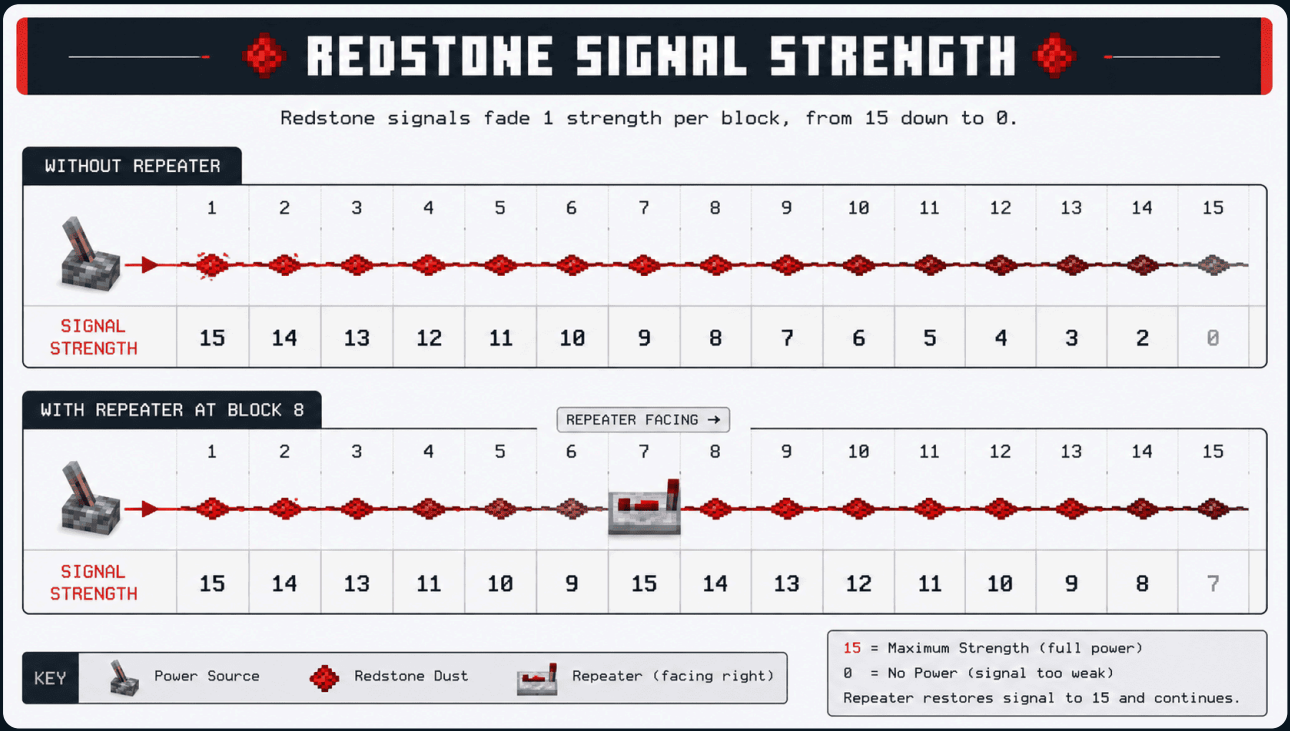
Chapter Three: Wiring, Repeaters, and Comparators

Signals need to travel, turn corners, climb walls, and sometimes be measured. Three components carry signal, and knowing what separates them is most of intermediate redstone.

Redstone dust: the wire

Dust is the basic wire. It carries the signal, branches to feed several places at once, and even climbs vertically if you run it up a staircase of blocks. Its

one great weakness is the decay rule from Chapter One: fifteen blocks and it dies. Dust also quietly powers the solid block it runs onto, which is how a signal travels from the floor up into a device. This is useful and occasionally maddening, because a stray line of dust can switch on something you did not mean to touch.



Top: an unassisted signal fading to nothing by block fifteen. Bottom: a repeater at the halfway point refreshes the signal to full strength, letting it travel on. This is how redstone covers distance.

The repeater: distance, direction, and delay

The repeater is the single most useful wiring component, because it does four jobs at once.

1. **It refreshes the signal back to 15.** Place one anywhere in a line and the signal past it starts fresh at full strength. This is how you send power across any distance: a repeater at least every fifteen blocks.
2. **It only allows signal one way.** A repeater is a diode. Signal passes through it in the direction of its arrow and cannot flow back. This stops circuits from "leaking" into each other.
3. **It adds delay.** Right click a repeater to set a delay of 1 to 4 redstone ticks (0.1 to 0.4 seconds). Chain several to build in longer, deliberate timing.
4. **It can lock.** Send a signal into the side of a repeater and it "locks," freezing whatever value it currently holds. That is a one block memory cell, and we will use it later.

The comparator: the component that measures

If the repeater is about moving a signal, the comparator is about **reading** one. It is the smartest wiring component, and it has two talents.

First, it can **measure the contents of a container**. Point a comparator out of a chest, barrel, furnace, hopper, or cauldron and it outputs a signal strength proportional to how full that container is. A nearly empty chest gives a weak signal; a full one gives a strong signal. This single trick powers item sorters, storage indicators, and auto brewing stands.

Second, it **compares two signals**. It has a rear input and a side input, and two modes you switch by right clicking the little torch on top:

- **Comparison mode** (torch unlit): the signal passes straight through from back to front, unless the side input is stronger, in which case the output is 0. It answers the question "is my main signal at least as strong as this other one?"
- **Subtraction mode** (torch lit): the output equals the back signal minus the side signal. Actual arithmetic, in a block.

Component	Job	Signal after it	One line summary
Redstone dust	Carry and branch	Decays 1 per block	The wire, but it fades
Repeater	Extend, direct, delay, lock	Refreshed to 15	The wire's amplifier and one way valve
Comparator	Measure and compare	Varies (analog)	The wire's brain

> ***WARNING*** *— The fifteen block limit is the most common bug in all of redstone. If a long line "just stops working," count the blocks. Nine times out of ten your signal died of old age somewhere in the middle. Drop in a repeater and it lives again.*

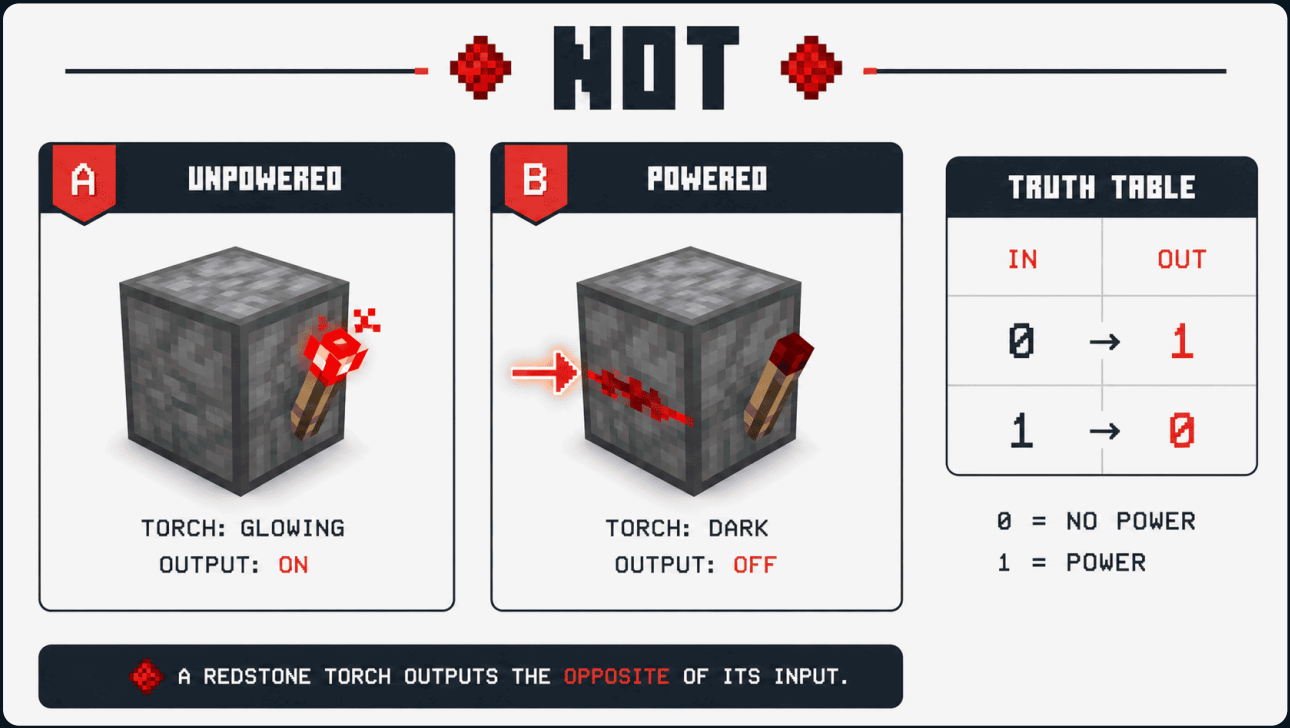
> *"The repeater moves a signal. The comparator understands it. Master the comparator and you can make redstone react to the state of your whole world."*

Chapter Four: The Inverter, and the Birth of Logic

We now arrive at the most important five minutes in this handbook. Everything so far has been plumbing: making signals and moving them around. Logic is where redstone starts to *think*, and it all begins with one component behaving in a way that seems, at first, completely backwards.

The redstone torch says "not"

A redstone torch, attached to a block, is **on by default**. It glows and outputs a signal of 15. But the moment you power the block it is attached to, the torch turns **off**.



The redstone torch is an inverter. It glows when its block is off, and goes dark when its block is powered. This backwards little behavior is the seed from which all redstone logic grows.

Read that again, because it is the whole game. Power in, light off. No power in, light on. The torch outputs the *opposite* of its input. In electronics this is called a **NOT gate** or an **inverter**, and it is the fundamental building block of every logic circuit ever made, in redstone or in silicon.

> ***DEFINITION*** *— An* ***inverter*** *(NOT gate) outputs the opposite of its input. On becomes off, off becomes on. In redstone, a single redstone torch on a block is a complete, working inverter. It is the smallest unit of "reasoning" a machine can do.*

Why does one flipped signal matter so much? Because "not" is the missing ingredient that lets you build every other kind of logic. With sources and wires alone, you can only ever say "if this, then that." Add the ability to say "if *not* this," and combine a few of them, and you can suddenly express any rule you can imagine: "turn on only if both switches are on," "only if exactly one is," "only if none are." The inverter is the hinge the whole door of logic swings on.

One more torch trick

The torch does something else worth knowing: it powers the solid block **above** it, and the dust around it. That means a torch is also your standard way to send a signal upward through a build and to feed power into the block a circuit sits on. Keep this in your pocket; vertical redstone leans on it constantly.

> *"Every computer that has ever existed is, when you strip it all the way down, a vast pile of NOT gates arranged very cleverly. Your redstone machine is the same idea, just bigger blocks."*

Chapter Five: Logic Gates

A **logic gate** takes one or more inputs (on or off) and produces an output according to a rule. String them together and you get behavior. Here are the ones worth knowing, described so you can actually build them.

OR: ”either one”

The easiest gate in redstone. Just **merge two lines of dust** into one. If either input is on, the output is on. That is an OR gate, and you have probably built one by accident. Use it when several different triggers should all be able to fire the same machine (three doors, any of which opens the same gate).

NOT: ”the opposite”

One redstone torch, from Chapter Four. Input on, output off.

AND: ”both, or nothing”

The AND gate turns on only when **every** input is on. It is slightly trickier because it needs inverters. The tidy way to think about it uses a rule logicians call De Morgan's law: *both A and B* is the same as *not (either A is off or B is off)*. So you invert each input with a torch, feed those into an OR, and invert the result. Three torches and some dust, and you have a gate that demands both switches before it acts. Use it for security: a door that opens only when two separate levers are both thrown.

XOR: ”one or the other, but not both”

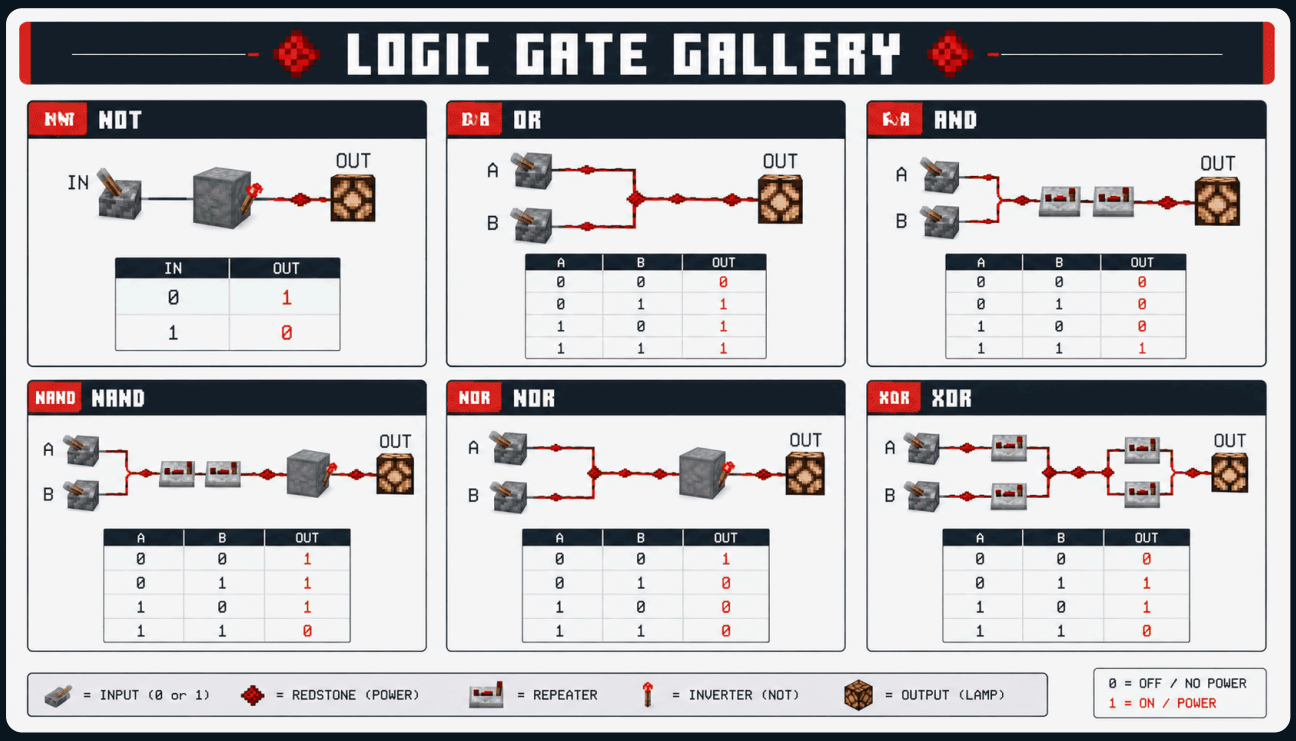
The exclusive OR turns on when its inputs are **different**, and off when they are the same. This is the famous "two light switches for one hallway" circuit: either switch, flipped, toggles the light, regardless of what the other is doing. XOR is fiddlier to build than the others, but it is so useful in a real base that it is worth memorizing a compact design and reusing it forever.

NAND and NOR: the inverted twins

NAND is simply AND with the output inverted (on unless *both* inputs are on). NOR is OR inverted (on only when *both* inputs are off). They sound like footnotes, but they hide a remarkable secret in the callout below.

A	B	AND	OR	NAND	NOR	XOR
0	0	0	0	1	1	0

0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0



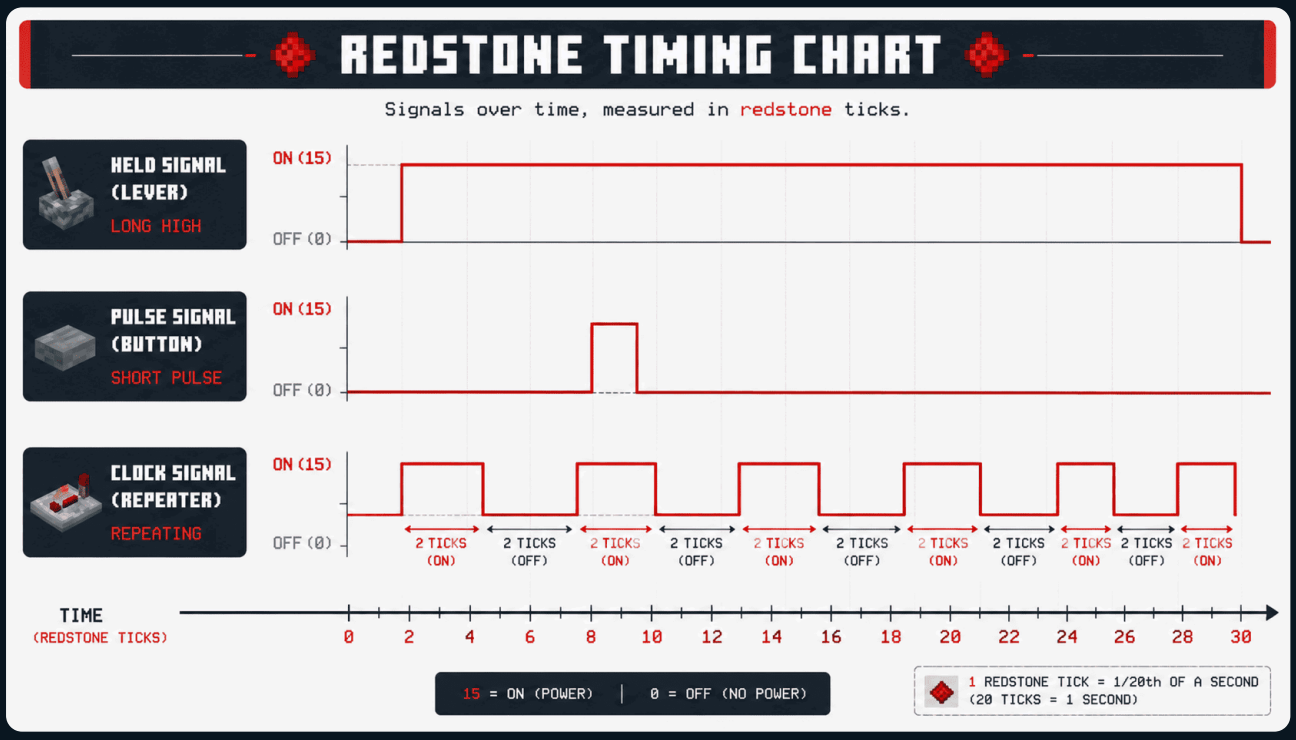
The six gates that build everything. You will use OR, NOT, and AND constantly; XOR is the secret behind the two switch light; NAND and NOR are their inverted cousins.

> ***INTERESTING FACT*** *— The NAND gate is "functionally complete," which means you can build every other gate, and therefore every possible circuit, using nothing but NAND gates. A working computer made entirely of one kind of gate is not just possible, it is a classic engineering exercise. Real players have built functioning CPUs in Minecraft on exactly this principle.*

> *"Logic gates feel abstract until the first time a door opens only because two friends both flipped their levers. Then you get it: you just built a rule out of rock and glowing dust."*

Chapter Six: Timing, Pulses, and Clocks

So far our signals have been simple: on or off. But *when* a signal happens, and for *how long*, is a whole dimension of redstone on its own.



Three shapes of signal over time. A held signal stays on, a pulse blips once, and a clock repeats forever. Controlling the shape of a signal in time is what separates a switch from a machine.

The redstone tick

Minecraft runs at 20 game ticks per second. Redstone timing is measured in **redstone ticks**, and one redstone tick equals two game ticks, or 0.1 seconds. That repeater delay of 1 to 4? Those are redstone ticks: 0.1 to 0.4 seconds. When builders say "add two ticks of delay," this is the unit they mean.

Pulses

A **pulse** is a signal that turns on and then off by itself. A button press is a pulse. An observer fires a very short one tick pulse. Sometimes a pulse is too long or too short for what you need, so builders use small **pulse circuits** to lengthen or shorten them: a pulse extender holds a brief signal on for longer, and a pulse limiter chops a long signal down to a quick blip. These are the redstone equivalent of adjusting how long a button "counts as pressed."

Clocks: signals that repeat

A **clock** is a circuit that turns itself on and off over and over, forever, producing a steady beat you can use to drive anything that needs to repeat: a flashing lamp, a repeating dispenser, an auto smelter.

Clock type	How it works	Speed	Adjustable?	Watch out for
Repeater clock	A loop of dust and repeaters	Medium	Yes, via repeater delays	Simple and reliable
Torch clock	A loop of an odd number of torches	Fast	Barely	Torches burn out if too fast
Hopper clock	Items bouncing between two hoppers, read by comparators	Slow, long	Yes, add items	Best for long timers (seconds to minutes)
Observer clock	Two observers facing each other	Very fast	No	Extreme speed causes lag

> ***WARNING*** *— A running clock never sleeps, and every tick costs the game a little processing. A base full of free running clocks is a base full of lag. Two rules save you: always wire a lever into your clock so you can switch it off when you are not using it, and never build a clock faster than the job actually needs. Also beware the torch clock: cycle a redstone torch on and off too quickly and it "burns out," going dark until it gets a block update. Add repeater delay to keep it healthy.*

> *"A lever gives you a state. A clock gives you a heartbeat. The moment your redstone has a heartbeat, it can do things while you walk away."*

Chapter Seven: Memory, or Making Redstone Remember

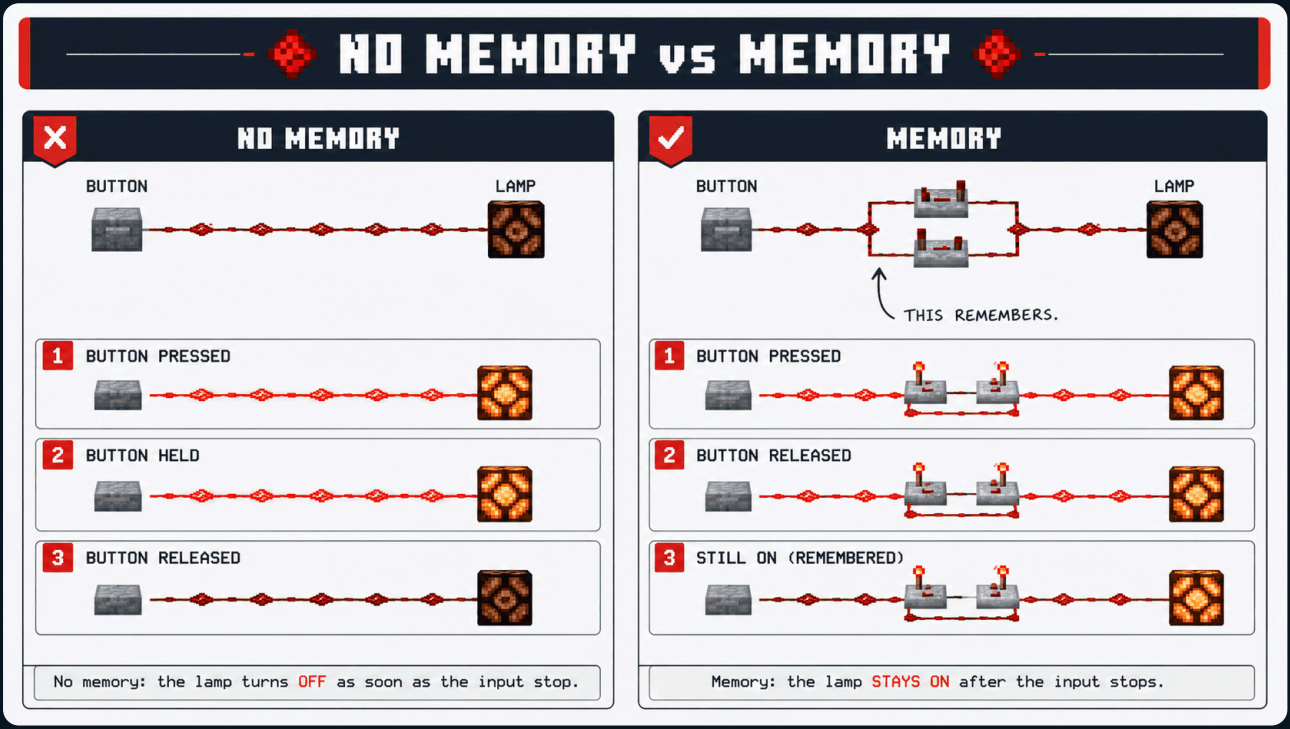
Redstone has no memory of its own. The instant you cut the power, it forgets everything and switches off. That is fine for a lamp on a lever, but it makes some obvious things surprisingly hard. How do you make a **single button** turn a light on, and turn it off again the next time you press it? The button only ever sends a brief pulse; the moment it resets, a plain lamp goes dark. You need a circuit that can *remember* what the last press meant. You need memory.

The RS latch: remember which button was pressed

The simplest memory circuit is the **RS latch**, built from two inverters (torches) cross wired so each feeds the other. It has two inputs, usually called **set** and **reset**. Hit set and the output turns on and *stays* on, even after you release the button. Hit reset and it turns off and stays off. It remembers which of the two you touched last. Use it for a light with separate on and off buttons, or to record "the alarm has been triggered" until someone clears it.

The T flip-flop: one button, toggle on and off

The circuit you will reach for most is the **T flip-flop** (the T is for "toggle"). It takes a single pulse as input and flips its output: press once for on, press again for off, from the same button. It turns a button into a lever, which is exactly what you want for a hidden door or a light you trigger from one spot.



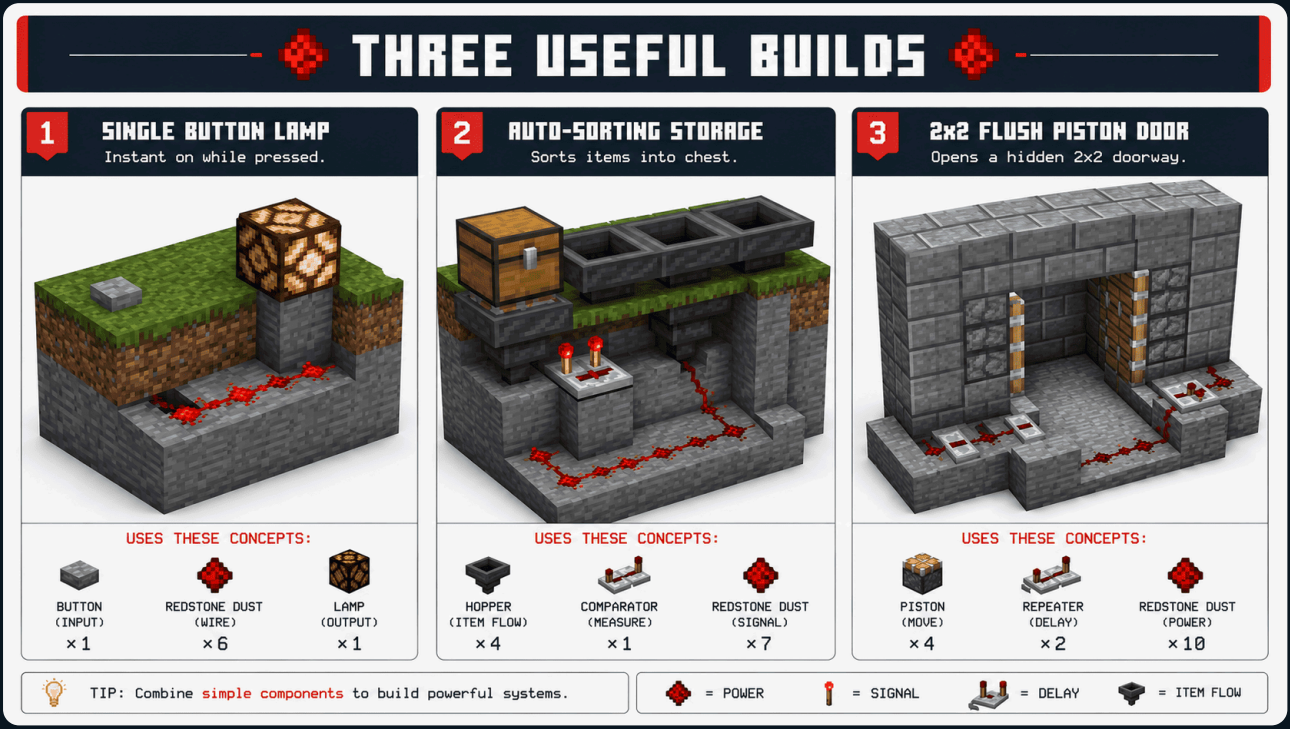
Without memory, redstone forgets the instant power stops. A memory circuit holds a state after the trigger is gone, which is what lets a single button behave like a switch.

The old way to build one was fiddly. The modern way is beautifully simple and worth memorizing: a **sticky piston** with a **block of redstone** on its face, watched by an **observer**, wired so each pulse pushes or pulls the redstone block into or out of the circuit. The piston physically holds the state, so it "remembers" with no power draw at all. Piston based memory like this is compact, lag friendly, and the backbone of countless survival builds.

> ***DEFINITION*** *— A* ***latch*** *holds one of two states based on separate set and reset inputs (it remembers which input fired last). A* ***flip-flop*** *changes state each time it receives a single repeated input (it remembers how many times it has been triggered, at least modulo two). Both are memory; they just answer different questions.*

> *"Logic lets redstone decide. Memory lets it remember what it decided. Put the two together and you no longer have a circuit. You have a machine with a past."*

Chapter Eight: Build It, Three Machines That Use Everything



Theory becomes real. Each of these three builds combines power sources, wiring, logic, timing, and memory from the previous chapters. Build them in order and you will have used every idea in this handbook.

Reading about redstone is like reading about swimming. Here are three projects, easy to hard, that force the ideas together.

1. The one button toggle lamp (beginner)

What it teaches: pulses and memory. Wire a button to a T flip-flop (use the piston and redstone block design from Chapter Seven), and wire the flip-flop's output to a redstone lamp. Press once, light on. Press again, light off. You have just made a button behave like a lever, which is the single most reused trick in decorative redstone.

2. The comparator item sorter (intermediate)

What it teaches: comparators reading containers, and analog signal. This is the machine that makes a survival base feel like magic: items travel along a line of hoppers, and each sorter module plucks out one specific item and drops it into its own chest, leaving everything else to flow onward.

The trick is a **filter hopper** holding a few of the target item plus filler, watched by a comparator. When the wanted item arrives, it tips the hopper's fill level past a set threshold, the comparator's signal ticks up, and that change unlocks the hopper to release the item into storage. Every other item fails the test and passes by. It is Chapter Three's "comparators measure containers" idea, turned into a sorting factory. Chain a module per item and your entire storage room organizes itself.

3. The 2x2 flush piston door (advanced)

What it teaches: everything. Timing, logic, sticky pistons, and clean sequencing. A flush hidden door uses **sticky pistons** to pull four blocks sideways into the wall, revealing a two by two gap, then push them back to seal it seamlessly. The challenge is timing: the blocks must retract in the right order and with the right delays (hello, repeaters) so nothing jams. Trigger it with a button through a pulse circuit, or with a hidden lever, or (for the paranoid) with a sculk sensor that only you know how to activate. Get this working and you are no longer a beginner.

> ***BEST PRACTICE*** *— Build every machine in creative mode first, flat and exposed, with levers for inputs and lamps for outputs so you can see the logic working. Only once it behaves perfectly should you compress it, hide the wiring, and move it into your survival world. Debugging a machine that is already buried inside a wall is the fastest way to quit redstone forever.*

Chapter Nine: Golden Rules and Troubleshooting

When a build misbehaves, resist the urge to tear it down. Redstone almost never fails randomly. Walk this list instead.

Is it actually powered? Trace the signal from the source. Is the source on? Did you wire into the right block?

Is it within fifteen blocks? The decay rule, again, because it really is that common. Count the blocks from the last repeater.

Is something backflowing? Two lines of dust bleeding into each other cause chaos. A repeater, which only lets signal pass one way, is the cure.

Does it need a block update? Some redstone states only refresh when a nearby block changes. Placing and breaking a block next to the circuit

("giving it an update") sometimes wakes a stuck component, especially a burned out torch.

Is it a Java versus Bedrock difference? This is the big one, described below.

> ***WARNING — Redstone is not the same in Java and Bedrock editions.*** *Circuits, especially compact ones from tutorials, frequently work in one edition and fail in the other. The most notorious cause is* ***quasi connectivity****, a Java only quirk where pistons, dispensers, and droppers can be activated by power reaching the block diagonally above them, not just directly. It is simultaneously a confusing bug and a beloved tool that many compact Java builds depend on. Bedrock does not have it. Always check which edition a tutorial was built for before you trust it.*

The golden rules

- **Test with levers, trigger with buttons.** Levers hold state for debugging; swap in the real trigger last.
- **A repeater every fifteen blocks, no exceptions.**
- **Use repeaters as diodes** wherever two circuits might bleed together.
- **Switch off your clocks** when idle, and never build them faster than needed.

- **Build flat and exposed first, hide it last.**
- **Label your edition.** What works in Java may not survive the trip to Bedrock.

Symptom	Likely cause	Fix
Long line dies partway	Signal decayed past 15 blocks	Add a repeater
Random extra thing turns on	Dust powering an adjacent block	Reroute, or isolate with a block
Circuit fires backwards	Signal backflow	Insert a repeater as a diode
Torch stuck off	Burned out from fast toggling	Slow the clock; give it a block update
Works for a friend, not you	Java vs Bedrock difference	Rebuild for your edition

> *"Redstone rewards the patient. Every builder who ever made a working computer out of blocks started exactly where you are now, staring at a line of dust that refused to light."*

Now go build something that shouldn’t be possible.

You have the fundamentals: signals, logic, timing, and memory. That is genuinely everything you need to build almost any machine in the game, from a self sorting storage hall to a working calculator. If you want to go further, Redwire's video course walks you through twenty real builds, block by block, in both Java and Bedrock, from your first piston door to a redstone powered adventure map. And the free weekly newsletter breaks down one clever community contraption every issue, fully annotated.

Redwire · Minecraft engineering, explained *Learn to think in redstone.*