Tutorials/Redstone

Redstone mechanics provide *Minecraft* with a loose analogue to electricity, which is useful for controlling and activating a variety of mechanisms. Redstone circuits and devices have many uses including automatic farms, controlling doorways, changeable or mobile buildings, transporting players and mobs, and more.

Redstone construction can range from fairly simple to deeply complex. While there is not a single overarching tutorial, there are many relevant pages under both the "Mechanics" and "Tutorials" trees. Some relevant pages include:

- Redstone dust: The core material that enables most redstone devices, being crafted into many of them and also placed to carry signals.
- Mechanics/Redstone: The basic game mechanics for redstone power and signals.
- Mechanics/Redstone/Components: The blocks that are used in and with redstone contraptions.
- Tutorials/Redstone tips: Hints and advice for building your redstone devices.
- Mechanics/Redstone/Circuit: Lists various types of reusable circuits that can be used to manipulate signals, with sub-pages giving examples of the various types.
- Help:Schematic: The "modern" way for redstone circuits to be represented on this wiki.
- MCRedstoneSim schematics: An older method for displaying redstone circuits.

Some pages dealing with specific blocks:

- Mechanics/Redstone/Piston circuits: A list of circuits making use of piston mechanics.
- Tutorials/Quasi-connectivity: Discusses the special mechanic of Quasi-connectivity.
- Tutorials/Hopper: All about how to use hoppers, including for item sorting.
- Tutorials/Observer stabilizer: Getting a better signal out of an observer
- Tutorials/Daylight detector, Tutorials/Day and night detector: Daylight sensors
- Tutorials/Automatic Respawn Anchor Recharger: Respawn Anchors

Circuits can be built into more complex devices:

- Tutorials/Mechanisms: Lists an assortment of complete devices using redstone.
- Tutorials/Minecarts: Large railway systems can benefit from redstone at the terminals.
- Tutorials/Redstone music: Creating music with Note Blocks and redstone circuits.
- Tutorials/Rube Goldberg machine: Complexity and spectacle!
- Tutorials/Block update detector: A specialized class of circuit; BUDs are mostly but not completely rendered obsolete by the Observer.
- Tutorials/Comparator update detector: An extension of BUDs that also spots inventory changes.
- Tutorials/Shulker box storage: Systems for loading and unloading shulker boxes.
- Tutorials/Zero-ticking: Exploiting a notable bug for rapid circuits.
- Tutorials/Combination locks: Creating combination locks
- Tutorials/Elevators: Vertical transportation
- Tutorials/Telegraph: Long-range signaling (through loaded chunks only)
- Tutorials/Command block: Use of creative-mode Command blocks
- Tutorials/Flying machines: Mobile machinery!
- Tutorials/Note block music: A working music machine!

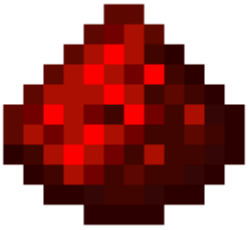
Perhaps the most ambitious redstone project of all is to build a working computer within *Minecraft*!

- Tutorials/Logic gates: Arithmetic logic.
- Tutorials/Advanced redstone circuits
- Tutorials/Redstone computers:
- Tutorials/Calculator: Build a calculator within *Minecraft*.
- Tutorials/Printing: And a printer/3D printer.

Most of the farming tutorials also include complete devices, including Tutorials/Egg farming and Tutorials/Cobblestone farming. Many traps also use redstone.

Redstone Dust

Inactive (connected)
Inactive (unconnected)
Active (connected)
Active (unconnected)



Renewable

Yes

Stackable

Yes (64)

Tool

Any tool

Blast resistance

0

Hardness

0

Luminous

No

Transparent

Yes

Flammable

No

Catches fire from lava

No

Redstone dust is a mineral that can transmit redstone power as a wire when placed as a block. It is also used in crafting and brewing.

Obtaining

Mining

See also: Redstone Ore § Natural generation

Redstone ore mined using an iron pickaxe or higher drops 4 or 5 redstone dust (or more with Fortune, averaging at 6 redstone dust with Fortune III). If mined with Silk Touch, the block drops itself instead of redstone dust.

Natural generation

15 lengths of redstone dust are naturally generated as part of the trap in each jungle pyramid. 5 lengths of redstone dust can be found in one type of jail cell room in a woodland mansion. In ancient cities, multiple pieces of redstone dust can be found integrated into circuitry.

Breaking

Redstone dust can be broken instantly using any tool, or without a tool, and drops itself as an item.

Redstone dust is removed and drops as an item if:

- its attachment block is moved, removed, or destroyed
- water or lava flows into its space
- a piston tries to push it or moves a block into its space

Mob loot

Witches have a chance of dropping 0–2 redstone dust upon death. This is increased by 1 per level of Looting, for a maximum of 0–5 redstone dust.

Chest loot

Item	Structure	Container	Quantity	Chance
	Java Edition			
Redstone Dust	**Dungeon**	Chest	1–4	26.6%
	Mineshaft	Chest	4–9	14.5%
	Stronghold	Storeroom chest	4–9	18.6%
		Altar chest	4–9	12%
	Village	Temple chest	1–4	44.8%
	Woodland mansion	Chest	1–4	26.6%
	Bedrock Edition			
Redstone Dust	**Dungeon**	Chest	1–4	26.6%

Mineshaft	Chest	4–9	14.5%
Stronghold	Storeroom chest	4–9	15.2%
	Altar chest	4–9	11.6%
Village	Temple chest	1–4	44.8%
Woodland mansion	Chest	1–4	26.6%

Crafting

Redstone dust can be crafted from blocks of redstone.

Ingredients	Crafting recipe
Block of Redstone	

Smelting

Name	Ingredients	Smelting recipe
Redstone Dust	Redstone Ore or Deepslate Redstone Ore + Any fuel	

	0.7

Trading

In *Java Edition*, novice-level cleric villagers sell two redstone dust for one emerald.

In *Bedrock Edition*, novice-level cleric villagers sell four redstone dust for one emerald.

Villager gifts

See also: Tutorials/Raid farming

In *Java Edition*, when the player has the Hero of the Village status effect, clerics might throw that player a redstone dust as a gift.

Usage

Redstone dust is used for brewing, crafting, and in redstone circuits by placing it on the ground to create redstone wire. It can also be used to power redstone components.

Brewing ingredient

Name	Ingredients	[hide] **Brewing recipe**
Mundane Potion	**Redstone Dust** + Water Bottle	

Increased Duration	**Redstone Dust** + Potion of Fire Resistance or Potion of Invisibility or Potion of Night Vision or Potion of Poison or Potion of Regeneration or Potion of Slowness or Potion of Strength or Potion of Swiftness or Potion of Water Breathing or Potion of Weakness or Potion of Leaping or Potion of Slow Falling	

Crafting ingredient

Name	Ingredients	Crafting recipe	[hide] Description
Block of Redstone	**Redstone Dust**		

Clock	Gold Ingot + **Redstone Dust**		
Compass	Iron Ingot + **Redstone Dust**		
Detector Rail	Iron Ingot + Stone Pressure Plate + **Redstone Dust**		
Dispenser	Cobblestone + Bow + **Redstone Dust**		The bow can be of any durability. Enchantments on the bow do not affect the resulting dispenser.
Dropper	Cobblestone + **Redstone Dust**		

Note Block	Any Planks + **Redstone Dust**		
Observer	Cobblestone + **Redstone Dust** + Nether Quartz		
Piston	Any Planks + Cobblestone + Iron Ingot + **Redstone Dust**		
Powered Rail	Gold Ingot + Stick + **Redstone Dust**		
Redstone Lamp	**Redstone Dust** + Glowstone		
Redstone Repeater	Redstone Torch + **Redstone Dust** + Stone		

Redstone Torch	**Redstone Dust** + Stick		
Target	**Redstone Dust** + Hay Bale		

Redstone component

When placed in the world, redstone dust becomes a block of "redstone wire"[*more information needed*], which can transmit redstone power.

Smithing ingredient

Ingredients	**Smithing** recipe	Description
Any Armor Trim + Any Armor Piece + Redstone Dust	Upgrade Gear	All armor types can be used in this recipe, a netherite chestplate is shown as an example.

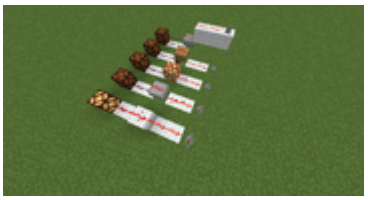
Trim color palette

The following color palette is shown on the designs on trimmed armor:

-

Placement

Examples of redstone wire configuration. *Top Left:* Redstone wire connects diagonally vertically through non-opaque blocks. *Top Right:* Redstone wire does *not* connect diagonally vertically through opaque blocks. *Center:* Redstone wire gets darker as its power level drops, to a maximum of 15 blocks from a power source.



Examples of redstone wire placements.

Redstone dust can be placed on opaque blocks as well as glowstone, upside-down slabs, glass, upside-down stairs, and hoppers. It can also be placed on some transparent blocks; see Opacity/Placement for more information. It cannot be placed suspended in midair, even with commands, which is not unintentional.[1]

Redstone wire configures itself to point toward adjacent redstone power components and transmission component connection points. Redstone wire also configures itself to point toward adjacent redstone wire one block higher or lower – unless there is a solid opaque block above the lower redstone wire.

If there is only one such adjacent redstone component, redstone wire configures itself into a line pointing both at the neighbor and away from it. If there are two or more such adjacent components, redstone wire connects them in the form of , , , or as needed.

When there are no adjacent components, a single redstone wire configures itself into a plus sign, which can provide power in all four directions. By right-clicking it can be changed into a dot, which does not provide power to any of the four directions.

In *Bedrock Edition*, redstone wire automatically configures itself to point toward adjacent blocks or mechanism components. In *Java Edition*, it does not. If such a configuration is desired, the other neighbors of the redstone wire must be arranged to create it, i.e the redstone dust must be placed in a way that it would be pointed at the block's location even if it were not there.

When redstone wire is reconfigured after placement, it does not update other redstone components around it of the change unless that reconfiguration also includes a change in power level or another component provides an update. This can create situations where a mechanism component remains activated when it shouldn't, or vice versa, until it receives an update from something else – a "feature" of redstone wire that can be used to make a block update detector.

Behavior

Redstone wire can transmit power, which can be used to operate mechanism components (doors, pistons, redstone lamps, etc.). Redstone wire can be "powered" by a number of methods:

- from an adjacent power component or a strongly-powered block
- from the output of a redstone repeater or redstone comparator
- from adjacent redstone wire. The powering dust can be a level higher or lower, but with restrictions:
    - Redstone dust can be powered by redstone dust that is one level lower, or on an opaque block one level higher. A transparent block cannot[*Java Edition only*] pass power downward.
    - The block "between" the two dust blocks must be air or transparent. A solid block there "cuts" the connection between the higher and lower dust.

The "power level" of redstone dust can vary from 0 to 15. Most power components power-up adjacent redstone dust to power level 15, but a few (daylight sensors, trapped chests, and weighted pressure plates) may create a lower power level.

Redstone repeaters output power level 15 (when turned on), but redstone comparators may output a lower power level.

Power level drops by 1 for every block of redstone wire it crosses. Thus, redstone wire can transmit power for no more than 15 blocks. To go further, the power level must be re-strengthened – typically with a redstone repeater.

Powered redstone wire on top of, or pointing at, an opaque block provides *weak* power to the block. A weakly-powered block cannot power other adjacent redstone wire, but can still power redstone repeaters and comparators, and activate adjacent mechanism components. Transparent blocks cannot be powered.

When redstone wire is unpowered, it appears dark red. When powered, it becomes bright red at power level 15, fading to darker shades with decreasing power. Powered redstone wire also produces "dust" particles of the same color.

While redstone wire always provides power to the directions it points into, it can still point into directions in which it cannot give power. If redstone wire comes in the form of a cross, the player can right-click to toggle it between a cross and dot. A redstone dot does not power anything adjacent to it, but powers the block under it.

Redstone mechanics

Redstone mechanics provide *Minecraft* with a loose analogue to electricity, which is useful for controlling and activating a variety of mechanisms. Redstone circuits and devices have many uses including automatic farms, controlling doorways, changeable or mobile buildings, transporting players and mobs, and more. Some relevant pages include:

- Redstone dust is the core of redstone mechanics. Mined from redstone ore, the dust can be placed to form "redstone wires" to carry signals, or crafted into other devices.
- Components lists and describes the various blocks which interacts with or are affected by redstone power.
- Redstone Circuits covers the basics of redstone circuitry, including summaries of the basic circuit types and the blocks (e.g., doors, pistons) which can be controlled by redstone.
- Tutorials/Mechanisms lists and describes a variety of more complex projects that can be made with redstone.

Redstone Concepts

Redstone tick

A **redstone tick** is a unit of time, in redstone, that is equal to two game ticks, 0.1 seconds. Most redstone components take a multiple of a redstone tick to change states. Redstone torches, redstone repeaters, and other redstone components require one or more ticks to change state, so it can take several ticks for a signal to propagate through a complicated circuit.

Redstone ticks differ from "game ticks" (20 per second) and "block ticks" (block updates that occur at each game tick). When discussing redstone circuits, the term "tick" should always be interpreted to mean a redstone tick, unless otherwise specified.

Redstone components

Main article: Redstone components

A Redstone component is a block that provides some purpose to a Redstone circuit.

- A **power component** provides power to other parts of a circuit—e.g., redstone torches, buttons, levers, redstone blocks, target blocks, etc. Some of these fall into one of three overlapping subgroups:
    - Switches provide power depending on request by the player. Buttons and levers are switches.
    - Sensors provide power or signals (see below) in response to some environmental condition. Pressure plates and Observers are sensors, and comparators can be used as sensors. Note that some pressure plates can be triggered by a player standing on them, which also qualifies them as switches.
    - Logic components provide power conditionally, depending on their input conditions. Redstone torches, and comparators are classic logic components; redstone wire and ordinary opaque blocks can also be used to combine signals in various ways.
- A **transmission component** passes power from one part of the circuit to another. Redstone dust (placed as redstone wire) is the most fundamental transmission component, but redstone repeaters and redstone comparators are also important.
- A **mechanism component** affects the environment (by moving, producing light, etc.)—e.g., Doors, pistons, redstone lamps, dispensers, etc.

Power

The Redstone Lamps are all **activated**, but are **powered** differently. From top to bottom:

1. Strongly powered: powers both Repeater and Dust.
2. Weakly powered: powers Repeater, but not Dust.
3. Not powered: powers neither.

Redstone components and blocks may or may not be powered. A "powered block" can be thought of as a block that has electricity running through it. Some blocks will show their powered state visibly (for example, redstone dust lights up, a redstone lamp illuminates its surroundings and a redstone torch turns off), but other blocks may give no visual indication of their powered state other than their effect on other redstone components.

An opaque block (e.g. stone, dirt, etc.) powered by a power component, or by a repeater or comparator, is said to be **strongly powered** or 'hard-powered' (a different concept from power level). A strongly powered block can power adjacent redstone dust (including dust on top of the block or dust beneath it).

An opaque block powered only by redstone dust (and no other components) is said to be **weakly powered** or 'soft-powered' because a block powered only by redstone dust will not power **other** redstone dust (but can still power other components or devices, such as repeaters and pistons).

No opaque block can directly power another opaque block—there must be dust or a device in between. A transparent block can't be powered by anything. "Strong"/"hard" vs. "weak"/"soft" power applies only to opaque blocks, not to dust or other redstone components.

A powered block (strong or weak) can affect adjacent redstone components. Different redstone components react differently to powered blocks—see their individual descriptions for details.

Signal strength

Redstone "signal strength" can be an integer between 0 and 15. Most power components provide an output of power level 15, but a few components provide a variable amount of power. These include daylight sensors and redstone comparators.

Redstone dust transmits power to adjacent redstone dust and blocks, but its strength decreases by 1 for each block the redstone power travels. Redstone dust can thus transmit power up to 15 blocks before needing to be *maintained* with a redstone comparator or *re-strengthened* with a repeater. Power level only fades with the dust-to-dust transmission, not between dust and a device or block.

The power level can also be adjusted directly with a redstone comparator in comparison or subtraction mode.

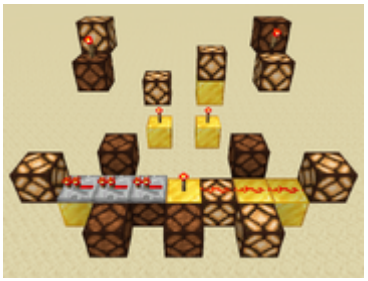
Signals and pulses

Circuits with a stable output are said to produce a **signal** — an ON signal (also "high" or "1") if powered, or an OFF signal ("low", "0") if unpowered. When a signal changes from OFF to ON ("rising edge") and then back to OFF ("falling edge"), that is described as a **pulse** (or ON pulse), while the opposite is described as an OFF

pulse. ON pulses are far more common, and in casual discussion, "a signal" often refers to an ON pulse.

Very short pulses (1 or 2 ticks) can cause problems for some components or circuits because they have different update sequences to change states. For example, a redstone torch or a comparator will not respond to a 1-tick pulse.

Activation



Activation of Mechanism Components — Mechanism components can be activated by power components (for example, redstone torches), powered blocks, redstone dust, repeaters, and comparators (not shown), but only if configured correctly.

Mechanism blocks (pistons, doors, redstone lamps, etc.) can be **activated** by incoming power, which causes the mechanism component to do something (push a block, open the door, turn on, etc.).

Activation behavior

There are two main variations for how things can respond to activation:

- Many types only perform an action when initially activated by a rising edge (command blocks execute a command, droppers and dispensers eject an item, note blocks play a sound) and won't do anything again until deactivated and then activated again
- Other mechanism components change their state when activated, and then change back when the activation ends; Redstone lamps stay on while the power continues, while hoppers stay disabled. Pistons will extend when powered, and retract when the power turns off. Doors,

fence gates, and trapdoors will open on a rising edge, and close on a falling edge; however, most of these (not iron doors or iron trapdoors) can *also* be opened or closed by players regardless of the redstone power. If they were already open when power turns on, or closed when power ends, they will simply remain so until their input changes again.

Powered vs. activated

Powered vs. Activated
The top lamp is both *activated* (the lamp is on) and *powered* (it powers the adjacent repeater).
The bottom lamp is activated but *not* powered.

For opaque mechanism blocks (command blocks, dispensers, droppers, note blocks, and redstone lamps), it is important to make a distinction between a mechanism component being *activated* and being *powered* (and this is the reason why mechanism components are described as activated instead of just saying they are powered).

- A mechanism component is **powered** if it could power adjacent redstone dust (**strongly**), or repeaters or comparators (**weakly**).
- A mechanism component is **activated** if it is doing something (or has done something and is waiting to be activated again).

Any method of powering a mechanism component (such as a redstone torch underneath it) will also activate it, but some activation methods (such as a redstone torch next to or above a mechanism component) won't actually power the component (following the usual rules for power components).

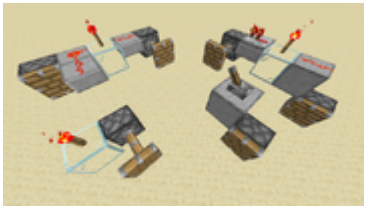
Non-opaque mechanism components (doors, fence gates, hoppers, pistons, rails, trapdoors) can be activated (they can do things), but cannot be powered (i.e. they can not then power adjacent redstone dust, etc.).

Normal activation rules

In general mechanism components are activated by:

- an adjacent active **power component**, including above or below.
    - *Exception:* A redstone torch will not activate a mechanism component it is attached to
    - *Exception:* A piston is not activated by a power component directly in front of it.)
- an adjacent **powered opaque block** (either strongly-powered or weakly-powered), including above or below.
- a powered **redstone comparator** or **redstone repeater** facing the mechanism component.
- powered **redstone dust** configured to point at the mechanism component (or on top of it, for mechanical components that can support redstone dust, but *not* beneath it), or adjacent "directionless" redstone dust; a mechanism component is *not* activated by adjacent powered redstone dust that is not configured to point at it

Special activation rules



Activation by Quasi-Connectivity — Pistons can also be activated by anything that activates the space *above* them. Note that the piston on the far left is *not* activated by quasi-connectivity because the redstone dust is running *past* the block above the piston, rather than directly into it, and thus would not power a mechanism there

Some mechanism components have additional ways of being activated:

- In *Java Edition*, **pistons**, **dispensers**, **droppers**, and can also be activated if one of the methods above *would* activate a mechanism component in the block above the component, even if there is no mechanism component there (even if the block above the component is air or a transparent block). This rule is often simplified to saying that the components can be powered by blocks diagonally above or two blocks above, but other methods of such activation exist (see image to the right). This method of activation is known as **quasi-connectivity** because the mechanism component's activation is somewhat connected to the space above it.
- **Doors** occupy two spaces, one above the other, and anything that activates either space also activates the other.

Redstone block updates

Block upates are how most redstone compenents "tell" each other that they need to change states.

When a change occurs somewhere in a redstone circuit, it can produce other changes in surrounding blocks in what is called a **block update.** Each of these changes can then produce other changes in their surrounding blocks. The update will propagate following the redstone circuit rules within loaded chunks (block updates will not propagate into unloaded chunks), usually very quickly. *Note: in Bedrock Edition, block updates and redstone are not connected.*

A block update simply notifies other Redstone components and blocks that a change has occurred nearby and allows them to change their own state in response, but not all updates will necessarily require changes. For example, if a redstone torch activates and updates the dust below it, the dust may already be powered from something else, in which case the dust won't change state and the update propagation will stop there.

Block updates can also be generated by any immediate neighbor block being placed, moved, or destroyed.

Solid blocks don't "know" if they're powered or not. Block updates simply update enough blocks around a redstone component to update other redstone components around the solid block (for example, a pressure plate updates its neighbors and the neighbors of the block it's attached to, which includes the space under that block which might be redstone dust).

In addition to block updates, comparators can be updated by containers (including detector rails with container minecarts on them) and certain other blocks, up to two blocks away horizontally when their state changes (for example, when their inventory changes). This is known as a comparator update.

The following redstone components produce block updates up to two blocks away by taxicab distance, including up and down:

- Redstone Dust (All directions)
- Redstone Torch (Up and down)
- Flat and slanted rails, activator rails, detector rails, and powered rails (Up and down if slanted, down only otherwise)

The following redstone components produce block updates in their immediate neighbors, including above and below, *and* in the immediate neighbors of the block they're attached to:

- Redstone Repeater (as if "attached" to the block it is facing)
- Redstone Comparator (as if "attached" to the block it is facing)
- Buttons
- Detector Rail (flat only; also produces comparator updates)
- Lever
- Pressure Plates

- Trapped Chest (as if "attached" to the block beneath; also produces comparator updates)
- Tripwire Hook
- Weighted Pressure Plates
- Observer

The following redstone components update only their immediate neighbors when they change their state, including above and below:

- Daylight Detector
- Inverted Daylight Detector (the Inverted Daylight Detector is not obtainable as an item)
- Note Block
- Leaves
- Scaffolding

This is an XOR gate.

- Tripwire (can also activate tripwire hooks in valid tripwire circuit)
- Piston and Sticky Piston (from both the piston base and the piston head when extended

The following redstone components do *not* produce block updates when they change their state (though any block will produce a block update in its immediate neighbors if moved or destroyed):

- Impulse Command Block (also produces comparator updates)

- Repeating Command Block (also produces comparator updates)
- Chain Command Block (also produces comparator updates)
- Dispenser (also produces comparator updates)
- Dropper (also produces comparator updates)
- Doors
- Fence Gates (can be moved)
- Hopper (also produces comparator updates)
- Redstone Lamp (can be moved)
- Trapdoors (can be moved)

Redstone components

SIGN IN TO EDIT

Redstone components are the blocks used to build redstone circuits. Redstone components include power components (such as redstone torches, buttons, and pressure plates), transmission components (such as redstone dust and redstone repeaters), and mechanism components (such as pistons, doors, and redstone lamps).

This article assumes familiarity with the basics of redstone structures. This article also limits its discussion of each component to its role in redstone structures; for full details about a component, see the main article for the block.

Power components

Power components create redstone signals, either permanently or in response to player, mob, and environmental activity.

Block of redstone

Block of Redstone's range of activation

It does not **power** any adjacent opaque block

Main article: Block of Redstone

A block of redstone provides constant power. It can be moved by pistons.

Activation

A block of redstone is always ON.

Effect

A block of redstone powers adjacent mechanism components (including those above or below) and adjacent redstone dust. It also powers adjacent redstone comparators or redstone repeaters facing away from it.

A block of redstone does not power adjacent opaque blocks.

Button

Button's range of activation

It **powers** the opaque block it is attached to

Main article: Button

A button is used to generate a pulse. A button may be of two types: wooden or stone.

Placement

A button can be attached to any part of most opaque blocks. If the attachment block is removed, the button drops into the item form.

Activation

A player can activate a button by right-clicking it. A stone button stays ON for 10 ticks (1 second), while a wooden button stays ON for 15 ticks (1.5 seconds). A wooden button can also be turned ON by an arrow that has been shot at it. In such a case, the button remains ON until the arrow despawns (after one minute) or is taken.

Effect

While activated, a button and its attachment block both power adjacent redstone dust (including beneath the button, and beneath and on top of the block), and all adjacent redstone comparators or redstone repeaters facing away from it. They also activate all adjacent mechanism components (including those above or below).

Calibrated sculk sensor

Main article: Calibrated Sculk Sensor

This section of the article is empty.

You can help by adding to it.

Chiseled bookshelf

Main article: Chiseled Bookshelf

This section of the article is empty.

You can help by adding to it.

Daylight detector

Daylight Detector's range of activation

It does not **power** any adjacent opaque block

Main article: Daylight Detector

A daylight detector can be used to detect the time of the *Minecraft* day.

Activation

A daylight detector, if exposed to the sky ("sl" greater than 0), remains activated while the sun is in in the sky. A daylight detector blocked from the sky ("sl" equals 0) remains activated while the moon is in the sky.

Effect

While activated, a daylight detector powers adjacent redstone dust (including beneath it), and all adjacent redstone comparators or redstone repeaters facing away from it, at a power level proportionate to the height of the sun or moon in the sky. It also activates all adjacent mechanism components (including those above or below).

A daylight detector does not power adjacent opaque blocks.

Detector rail

Detector Rail's range of activation

It **powers** the opaque block it is attached to

Main article: Detector Rail

Detector rail as power component

A detector rail is used to detect the passage of a minecart.

Placement

A detector rail can be attached to the **top** of any opaque block, or to the **top** of an upside-down slab or upside-down stairs. If the attachment block is removed, the detector rail drops as an item.

When placed, a detector rail lines up with adjacent rails, powered rails, and other detector rails, as well as adjacent rails one block above it. If there are two adjacent rails not on opposite sides, or three or more adjacent rails, the detector rail lines up in the east-west direction. If there are no adjacent rails, the detector rail lines up in the north-south direction. If there is an adjacent rail one block

above, the detector rail slants to match it (when there is more than one adjacent rail to slant toward, the order of preference is: west, east, south, and north). Other configurations can be created by placing and removing various rail.

Activation

A detector rail turns ON when a minecart passes over it, and turns OFF when it leaves.

Effect

While activated, a detector rail and its attachment block (unless attached to a slab or stairs) both power adjacent redstone dust (including beneath the block), and all adjacent redstone comparators or redstone repeaters facing away. They also activate all adjacent mechanism components (including those above or below).

Jukebox

A jukebox with a music disc playing emits a redstone signal of strength 15.

Lectern

Lectern's range of activation

It **powers** the opaque block beneath it

Main article: Lectern

A lectern is a block that can hold written books so they can be read.

Placement

Lecterns can face north, south, east or west, facing toward the player when placed. This has a redstone effect judging on the book page.

Activation

When the page of the book it is holding is turned, the lectern emits a redstone pulse that is one game tick long (0.5 redstone ticks).

Lever

Lever's range of activation

It **powers** the opaque block it is attached to

Lever as power component

A lever is used to switch circuits on or off, or to permanently power a block.

Placement

A lever can be attached to any part of most opaque blocks, or to the **top** of an upside-down slab or upside-down stairs. If the attachment block is removed, the lever drops as an item.

Activation

A player can turn a lever ON or OFF by right-clicking it.

Effect

While activated, a lever and its attachment block (unless attached to a slab or stairs) both power adjacent redstone dust (including beneath the lever, or beneath or on top of the block), and all adjacent mechanism components (including those above or below it). They also activate all adjacent redstone comparators or redstone repeaters facing away.

Lightning rod

Lightning Rod's range of activation

It **powers** the opaque block it is attached to

A lightning rod emits a redstone signal strength of 15 when struck by lightning.

Observer

Observer's range of activation

It does not activate adjacent components

It **powers** the opaque block behind it

Main article: Observer

An observer can be used to detect block changes.

Placement

An observer can be placed anywhere and can face in any direction, including up or down. When placed, the observer's side that detects block changes (its face) faces away from the player and the side that produces a pulse faces the player.

Activation

An observer turns ON when the block in front of its face changes state (for example, a block being placed or mined, water changing to ice, a repeater having its delay changed by a player, etc.). The observer stays ON for 2 redstone ticks (4 game ticks, or 0.2 seconds barring lag) and then turns OFF automatically.

An observer also turns ON for 1 game ticks after it is moved by a piston.

Effect

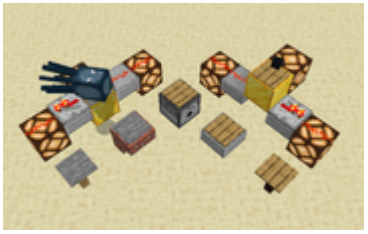
When activated, an observer produces a 1tick pulse from the side opposite its face.

Pressure plate

Pressure Plate's range of activation

It **powers** the opaque block that supports it

Main article: Pressure Plate



Pressure plate as power component

A pressure plate can be used to detect mobs, items, and other entities. A pressure plate may be of two types: wooden or stone.

Placement

A pressure plate can be attached to the **top** of any opaque block, or to the **top** of a fence, nether brick fence, an upside-down slab or upside-down stairs. If the attachment block is removed, the pressure plate drops as an item.

Activation

A pressure plate turns ON when an entity (mob, item, etc.) crosses or falls on it, and turns OFF when the entity leaves or is removed. A wooden pressure plate may be turned ON also by falling items and arrow shots. A wooden pressure plate that is activated in this way turns OFF when the object is picked up or despawns (after one minute for a shot arrow, or up to five minutes for an item).

Effect

While activated, a pressure plate and its attachment block (unless attached to a fence, nether brick fence, slab, or stairs) both power adjacent redstone dust (including beneath the block), and all adjacent mechanism components (including those above or below). They also activate all adjacent redstone comparators or redstone repeaters facing away.

Considerations

A pressure plate is not solid (it cannot be used as a wall or platform). Usually a block under a pressure plate provides solid ground (for mobs to walk across, items to fall on, etc.), but when a pressure plate is placed on a block with a small

collision mask, like a fence or nether brick fence, it is possible for entities to move through the pressure plate and still activate it. Thus, a pressure plate on a fence can be used to detect entities without stopping them (more compactly than a tripwire circuit).

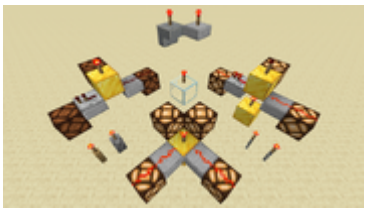
Redstone torch

Redstone Torch's range of activation

It **powers** the opaque block that is **above** it

It does not power/activate the block/component it is attached to

Main article: Redstone Torch



Redstone torch as power component

A redstone torch powers circuits (horizontally and vertically), and can invert signals.

Placement

A redstone torch can be attached to any surface (except the bottom) of any opaque block, or to the **top** of: a cobblestone wall, a fence, glass, nether brick fence, an upside-down slab or upside-down stairs. If the attachment block is removed, the redstone torch drops as an item.

Activation

A redstone torch turns OFF when its attachment block receives power from another source and turns back on when the block loses power.

Effect

While activated, a redstone torch and any opaque block above it both power adjacent redstone dust (including beneath the redstone torch, or on top of the

block), and all adjacent mechanism components (including those above or below it). They also activate all adjacent redstone comparators or redstone repeaters facing away from it.

A redstone torch does not affect the block it is attached to (even if it is a mechanism component).

Considerations

A redstone torch can burn out (stop turning on) when it is forced to flicker on and off too quickly (by powering and de-powering its attachment block). After burning out, a redstone torch re-lights when it receives a redstone update, or randomly after a short time.

One way to cause a burnout is with a **short-circuit** – using a torch to turn itself off, which then allows the torch to turn back on, etc. For example, placing redstone dust on top of a block with a redstone torch on its side, then putting another block above the torch, causes the torch to power the top block, which activates the dust, which powers the first block, turning the torch off – this feedback loop causes the redstone torch to flicker and burn out. When putting a torch underneath a block, make sure that the block isn't adjacent to redstone dust or the torch can burn out.

Sculk sensor

A sculk sensor emits a redstone signal when it detects a vibration

Target

Target's range of activation

It does not **power** any adjacent opaque block

Main article: Target

A Target emits a redstone signal when hit by a projectile (including arrows, tridents, eggs, snowballs, splash potions, fire charges fired from dispensers, and lingering potions, but excluding ender pearls and eyes of ender).

Activation

Arrows and Tridents emit a pulse of 10 redstone ticks, while other projectiles emit a pulse of 4 redstone ticks. The closer the projectile is to the center of the block, the stronger the signal it produces is, from 1 (at the edge) to 15 (in the center).

Trapped chest

Trapped Chest's range of activation

It **powers** the opaque block beneath it

Main article: Trapped Chest

A trapped chest can be used to detect when a player tries to take from it.

Activation

A trapped chest is turned ON when a player accesses its contents.

Effect

While activated, a trapped chest and any opaque block beneath it both power adjacent redstone dust (including beneath the block), and all adjacent mechanism components (including those above or below it). They also activate all adjacent redstone comparators or redstone repeaters facing away from it, at a power level equal to the number of players simultaneously accessing its contents (maximum 15).

Tripwire hook

Tripwire Hook's range of activation

It **powers** the opaque block it is attached to

Main article: Tripwire Hook

Tripwire hook as power component – The tripwire hooks and the blocks they are attached to provide power, but the tripwire does not.

A tripwire hook is used to detect mobs, items, and other entities over a large area.

Placement

A tripwire hook can be attached to the **side** of most opaque blocks. If the attachment block is removed, the tripwire hook drops as an item.

In order to function correctly, a tripwire hook must be part of a tripwire circuit: two opaque blocks attached to tripwire hooks, at the ends of a tripwire line (one or more blocks of tripwire).

To place tripwire, right-click on an adjacent block with a string. Tripwire can be placed on the ground or in the air, and forms a valid tripwire line only if all the tripwire is of the same type. Tripwire is considered on the ground if placed on any opaque block, or on a block of redstone, a hopper, an upside-down slab, or an upside-down stairs. Tripwire is considered in the air if placed on or above any other block. Tripwire on the ground has a short hitbox (1/8 block tall), while tripwire in the air has a taller hitbox (1/2 block tall).

If the attachment block under ground tripwire is removed, the tripwire drops as string.

A tripwire circuit is properly placed when the tripwire hooks are fully extended and the tripwire line runs continuously between the tripwire hooks. Tripwire lines from separate tripwire circuits can be placed next to each other (in parallel), above each other, and can even intersect each other.

Activation

A tripwire hook turns ON when an entity (mob, item, etc.) crosses or falls on the hook's tripwire line (but *not* the tripwire hook), and turns OFF when all entities leave or are removed from the tripwire line. A tripwire hook also turns ON for 5 ticks (1/2 second) when any of its tripwires are destroyed, except when using shears to cut the tripwire. Breaking the tripwire hook, or its attachment block, does not generate a pulse.

Effect

While activated, a tripwire hook and its attachment block both power any adjacent redstone dust (including below the tripwire hook, or beneath or above the block), and all adjacent mechanism components (including those above or below it). They also activate all adjacent redstone comparators or redstone repeaters facing away from it.

Tripwire itself provides no power.

Transmission components

Transmission components propagate signals and pulses from power components to mechanism components. Complex effects can also be produced by allowing a signal to affect itself or its circuit.

Redstone dust

Redstone Dust's range of activation

It **weakly powers** the opaque blocks beneath it and it points to

Redstone dust as redstone component

Redstone dust transmits power.

Main article: Redstone Dust § Redstone component

Placement

Redstone dust is placed by right-clicking with redstone dust. Redstone dust can be attached to the **top** of any opaque block, or to the **top** of glowstone, an upside-down slab or upside-down stairs. If the attachment block is removed, the redstone dust drops as an item.

When placed, redstone dust configures itself to point toward adjacent redstone dust (at the same level or one level up or down), correctly-facing redstone repeaters and redstone comparators, and power components. If there is only one such neighbor, redstone dust forms a line pointing toward and away from that one neighbor (which can cause it to point toward blocks it wouldn't normally point toward). If there are multiple such neighbors, redstone dust forms either a line, an "L", a "T", or a "+". If there are no such neighbors, redstone dust forms a large directionless dot. Redstone dust does not automatically configure itself to point toward adjacent mechanism components, it must be arranged to do so.

When two redstone dust trails are placed vertically diagonally (one block over and one up, or one over and one down), the lower dust trail appears to crawl up the side of the higher block to join the other dust. This linking can be cut by an opaque block above the lower trail, which prevents the two trails from connecting. If the higher trail is on an upside-down slab or upside-down stairs, the higher trail configures itself to point toward the lower trail (and other adjacent dust), but the lower trail (although visually) does not configure itself to point toward the higher trail (including not appearing to crawl up the side of the slab or stairs).

The directions in which redstone dust configures itself can affect whether it powers adjacent opaque blocks and mechanisms.

Activation

Redstone dust can be turned ON by any adjacent power component, redstone repeater pointing at it, or strongly-powered opaque block. Redstone dust can also be turned ON by other adjacent powered redstone dust, but the power decreases with distance from a strongly-powered block. Redstone dust transmits power up to 15 blocks away.

Redstone dust can transmit power diagonally upward to dust on an upside-down slab or upside-down stairs, but not diagonally downward from an upside-down slab or upside-down stairs.

Effect

Powered redstone dust turns ON any mechanism component it is configured to point at. It powers, weakly, an opaque block that it lies on or points to.

Redstone repeater

It **powers** the opaque block it points to

Main article: Redstone Repeater

Redstone repeater as redstone component

A redstone repeater is used to transmit power, strengthen redstone dust signals weakened by distance, delay a signal, and redirect a signal.

Placement

A redstone repeater can be attached to the **top** of any opaque block, or to the **top** of an upside-down slab or upside-down stairs. If the attachment block is removed, the redstone repeater drops as an item.

A redstone repeater is marked with an arrow pointing toward its front. The repeater reacts only to signals from the block behind it and propagates signals only to the block in front of it (in the direction of the arrow). It also has an adjustable delay that can be set from 1 to 4 ticks by right-clicking it.

Activation

A redstone repeater is turned ON by any powered component at its back and is unaffected by the powered state of any block beside, above, below, or in front of it (but see below about locking a repeater).

Effect

A powered redstone repeater turns ON redstone dust or a mechanism component in front of it, or strongly powers an opaque block in front of it. It has no effect on the blocks under, above, beside, or behind it.

A redstone repeater not only strengthens it for further transmission, it also delays it by 1 to 4 ticks. A redstone repeater also increases the duration of any pulse shorter than its delay to match the duration of its delay.

A redstone repeater can be locked by powering it from the side with another redstone repeater or with a redstone comparator. A locked repeater does not change its output state until unlocked, even if its input changes. A locked repeater displays its locked status with a bedrock bar.

Redstone comparator

It **powers** the opaque block it points to

A redstone comparator is used to compare or subtract two signals, or to measure how full a container is.

Placement

A redstone comparator can be attached to the **top** of any opaque block, or to the **top** of an upside-down slab or upside-down stairs. If the attachment block is removed, the redstone comparator drops as an item.

A redstone comparator is marked with an arrow that point toward its **front**. The comparator takes a signal from its back as its input, and outputs a signal to the block in front of it, but can also be affected by signals from its sides (see below).

	Containers and Slots						
Power Level	**4**	**27**	**54**	**9**	**3**	**5**	**—**
0	0	0	0	0	0	0	—
1	1i	1i	1i	1i	1i	1i	"13"
2	19i	1s 60i	3s 55i	42i	14i	23i	"cat"
3	37i	3s 55i	7s 46i	1s 19i	28i	46i	"blocks"

4	55i	5s 51i	11s 37i	1s 60i	42i	1s 5i	"chirp"
5	1s 10i	7s 46i	15s 28i	2s 37i	55i	1s 28i	"far"
6	1s 28i	9s 42i	19s 19i	3s 14i	1s 5i	1s 51i	"mall"
7	1s 46i	11s 37i	23s 10i	3s 55i	1s 19i	2s 10i	"mellohi"
8	2s	13s 32i	27s	4s 32i	1s 32i	2s 32i	"stal"
9	2s 19i	15s 28i	30s 55i	5s 10i	1s 46i	2s 55i	"strad"

10	2s 37i	17s 23i	34 s 46i	5s 51i	1s 60i	3s 14i	"war d"
11	2s 55i	19s 19i	38 s 37i	6s 28i	2s 10i	3s 37i	"11"
12	3s 10i	21s 14i	42 s 28i	7s 5i	2s 23i	3s 60i	"wait "
13	3s 28i	23s 10i	46 s 19i	7s 46i	2s 37i	4s 19i	"pigs tep"
14	3s 46i	25s 5i	50 s 10i	8s 23i	2s 51i	4s 42i	"othe rside " "Reli c"
15	4s	27s	54 s	9s	3s	5s	"5"

A redstone comparator has two **modes**. Right-clicking it toggles between comparison mode (front torch down/off) and subtraction mode (front torch up/on).

Activation

A redstone comparator is turned ON by a power source at its input or a power source separated by one opaque block from its input. Power sources include any powered component, a non-empty container, a container minecart on a detector rail, a command block that has run its last command successfully, a cauldron containing water, an end portal frame with an eye of ender, or a jukebox with a record. Either at its back or separated from its back by an opaque block. It is not affected by blocks beneath it or above it, but its signal strength can be modified by signals from its sides (see below).

Effect

A powered redstone comparator turns ON redstone dust, a properly-facing redstone comparator or redstone repeater, or a mechanism component in front of it; or strongly powers an opaque block in front of it – all at the same power level as its input signal (unless modified by a side signal, see below). It has no effect on blocks in other adjacent positions (including the block beneath it).

The output of a redstone comparator can be affected by a signal provided from its side by a transmission component (redstone dust, redstone repeater, or another redstone comparator only):

- In **comparison mode**, a redstone comparator propagates its input signal only if the input signal is greater than the side signal, and outputs no signal if not.
- In **subtraction mode**, a redstone comparator outputs a power level equal to the difference of the power level of the input signal minus the power level of the side signal.

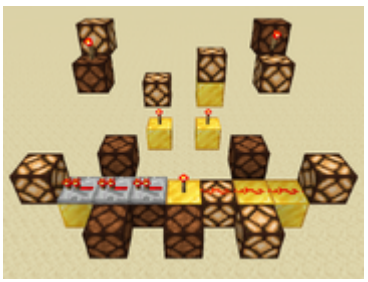
A redstone comparator that is activated by a container outputs a power level in proportion to how full the container is (rounded up, so a single item in a container produces a power level of at least 1). A container's fullness is measured by stacks: for example, a single shovel (a non-stackable item), 16 signs, or 64 sticks are all considered to be equivalent, full stacks.

The **Comparator Output Table** (right) shows the minimum stacks ("s") plus items ("i") required to produce a specific power level from a container. For example, to get power level 5 from a hopper, put 1 stack plus 28 items in the hopper. Divide items by 4 and round up for items with a stack maximum of 16. The values for the chest, dispenser, furnace and hopper apply to minecarts with those components as well (when on a detector rail).

Some blocks (such as crafting tables, enchantment tables, etc.) can hold items temporarily while the player uses the block's interface – the items are returned to the player if the player exits the interface with items still inside. Other blocks (such as beacons) only consume items. Putting items in these blocks never activates a redstone comparator.

Mechanism components

Mechanism components are blocks that react to redstone power by affecting the environment – by moving themselves or other entities, by producing light, sound, or explosions, etc.

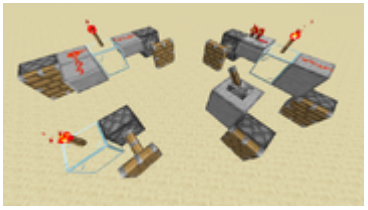


Activating a mechanism component (in this case, a redstone lamp)

Activation

Mechanism components are turned on by:

- an adjacent active power component (*Exceptions:* a redstone torch does not activate a mechanism component it is attached to, and a piston is activated only by a power component directly in front of it if the component is connected to it.)
- an adjacent powered opaque block (strongly-powered or weakly-powered)
- a powered redstone repeater or redstone comparator facing the mechanism component
- powered redstone dust configured to point toward the mechanism component (or on top of it, for opaque mechanism components); a mechanism component is *not* turned ON by adjacent powered redstone dust that is not configured to point toward it.



Activating a piston by quasi-connectivity – Note that the piston on the left is *not* powered by quasi-connectivity because the redstone dust is running *past* the block above the piston, rather than directly into it, and thus would not power a mechanism there)

Quasi-Connectivity

This feature is exclusive to *Java Edition*.

In addition to the methods above, **pistons, dispensers**, and **droppers** can also be turned ON if a block above it receives a block update (including a redstone update within two blocks of the component) and is powered by any of the above means, even without a mechanism component (e.g.; even if the block above the component is air or a transparent block). This rule is often simplified to say that

the components can be powered by blocks diagonally above or two blocks above, however other methods of activation by connectivity exist (see image to the right). This method of activation is also known as "connectivity", "piston connectivity" (as it originated with pistons), or simply "indirect power".

Activated vs. Powered – The top lamp is both *activated* (the lamp is on) and *powered* (it can power the repeater), while the bottom lamp is activated, but *not* powered.

Activated vs. Powered

For opaque mechanism components (command blocks, droppers, dispensers, note blocks and redstone lamps), it's important to make a distinction between a mechanism component being *activated* (so that it performs an action) and being *powered* (so that a redstone signal could be drawn from it by a transmission component). Any method of powering a mechanism component (such as a redstone torch underneath it) also activates it, but some activation methods (such as a redstone torch next to or above a mechanism component) does not actually power the component (following the usual rules for power components).

Activator rail

An activator rail is used to activate a minecart.

Placement

An activator rail can be attached to the **top** of any opaque block, or to the **top** of an upside-down slab or upside-down stairs. If the attachment block is removed, the activator rail drops as an item.

When placed, an activator rail configures itself to line up with adjacent rails, activator rails, powered rails, and detector rails, as well as such adjacent rails one

block above. If there are two such adjacent rails not on opposite sides, or three or more such adjacent rails, an activator rail lines up in the east-west direction. If there are no such adjacent rails, an activator rail lines up in the north-south direction. An activator rail slopes upward to match with a rail above it (when there is more than one such rail, the order of preference is: west, east, south, and north). Other configurations can be created by placing and removing various rails.

Activation

In addition to the methods above, an activator rail can also be activated by an activator rail adjacent to it that is activated. Activator rail can transmit activation up to 9 rails (the first originally-activated activator rail, and up to eight additional activator rails). Activation transmitted in this way can power only activator rails.

Effect

An activator rail affects certain minecarts passing over it. The effects vary with the type of minecart activated:

- A minecart with command block executes its command every 2 redstone ticks (5 times per second).
- A minecart with hopper is deactivated by an activated activator rail (it stops sucking up items in its path, or transferring items to containers as it passes them), and re-activated by an unactivated activator rail.
- A minecart with TNT is ignited by an active activator rail.
- Regular minecarts with an entity riding it (mob or player) eject that entity if the activator rail is active.
- Other minecarts are not affected by an activator rail.

Bell

Bells can be rung using a redstone signal.

Dispenser

A dispenser is used to automatically affect the environment by throwing items.

Activation

See Quasi-Connectivity above.

Effect

When activated, a dispenser ejects one item. If multiple slots are occupied by items, a random item is ejected.

The effects of being activated vary with ejected item:

Item	Effect
ArmorElytraHeads Shield	Equips on a player within a one-block distance (any armor, made from any material)
ArrowBottle o' Enchanting Egg Fire Charge Snowball Splash Potion	Fired in the direction the dispenser is facing, as if a player had used the item him or herself
Boat	Placed as entity (i.e., a right-clickable vehicle) onto the block in front of the

	dispenser, if it is water or air above water; otherwise dropped (see below)
Firework Rocket	Placed as entity (i.e., a flying firework) onto the block in front of the dispenser
Bone Meal	Increments the growth stage of carrots, cocoa pods, crops, melon stems, potatoes, pumpkin stems, and saplings in front of the dispenser; grows grass, dandelions, and roses, if a grass block is in front of the dispenser; grows a huge mushroom if facing a mushroom; otherwise remains unused
Bucket	Collects lava or water in front of the dispenser (replacing the empty bucket in the dispenser with a lava bucket or water bucket); otherwise dropped (see below)
Flint and Steel	Ignites the block the dispenser is facing; reduces the remaining durability of the used flint and steel

Lava Bucket Water Bucket	Places lava or water in the block in front of the dispenser (replacing the lava or water bucket in the dispenser with an empty bucket), if the block in front of the dispenser is one that the player could use a lava or water bucket on (e.g., air, flowers, grass, etc.); otherwise dropped (see below)
Minecart Minecart with Chest Minecart with Command Block Minecart with Furnace Minecart with Hopper Minecart with TNT	Placed as entity (i.e., a right-clickable vehicle) in the block in front of the dispenser, if the dispenser is in front of a type of rail; otherwise dropped (see below)
TNT	Ignites TNT on the block in front of the dispenser
Shears	Shears sheep within a one-block radius

Glowstone	If a respawn anchor is one block away, it fills the respawn anchor by 1 as if a player had right clicked with glowstone. If the respawn anchor is full, the dispenser does nothing
Others	Dropped—ejected toward the block in front of the dispenser, as if the player had used the Drop control (default Q)

Considerations

A dispenser is an opaque block, so powering it directly can activate adjacent mechanism components (including other dispensers) as well.

Door

A door is used to control or prevent the movement of mobs, items, boats, and other entities. A door may be of two types: a wooden door can be opened and closed by redstone power or by a player right-clicking on it, while an iron door can be operated only by redstone power.

Placement

A door can be attached to the **top** of most opaque blocks, or to the **top** of an upside-down slab or upside-down stairs. If the attachment block is removed, the door drops as an item.

A door is placed on the edge of the block facing the player. By default the door's hinge is positioned on the left side, but another door or block can force the hinge to the right side.

Effect

While activated, a door re-positions to the other side of its hinge, allowing movement through its former position and denying movement through its current position. When activated, any entities on the door fall off.

A door doesn't actually move (the way a piston arm or a pushed block moves), it simply disappears from one side and re-appears on another; therefore, it does not push entities as it opens.

Dragon Head

The dragon head opens and closes its mouth repeatedly like the ender dragon when placed and powered by redstone.

Dropper

A dropper is used to eject items or push them into containers (including other droppers).

Placement

A dropper can be placed so that its output faces in any direction.

Activation

See Quasi-Connectivity above.

Effect

When activated, a dropper ejects one item. If multiple slots are occupied by items, a random occupied slot is chosen for ejection.

If the dropper is facing a container, the ejected item is transferred into the container. Otherwise, the item is ejected in the direction the dropper is facing, as if the player had used the Drop control.

Considerations

A dropper is an opaque block, so powering it directly can cause adjacent mechanism components (including other droppers) to activate as well.

Fence gate

A fence gate is used to control or prevent the movement of mobs, items, boats, and other entities.

Placement

A fence gate can be placed on the **top** of most blocks. Once placed, the block beneath it may be removed without popping the fence gate.

Effect

While activated, a fence gate re-positions its two gates to either side, allowing movement through it. When activated, any entities on the fence gate falls down.

A fence gate doesn't actually move (the way a piston arm or a pushed block moves), it simply disappears from one state and re-appears in another, so it does not push entities as it opens.

Unlike a door or trapdoor, while active, a fence gate is completely non-solid (lacks a collision mask) to all entities.

Hopper

A hopper is used to move items to and from containers (including other hoppers).

Placement

A hopper can be placed so that its output faces in any direction except up.

Effect

While *not* activated, a hopper pulls items from a container above it (or item entities in the space above it) into its own slots and pushes items from its own slots into a container it is facing. Both types of transfers occur every 4 redstone ticks (0.4 seconds), and pushes are processed before pulls. A hopper always pulls items into the leftmost available slot, and pushes items from leftmost slots before rightmost slots (it does not start pushing items from the second slot before the first is empty, from the third slot before the second is empty, etc.).

While activated, a hopper does not pull items from above or push them out, but may receive items from other mechanism components such as droppers, and may have its items removed by another hopper beneath it.

Note block

A note block is used to produce a player-chosen sound.

Placement

After being placed, a note block's pitch can be adjusted over a two-octave range by right-clicking the note block, and its timbre can be adjusted by placing different blocks beneath it.

Effect

When activated, a note block produces a sound and send out block updates to all adjacent blocks. A note block must have air above it to activate.

Considerations

A note block is an opaque block, so powering it directly can cause adjacent mechanism components (including other note blocks) to activate as well.

Piston

A piston is used to move blocks or entities. A piston may be of two types: a regular piston only pushes blocks, while a sticky piston pushes and pulls blocks.

Placement

A piston has a stone base and a wooden head, and can be placed so the head faces in any direction (its front).

Activation

See Quasi-Connectivity above.

Effect

When activated, a piston pushes the block in front of its arm, and up to 11 more blocks in front of that (up to 12 blocks total). When deactivated, a regular piston pulls its arm back (leaving an air block in front of the piston), while a sticky piston pulls back both its arm and one block (leaving an air block on the other side of the pulled block).

A moving piston or block can also push an entity such as a mob or item.

Some blocks (bedrock, obsidian, end portal frame, etc.) cannot be moved by a piston. Some blocks (flowers, leaves, torches, etc.) is destroyed, but may drop items (as if destroyed by the player). For full details of how pistons interact with other blocks, see Pushing Blocks.

Slime blocks stick to blocks and make them move when adjacent blocks are moved. The 12 block limit still holds.

Considerations

When a sticky piston is activated by a pulse *shorter than 1.5 ticks*, it *pushes* the block in front of it, but *fails to pull back* the pushed block on the end of the pulse. If that sticky piston is activated again by any pulse, it *can still pull back* the block. Thus, a sticky piston running on fast pulses (for example, 1-tick pulses) pushes and pulls a block every *other* pulse.

A piston is a transparent block, so powering it directly does not cause adjacent mechanism components (including other pistons) to activate (for exceptions see Quasi-Connectivity above).

Powered rail

A powered rail is used to propel a minecart.

Placement

A powered rail can be attached to the **top** of any opaque block, or to the **top** of an upside-down slab or upside-down stairs. If the attachment block is removed, the powered rail drops as an item.

When placed, a powered rail configures itself to line up with adjacent rails, powered rails, and detector rails, as well as such adjacent rails one block up. If there are two such adjacent rails on non-opposite sides, or three or more such adjacent rails, a powered rail lines up in the east-west direction. If there are no such adjacent rails, a powered rail lines up in the north-south direction. If a rail it would line up with is one block up, a powered rail slants upward toward it (with multiple options to slant upward to, a powered rail prefers, in order: west, east, south, and north). Other configurations can be created by placing and removing various rail.

Activation

In addition to the methods above, a powered rail can also be activated by other adjacent activated powered rails. A powered rail can transmit activation up to 9 rails (the first originally-powered powered rail, and up to eight additional activated rails). Activation transmitted in this way cannot power any redstone components except powered rails, but the power change states can be detected by observers.

Effect

While activated, a powered rail boosts the speed of a minecart passing over it, or starts a minecart moving away from an adjacent solid block it is in contact with.

While not activated, it acts as a brake, reducing the speed or even stopping a minecart passing over it.

Rail

Rails and powered rails as mechanism components

A rail is used to switch the track of a minecart.

Placement

A rail can be attached to the **top** of any opaque block, or to the **top** of an upside-down slab or upside-down stairs. If the attachment block is removed, the rail drops as an item.

When placed, rail configures itself to line up with adjacent rails, powered rails, and detector rails, as well as such adjacent rails one block up. If there are two such adjacent rails on non-opposite sides, the rail curves from one to the other. If there are three or four such adjacent rails, the rail curves between two of them (when choosing which directions to curve between, a rail prefers south over north, and east over west). If there are no such adjacent rails, the rail lines up in the north-south direction. If a rail it would line up with is one block up, a rail slants upward toward it *without* curving (with multiple options to slant upward to, a rail prefers, in order: west, east, south, and north). Other configurations can be created by placing and removing various rails.

Effect

When activated, a rail in a "T" junction flips to curve the other way (activating a rail in another configuration has no effect).

Redstone lamp

A redstone lamp is used to provide light.

Activation

A redstone lamp activates normally, but takes 2 ticks to deactivate.

Effect

While activated, a redstone lamp has block light level 15 (so produces block light level 14 in all adjacent transparent spaces). An activated redstone lamp is transparent to sky light.

Considerations

A redstone lamp is an opaque block, so powering it directly can cause adjacent mechanism components (including other redstone lamps) to activate as well.

TNT

TNT is used to create an explosion.

Activation

In addition to the methods above, TNT can also be activated by fire and explosions, as well as flaming arrows.

Effect

When activated, TNT ignites and becomes primed TNT, an entity that can fall like sand or be pushed by pistons (but isn't moved by water). Primed TNT explodes 40 ticks (4 seconds) after being ignited by redstone power (10-30 ticks for TNT ignited by an explosion).

Considerations

A TNT is a transparent block, so powering it directly does not cause adjacent mechanism components (including TNTs) to activate.

Trapdoor

A trapdoor is used to control or prevent the movement of mobs, items, boats, and other entities. A trapdoor may be of two types: a wooden door can be opened and closed by redstone power or by a player right-clicking on it, while an iron door can be operated only by redstone power.

Placement

A trapdoor can be attached to the top, bottom, or the **side** of blocks. If the attachment block is removed, the trapdoor does not drop.

Effect

While activated, a trapdoor re-positions itself in a vertical state, allowing vertical movement through it. When activated, any entities on the trapdoor fall down.

Similar to a door, a trapdoor doesn't actually move (the way a piston arm or a pushed block moves), it simply disappears from one state and re-appears in another, so it does not push entities as it opens.

Command block

A command block is used to execute a server command. Command blocks can be obtained only by placing it or giving it to the player with commands.

Types

A command block have 3 types: impulse (execute a command once), chain (execute a command when triggered) and repeat (execute a command for 1 or more redstone ticks when powered)

Placement

After being placed, the player can set the command to be executed by right-clicking on the command block.

Effect

When activated, a command block executes its defined command *once*. To make a command block constantly execute its command, it must be run on a clock circuit or using a repeating command block.

Like other mechanism components, an already-activated command block does not respond to other redstone signals. To make a command block execute its defined command more than once it must be deactivated and re-activated repetitively.

Considerations

A command block is an opaque block, so powering it directly can activate adjacent mechanism components (including other command blocks) as well.

Structure block

A Structure Block is used to save and load structures. Structure blocks can be obtained only by placing it or giving it to the player with commands.

Placement

After being placed, the player can set the mode or the structure to save/load by right-clicking the structure block.

Effect

Redstone signals can be used to automate some of the structure block's functions.

Like other mechanism components, an already-activated structure block does not respond to other redstone signals. To make a command block execute its defined command more than once it must be deactivated and re-activated repetitively.

Mobile components

Minecart

A minecart is used to transport a mob or player over rails.

Behavior

The player can move a minecart by pushing against it while outside the minecart (whether the minecart is on rails or not), or by pressing the Forward control key (by default, W) while inside the minecart (only while the minecart is on rails). A minecart resting on powered rails configured to point at an adjacent opaque block is propelled away from the opaque block when the powered rails are activated. A minecart traveling over activated powered rails gets a speed boost. When a minecart passes over an activated activator rails, the entity inside it is ejected out.

Minecart with chest

A minecart with chest (a.k.a. chest minecart, storage minecart) is used to store and transport items over rails.

Behavior

A minecart with chest accepts items from a hopper and allows a hopper underneath it to pull items from it.

Minecart with command block

A minecart with command block (a.k.a. command minecart, command block minecart) is used to execute commands.

Behavior

A minecart with command block executes its command every 2 redstone ticks while on an activator rail.

Minecart with furnace

A minecart with furnace[*Java Edition only*] (a.k.a. furnace minecart, powered minecart) is used to push other minecarts over rails.

Behavior

A minecart with furnace propels itself and other minecarts without requiring powered rails.

Activation

A minecart with furnace can be activated by pressing the use key while facing the minecart with furnace and holding fuel (coal, lava, wood, etc.). It continues to move until the fuel runs out.

Minecart with hopper

A minecart with hopper (a.k.a. hopper minecart) is used to collect, transport, and distribute items over rails.

Behavior

A minecart with hopper pulls items from containers above it and push items into *hoppers* below it (the number of items transferred can depend on how long its velocity allows it to remain within reach of the containers). It also picks up items that have fallen on the rails. If a minecart with hopper passes over a powered activator rail, it stops transferring items indefinitely until it passes over an unpowered activator rail.

Minecart with TNT

Main article: Minecart with TNT

A minecart with TNT (a.k.a. TNT minecart) is used to create explosions.

Behavior

A minecart with TNT that passes over a powered activator rail explodes.

Miscellaneous components

Powering opaque blocks

An opaque block can be powered differently (in this case, a Redstone Lamp). From top to bottom:

1. **Strongly powered**: powers both Repeater and Dust.
2. **Weakly powered**: powers Repeater, but not Dust.
3. Not powered: powers neither.

Opaque blocks obstruct light and vision (with some exceptions: for example, glowstone is not considered an opaque block).

Opaque blocks are used to support redstone components and to transmit power.

Strongly powered vs. weakly powered

An opaque block is strongly powered by an active power component (except a block of redstone or a daylight detector), an active redstone repeater, or an active redstone comparator.

An opaque block is weakly powered **only** by powered redstone dust on top of it, or pointing to it.

Effect

A powered opaque block turns OFF any attached redstone torch, turns ON any adjacent redstone repeater or redstone comparator facing away from it, and activates any adjacent mechanism component.

A strongly-powered opaque block turns ON any adjacent redstone dust, including redstone dust beneath or on top of the opaque block; but a weakly-powered opaque block does not.

Use of transparent blocks

Properties of some transparent blocks.

Transparent blocks either can be seen through fully (for example, glass) or partially (for example, stairs), or allow light to pass through (for example, leaves).

Transparent blocks cannot be powered, but can be used as insulators in compact circuits. Some transparent blocks have special properties that make them useful in redstone circuits:

Fences

Main article: Fence

A redstone torch or a pressure plate can be attached to the **top** of a fence or nether brick fence.

Glass

Main article: Glass

Glass behaves as an opaque block as it does not effect how redstone components can be placed on it.

Glowstone

Main article: Glowstone

Glowstone behaves as an opaque block as it does not effect how redstone components can be placed on it.

Redstone dust on top of glowstone cannot transmit power diagonally downward to other redstone dust. Because glowstone is not opaque, it cannot power an adjacent block (including an attached trapdoor), but redstone dust on top of it can.

Slabs and Stairs

Main articles: Slab and Stairs

Any redstone component that can be attached or placed on an opaque block can also be attached or placed on an upside-down slab or upside-down stairs.

Redstone dust on top of an upside-down slab or upside-down stairs cannot transmit power diagonally downward to other redstone dust. Because slabs and stairs are not opaque, they cannot be powered by power components and cannot provide power to adjacent blocks.

Walls

Main article: Wall

A redstone torch can be attached to the **top** of a wall. Walls change states when a block is moved to or away from a wall, and this output can be detected using observers.

Redstone tips

This tutorial gives general advice for building with redstone. There are a variety of pages discussing how redstone works, mostly collected in the page Tutorials/Redstone.

Planning

The first step in building a redstone circuit is to decide what it will do and how, in general, it will operate.

- How and where will it be controlled?
    - Will the circuit be controlled by the player, by mob movement, or something else?
- What mechanism components will it control?
- What is an efficient first design?
    - Although refinement often occurs in later stages of the build, starting on a strong foot to tackle the idea will be beneficial later on. Allowing an inefficient/flawed design to manifest can hinder development.
- How will the signal be transmitted from the controls to the mechanisms?
    - Will signals need to be combined from multiple sources?

Size

When making redstone, its important to make it a reasonable size.

You shouldn't use a huge amount of space for a single contraption. Large builds take up a lot of space and are inconvinent. However, you also shouldn't try to create a fully functional redstone circuit in a tiny area. Complex redstone circuits often need plenty of space to function. For example, you cannot create a redstone computer which can perform number operations and display multiple items on a screen at once in only 1 chunk.

For the best redstone results, make your contraption as small as you can with it still functioning, but if you find you're having any troubles with that size, make it bigger. Also, make sure to never underestimate how much time, space or materials you will need. It's much better to overestimate and bring more than you need so that you have extras for next time.

Creative Mode

A complex redstone project for a Survival world can be designed in Creative mode first, before investing resources and effort in a survival world. It's handy to keep a creative-mode world handy for such laboratory work, usually a superflat with cheats on. You can also manipulate the game rules for your testing world to your liking, such as to make it permanent day or avoid mob spawning. Creative mode is great for building, because you have an infinite number of blocks, you can break blocks right away, and you can fly around to look all around your structures. You can also press F3 + N to invoke spectator mode, then fly through look inside your circuit.

Once you have finished your redstone contraption and gotten it working, look it over to make sure you understand how it's working *now*. You may be able to make some improvements here. But eventually, you go back to your survival-mode world, gather the materials, and just copy your design from creative mode. Optionally, you can

count how many of each material you used when building in creative mode, so that you will know exactly how much of a certain material to gather when in survival.

Gathering Resources

When making very large redstone contraptions, you may need farms for renewable resources. Here are some materials you may need to farm:

- Redstone: There are only a couple of *renewable* sources of redstone: killing witches (witch and raid farms are rather slow) or trading with Clerics (which is even slower). However, it is fairly plentiful in the underground, especially once you have a Fortune III pickaxe to multiply its drops.
- String: A spider farm can help if your contraption includes a lot of tripwires and/or dispensers. Piglin bartering is an alternative
- Iron ingots (Make an iron golem farm if you need a lot of hoppers or minecarts)
- Slimeballs: (For sticky pistons and/or slime blocks)
- Honey Blocks: Used in some mobile constructions. Requires farming bees.
- Stone and Cobblestone: Smooth stone for repeaters and comparators, cobble for pistons, dispensers/droppers, etc.
- Nether Quartz: Plentiful in the Nether, can be renewable farmed with bartering. With it, you get to use comparators, observers, and daylight detectors.
- Glowstone: Used for Redstone Lamps, or lighting. Can be found fairly easily in the Nether, or it can be purchased from Clerics or Wandering Traders. Witches can also drop small amounts of the dust.

Construction

It can be helpful to choose a specific set of blocks the player uses to construct circuits. Then, when the player runs into these blocks during the excavation of new rooms in the base, the player knows they are about to damage a previously-built circuit. Common choices include stone bricks, snow block, wool and concrete. (Using different colors of wool and concrete is also a great way to keep track of different circuits)

Be cautious when building circuits near water or lava. Many redstone components will "pop off" (turn into items) when washed over by liquids, and lava will destroy any items it contacts.

Be careful when building circuits to activate TNT (traps, cannons, etc.). Circuits in mid-construction can sometimes briefly power up unexpectedly, which might activate TNT. For example, placing a redstone torch on a powered block, it won't "realize" that it should be turned off until the next tick, will therefore be powered for one tick, and can briefly power another part of the circuit during that one tick. Placing TNT after the rest of the circuit is complete will help to avoid such problems and the destruction of the device itself. This also applies to any other features of the circuit that may be accidentally activated with such actions (e.g., activating a dispenser before the circuit is ready). Temporarily placing a redstone lamp or piston can quickly test whether a given space is powered.

Color coding

This is a simple yet very effective tip, especially if you create redstone contraptions that have many different parts to them, such as comparator clocks mixed with other redstone items. It is best to use different colored wool, concrete, or terracotta for different parts of the circuit. If you build all the redstone using the same building block, for example, out of dirt(which you shouldn't be using for redstone anyway if you are in Survival because an Enderman may break it), soon you may completely

forget how your redstone works due to not remembering where each circuit goes. Furthermore, this is important if you want to show off the redstone contraptions on YouTube, so people can copy your design in their Minecraft world or you want to be able to go back to your project and understand what parts of the circuit perform what function.

If you don't want to use wool, concrete, or terracotta, you can find other blocks that are different colors from each other. For example, you can use stone variants and wood-related blocks. However, try not to use blocks of similar color, such as a block of coal and black concrete on 2 different parts of a circuit. You can also use different colors or variants to mark switch-supporting blocks (input) or potential output locations for a circuit, e,g, if most of the circuit is built on stone brick, you might choose carved stone brick for switch blocks, polished granite for output locations, and diorite for mobile blocks. Glass (which can also be tinted) can be used to display the workings of a circuit; it can also make sure that lava and water (e.g., in a cobblestone generator) are visible but not open to unwary players.

All this may take extra time and effort, but the benefits are worthwhile.

Troubleshooting

When the circuit isn't working the way it should, take a look at it and try to find the problem. Work through the circuit and test various inputs to find where a signal is "dropped" or gained inadvertently.

- What part of the wiring actually is not behaving as expected?. Unexpected output behavior is usually only a symptom, where the actual problem resides somewhere in the wiring.
- Are signals out of sync due to timing issues?
- Are parts of the circuit activating when they shouldn't be? Maybe accidentally "crossed wires" are allowing a signal from one part of the

circuit to activate another part of the circuit, or a repeater's output is being cycled back into its input.

- Has the wiring been damaged by pistons pushing the wiring around?
- Trying to draw power from a weakly-powered block? Maybe a redstone repeater is needed to either strongly-power the block or to pull power out of it.
- Trying to transmit power through a non-opaque block? Replace it with an opaque block, or go around it.
- Was a short-circuit created and a redstone torch that should be powered is now burned out? Fix the short-circuit and update the torch to get things going again.
- Are pistons, dispensers, or droppers being indirectly powered when they shouldn't be?
- Is the circuit based on a tutorial from an older version of Minecraft which no longer works in the current version?

Refining

Once the circuit is working, consider if it can be improved (without breaking it).

- Can the circuit be faster?
    - Shorter delays and pulses can make most circuits faster.
    - Reducing the number of components or distance a signal has to travel through can speed up the circuit.
- Can the circuit be smaller?
    - Can fewer blocks be used?
    - Is there a more efficient way of doing the same thing?
    - Can the redstone dust lines be shortened?
    - Are unnecessary components used?
- Can the circuit be more robust?

        - Will the circuit still work when activated by a very short pulse?
        - Will the circuit still work when activated and deactivated rapidly in succession?
        - If either of the above are a problem, can this be fixed by filtering the input?
        - Can the circuit be damaged if unloaded? Be careful with constantly running clocks.
- Did an update create the opportunity for a better circuit? (e.g., comparators, locking repeaters, observers, etc.)
- Can the circuit be quieter?
    - Fewer sound-producing blocks (e.g. pistons, dispensers and droppers, doors, trapdoors, fence gates, and note blocks) will make your device more stealthy around other players.
- Can any lag be reduced? Machines with many redstone components frequently changing state can cause light, sound, particle, or update lag.
    - Hoppers and hopper minecarts especially try to do several things every tick (accept items pushed into them, push items into other containers, check for item entities above them).
    - Redstone torches and redstone lamps change their light level when they change state. Light changes can cause block light updates in *hundreds* of block tiles around each component. Concealing the component in opaque blocks or placing permanent light sources (torches, glowstone, etc.) nearby can reduce light updates. While light updates no longer create lag on since they are on a separate thread, excessive light updates can light suppress which lags other actions such as chunkloading.

- Several redstone components produce particles (redstone torches, redstone dust, but especially fireworks fired from dispensers). Too many particles may overload *Minecraft's* particle rendering and then some particles may fail to render until old particles have disappeared.
- Every time a block is moved by a piston, the piston makes many checks for movement and it produces block updates in its neighbors, so moving too many blocks at once can produce lag.

Redstone circuits

A **redstone circuit** is a contraption that activates or controls mechanisms. Circuits can act in response to player or entity/mob activation, continuously on a loop, or in response to non-player activity (mob movement, item drops, plant growth, etc).

A useful distinction can be made between a **circuit** performing operations on signals (generating, modifying, combining, etc.), and a **mechanism** manipulating the environment (moving blocks, opening doors, changing the light level, producing sound, etc.). Making this distinction lets us talk about the various circuits separately, and let players choose whichever circuits are useful for their purposes. The machines controlled by redstone circuits can range from simple devices such as automatic doors and light switches to complex devices such as elevators, automatic farms, or even in-game computers. However, *this* article provides only an overview of redstone *circuits* as above. These can be used to control simple mechanisms, or combined as parts of a larger build. Each circuit type on this page has links to its own page, which provides greater detail about them and give schematics for multiple variations of each.

Before working with any but the most basic Redstone circuits, an understanding of some basic concepts is required: "power", "signal strength", "redstone ticks", and "block updates". Other relevant articles:

- The Redstone mechanics article provides more information on these concepts.
- The Redstone components article adds a list and description of all blocks which interact with redstone power.
- The Mechanisms tutorial complements this article with an assortment of mechanism designs using circuits described here.
- The Redstone tips tutorial gives general advice about building.

Describing Circuits

Most circuits are described using Schematic diagrams; some of these require multiple images to show one or two layers per image. See the Help:Schematic page for details on how various blocks and components are represented.

Size

The wiki describes circuit size (the volume of the rectangular solid it occupies) with the notation of *shorter width* × *longer width* × *height*, including support/floor blocks, but not including inputs/outputs.

Another method used for describing circuit size in the *Minecraft* community is to ignore non-Redstone blocks simply used for support (for example, blocks under Redstone dust or repeaters). However, this method is unable to distinguish between flat and 1-high circuits, as well as some other circuit differences.

Sometimes it is convenient to compare circuits simply by the area of their footprint (e.g., 3×4 for a circuit three-block wide by four blocks long), or by a single dimension

important in a particular context (e.g., length in a sequence of sub-circuits, height in a confined space, etc.).

Features

Several features may be considered desirable design goals:

1-high

A structure is 1-high (aka "1-tall") if its vertical dimension is one block high (meaning it cannot have any redstone components that require support blocks below them, such as redstone dust or repeaters). Also see flat.

1-wide

A structure is 1-wide if at least one of its horizontal dimensions is exactly one block wide.

Flat

A structure is flat if it generally can be laid out on the ground with no components above another (support blocks under redstone components are okay). Flat structures are often easier for beginners to understand and build, and fit nicely under floors or on top of roofs. Also see 1-high.

Flush

A structure is flush if it doesn't extend beyond a flat wall, floor, or ceiling and can still provide utility to the other side, though redstone mechanisms may be visible in the wall. Flush is a desirable design goal for piston-extenders, piston doors, etc. Also see hipster and seamless.

Hipster

A structure is hipster if it is initially hidden behind a flat wall, floor, or ceiling and can still provide utility to the other side. See also flush and seamless.

Instant

A structure is instant if its output responds immediately to its input (a circuit delay of 0 ticks).

Seamless

A structure is seamless if no redstone components are visible both before and after it completes its task (but it's okay if some are visible during operation). Seamless is a desirable design goal for piston-extenders, piston doors, etc. See also flush and hipster.

Silent

A structure is silent if it makes no noise (such as from piston movement, dispenser/dropper triggering when empty, etc.). Silent structures are desirable for traps or peaceful homes.

Stackable

A structure is stackable if it can be placed *directly* on top of other copies of itself, and they all can be controlled as a single unit. Also see tileable.

Expandable

A structure is Expandable if it can be placed *directly* next to other copies of itself, and they all can be controlled as a single unit. Also see tileable.

Tileable

A structure is tileable if it can be placed *directly* next to or on top of other copies of itself, and each copy can still be controlled independently. Also see stackable.

Structures might be described as "2-wide tileable" (tileable every two spaces in one dimension), or "2×4 tileable" (tileable in two directions), etc. Some structures might be described as "alternating tileable", meaning they can be placed next to each other if every other one is flipped or a slightly different design.

Other design goals may include reducing the delay a sub-circuit adds to a larger circuit, reducing the use of resource-expensive components (redstone, nether

quartz, etc.), and re-arranging or redesigning a circuit to make it as small as possible.

Some components are not available before a player has access to the Nether, which limits the designs available. In particular, redstone comparators, observers and daylight detectors require nether quartz, which is available only from the Nether. Additionally, redstone lamps require glowstone, which is occasionally available from trading or witches, but is much more plentiful in the Nether.

Circuit types

Although the number of ways to construct circuits is endless, certain patterns of construction occur repeatedly. The following sections attempt to categorize the circuits that have proven useful to the *Minecraft* community, while the main articles describe the specific circuits that fall into those categories.

Some of these circuits might be used by themselves for simple control of mechanisms, but frequently the player needs to combine them into more complex circuits to meet the needs of a mechanism.

Transmission circuit

Main article: Transmission circuit

Some aspects of signal transmission can be helpful to understand: transmission types, vertical transmission, repeaters, and diodes.

Vertical transmission

Transmitting signals upward

Transmitting signals downward

Examples of two-way vertical ladders in Bedrock Edition

Although horizontal signal transmission is straightforward, vertical transmission involves options and trade-offs.

- *Redstone staircases:* The simplest way to transmit signals vertically is by placing redstone dust on blocks diagonally upward, either in a straight staircase of blocks, in a 2×2 spiral of blocks, or in another similar variation. Redstone staircases can transmit signals both upward and downward but can take up much space and require repeaters every 15 blocks.
- *Redstone ladders:* Because glowstone, top slabs, glass, and upside-down stairs can support redstone dust but don't cut redstone dust, signals can be transmitted vertically (upward only) by alternating these blocks in a 2×1 "ladder". Redstone ladders take up less space than redstone staircases, but also require repeaters every 15 blocks. In *Bedrock Edition*, glass and pistons can be used to create two-way vertical ladders that transmit signals both upward and downward

(glowstone, hoppers, and slabs still allow the dust to power upward but not downward).

- *Torch towers and torch ladders:* A redstone torch can power a block above it, or redstone dust beneath it, allowing vertical transmission both upward and downward (different designs are required for each). Because it takes each torch a little time to change state, a torch tower can introduce some delay into a circuit, but no repeaters are necessary. However, every torch inverts the redstone signal (i.e. changes it from powered to unpowered), so having an even number of torches is required.
- *Observer towers:* An observer can power a block of a redstone circuit above or below it, allowing vertical transmission both upward and downward. Placing blocks that can be activated, such as redstone dust, noteblocks, or doors, both above and below it creates a state change when the observer is looking downward or downward when the observer is looking upward. Repeating this pattern means that updates are chained.
- *Daylight detector exploiting:* You can use daylight detectors to send a Redstone signal downward in 1 tick, but the path needs to be unobstructed by anything. You need to have a piston push a block over the sensor. It detects the change in light and emits a Redstone pulse. This design is extendable upward as far as you want, but you need to have the original hole open to sunlight. It also works only during the day, because it uses shadows to activate.
- *Bubble columns:* An observer can be used to detect the block update that occurs when a water source changes to a bubble column (or vice versa). When swapping the block below a column of water sources to soul sand or a magma block from some other block, the entire column immediately changes to bubble column blocks. This can be

used to quickly transmit a redstone signal upward to an observer facing the top water source/bubble column block.

- *Wall updating*: A setup that can carry a pulse signal downwards across any distance involves walls of any type of stone, a piston, and an observer. When a wall block has a solid block on two opposing sides and non-solid blocks (e.g., air) on the other two sides, it takes a flat shape. This is vertically repeatable up to any height. However, when a wall/solid block is placed into one of the two air blocks around a flat wall, the flat wall block *and every flat wall block below it* are updated to a different version of the wall with a column in the middle. This update is instant and can be detected by an observer watching any flat wall in the tower. The update can be made repeatable by having a regular piston face the flat wall at the top of the tower, since the piston head also triggers the wall update.

Repeater

To "repeat" a signal means to boost it back up to full strength. The easiest way to do this is with a redstone repeater. Variations include:

- *Instant repeater:* Repeats a solid signal without the delay introduced by a redstone repeater.
- *Two-way repeater:* Repeats a signal in both directions.

Diode

A "diode" is a one-way circuit that allows a signal to travel in one direction. It is used to protect another circuit from the chance of a signal trying to enter through the output, which could incorrectly change the circuit's state or interfere with its timing. It is also used in a compact circuit to keep one part of the circuit from interfering with another. Common choices for a diode include a redstone repeater

or a height elevation to glowstone or a top slab, which does not transmit a signal back down.

Many circuits are already one-way simply because their output comes from a block that can't take input. For example, a signal cannot be pushed back into a circuit through a redstone torch except through the block it's attached to.

Logic circuit

Main article: Logic circuit

It's sometimes necessary to check signals against each other and output a signal only when the inputs meet some criteria. A circuit that performs this function is known as a **logic gate** (a "gate" that allows signals through only if the logic is satisfied).

In electronic or programming diagrams, logic gates are typically shown as if they were individual devices; However, when building redstone devices in *Minecraft*, all logic gates are formed from multiple blocks and components, which interact to produce the desired results.

NOT gate

A NOT gate (aka "inverter") is on if its input is off. The simplest NOT gate is an input block with a redstone torch attached.

OR gate

An OR gate is on if *any* of its inputs are on. The simplest OR gate is to feed multiple signals into a single block or redstone wire.

NOR gate

A NOR gate is on only if *none* of its inputs are on. The simplest NOR gate is to feed multiple signals into a block with a redstone torch attached.

AND gate

An AND gate is on only if *all* of its inputs are on.

NAND gate

A NAND gate is on if *any* of its inputs are off.

XOR gate

An XOR gate is on if its inputs are *different*.

XNOR gate

An XNOR gate is on if its inputs are *equal*.

IMPLY gate

An IMPLY gate is on unless the first input is on and the second input is off.

A B					Question Answered
A AND B	ON	off	off	off	Is A and B on?
NOT (A IMPLIES B)	off	ON	off	off	Is A on and B off?
NOT (B IMPLIES A)	off	off	ON	off	Is B on and A off?
A NOR B	off	off	off	ON	Are both inputs off?
A	ON	ON	off	off	Is A on?
A XOR B	off	ON	ON	off	Are the inputs different?

NOT A	off	off	ON	ON	Is A off?
A XNOR B	ON	off	off	ON	Are the inputs the same?
B	ON	off	ON	off	Is B on?
NOT B	off	ON	off	ON	Is B off?
A NAND B	off	ON	ON	ON	Is either input off?
A IMPLIES B	ON	off	ON	ON	If A is on, is B also on?
B IMPLIES A	ON	ON	off	ON	If B is on, is A also on?
A OR B	ON	ON	ON	off	Is either input on?

Pulse circuit

Main article: Pulse circuit

Some circuits require specific pulses; other circuits use pulse duration to convey information. Pulse circuits manage these requirements.

A circuit that is stable in one output state and unstable in the other is known as a monostable circuit.[note 1] Many pulse circuits are monostable because their OFF state is stable, but their ON state soon reverts to OFF.

Pulse generator

A pulse generator produces a pulse of a specific duration.

Pulse limiter

A pulse limiter (aka pulse shortener) reduces the duration of pulses that are too long.

Pulse extender

A pulse extender (aka pulse sustainer, pulse lengthener) increases the duration of pulses that are too short.

Pulse multiplier

A pulse multiplier outputs multiple pulses for every input pulse (it multiplies the number of pulses).

Pulse divider

A pulse divider (aka pulse counter) outputs a signal only after a certain number of pulses have been detected through the input (the number of pulses is indicative of the number of loops).

Edge detector

An edge detector reacts to either a redstone signal changing from OFF to ON (a "rising edge" detector), from ON to OFF (a "falling edge" detector), or switching between ON and OFF in either order(a "dual edge" detector).

Pulse length detector

A pulse length detector reacts only to pulses in a certain range of durations (often only to pulses of one specific duration).

Clock circuit

Main article: Clock circuit

A clock circuit is a pulse generator that produces a loop of specific pulses repeatedly. Some are designed to run forever, while others can be stopped and started.

A simple clock with only two states of equal duration is named for the duration of its ON state (e.g., for example, a clock that alternates between a 5-tick ON state and a

5-tick OFF state is called a 5-clock) while others are usually named for their period (the time it takes for the clock to return to its original state; for example, a "1-minute clock" might produce a 1-tick pulse every 60 seconds).

Observer clock 1

A repeating clock made with Observers and Pistons (an Observer looking at a piston).

Observer clock 2

A repeating clock made with two Observers with their faces facing each other.

Repeater clock

A repeater clock consists of a loop of repeaters (usually either redstone repeaters or redstone torches) with occasional dust or blocks to draw off the appropriate pulses.

Hopper clock

A hopper clock produces timed pulses by moving items back and forth between 2 hoppers feeding into each other and taking a redstone output with comparators.

Piston clock

A piston clock produces a loop of pulses by passing a block back and forth (or around, with many pistons) and drawing off a redstone pulse when the block is in a certain location.

Comparator clock

The clock of short or moderate cycle length utilizing comparator's subtraction or signal fading feature. Clocks can also be built using daylight sensors, minecarts, boats, water flow, item despawn, etc.

Memory circuit

Main article: Memory circuit

Unlike a logic circuit whose state always reflects its current inputs, a memory circuit's output depends not on the current state of its inputs, but on the *history* of its inputs. This allows a memory circuit to "remember" what state it should be in, until told to remember something else. There are five basic types of memory circuits. (A few circuits combine two different types.)

RS latch

An RS latch has two inputs, one to set the output on and another to reset the output back to off. An RS latch built from NOR gates is known as an "RS NOR latch", which is the oldest and most common memory circuit in *Minecraft*.

T flip-flop

A T flip-flop is used to toggle a signal (like a lever). It has one input, which toggles the output between on and off.

Gated D latch

A gated D latch has a "data" input and a "clock" input. When the clock input turns on, it sets the output to equal its data input. Not to be confused with a D flip-flop, which sets the output equal to its data input on a clock rising transition.

JK latch

A JK latch has two inputs, one to set the output on and another to reset the output back to off (like an RS latch), but when both turn on simultaneously it toggles the output between on and off (like a T flip-flop).

Counter

Unlike T flip-flops and RS latches, which can hold two states (ON or OFF), a counter can be designed to hold a greater number of states.

Many other memory circuits are possible.

Piston Circuits

Main article: Piston circuits

Pistons have allowed players to design circuits that are smaller and/or faster than the standard, redstone-only counterparts. An understanding of standard **redstone circuits** is helpful, as this tutorial is focused on the circuit design rather than the function. The main components here are sticky pistons, redstone wire, repeaters, and redstone torches. Regular pistons can also see use, especially combined with gravity blocks.

There are several benefits of piston circuitry:

- Neither repeaters nor pistons 'burn out', unlike redstone torches.
- Piston circuits are often (not always) smaller and/or faster than their redstone counterparts. This allows building devices such as fast clocks and "instant" signal transmission.
- Pistons' ability to move blocks within the world makes them a natural for memory circuits, as well as the obvious doorways and switchable bridges. With slime or honey blocks involved, entire structures can "get up and move" (see also the Flying Machines tutorial).
- Piston circuits can sharply reduce the use of redstone in favor of wood, stone, and iron.

Miscellaneous circuits

Main article: Miscellaneous circuits

These circuits aren't generally needed for redstone projects, but might find use in complex projects, proofs of concept, and thought experiments. Some examples:

Multiplexers and relays

A multiplexer is an advanced form of logic gate that chooses which of two inputs to let through as output based on an additional input (for example, if input A is ON then output input B, otherwise output input C). The reverse of this is a relay,

which copies a data input to one of two outputs, depending on whether the additional input is ON or OFF.

Randomizers

Main article: Tutorials/Randomizers

A randomizer produces output signals unpredictably. Randomizers can be designed to produce a pulse at random intervals, or to randomize which of multiple outputs are turned ON (such as random number generators, or RNGs). Some randomizers use the random nature of *Minecraft* (such as cactus growth or dispenser slot selection), while others produce pseudo-randomness algorithmically.

Multi-bit circuits

Multi-bit circuits treat their input lines as a single multi-bit value (something other than zero and one) and perform an operation on them all at once. With such circuits, possibly combined with arrays of memory circuits, it's possible to build calculators, digital clocks, and even basic computers inside *Minecraft*.

Block update detectors

Main article: Tutorials/Block update detector

Main article: Tutorials/Comparator update detector

A block update detector (BUD, or BUD switch) is a circuit that reacts to a block changing its state (for example, stone being mined, water changing to ice, a pumpkin growing next to a pumpkin stem, etc.). BUDs react by producing a pulse, while T-BUDs (toggleable BUDs) react by toggling their output state. These are generally based on subtle quirks or glitches in device behavior; current circuits most often depend on pistons. As of *Java Edition* 1.11, many of the functions of BUDs were condensed into the observer, however, a BUD circuit can also detect other changes undetectable by observers, like a furnace finishing

smelting or something being crafted in a crafting table. The addition of this was made to move toward feature parity with *Bedrock Edition* versions.

Advanced redstone circuits

< Tutorials

SIGN IN TO EDIT

This article needs cleanup to comply with the style guide. [discuss]

Please help improve this page. The talk page may contain suggestions.

Reason: Seems a bit chatty and very unformatted (messy); also there is too much whitespace.

It has been suggested that this page be split into *Arithmetic logic*, *Passcode locks*, and *Base changers*. [discuss]

If this split may potentially be controversial, do *not* split until a consensus has been reached.

Reason: *There's already a tutorials section for this and it has become a disorganized mess. The pages above are only a suggestion.*

This article uses MCRedstoneSim schematics.

These should be converted to use `{{schematic}}` if possible.

Advanced redstone circuits encompass mechanisms that require complicated redstone circuitry. They are usually composed of many simpler components, such as logic gates. For simpler mechanisms, see electronic mechanisms, wired traps, and redstone.

Computers

> *Main article: Tutorials/Redstone computers*

In *Minecraft*, several in-game systems can usefully perform information processing. These systems include water, sand, minecarts, pistons, and redstone. Of all these systems, only redstone was specifically added for its ability to manipulate information, in the form of redstone signals.

Redstone, like electricity, has high reliability and high switching-speeds, which has seen it overtake the other mechanical systems as the high-tech of *Minecraft*, just as electricity overtook the various mechanics such as pneumatics to become the high-tech of our world.

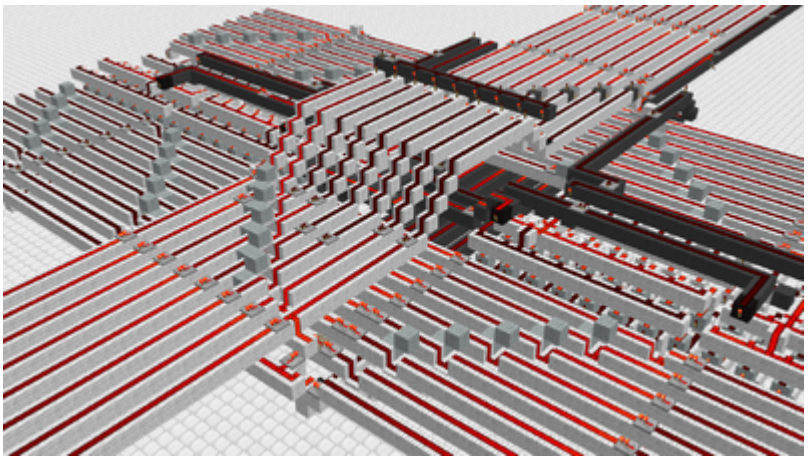
In both modern digital electronics and redstone engineering, the construction of complex information processing elements is simplified using multiple layers of abstraction.

The first layer is that of atomic components; redstone/redstone torches/redstone repeaters/blocks, pistons, buttons, levers and pressure plates are all capable of affecting redstone signals.

The second layer is binary logic gates; these are composite devices, possessing a very limited internal state and usually operating on between one and three bits.

The third layer is high-level components, made by combining logic gates. These devices operate on patterns of bits, often abstracting them into a more humanly comprehensible encoding like natural numbers. Such devices include mathematical adders, combination locks, memory-registers, etc.

In the fourth and final layer, a key set of components are combined to create functional computer systems which can process any arbitrary data, often without user oversight.



An 8-bit register page would be in the third layer of component abstraction

Converters

These circuits simply convert inputs of a given format to another format. Converters include Binary to BCD, Binary to Octal, Binary to Hex, BCD to 7-Segment, etc.

Piston mask demultiplexer

You can understand this design as a combination of AND gates.

Demultiplexer is a circuit that uses the following logic:

Output 0 = (~bit2) & (~bit1) & (~bit0)

Output 1 = (~bit2) & (~bit1) & (bit0)

The most obvious way to implement a demultiplexer would be to put a whole bunch of logic gates and connect them together, but even with 3 or 4 bits it turns into a mess.

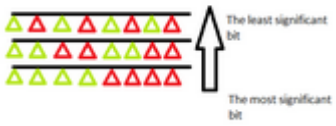
If you look at the binary numbers table, you can notice a pattern.

N	**Bit 2**	**Bit 1**	**Bit 0**
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1

6	1	1	0
7	1	1	1

If the number of bits is Q, the most significant bit reverses every Q/2 numbers, the next bit reverses every Q/4 numbers an so on until we get to the Qth bit.

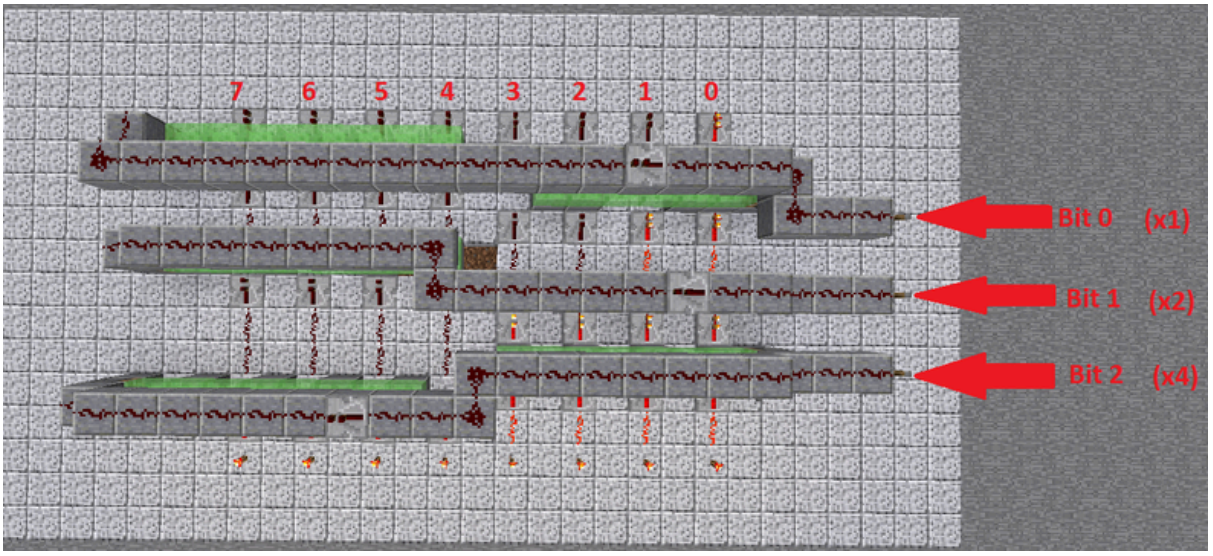
Therefore, we should make a circuit that looks like this:

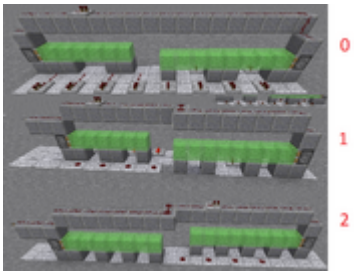


Where the green triangles are non-reversing and red triangles are reversing. The black lines are imaginary AND gates.

We can easily implement this using 3 "punch cards" that consist of solid blocks and air. The "punch cards" or the masks are being moved by pistons with slime blocks.

So the signal is only being propagated if all three layers of masks align in a specific way.



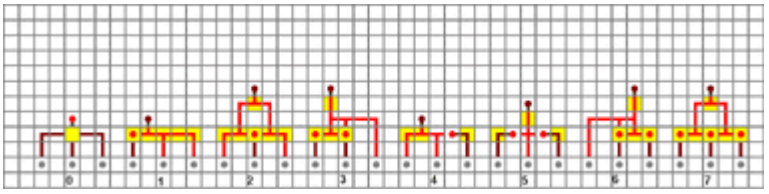


Open the picture to see the layers.

As you can see, this system is very compact and comprehensible.

You can use this in reverse as well (not as a multiplexer, but if you reverse the repeaters the signal from every ex-outptut (0–7) will only propagate if it matches the current state of the demultiplexer, so it works like "Output3 = (Input3) AND (Demux=011)").

Binary to 1-of-8



3-bit Binary to 1-of-8 gates.

A series of gates that converts a 3-bit binary input to a single active line out of many. They are useful in many ways as they are compact, 5×5×3 at the largest.

As there are many lines combined using implicit-ORs, you have to place diodes before each input into a circuit to keep signals from feeding back into other inputs.

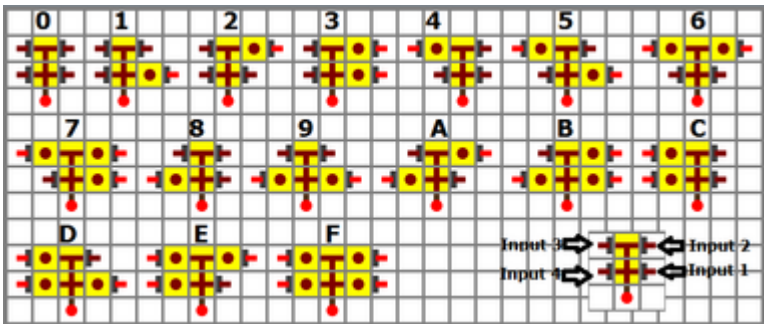
Requirements for each output line (excluding separating diodes):

Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Size	5×3×2	5×3×3	5×5×3	5×5×3	5×3×3	5×4×3	5×5×3	5×5×3
Torches	1	2	2	3	2	3	3	4
Redstone	7	7	12	10	7	7	10	10

Binary to 1-of-16 or 1-of-10

A series of gates that converts a 4-bit binary input to a single active line out of many (e.g. 0-9 if the input is decimal or 0-F if the input is hexadecimal). They are useful in many ways as they are compact, 3×5×2 at the largest.

As there are many lines combined using implicit-ORs, you have to place diodes before each input into a circuit to keep signals from feeding back into other inputs.



4-bit Binary to 1-of-16 gates.

Requirements for each output line (excluding separating diodes):

Number	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Size	3×3×2	3×4×2	3×4×2	3×4×2	3×4×2	3×5×2	3×5×2	3×5×2	3×4×2	3×5×2	3×5×2	3×5×2	3×5×2	3×5×2	3×5×2	3×5×2
Torches	1	2	2	3	2	3	3	4	2	3	3	4	3	4	4	5
Redstone	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

1-of-16 to Binary

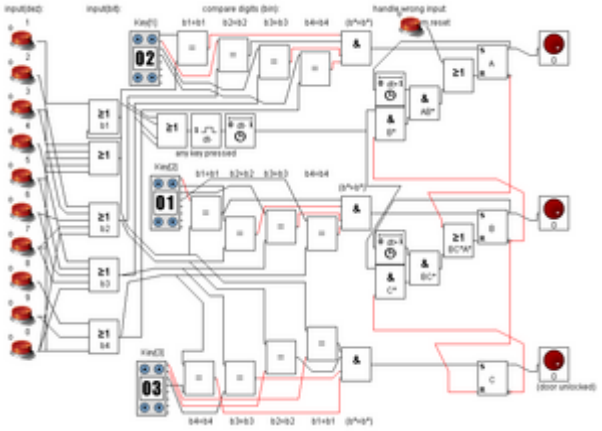
You also can convert a 1-of-16 signal to a 4-bit binary number. You only need 4 OR gates, with 8 inputs each. These have to be isolating ORs to prevent signals from feeding back into other inputs.

For every output line, make an OR gate with the inputs wired to the input lines where there is a '1' in the table below.

Number	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

4-bit	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
3-bit	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
2-bit	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1-bit	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Example



Logic for a 3-digit key log, with digits 0-9. It's order-sensitive

The example on the right uses ORs (>=1), XNORs (=), RS NOR latches (SR) and some delays (dt*). For the XNORs I would prefer the C-design.

The example on the right uses a 4-bit design, so you can handle a hexadecimal key. So you can use 15 various digits, [1,F] or [0,E]. You only can use 15, because the state $(0)_{16} = (0000)_2$ won't activate the system. If you want to handle 16 states, you edit the logic, to interact for a 5-bit input, where the 5th bit represents the $(0)_{16}$ state.

In the following we'll use $(0)_{16} = (1111)_2$. And for [1,9] the MUX-table upon. So the key uses decimal digits. Therefore, we have to mux the used buttons to binary data.

Here look through the first two columns. The first represents the input-digit in (hexa)decimal, the second represents the input-digit in binary code. Here you can add also buttons for [A,E], but I disclaimed them preferring a better arranging. The /b1\-box outputs the first bit, the /b2\-box the second, and so on.

Now you see Key[i] with i=1..3, here you set the key you want to use. The first output of them is the 1-bit, the second the 2-bit and so on. You can set your key here with levers in binary-encryption. Use here the MUX-table upon, and for $(0)_h := (1111)_2$. If we enter the first digit, we have to compare the bits by pairs (b1=b1, b2=b2, b3=b3, b4=b4). If every comparison is correct, we set the state, that the first digit is correct.

Therefore, we combine (((b1=b1 & b2=b2) & b3=b3) & b4=b4) =: (b*=b*). In *Minecraft* we have to use four ANDs like the left handside. Now we save the status to the RS-latch /A\. The comparison works the same way for Key[2], and Key[3].

Now we have to make sure, that the state will be erased, if the following digit is wrong. Therefore, we handle a key-press-event (--/*b1 OR b2 OR b3 OR b4*\--/*dt*-\--/*dt*-\--). Search the diagram for the three blocks near "dt-". Here we look, if any key is pressed, and we forward the event with a minor delay. For resetting /A\, if the second digit is wrong, we combine (key pressed) & (not B). It means: any key is pressed and the second digit of the key is entered false. Therewith /A\ will be not reset, if we enter the first digit, /A\ only should be reset, if /A\ is already active. So we combine (B* & A) =: (AB*). /AB*\ now resets the memory-cell /A\, if the second digit is entered false and the first key has been already entered. The major delay /dt+\ must be used, because /A\ resets itself, if we press the digit-button too long. To prevent this failure for a little bit, we use the delay /dt+\. The OR after /AB*\ is used, for manually resetting, i.e. by a pressure plate.

Now we copy the whole reset-circuit for Key[2]. The only changes are, that the manually reset comes from (not A) and the auto-reset (wrong digit after), comes from

(C). The manual reset from A prevents B to be activated, if the first digit is not entered. So this line makes sure, that our key is order-sensitive.

The question is, why we use the minor-delay-blocks /dt-\. Visualize /A\ is on. Now we enter a correct second digit. So B will be on, and (not B) is off. But while (not B) is still on, the key-pressed-event is working yet, so A will be reset, but it shouldn't. With the /dt-\-blocks, we give /B\ the chance to act, before key-pressed-event is activated.

For /C\ the reset-event is only the manual-reset-line, from B. So it is prevented to be activated, before /B\ is true. And it will be deactivated, when a pressure-plate resets /A\ and /B\.

Pros

- You can change the key in every digit, without changing the circuit itself.
- You can extend the key by any amount of digits, by copying the comparison-circuit. Dependencies from previous output only.
- You can decrease the amount of digits by one by setting any digit (except the last) to $(0000)_2$.
- You can open the door permanently by setting the last digit to $(0000)_2$

Cons

- The bar to set the key will be get the bigger, the longer the key you want to be. The hard-coded key-setting is a compromise for a pretty smaller circuit, when using not too long keys. If you want to use very long keys, you also should softcode the key-setting. But mention, in fact the key-setting-input will be very small, but the circuit will be much more bigger, than using hard-coded key-setting.

Not really a con: in this circuit the following happens with maybe the code 311: 3 pressed, A activated; 1 pressed, B activated, C activated. To prevent this, only set a

delay with a repeater between (not A) and (reset B). So the following won't be activated with the actual digit.

If you fix this, the circuit will have the following skill, depending on key-length. (digit = 2^n-1, possibilities: $digit^{Length}$)

Length	1	2	3	4	5
2 bit	3	9	27	81	243
3 bit	7	49	343	2.401	16.807
4 bit	15	225	3.375	50.625	759.375
5 bit	31	961	29.791	923.521	28.629.151

Miscellaneous

Combination locks

Main article: Tutorials/Combination locks

Combination locks are a type of redstone circuit. They generally have a number of components which must be set in the right combination in order to activate something such as a door. Combination locks can be very useful in creating adventure maps. Note that if you are playing in survival multiplayer, other players will still be able to break into the mechanism and cause it to activate without knowing the password.

Sorting device

This is a device which sorts the inputs, putting 1s at the bottom and 0s at the top, in effect counting how many 1s and how many 0s there are. It is designed so that it is easily expandable, as shown in the diagram. The 5×5 center square is tileable. The inputs are at the bottom and right and the outputs are at the top and left

Truth table for a three-bit sorting device:

A	B	C	1	2	3
0	0	0	0	0	0
1	0	0	1	0	0
0	1	0	1	0	0
0	0	1	1	0	0
1	1	0	1	1	0
0	1	1	1	1	0
1	0	1	1	1	0
1	1	1	1	1	1

Timer

Timers can detect the time difference between the first input and the second.

A timer. The extra repeater at the bottom is to compensate for the delay of the upper repeaters.Example of a timer in action. This one determines the time difference between the input and output of a 2-tick repeater.

The amount of time can be determined by how far the signal travels. For example, if 5 of the locked repeaters are powered, it means the time difference was 0.4-0.5 seconds, ignoring lag. If the time difference is exactly 0.4 seconds, 4 repeaters will be powered.

The repeaters that will lock can be set to different delays. For example, if they are set to 4 ticks and the first 3 are active, it means the time difference was 0.8-1.2 seconds. You can even have a mix, which can be handy if you know what the range is likely to be. However, you will need to be careful when reading these timers.

If you are measuring higher scales, the second signal might not reach all of the repeaters. You will need repeaters to replenish the signal.

A section of the timer that replenishes the signal. Since the upper repeater has a delay, another repeater is required in the lower section.

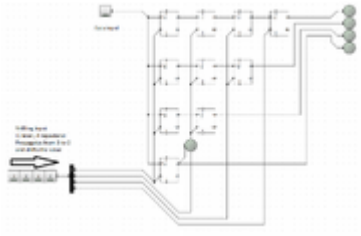
If the signals are short times (like if you are using observers), you may not have time to read the data.

You can also measure how long a signal lasts.

Please note the following when making a duration timer:

- Because of the delay that the redstone torch adds, the delay of the initial repeater, the one that stays unlocked, must be increased to 2 ticks.
- The data from the timer will be preserved.
- Because the repeaters will still be powered when the timer is used again, the circuit must be obstructed between uses in order to unlock the repeaters. To do this mine the redstone torch, wait for all of the repeaters to deactivate, and put the redstone torch back.

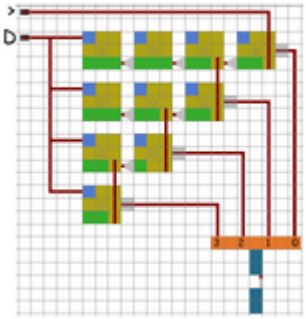
Serial interface lock with D flip-flops

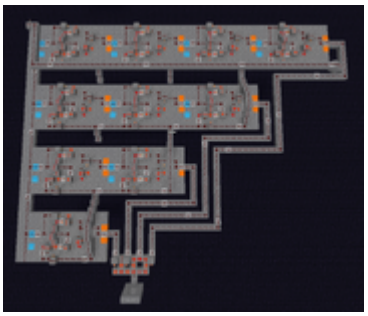


D flip-flop is an electronic component that allows you to change its output according to the clock. It's and RS NOR latch that sets its value to the D input when the ">" (clock) input is changing its state from low to high (in some cases from high to low).

Basically, it's equivalent to the expression: "Set the output Q to the input D when the input C goes from 0 to 1".

For example, you can use D flip-flops to shift the value from left to right.





In this lock, the > signal propagates from the rightmost flip-flop to the leftmost, so the signal shifts to the right. This circuit allows you to input a 4-bit number with two

levers. You can use any number of bits, but this configuration is already pretty secure even if someone figures out what a lock it is.

So, if you want to input the combination 1-0-1-0, follow these steps:

1. D = 1
2. > = 1
3. > = 0
4. D = 0
5. > = 1
6. > = 0
7. D = 1
8. > = 1
9. > = 0
10. D = 0
11. > = 1
12. > = 0

In theory, you can program the lock from this serial interface as well. Just attach 4 RS NOR latches and a hidden place for the programming levers.

This design is not very practical as a lock, but might be a nice feature on something like a puzzle challenge map.

Redstone computers

This article is a stub.

You can help by expanding it.

This article uses MCRedstoneSim schematics.

These should be converted to use `{{schematic}}` if possible.

This article aims to examine the design and implementation of **redstone computers** in Minecraft. This Article is extremely complicated, for nerds.

See Chapter 1, **Tutorial on Building a Computer**, for a detailed tutorial on building a computer in Minecraft and how to expand and improve on the example. Does not require any extensive knowledge of computer science. NOT FINISHED.

See Chapter 2, **Planning a Redstone Computer**, for basic computer concepts of designing and understanding a redstone computer in Minecraft. Does not require any extensive knowledge of computer science.

Overview

Computers facilitate the implementation of ideas that are communicated from humans through programming.

This article will explain the basics of designing and building a computer in Minecraft, assuming the reader is fairly familiar with redstone and computers to a basic level.

There really is no way of building a computer without knowing how a computer works. The tutorial attempts to explain everything that you need to know but does require a bit of understand here and there of computer science, which is stated in the prerequisites section of each tab. The deepest part we cover is up to IGCSE CS.

All computer systems have at least one processing unit. During operation, processing units execute instructions stored in the computer's memory. For a good start on Minecraft computers you should learn computer science. There are many sources and tutorials to learn computer science but for a basic start, it is recommended to watch Crash Course on Computer Science especially episodes 1–8. Although it isn't completely thorough, it can work as a basis in your understanding of computers.

Most computers in Minecraft are made of redstone dust, redstone torches, and repeaters, leading into sticky pistons or redstone lamps which are controlled using a series of buttons, levers, pressure plates, etc. Other proposed ideas (not covered) are to use hoppers, mine carts, or boats with redstone.

See chapter 1, **Tutorial on Building a Computer**, for a detailed tutorial on building a computer in Minecraft and how to expand and improve on the given example. It does not require any extensive knowledge of Computer Science as it will be explained but will delve quite deep into it.

See chapter 2, **Planning a Redstone Computer**, for basic computer concepts of designing and understanding a redstone computer in Minecraft. It does not require any extensive knowledge of Computer Science but will delve quite deep into it.

Implementations

Computers can be used in many ways, from creating a smart house to using it to run an adventure map. However, due to the limitations of computers in Minecraft, stated below, they remain an abstract concept and serve as good tools to understand lower-level concepts of CPU architecture and embedded systems.

The thing that sets apart computers and calculators are that calculators cannot perform multiple instructions in a row without user input. A computer can compare and assess instructions in a flow to perform tasks.

However, in Minecraft, they are extremely slow and with their large size, redstone computers are difficult to find practical applications for. Even the fastest redstone computers take seconds to complete one calculation and take up a few thousand blocks of space. Command blocks are far superior to computers in Minecraft because of their speed and legible, higher-level commands.

Mods can change the computer's speed such as TickrateChanger will change the tick rate of the game.

Chapter 1: Tutorial on Building a Computer

Introduction & Prerequisites

Redstone logic closely reflects simple binary logic, as redstone can be either on or off, and can, therefore, be interpreted as 1s or 0s. We will be referencing in this tutorial, basic binary logic and various simple computer science terms. There is an excellent article which explains binary and conversion to binary. Please read the *Architecture of building the Computer section* as we will be following that to plan our computer, it is located in this article, thank you.

This chapter will focus on the application of the knowledge and manipulation of redstone to create a simple 8-bit computer, and will describe how to make one and how it works.

All subjects will be split into (THEORY) and (PRACTICE), THEORY will go in-depth of exactly what will go on. PRACTICE will cover how to build it in Minecraft, what it will look like and possibly world downloads.

There are three primary design objectives for a computer in Minecraft, to make your computer most suitable for your task at hand. There are trade offs to consider, such as the larger the computer, the slower it will get because the number of redstone repeaters will increase by distance. The more memory, the less speed and larger size.

Compactness

How small is the computer? In Minecraft, designing a survival computer will most likely emphasize on this point. The number of repeats required will increase as size increases.

Memory

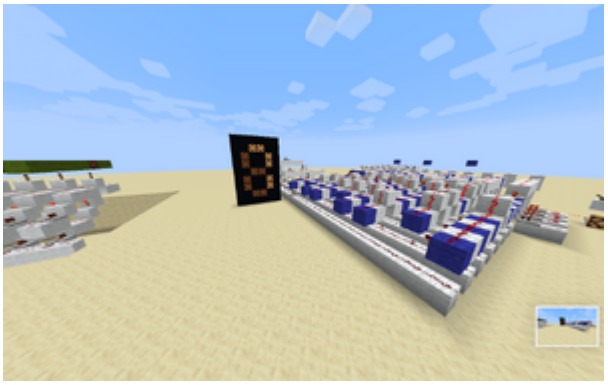
How much memory can it hold? How many bits and numbers can it count up to? This is important for large-scale computers, say ones which can do more complex algorithms and require larger instruction sets (e.g. doing square roots or trigonometry). The larger the memory size or bit architecture, the more complex the computer will get.

Speed/Performance

How fast can it do operations? Is it optimized to run its tasks? Custom designing and building a computer will significantly increase its speed as more redundant wiring and code could be switched to purpose-built hardware and software. This is apparent in some real-world supercomputers which are programmed to run one task very, very efficiently. The speed of computers in Minecraft is very slow, therefore a mod could be installed for the client to significantly increase the speed of the game, and therefore the computer.

The MASIC Computer

The work in progress computer which we will be making in the tutorial. 8 bits, 16 bytes of RAM. I/O is a seven-segment display (for hex and decimal) and a control panel which we will make.



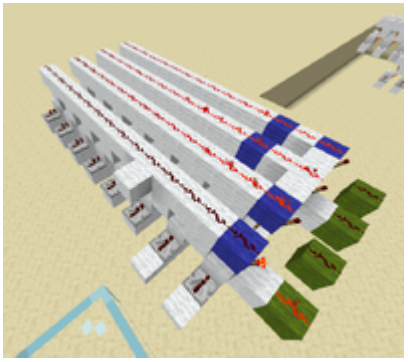
The MASIC computer aims to be a one-size-fits-all computer and does not specialize in one task, so it is fully programmable by reading its own memory (explained in Section 2: instruction sets). The simple I/O is great for multipurpose use and the memory is sufficiently sized. It runs at quite a fast speed (because of its small size).

Step 1: Memory and Address Decoders (THEORY)

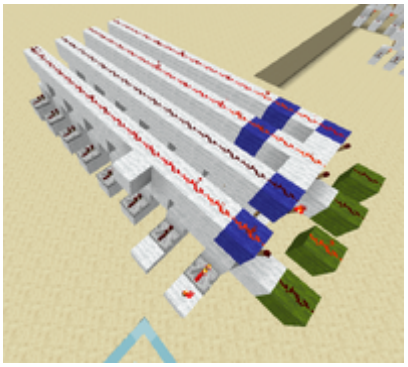
Decoders convert binary figures into decimals. For example, looking at the 8-bit decoder, 00 turns on the first lamp which stands for 0. 01 turns on the second lamp which is 1. 10 turns on the third which is 2. 11 turns on the last one which is 3.

Step 1: Memory and Address Decoders (PRACTICE)

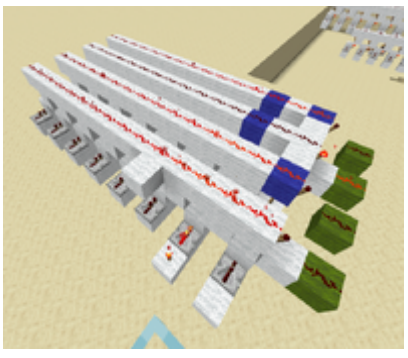
Address Decoder



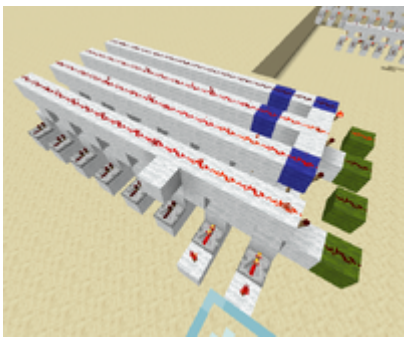
0000 0000 (notice output 1 is lit)



0000 0001 (notice 2nd output is lit)

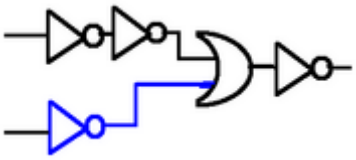


0000 0010



0000 0011

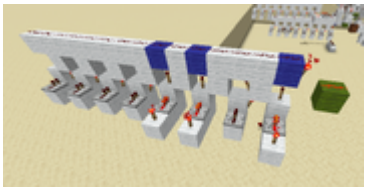
This is the design for the address decoder we are going to build.

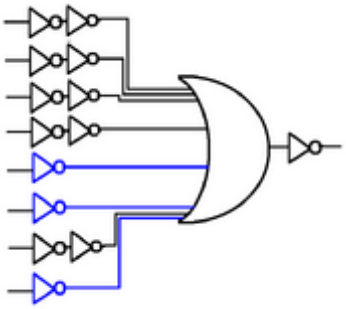


Above is a simple 2-bit state, so it has two inputs (left and right) through the repeaters. The output is the redstone line above which will turn OFF when the state is met. The state is whether the redstone input will turn OFF the redstone line above; if so, the state is the redstone inputs. In the above case, the left must be turned OFF (0) and the right (blue) must be turned ON (1) to yield an OFF on the top redstone line. So it expects a state of OFF ON (aka 01 for binary).

They are colored blue for bits which should be ON (1) for it to stop powering the top redstone line. Once every bit stops powering the redstone line, it then turns off.

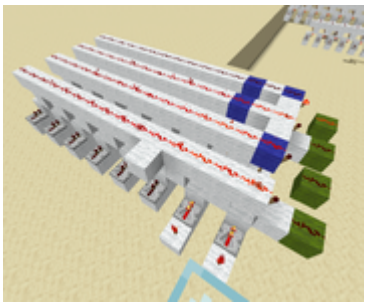
These are basically either one or two NOT gates feeding into a OR gate and then NOT the output.





Above is an 8-bit state, it expects 8 inputs in exactly the order 0000 1101. So that state it expects is 0000 1101. So the redstone torches power the inputs, and so we see the redstone line on the top turns OFF (only when exactly three redstone torches are placed in that exact order of 0000 1101).

Now if we put multiple of these together, we can count up in binary with the blue bits to get all 255 states of 8 bits. The one below is 8 bits, and has four state expectations. See the right images to see it in action. Now each green output can be a memory cell, and if we continue counting in binary, it will reach 255.



The input is 0000 0011 (see the redstone torches for input) and where the blue bits match the current state, the green output is ON.

- 0000 0000 - first signal out (on the images on the right)
- 0000 0001 - second signal out
- 0000 0010 - third signal out
- 0000 0011 - fourth signal out

So now we keep counting up in binary to get up to 0000 1111 and stop there; we should now have 2^4 (16) state expectors. Now we're done with the address decoder. We do not continue counting up to 1111 1111 because of instruction set limitations, explained in section 3: instruction sets

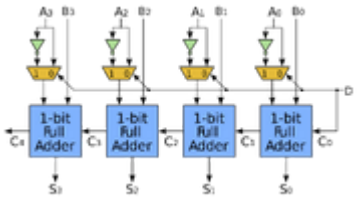
Step 2: Building an Arithmetic Logic Unit (THEORY)

The Arithmetic Logic Unit referred to as the ALU will compare and perform mathematical operations with binary numbers and communicate the results with the Control Unit, the central component of the computer (and Central Processing Unit but that is going to be as big as the computer itself). Many tutorials will want the reader to build an ALU first, and therefore the topic is covered very widely around the internet.

The ALU we will be building can perform four important operations on two inputs and return a correct output. A, B, being both 8-bit inputs

- A + B (Add A to B)
- A >> (bitshift A right (the same as binary divide by 2))
- << A (bitshift A left (the same as binary multiply by 2))
- NOT A (The opposite of A)

There can also be multiple ALUs inside a computer, as some programs require a lot of operations to run, which do not depend on the previous operations (so they can be threaded) so delegating them to different ALUs could significantly speed up the program.



binary adder

Adding two numbers

In an adding unit, for each bit (for our computer, we require four, hence 4-bit), there is a full adder. The full adder will take three inputs, each input can be either 1 or 0. The first two will be the user's input and the third will be the *carry* input. The *carry* input is the output of the previous full adder, this will be explained later. The adder will output two statements: first, the output and then the *carry* output, which is sent as input into the next full adder, a place value up. For example, I wish to add the number 0101 to 1011. The first full adder will consider the first place value, 1 and 1 as their two inputs (we are reading right to left). There is no *carry* input as there is no previous full adder. The full adder will add 1 and 1; which is 0, and carries a 1 to the next place value. The next full adder would add 0 and 1 and the carry input would be 1 which the previous full adder stated. The output of 0 and 1 would be 1 but there is a carry input of 1 and therefore will add 0 and 1 and 1, which is 0 and carries a 1 to the next place value. Reviewing addition in binary should resolve any confusion.

All ALUs, to perform adding operations, require the presence of multiple adders. Every two bits will feed into an adder which, when joined with other adders, will produce an output which is the sum of the two bytes added together. An adder has an input, an output, and two carry input/output as would a person carry when doing the addition of 9 + 1 or 01 + 01. The adders are made of logic gates which is possible by the nomenclature of binary. Tutorials/Arithmetic logic gives a very detailed look into full adders and half adders, for now, there is a schematic of how to construct one. It gives four inputs/outputs and should be connected with other adders to create a unit. For this example, we will connect four adders together in our

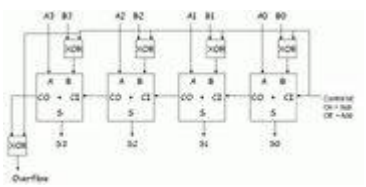
four-bit computer so that we can take in all four bits to make an output. There will be an input carry missing from the first adder, this is because there is nothing to carry from the bit before it, it is the first bit. The input carry will remain at zero. There will also be an output carry missing from the fourth adder, and the output of this will be ignored as we can only support four bits. The additional fourth carry output is wired to the overflow flag to signify the operation couldn't be done. This is called a binary overflow.

So basically, go into Minecraft and build a full binary adder (picture show) and connect them up. There should be eight inputs and outputs. Try placing levers and redstone lamps at the respective ends to test your creation. So 0010 + 0011 should yield 0101 (2 + 3 = 5, we are reading right not left).

Fractional numbers

A computer takes care of numbers less than one by form of float-point arithmetic, it is only so useful in larger-bit computers (16-64 bits) and computers which do need to use numbers less than one. Floating-point arithmetic or arbitrary-precision arithmetic are two ways to achieve this. Another simpler but less efficient way would be to assign all numbers a power of two so that they are 'bumped up' by the power of two chosen. The player must do this to every number and assume the one as one times the power of the two you have chosen. For example, $5 = 101_2$ so $5 \times 2^3 = 101000_2$; five is bumped up by three. So now, one in your new system would be $1 \times 2^3 = 1000_2$ and that would leave room for 0.1, 0.01 or 0.001; $0.01 * 2^3 = 10_2$. This leads to a more complicated setup for your computer.

Subtracting two numbers



An adder with all labeled parts.

The subtraction of numbers is surprisingly simple. The ALU first must change the second number (the value subtracting by) and convert it from a positive number to a negative number. A two's complement is when you invert the binary number (so that all the 0s are 1s and 1s are 0s) and add one to it.

Example: do 10 subtract 9

1. 0000 (9 in binary, we want -9, not 9)
1001

2. 1111 (Invert 9, so that all 0s are 1s and 1s are 0s)
0110

3. 1111 add one (this the two's complement of 9)
0111

4.

0000 1010 (10 in binary)

+1111 add two's complement of 9 (aka -9)
0111

0000 0001 result (10 + (-9) = 1) (there is an overflow, this just means that the result is not a negative number)

This poses the complexity of signed numbers.[1] This is a weight to the binary number to assign it as a positive or negative number. Whether the result is a

negative or positive number is determined by the overflow flag. If there is an overflow, this means that the number is positive and otherwise, negative.

To implement this, you can ask the ALU to do 3 operations. To do A subtract B, the operations are

Operation: A SUB B

- NOT B
- (set B to) B ADD 1
- (set A to) A ADD B
- RETURN A

Multiplying two numbers
Multiplication is repeated addition, so the easiest (inefficiently) is to add A to a variable B amount of times.

Here's pseudomachine code for it

Operation: A * B

- C = 0
- (set C to) C ADD A
- (set B to) B SUB 1
- JUMP IF (B > 0) TO LINE 2
- RETURN C

However, there are more efficient ways of multiplication. A good method is to repeatedly bitshift the first number to the location of each 1 in the second number and sum it.

There are underscores to mark indents, since padding with 0s are less intuitive. subscript 2 means in binary, and decimal numbers are also in bold

__ __11 **3** (notice that there are **2** 1s)

x_ 1011 **11**

__ __11 We shift 11_2 by $\mathbf{0_{10}}$ since the **1st** bit of 1011_2 is 1_2

+_ _110 We shift 11_2 by $\mathbf{1_{10}}$ since the **2nd** bit of 1011_2 is a 1_2

+1 1000 We shift 11_2 by $\mathbf{3_{10}}$ since the **4th** bit of 1011_2 is a 1_2

---- the **3rd** bit of 1011_2 is 0_2 so we do not add a 11_2 there

10 0001 **33** (result)

so this is more efficient for larger numbers.

Operation: A * B

- C = 0
- D = 0
- (Set A to) << A (bitshift A to the left)
- JUMP IF (BIT (D) OF B == 0) TO LINE 6
- (Set C to) C ADD A
- (Set D to) D ADD 1
- JUMP IF (D < LENGTH OF B) TO LINE 3
- RETURN C

Don't forget that

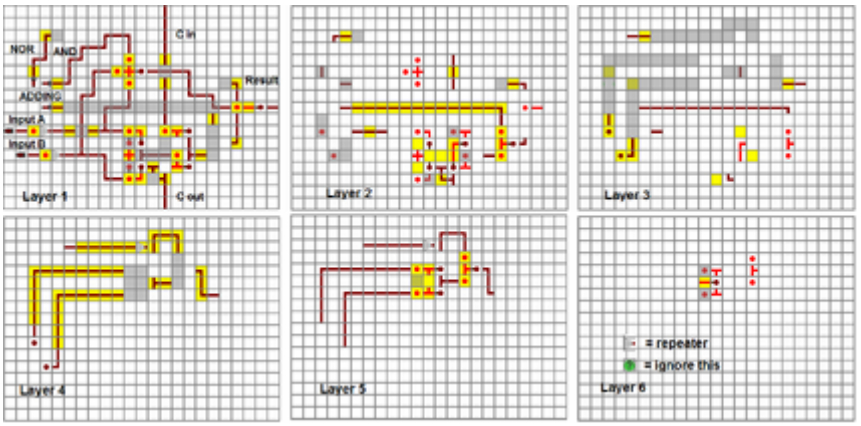
<< A (bitshift to the left) is effectively, A * 2

and

>> A (bitshift to the right) is effectively, A / 2

If the numbers are predictable or the CPU must do a lot of similar numbers in bulk, consider using a look-up table to quickly get results to frequently called multiplication. Is this a way of hard-coding your answers and is used in extreme cases.

Step 2: Building an Arithmetic Logic Unit (PRACTICE)



Step 3: Instruction set and machine architecture (THEORY)

This is pretty fun, this part.

Elaborating on Chapter 2: Instruction Set, we will be creating one for ours.

For the MASIC Computer, the computer which we are building, has an 8-bit system, so that means each instruction on each slot of the stack memory will be 8 bits. The stack memory is the memory where any information can be stored and is on the RAM. There will be a counter, called the program counter, which increments by 1 every cycle. A cycle is the CPU fetching the instruction, decoding the instruction

(finding out what to do with the instruction) and executing the instruction (doing what it tells it to do). Then it moves on to the next one by incrementing the program counter and reading the information at that location in the stack memory.

So each byte in the stack memory has 8 bits for us to work with.

0000 0000

and some instructions require an address, say loading memory into a register so that we can perform operations (such as addition) on it. Each instruction will be split into two parts, each 4 bits. The first is the TYPE. the TYPE will specify what the computer must do and the ADDRESS will be where the value we will perform our operations are located.

OPCODE OPERAND

so 4 bits for the TYPE, we can have 2^4 types, so 16 different ones. Our computer will have two registers, so one bit will be for specifying the register the operation will executing on and is denoted by an **x**.

Instructions are put in the same place as memory and as the ADDRESS part of the instruction is only four bits, we can only reference memory from 1-16 lines, requiring some clever programming to fit larger programs. Memory is also limited to 16 bytes per program. Values and instructions are essentially the same thing, so if you write an instruction to store it onto a line that previously-stored an instruction, that effectively overwrites the instruction with a value. Accidental execution of values might be a problem, so a STOP command must be used to prevent any errors. This is a whole lot to understand, so good sources are https://www.computerscience.gcse.guru/theory/high-low-level-languages and

https://scratch.mit.edu/projects/881462/ <-- really helpful actually. and also don't forget to take both CS and ICT for your IGCSEs.

Prerequisites

The section will cover simple topics and components commonly found in a computer, so information from chapter 2 will be used, such as the ALU, RAM, registers and binary manipulation.

The MASIC Instruction Set

Since the computer Here is the first draft of the instruction set, with only essentials. This is based on other assembly languages, but changed to adapt to our architecture. There are two registers, so we need instructions to perform operations on both registers.

BINARY	OPCODE	COMMENT
0000	LOAD R1	Load the ADDRESS into register 1
0001	STORE R1	Store contents of register 1 into ADDRESS
0010	JUMP R1 IF	Jump to line ADDRESS if register 1 is equal to 0
0011	ADD R1	Add contents at ADDRESS to register 1
0100	<<R1	Bitshift register 1 left

0101	NOT R1	Bitwise NOT register 1
0110	JUMP	Jump to line OPERAND
0111	STOP	Terminate the program.
1000	LOAD R2	Load the ADDRESS into register 2
1001	STORE R2	Store contents of register 2 into ADDRESS
1010	JUMP R2 IF	Jump to line ADDRESS if register 2 is equal to 0
1011	ADD R2	Add ADDRESS to register 2
1100	<<R2	Bitshift register 2 left
1101	NOT R2	Bitwise NOT register 2
1110	OUT R1	Outputs register 1

1111		

To translate:

1000 0011 means LOAD R2 3 because LOADR2 is **1000** and 0011 is 3.

These can be in a process so that functions can be performed.

Writing programs

This one does the Fibonacci sequence: (0,1,1,2,3,5,8... etc.)

FIBONACCI			
LINE	**BINARY**	**INSTRUCTION**	**COMMENT**
1	0000 1110	LOAD R1 14	set register 1 to 0 (the value at line 14)
2	0011 1111	ADD R1 16	add the value at line 16
3	1110 0000	OUT R1	output the register

4	0001 1111	STORE R1 16	put that in line 16
5	0011 1110	ADD R1 15	add the value at line 15
6	1110 0000	OUT R1	output the register again
7	0001 1110	STORE R1 15	now put the output back
8	0110 0010	JUMP 2	we don't have to reset the register so we loop back to line 2.
...			
14	0000 0000	0	
15	0000 0001	1	
16	0000 0001	1	

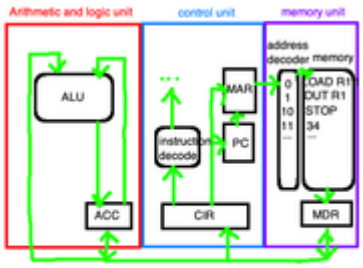
The previous is an example of a low-level assembly language. If it was written in a high level language such as C++, it would look more like this:

```
#include <iostream>
using namespace std;
int main()
{
    int n, t1 = 0, t2 = 1, nextTerm = 0;
    cout << "Enter the number of terms: ";
    cin >> n;
    cout << "Fibonacci Series: ";
    for (int i = 1; i <= n; ++i)
    {
        // Prints the first two terms.
        if(i == 1)
        {
            cout << " " << t1;
            continue;
        }
        if(i == 2)
        {
            cout << t2 << " ";
            continue;
        }
        nextTerm = t1 + t2;
```

```
        t1 = t2;
        t2 = nextTerm;

        cout << nextTerm << " ";
    }
    return 0;
}
```

Instruction Cycle



Rounded squares are components, squares are registers. Green arrows are busses

The instruction set is the lower assembly language, so we want to integrate that more with the hardware side. This revolves around the fetch-decode-execute cycle (explained above). In the CPU, there will be 4 important registers,

the Program Counter (PC), keeps track of which program the computer is currently on

the Memory Address Register (MAR), keeps track of where the next memory location will be

the Memory Data Register (MDR), keeps track of what the memory AT the location is

the Current Instruction Register (CIR), keeps track of what instruction is currently being worked on

and the ALU Accumulator (ACC), keeps track of the input and output from the ALU

There are also four components to keep in mind, the Address Decoder, the memory, the Instruction Decoder and the ALU.

FETCH The program will get the next instruction.

1. PC sends the instruction number to the MAR
2. PC increments by 1, to prepare for the next instruction
3. Address Decoder decodes the address, and requests information at that address from the memory
4. MDR receives the requested information (in the case of the picture, if the MAR is 0001, it receives 'LOADR1 1')

DECODE The program will identify what the instruction is

1. CIR receives the information from the MDR, through the information flow
2. Instruction Decoder decodes the instruction and what to do

EXECUTE The program will execute the instruction

1. In the case of the picture, the program receives 'LOADR1 1' as the instruction, the Instruction Decoder splits the instruction up into the opcode and the operand.

The opcode is 'LOADR1' and the operand is '1'.

1. Operand is sent to the MAR, to get the information at that address
2. MDR receives the information at that address (in the example, it is the same line)

Now four things could happen depending on what the instruction is.

If the instruction is an ADD instruction, the ACC will be told to receive the information from the information flow and the ALU will perform operations on it, outputting it to the ACC again.

If the instruction is a LOAD instruction, the CU will load the instruction to the register.

If the instruction is a STORE instruction, the CU will instead SET the data at the location specified by the MAR in the memory.

If the instruction is an OUT instruction, the CU will send the instruction to the output peripheral.

REPEAT The instruction cycle repeats itself until it reaches a STOP instruction or runs out of memory

Step 3: Instruction set and machine architecture (PRACTICE)

Chapter 2: Planning a Redstone Computer

A redstone computer can be planned very much like a real computer, following principles used in computer design and hardware architecture. There are several key design decisions that will affect the organization; the size and performance of your prospective computer should be made concretely prior to the construction of specific components.

Building a redstone computer will require an understanding of these five concepts and consider the most suitable approach, which would be most practical for your computer.

- Machine-Architecture (Components of a computer, what are they and what they do)
- Execution Model (The organization of components, making them efficient)
- Word Size (How many bits the system uses. Usually, powers of 2, around 4, 8, 16 bit is normal in Minecraft)
- Instruction Set (The instructions to be performed by the CPU)

and we will be applying this knowledge and plan the architecture of our CPU in the last section. This CPU will then be built in the next chapter.

Fundamentals of a Computer

A computer is a machine which has the ability to

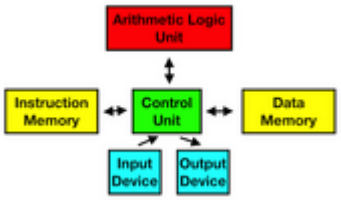
- Read and write from a memory which can be addressed
- Perform comparisons on the state of the memory, and perform an operation as a result of that. These operations include rewriting memory.
- Start functions based on content written in the memory. We call such content "programs + data", and the act of writing them programming.

A very notable example of this is the most basic concept of computing, a Turing machine, where the machine will read from one infinite line of code and instruction set in order to complete a function.

Designing and building a Turing machine in Minecraft is possible. This however, is not covered as we will be designing something more basic.

Machine-Architecture

There are five fundamental components in a basic modern computer. These are essential in order to produce a functioning computer and manipulate data by performing computations.



five components of a computer

Arithmetic Logic Unit (ALU) (optional, but is normally present)

- Perform adding and subtracting
- Compare booleans using logic gates

Control Unit (CU)

- Perform/Execute instructions sent to it
- Communicate with all components

Data Memory

- Store and return data from memory

Instruction Memory

- Return instructions, sent to the CU
- Can be set, but doesn't need to be as often as the Data Memory

Input/Output devices (I/O)

- Allows the computer to communicate with the world and the player.
- Can input information the computer (button push, daylight sensor)
- Can output information from the computer (redstone lamp, note block)

Computer Data Storage

There are many methods of storing data, in Minecraft or in real life. The states of memory usually are binary, either on or off and can be computed with boolean logic.

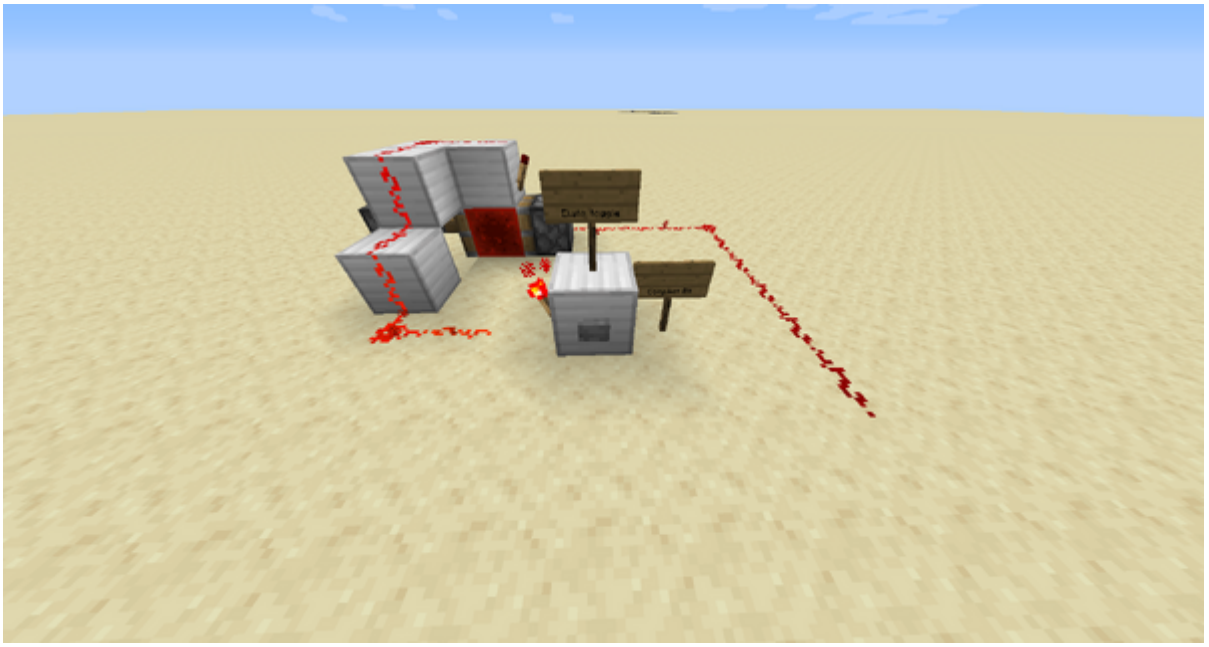
On a computer, there are three types of storage. Keeping in mind that increasing the device's capacity would increase its size, each type would have speed and capacity appropriate to it.

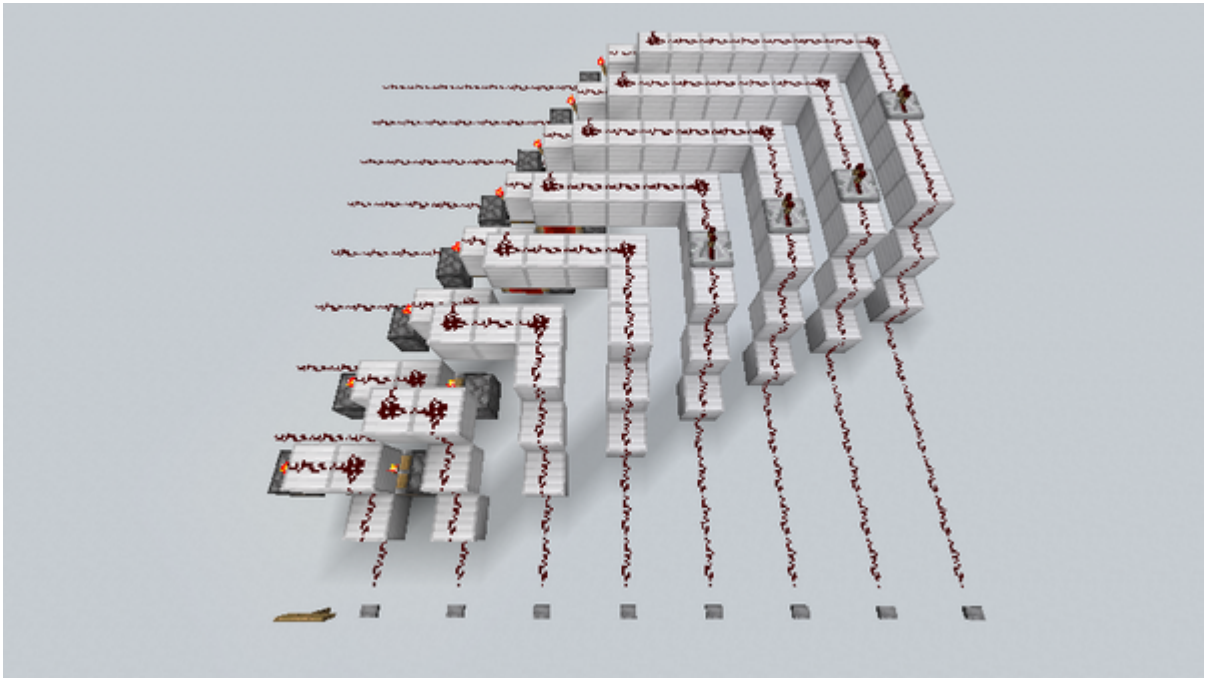
Primary Storage

These are the storage which directly accessible to the CPU, referred to as memory and is fastest to access but usually is smaller in capacity for it to be addressed quicker.

Registers & Flags

Fastest is the memory stored within the CPU. These are registers and flags as they can be set almost instantaneously and do not require any address sent to it as there is only one byte stored in each register. Redstone bits that can be toggled are extremely large but can be toggled within 2 ticks. This requires a very large amount of space but is perfect for caches and registers. The redstone is also required for logic gates (not shown) to set the bit, as in the images, sending an input would cause the bit to flip. The gate would take up more space. Registers could also utilize locking redstone repeaters and timing them correctly. This is explained below, in RAM). With the use of a computer clock, it may not be necessary to build registers. Registers are useful when the data goes through the line before either the CU or ALU is ready to process it. It would save it to the register and wait until the CU or ALU can perform its function.





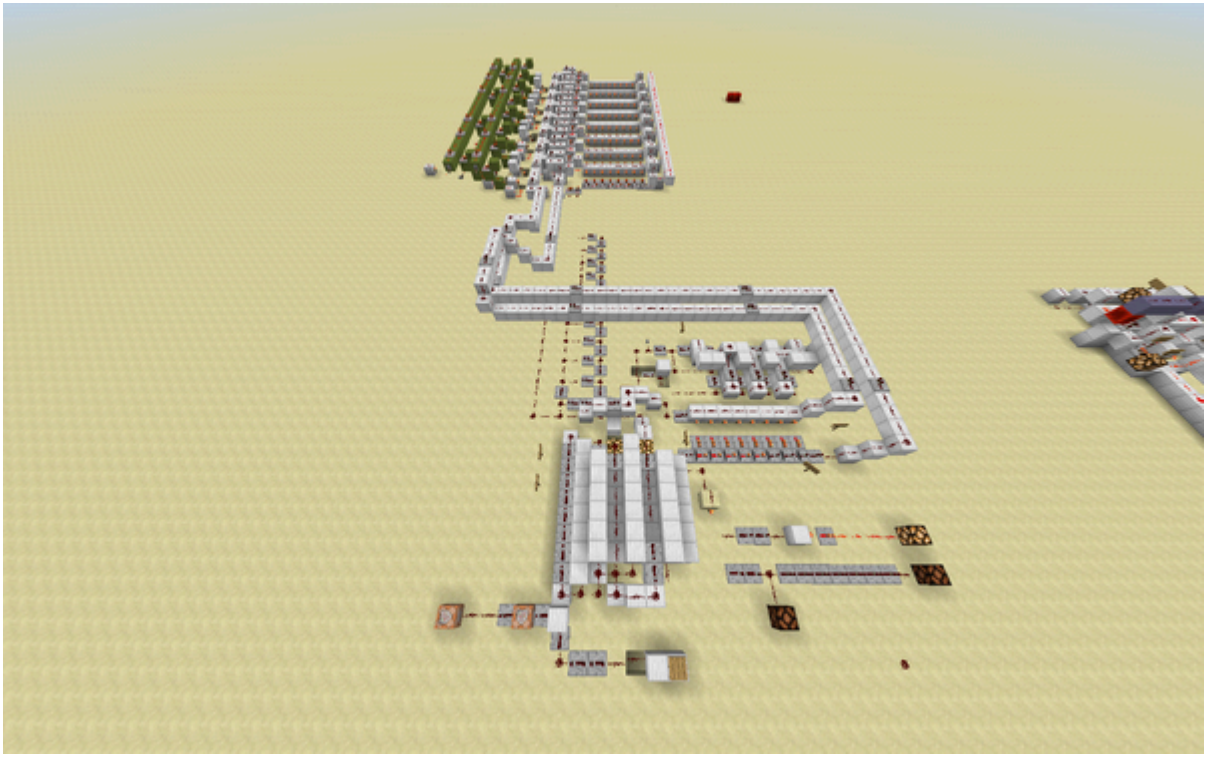
Caches

Second to those are caches, which feed information into the processor. In real life, they are separated into levels, each one with separate speed and capacities. It is useful for the same reason as the registers.

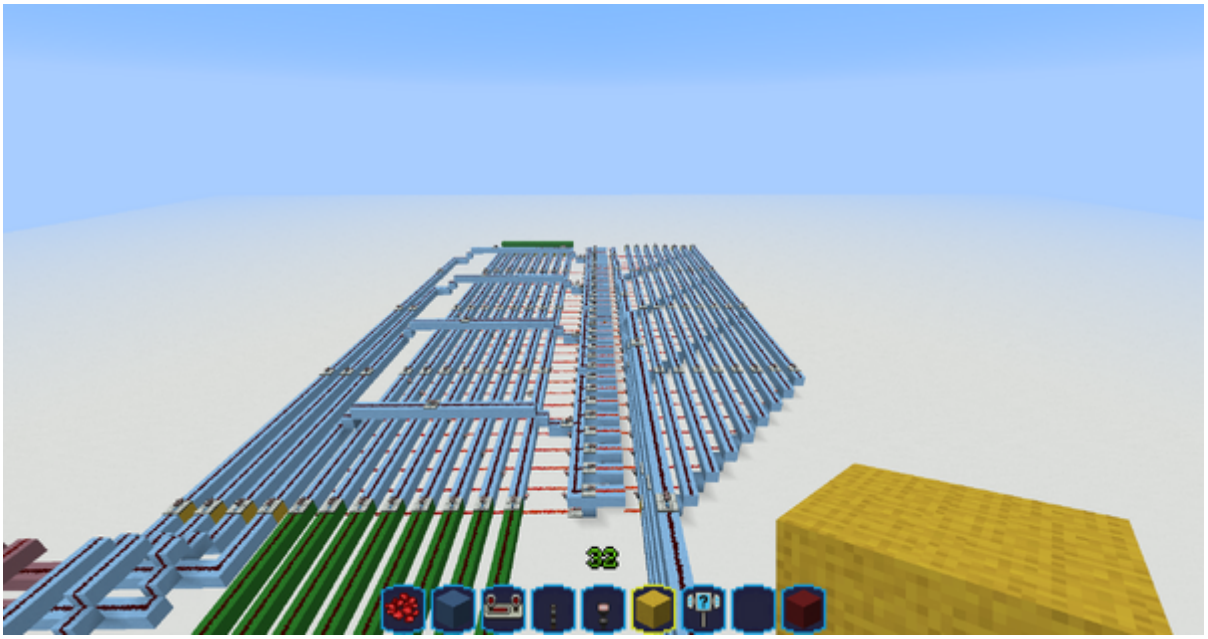
Random Access Memory (RAM)

Thirdly is Random Access Memory (RAM), this is much slower than the caches and registers as they have address systems. They are connected to three busses, data bus, control bus and the address bus. The data is sent through the data bus, either setting the RAM or getting values from the RAM. The control bus tells it whether it is being *get* or *set*. The address bus tells the RAM where the byte is. Refer to the Architecture of the Computer to understand this more in-depth. RAM is very useful and could fully replace tertiary memory (explained below) because of its non-volatility in Minecraft. Volatile means that when power is lost, it will lose information. The RAM will not lose information unlike in real life, and therefore in an excellent method of storing information.

The RAM in the first case is utilizing the locking redstone repeaters with the correct timing. This requires a bit of a plan but is very space-efficient. The conversion of a bus to the lines in order to lock the redstone repeaters also requires setting timings. This is time-consuming, much more than the registers, however, it is very compact and efficient. The address bus (green) would turn in binary to *unlock* a certain byte, either to be read or set by the control bus (second line, on the left).



Most often, making it volatile has no use in Minecraft, so the easiest way to make some is to use d-flip-flops and to add a reading and writing function. The bottom image shows instead of locking repeaters, it uses d-flip-flops which is much more space inefficient but simpler to build. D-flip-flops work more or less like locked repeaters, one input - if on, unlocks in until the input is off and the other will set it once unlocked. The output can be read as a bit and with a NAND gate, be ignored or put onto the bus. This is gone over in detail in the second chapter, *Tutorial on building a Computer*. Excuse the texture pack.



Random Access Memory also known as **RAM** is a kind of memory used by programs and is volatile. Volatile means that when the power is lost, it will lose information. Most often, making it volatile has no use in Minecraft, so the easiest way to make some is to use d-flip-flops and to add a reading and writing function.

Secondary Storage

These are equivalent of HDDs and SSDs. There is a very compact storage technique, involving redstone comparators with the ability to store up to 1KB, being practically sized.

Tertiary Storage

Third and last, is a tertiary memory, which requires a lot of time to read/write but can hold massive amounts of information at the expense of speed. Real-world tertiary storage use a mechanism of mounting the memory which takes about a minute for each drive. This is used for archival purposes and for memory which is rarely used. In Minecraft, a system where shulker boxes are used and block in the shulker boxes must be sorted out by a sorting system to represent a form of data. This can also be used to create removable storage. The read/write speed is fairly slow due to the massive amount of comparators and a lot of time is required. The aforementioned

mods could speed up tick rate and eliminate this problem, however. This is used for storing long-term data that needed to be loaded at the beginning of a program or rarely due to its poor read/write speed and large capacity. This is the equivalent of a real computer's hard disk or solid-state drive.

Execution Model

The technique of storing blocks of instructions called programs within memory is what allows computers to perform such a variety of tasks.

The apparatus employed by a computer for storing and retrieving these programs is the computer's Execution Model.

Two of the world's most successful execution models, Harvard and von Neumann, run on nearly 100% of the computers available today.

This is more advanced, and is for inquisitive and curious readers

Harvard

The Harvard architecture physically separates the apparatus for retrieving the instructions which make up an active program from that of the data access apparatus which the program accesses during execution.

Programs written for computers employing a Harvard architecture may perform up-to 100% faster for tasks that access the main memory bus. Note however that certain memory circuitry is necessarily larger for those who select a Harvard architecture. Harvard architecture is very important.

von Neumann

The von Neumann architecture uses a two-step process to execute instructions. First, the memory containing the next instruction is loaded, then the new instruction just loaded is allowed to access this same memory as it executes; using a single

memory for both program and data facilitates Meta-Programming technology like compilers and Self-modifying Code.

The von Neumann architecture was the first proposed model of computation and almost all real-world computers are von Neumann in nature.

Word sizes

Word-size is a primary factor in a computer's physical size.

In Minecraft, machines from 1-bit all the way up to 32-bits have been successfully constructed.

Common word-size combinations:

Data	Instruction
4	8
8	8
8	16
16	16

Data-Word

The amount of information a computer can manipulate at any particular time is representative of the computer's data word-size.

In digital binary, the computer's data-word size (measured in bits) is equal to the width or number of channels in the computer's main bus.

Data-Words commonly represent integers or whole numbers encoded as patterns of binary digits.

The maximum sized number representable by a Binary encoded integer is given by $\mathbf{2^{data\text{-}word\ width\ in\ bits} - 1}$.

For example, a computer with a data-word size of 8-bit will have eight channels on its bus (set of wires, connecting components) and therefore, we can count up to (2^8 - 1). 255. Counting further than 255 is not possible with eight bits, as the operation 255 + 1 carries over a one, which requires a ninth bit or what is called a *binary overflow* will occur, returning 0 as the answer, which is incorrect.

This is simply visualized;

	1	1	1	1	1	1	1	1	255
+	0	0	0	0	0	0	0	1	1
=	0	0	0	0	0	0	0	0	0

Some common Integer data sizes are:

Max Representable Number	Number of Bits Required
1 = (2^1 - 1)	1

$7 = (2^3 - 1)$	3
$15 = (2^4 - 1)$	4
$255 = (2^8 - 1)$	8
$65535 = (2^{16} - 1)$	16
$4294967295 = (2^{32} - 1)$	32

Data-Word size also governs the maximum size of numbers which can be processed by a computer's ALU (Arithmetic and Logic Unit).

Instruction-Word

The amount of data a computer needs in order to complete one single instruction is representative of a computer's instruction word-size.

The instruction-word size of a computer is generally a multiple of its Data-Word size, This helps minimize memory misalignment while retrieving instructions during program execution.

Instruction Set

This is a collection of instructions the control unit (CU) can decode, and then execute.

Instructions are essentially functions run by the computer, examples of instructions include:

- Add, subtract, multiply and divide

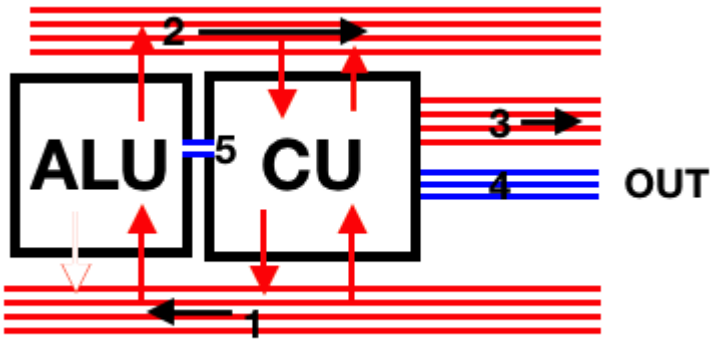
- Read/Write from RAM/ROM/tertiary memory
- Load and unload data into the RAM
- Branching to other parts of the code
- Comparing registers
- Selecting a logic function (NAND, NOR, NOT etc.)

Instructions can be programmed into the RAM, loaded from ROM or directly activated by using a lever or button. Each instruction would have its own specific binary string assigned to it (e.g. 0000=Load data from register 0001=add A and B 1011=Save RAM into tertiary memory etc.) and would probably require its own binary to decimal or binary to BCD to decimal encoders and buses to the ALU/registers.

Architecture of the Computer

Inside the computer, there is a Central Processing Unit (not to be confused with the Control Unit (CU), a component inside the CPU), which in real life, is a very small and powerful component that acts as more or less, the brain of the computer. In Minecraft, it is difficult to compact it to the scale we see in real life so don't worry if it looks *wrong*.

We will first be designing our 4-bit Central Processing Unit in the next chapter, as it is the most important thing in our computer with the Execution Model (the method of communication and organization of the CPU) in mind, (talked about in this page, before, in the Execution Model section) we can map out the construction of the computer.



Map of the CPU, based on the Havard Execution Mode

The CPU follows a cycle four steps, fetch, decode, execute and (sometimes) stores to perform instructions. The CPU first fetches the instruction from RAM, decodes what it means (the instruction will most likely be a number, and the CPU must find out what number it is), and once it understands what the instruction is, it will perform that action. This sometimes requires the data to be put back into the storage, therefore it will store the data. The cycle is then repeated.

Busses

There are five busses in the CPU, each to carry information from one component to the next. Busses are channels of redstone connecting each component. Since we are building a 4-bit computer, we only need four channels in our bus. These are the red and blue lines connecting the components inside the CPU. Notice that the blue buses have less than four lines, this is because they do not carry data. Since busses can only carry data one way (in Minecraft, due to repeaters only working one way), there are two buses connecting the CPU to the outer computer.

The first bus is the data bus, this is to transfer information from the storage or I/O devices to the CU. Instructions are also sent through this line The CU can also use this bus to transfer data to the ALU. The ALU cannot transfer data onto this bus

because buses only work one way and once the information is taken by the ALU, the bus cuts off beyond the ALU. Information from the ALU is passed through bus 2.

The second bus is the data bus, but returns the data from the ALU to the CU. The CU cannot send data through this bus to the ALU because the bus goes from left to right and works in one direction only. The CU can send information back to the storage units though, and is used to set values of storage devices.

The third bus is the address bus, which the CU can send the address of storage. This is where the information resides. For example, the CU asks for the address of the byte living in 0001. It sends the address (0001) through the address bus and the RAM will return the value of the byte through the first bus. 0001 is the *location* of the byte, not the value of it.

The fourth bus is the control bus, which the CU will communicate with the RAM with. For example, one wire could tell the RAM to set the byte to the value to the data sent to it by the CU. Another wire could tell the RAM to get the byte from the address sent to it by the CU.

The fifth bus is another control bus, linking with the ALU, which sends flags from the ALU. Flags are notes which could be error messages. For example, the CU could ask the ALU to add 15 and 1 in a 4-bit system. Adding 15 and 1 in 4 bits would yield 0 (explained above) and this is called a *binary overflow*. This is an error and the ALU will tell the CU about this through the fifth bus as a flag. The CPU could also send data to the ALU and ask for it to perform an action with that data.

Components

Control Unit (CU) will fetch instructions from the instruction ROM (for other computers, instructions can be changed and therefore is RAM. For our case, we are running a fixed program and do not need to change the instructions. This simplifies

the process entirely and the instruction is Read-Only Memory (ROM)). Inside the CU, it will then decode the instruction, which is normally a number, into a sensible action. It will then perform that action and if the instruction requires, store the result into the RAM. It communicates with the RAM through the control bus and receives flags from the ALU. It can also ask the ALU to perform actions on data it sends to the ALU (e.g. addition). To communicate with the RAM, for example, one wire could tell the RAM to set the byte (the location of it is specified through the third, address bus) to the value to the data sent to it by the CU through the second, data bus.

Arithmetic logic unit (ALU) will execute instructions sent to it from the CU and will compare binary numbers and communicate with the Control Unit. It can do simple addition and subtraction which can be repeated to do multiplication and whole-number division, outputting a whole number (then division). There are also logic gates for booleans, the fundamental logic gates are required, such as the *NOT* gate and the *NAND* gate.

Now we can choose from a range of designs of busses, each contributing to the aforementioned three key designing goals of a Minecraft computer.

MCRedstoneSim schematics

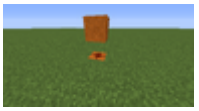
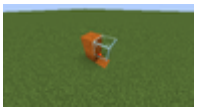
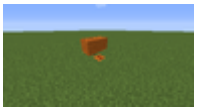
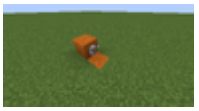
This article uses MCRedstoneSim schematics.

These should be converted to use `{{schematic}}` if possible.

MCRedstoneSim (from this point MCRS), uses a basic symbolism to represent redstone circuits. This same symbolism is used on the circuits page.

The symbols used in MCRS are shown below:

Symbol	Description	Screenshot
	Air over air (blank space).	
	Air over block.	
	Block over block.	
	Air over wire.	
	Air over torch.	
	Air over torch (attached to adjacent block).	
	Wire over block.	
	Torch over block.	
	Torch (attached to adjacent block) over block.	

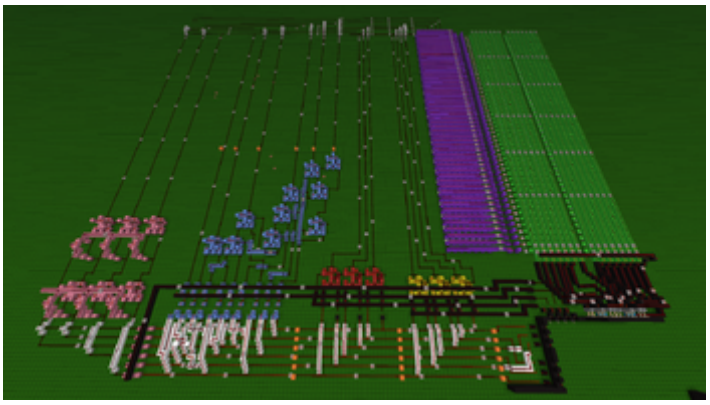
	Block over wire.	
	Block over torch.	
	Block over torch (attached to adjacent block).*	
	Torch (attached to adjacent block) over wire.	
	Wire over wire.	
	Air over lever.	
	Air over lever (attached to adjacent block).	
	Air over button.	
	Door (unpowered).	
	Door (powered).	

	Shadow (used for visual effect).	n/a
**Glass blocks do not interact with redstone. They are used in screenshots purely for visibility and must be replaced with opaque blocks in actual circuits.*		

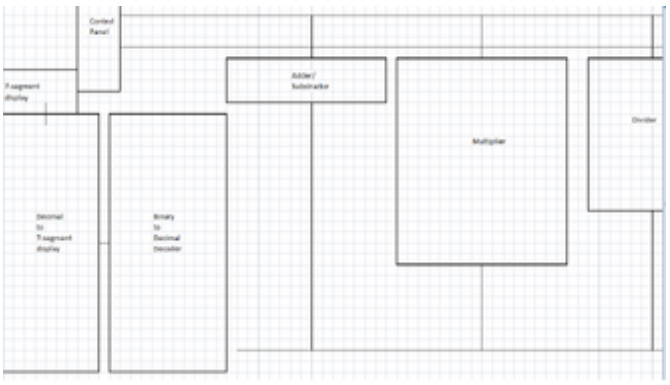
Tutorials/Calculator

Resume

See the following image for an example of a calculator:



Calculator Plan



Calculator Scheme

Of course, all compilations are made with binary code. This is why this calculator has many different decoders.

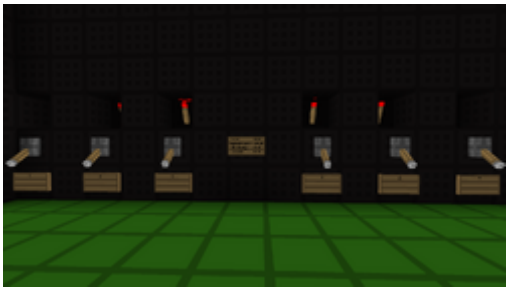
Current parts

The following are the components of a calculator. They are in a somewhat logical order.

Control panel (room)

The control panel is the room from which you set the inputs and decide of the operation.

Numbers input panel



Numbers Input Panel

Here, the users will decide what numbers they want to use. In the picture, a lever-based binary input system is used, so the users must decompose the numbers they want to use into powers of two.

Operation panel

From this panel, the user chooses between the operations he is going to use: add (+), subtract (-), multiply (*) and divide (/). Like for the Number's Input Panel, this picture of the Operation Panel uses the lever system.

Operation Panel

Input wires (white and orange)

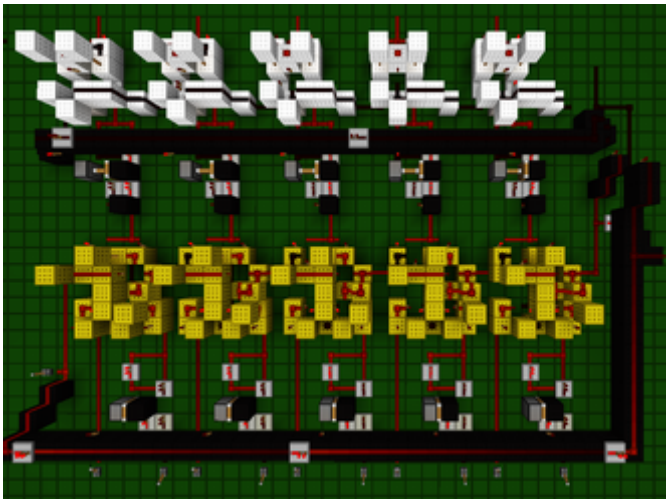
These wires link the input panel and the operation panel to the different logic units. Try to rearrange them in a manner where the same values go together. So, your wires should look like, from left to right: A1; B1; A2; B2; A4; B4; ...

Logic units

The logic units in a calculator are the machines that perform the operations.

Adder/Subtractor (yellow and red)

The picture to the left shows a version of an adder/subtractor. Its construction is simple because it is modulated (made of many same parts). That means that if you use more bits, you can just add more parts on the side. However, this means that you'll have to change some links.



2-in-1 Adder/Subtractor

In this machine, your inputs (in binary code) go into the bottom (yellow) full adders. Each adder needs the two inputs (A and B) with the same values. Also, the least significant bit has to be on the left, so they should all be connected by their carries. Basically, your inputs look like the wires in the #Input Wires (white and orange) section. Use basic bridges to pass wires over the others without connecting them. Your A inputs (left) are the minuend (X in X-Y=Z), and go straight in the adders. The B inputs, your subtrahend (Y in X-Y=Z), have to pass through a multiplexer, made out of a modified version of a XOR gate which gives to the adder an inverted signal in case of a subtraction. The multiplexer is controlled by a switch (on the picture, that switch is on the left). The sums go into another multiplexer, which, again, gives an inverted input in case of a subtraction. This is controlled by an IMPLIES gate (in the top right) which gives a true output if the switch is on "Subtraction" AND if the last carry is true. This is required because on a subtraction, that last carry actually means the "-" (minus) sign.

The white machines are half-adders, that use, as inputs, the carry of the last adder and the sum of their respective full adder. We need this because, if the answer is negative, it uses the equation " -A = !A (inverted A)+ 1 ", as explained here. The final outputs are all of the top-most wires you can observe, plus the wire on the right (the carry from the last full adder) and the carry that goes in the first (left) half-adder, as the negation sign.

Multiplier (light blue)

The multiplier is probably the most complicated part of the calculator. For our purposes, multiplication is a repeated addition. That means that, once again, adders will be used here. Before you add the adders, you actually have to set up an AND gate (not including the control one). Its use is simple: in binary multiplication,

because only 0's and 1's are used, the only way that we can have an output is by multiplying 1 by 1.

Here is how to construct a multiplier, in order from the least to the most significant bits.
Least significant bits : 1*1 = 1. That means that the output of the second AND gate (the control one) goes straight to the output collective wires.
Second to last: 1*2 = 2 and 2*1 = 2. Those two outputs meet in a full adder. The sum goes to the output, and the carry goes to the next bit.
Next: 1*4 = 4; 2*2 = 4; and 4*1 = 4 The carry from the last bit goes in the first adder as the carry input. The two normal inputs are 2 of the 3 AND gates. The sum of this goes in a second adder, where the second input is the third AND gate. Both carry outs go to the next stage, and the sum goes to the output.

You continue like this until you run out of AND gates, or equations.

Divider (pink)
Dividers on a redstone calculator are less complicated than multipliers. Again, full adders should be used here. Basically, for each A input, set up ***n*** adders where ***n*** is the quantity of inputs by B. Also, this time, you have to "reverse them". Now the most significant bit should pass its carry downwards.

Output wires

Output wires have to get every output from every machine and redirect them to the next part using redstone dust.

Binary-to-decimal decoder

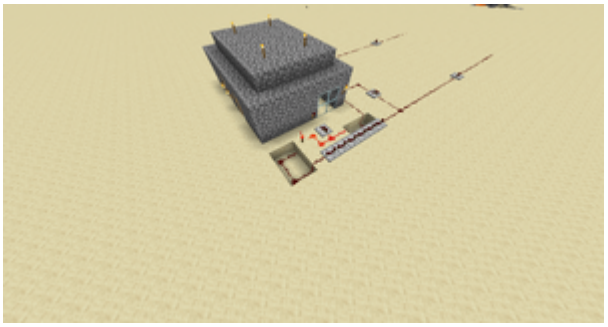
This transforms your binary code into a decimal output. The size of it will be (Binary inputs*2)*(Decimal output)

Quick note It uses a "programmable" XOR gate cane tend to a not gate. This activates a line of preset redstone torches to output the correct answer.

Tutorials/Telegraph

This article uses MCRedstoneSim schematics.

These should be converted to use `{{schematic}}` if possible.



Outside of a mid-size telegraph system.

A **telegraph** in *Minecraft* works identically to a real telegraph device, sending a series of redstone pulses over long distances to be decoded and interpreted by a receiving party. There are multiple designs, from the simple flashing redstone torch, to massive machines capable of reviewing, deleting, and editing messages before sending them to their destination. This tutorial will explain how to operate these devices, and how to create your own.

How they work

All telegraphs, no matter how basic or complex they are, require four things: a sending device, an inverter, redstone wire, and a receiving device. The sender will just about always be in the form of a lever. Anything else, such as a button or pressure plate, cannot be easily used to create a message, as they will remain activated for at least 0.9 seconds. A lever, on the other hand, can be shut off as soon

as the player wants to, allowing for the quick pulses needed for telegraph languages such as Morse code. After the sender comes the inverter, a simple logic gate that inverts its input to create the output. For instance, if the input wire is powered, the output wire will be off. These are almost always used in telegraphs because redstone torches (a common type of receiver) create their own power. This means that without an inverter, the torch will turn off when a pulse is sent. Although this is only an issue of appearance, it may lead to confusion during interpretation of the message, which is tricky enough by itself. Then comes the all-important wire, which allows you to send your message as far as you want (although the longer the wire, the more likely it is that something goes wrong). The wire consists of redstone wire, and, depending on its length, redstone repeaters placed on every fifteenth block. This will stretch out for as long as needed, until it reaches its destination; the receiver. Receivers can be anything from a redstone torch that blinks when a single pulse is received to a room filled with redstone repeaters, displaying the entire message before the player.

Telegraph components

Sending devices

A lever in a telegraph room.

A sending device is the mechanism used to transmit a series of pulses to the receiver, whether it creates those pulses or simply allows a looping message to enter the telegraph line. It can be one of three objects; a lever, a button, or a pressure plate.

Lever

Levers are by far the most common of all sending devices, and are the most practical choice in nearly every scenario. They can create as long or as short a pulse as the sending party wishes, and are the most accurate representations of real life telegraph keys.

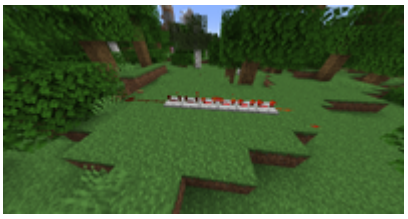
Button

The only situation that a button could be used is in a one-way telegraph system for distress signals to other players in multiplayer. However, the receiver would not remain activated due to buttons automatically returning to the 'off' position after only 0.9 seconds. It is also a small target to aim for, especially when a player is in the kind of situation that a distress signal seems like a good idea. So when it comes to telegraphs, use of buttons should probably be avoided entirely.

Pressure plate

Although they are similar to buttons in their general impracticality, the player could potentially utilize the pressure plate's unique ability to be activated by mobs. Multiple pressure plates could be positioned at different points of a cave, all of which could be connected to the one-way distress telegraph mentioned earlier. If any mob (only harmful mobs would spawn in a deep cave) stepped on one, it would trigger the distress signal, and players elsewhere would be alerted and be able to take action.

Receivers



A 6-bit receiver using redstone repeaters.

The receiver is the means of displaying a message sent from another telegraph room. They can be either redstone torches or redstone repeaters.

Redstone torch

The redstone torch is a primitive, but compact receiver, displaying each pulse as it arrives from the sender. It does not record, loop or display the entire message, and it cannot be interpreted easily without a solid understanding of the language being transmitted. It is most common in the one-way and classic telegraph systems, where simplicity and instantaneous viewing are most valuable. In addition, they are the only type of receiver that can be wall-mounted.

Redstone lamp

The addition of redstone lamps makes display a lot simpler by eliminating the need for an inverter.

Redstone repeater

Redstone repeaters, in their simplest form, can be used like the redstone torch receiver. However, they cannot be sent pulses as quickly or be mounted on walls, making them much more useful when placed in lines of two or more. This allows the receiving party to view a larger portion of the message as it is received, or, if the line is long enough, the entire message. The speed at which each pulse is displayed and lost can be adjusted by right-clicking each redstone repeater. This will increase or decrease the delay on each repeater before it passes on the signal. Such a system can be looped, so that the message is repeated to the receiving party indefinitely (note that looping a received message requires an erasing device and a message no longer than the receiver can display at one time).

In addition to being able to loop, redstone repeaters have the advantage of being able to be "locked" to a state. Thus, if a message is sent in a pre-configured pulse, it's possible to have a pulse preceding the message to lock the message in place in the repeaters, removing the need for a loop, and adding the possibility of a "timed eraser" if executed correctly. To lock a repeater in its current state, a repeater must power the side of it, then locking the current state.

Pistons

Pistons can be used in multiple bit receivers so that they look like dots and dashes.

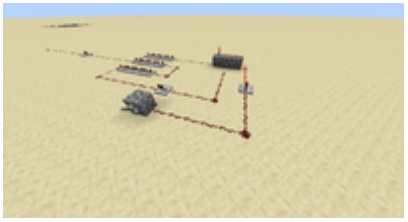
Note block

When coupled with an inverted rapid pulser, note blocks can be used for an auditory means of displaying messages (the inverted pulser is required in order to tell dots and dashes apart).

Message displays

Message displays are basically receivers are able to be viewed by the sender. Using these, one can view their message before it is sent to its destination. Either redstone torches or redstone repeaters can be used, usually corresponding to the type of receiver in the other telegraph room. When using a series of connected redstone repeaters, one can upgrade the display with a loop and eraser system.

Display loops and erasing devices



A 12-bit receiver with a loop and eraser system.

A display loop is a useful addition to both receivers and message displays. It connects the end of a redstone repeater display to its beginning, showing the message again and again until the eraser is activated. Erasing devices are an absolute necessity when creating a display loop. They end the looping message, and allow a new one to be shown before the two messages overlap pulses. A full loop and eraser system requires a customized NOR gate, through which the message is transferred and looped.

Erasers

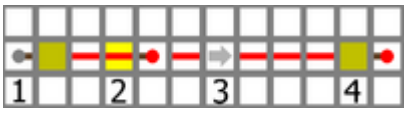
An erasing device, though it can be as simple as an exposed, destroyable piece of the loop, is generally a lever wired to the custom NOR gate. Although they do take extra space, receiver loops should be built so that the input is on the opposite side of the NOR gate, so that no wires should have to cross paths. Also, remember, you can always use multiple levels of wire to avoid other parts of the telegraph.

NOR gate

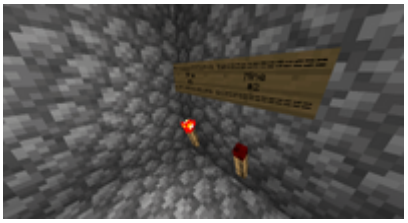
A NOR gate can be used in holding loops until you wish to send, delete or edit the message. You have to pull both levers back in order to activate it. Having both levers forward or one forward and one backwards will not activate the redstone torch.

Types of telegraphs

One-way telegraph



A one-way telegraph with a redstone torch receiver.



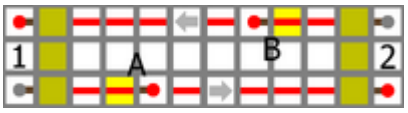
Example of one-way telegraph use.

As indicated by the name, messages created in a one-way telegraph may only be sent in one direction, greatly limiting its capabilities. The schematic shown demonstrates a short configuration, complete with the required sender (1), inverter (2), wiring (3) and receiver (4). However, it is the only kind of telegraph with any real practicality in single player. If one were to have the sender in a mine, and the receiver in their house, they could effectively use it like a post-it stamp, leaving a message in their house to remind them, for example, that there was diamond in that

mine. Expanding on this idea, they could have a series of redstone torches in their home indicating which of their mines had diamond or some other valuable element. Whenever another mine was discovered to have this element, they could simply flip a switch, and the mine's corresponding torch would activate in the house. An example of this is shown on the left. Another use would be as a simple distress signal in multiplayer, as it is always an issue of having to simultaneously type your call for help and run away from whatever is troubling you.

Classic telegraph

A typical classic telegraph room.

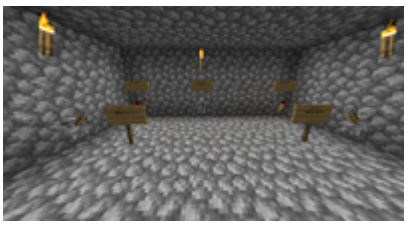


A two-way telegraph with control rooms on both sides.

The most basic system, the classic telegraph is quite easy to build, but not so easy to use. It consists of one lever, one inverter, and one redstone torch for each direction of communication. The schematic of a simple two-way telegraph is shown on the right. 1 and 2 are small control rooms, and A and B are inverters. Note that the area in between the two inverters can be elongated, with appropriately spaced redstone repeaters. The classic telegraph allows for a speedy construction and a space as small as 1x2x2 for each control room. In addition, due to its limited size and capability, it requires little to no knowledge of logic gates or circuitry, making it ideal for beginners or when you're low on redstone. The downside is that while you may be new to *Minecraft*, you have to be remarkably fluent in whatever language you decide to send the telegram in. This goes for the receiving party as well. In fact, they

must begin reading the message the instant you begin sending it, or risk losing its meaning entirely. This quickly becomes problematic, especially when the server is lagging. Despite this, players use it anyway, often because they enjoy the great similarity it holds to real telegraphs. So while the classic telegraph is compact, simple, and highly realistic, it may be worthwhile to look into a more advanced system.

Multi-directional telegraph



A three-way telegraph room with a 'send to all' switch.

Although the classification spans most telegraph types, the multi-directional telegraph is quite unique in its general design, and can be challenging to configure. This is mainly due to the amount of wiring required, and, in turn, the amount of space needed. For instance, a three direction telegraph requires a minimum of six separate lengths of wire, and that's just to create a classic one-torch receiver system, with no additional capabilities. While this is less of a problem over short distances, acquiring the massive quantities of redstone needed for such a project can be problematic, and it is good practice to make a rough estimate of how much you'll need before beginning its construction.

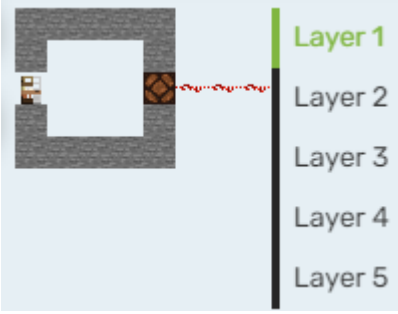
Simple multi-directional Morse telegraph

This is a really simple Morse code receiver and sender. The redstone lamp aligned near the lever shows what the sending is, and the lamp in the wall on layer 1 is the receiver. The signs are to show which telegraph room is receiving the message. Build a mirror of the structure to complete a multi-directional Morse telegraph sender

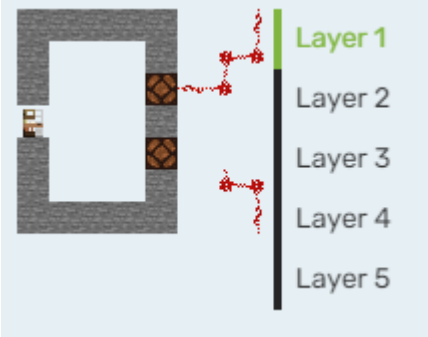
and receiver room. The redstone on the floor is the sender and the line on the slab is the receiver.

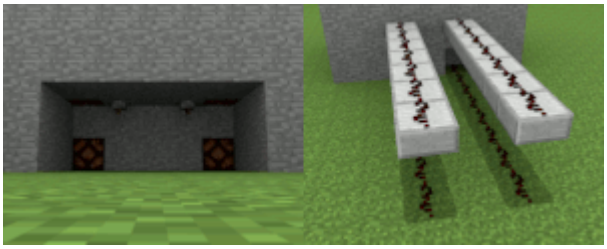
- Structure starts above ground
- Slabs are on upper section
- Levers hooked onto top wall

Single direction



Multi-directional

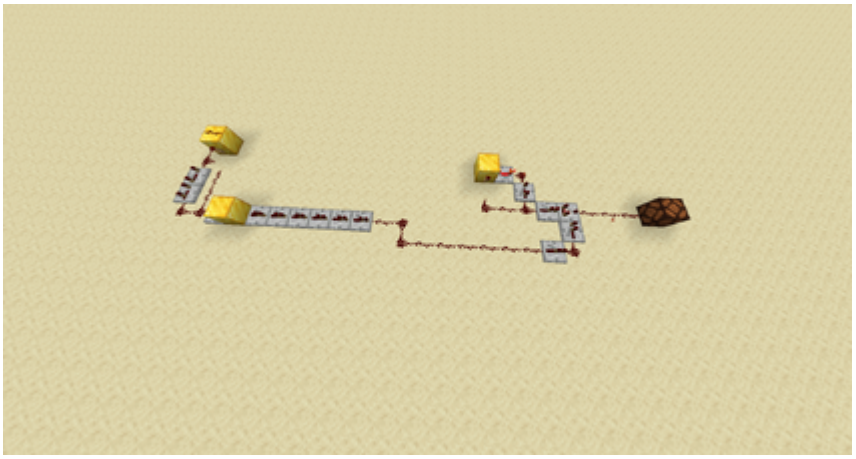




A multi-directional telegraph room in use from outside and inside

Alternative designs

Frequency matching



This device allows you to transmit different signals without any modification to the transmission line (it can remain a combination of redstone wires and repeaters). In this system, different frequencies represent different signals.

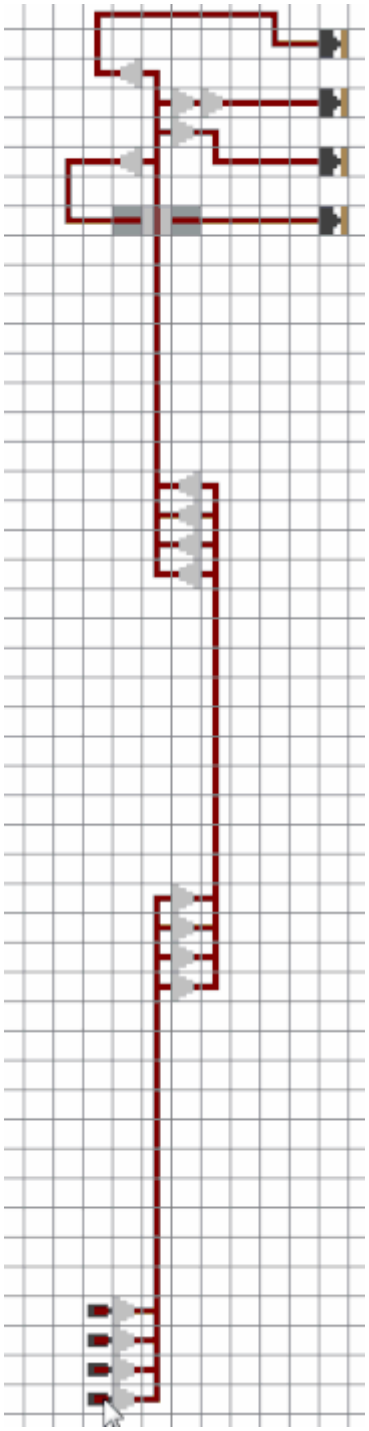
To build this circuit, you should build a transmitter — a simple oscillator — and attach it to the transmission line with a repeater. At the other end of the line, build another oscillator with the same delay. Configure the output of the transmission line in order to match the oscillators (they should turn on and off exactly at the same time, this is very important).

Then, connect the receiver oscillator and the output of the transmission line to a comparator in subtraction mode. If the received waveform is different from the receiver oscillator waveform, the comparator will generate a blinking signal at its output.

You can connect the transmission line to different receivers with different frequencies, and don't forget to match the timing.

The receiver circuit might get bulky, but this system allows you to spend much less redstone on the transmission line, if you have the transmitter and the receiver far away from each other.

Analog telegraph



Analog transmission line

Analog telegraph uses up to 16 states of a redstone wire, instead of just two.

Animation on the left is pretty self-explanatory: connect the repeaters as shown. The distance between the matching repeaters should be exactly 15.

Redstone clock telegraph

Steps

1. First you need to make a building, at smallest 16 blocks by 5 blocks, and if you want, you can make it look like a realistic building in real life, such as a post office.
2. Next, make a long hole in a wall. It must be 12 blocks long for a building 16 by 5 (If you made a building bigger, leave four blocks on the wall or make space somewhere else). You *must* have space for four blocks.
3. Place repeaters along the hole on full delay.
4. Hook up to a redstone clock (the bigger the better). A good way to save space is to go up and down repeatedly in rows. That will be the storer.
5. The next thing to build is the sender. Place a lever on one of the spaces. So that you can remember that this is the sender, it is recommended to place a sign on top of the lever saying sender.
6. After that, make a line of repeaters on full delay to the other telegraph office (A good way to do this is to go back and forth between solid blocks and repeaters). To make a turn, place a block, and then place a repeater facing towards the place. Hook up to the redstone the clock on the other one telegraph office.
7. To use the sender, do it in Morse code. To make a dot, turn the telegraph on and off as fast you can. To make a dash, turn on, count to three, and then turn off again.
8. The next thing to build is the deleter. Place blocks in the clock with sticky pistons touching them when on, so you can take them out of the clock stopping the pulse. (Make the sticky pistons so that it won't break things, but just take the block out.)
9. The final thing to build is the canceller. If you were to send a telegraph, make a mistake, and want to stop it from sending, just make a deleter. It's good to make plenty up the track, as with the clock.

Pros and cons

Pros

- Can leave messages.

- Fast.
- Easy to use.

Cons

- Needs lots of repeaters.
- Morse code needed.
- Lags a fair bit (A good way to stop this is to keep it well lit in the redstone clock.)
- Needs a large clock to do 7 letter words.

Tips

- Write the Morse code equivalent to each letter in the alphabet on a piece of paper.
- Place signs on top of levers.
- A nice touch is to hook note blocks up to the line, not the clock.
- Make more clocks to take more messages.

Redstone circuits/Transmission

A **transmission circuit** is a redstone circuit that allows redstone signals to move from one place to another.

Signal transmission

Redstone signals can be transmitted from one place to another with **redstone wire** – a line of redstone dust. Redstone wire can transmit a signal only 15 blocks – after that it needs a repeater to boost the signal back up to full strength.

Transmission crossing

When crossing each other, redstone wires must be kept isolated so they don't interfere with each other.

Redstone bridge

Redstone bridge

The center element consists of powered redstone dust on top of a block over unpowered redstone dust.

1×3×4 (12 block volume)

1-wide, instant, silent

circuit delay: none

The fastest method for crossing wires is by building a bridge to take one wire over the other.

Variations: A common variation is to drop the center block one level, and cut a three-block passage into the ground under it for the north-south wire.

Repeater bridge

Repeater bridge

2×3×3 (18 block volume)

silent

circuit delay: 1 tick

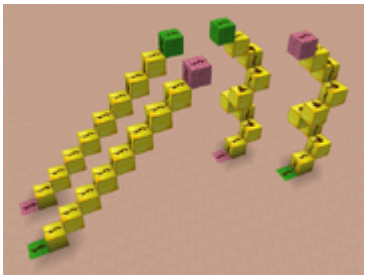
A repeater bridge takes up less vertical space than a redstone bridge, but it adds 1 tick of delay to both wires.

Vertical transmission

While horizontal transmission can be relatively straightforward, vertical transmission requires trade-offs.

Schematic Gallery: Vertical Digital Transmission [show] [edit]

Redstone staircase



Redstone Staircase – [schematic]

Up or Down

1×N×N

1-wide, silent

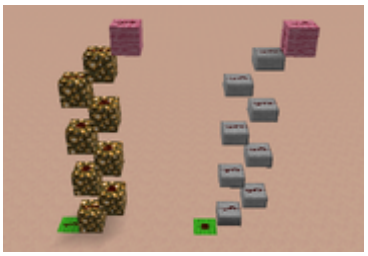
circuit delay: 1 tick per 15 blocks

Redstone dust propagates a signal to adjacent redstone dust one block up or down as long as no opaque block "cuts" the signal. This allows "staircases" of

blocks to carry redstone signals up (actual blocks of stairs aren't required, but can be used if placed upside-down).

Variation (Circular Staircase): By turning 90 degrees in the same direction each time the wire goes up a block, a "circular" staircase can be created in a 2×2 footprint. This variation is 2-wide tileable in both horizontal directions as long as the rotation direction is alternated in each direction (clockwise, anticlockwise, clockwise, etc.), or 2×4 alternating tileable with repeaters.

Redstone ladder



Redstone Ladder – [schematic]

Up Only

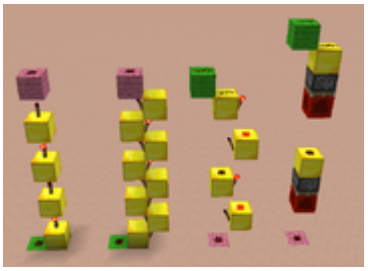
1×2×N

1-wide, silent

circuit delay: 1 tick per 15 vertical blocks

Transparent blocks that support redstone dust do not "cut" redstone dust, so "ladders" of these blocks can be made zig-zagging back and forth upward. Glowstone and upside-down slabs are the most commonly used supporting blocks, but upside-down stairs, glass, and hoppers also can be used. Redstone ladders are 2×2 alternating tileable for short runs, or 1×4 alternating tileable with repeaters.

Torch tower



Left: torch tower
Center Left: torch ladder
Center right: torch cascade
Right: piston tower
[schematic]

Up only

1×1×N

1-wide, silent

circuit delay: 1 tick per 2 vertical blocks

Redstone torches can power blocks above them, allowing transmission upward. Torch towers are 1×1 tileable.

Torch ladder

Up Only

1×2×N

1-wide, silent

circuit delay: 1 tick per vertical block

Redstone torches can power redstone dust beneath them, allowing transmission downward. Torch ladders are 1×2 tileable upward but 2×2 alternating tileable downward.

Torch cascade

Down Only

1×2×N

1-wide, silent

circuit delay: 1 tick per 2 vertical blocks

Piston tower

Down Only[*Java Edition only*], *Up or Down*[*Bedrock Edition only*]

1×1×N

1-wide

circuit delay: 1.5 ticks per 5 vertical blocks (rising edge) and none (falling edge)

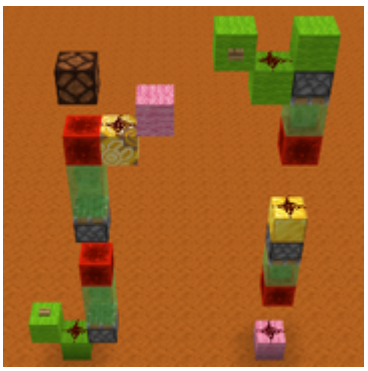
A sticky piston pointing downward can push a block of redstone into the space above redstone dust that is placed on top of a solid block. This can be repeated straight down, i.e. another sticky piston placed underneath that solid block, pointing downward, then another redstone block, a space, redstone dust on a solid block, and so on, allowing 1×1 downward transmission.

Because of the difference in rising and falling edge behavior, off-pulses are extended by 1.5 ticks per piston and on-pulses are shortened by 1.5 ticks per

piston and may possibly be erased altogether. This makes piston towers less useful for rapidly changing states.

An upward-directed transmission using pistons is not possible in *Java Edition* due to effects of quasi-connectivity, unless slime blocks are used to move the blocks of redstone.

Piston-slime block tower



Left: upward piston-slime block tower
Right: downward piston-slime block tower
[schematic]

Up or down

1×1×N

1-wide

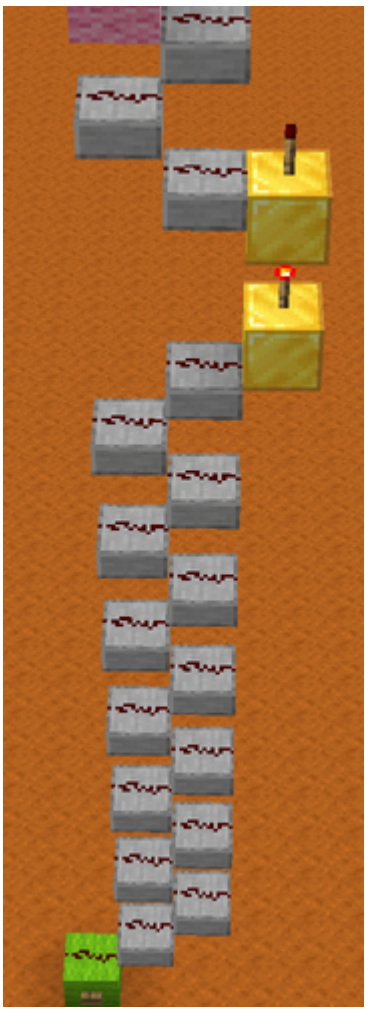
circuit delay: 1.5 ticks per piston (rising edge) and none (falling edge)

This is a variation of the simple piston tower that uses slime blocks.

Using slime blocks together with a basic piston tower may drastically improve its performance, because less pistons are used for the same height in comparison to a simple piston tower, and piston moves more blocks. Up to 11 slime blocks per piston can be used.

If using more than two slime blocks per piston, the piston tower can transmit a signal upward. A simple piston tower can't transmit signals up because effects of quasi-connectivity cause the tower to freeze in its state and not disable upon falling edge. If using only one slime block per piston, an upward piston tower turns into a block update detector.[*Java Edition only*]

Combined upward ladder



Combined Upward Ladder – [schematic]

Up Only

1×3×N

1-wide, silent

circuit delay: 2 tick per 17 vertical block

A vertical transmission circuit can be made by combining both torch tower and redstone ladder, resulting in a ladder with maximum height and minimum delay, as seen in the schematic. Additionally, the first torch can be moved to the middle replacing the top slab and the torch with redstone wire, and adding another top slab before the second torch, which adds one block of height (as seen in the picture).

Observer wire

Observers can also be used to create a vertical ladder. Each observer carries block updates to the next observer up or down. A more advanced observer wire can be

made by alternating, in upward order, observers, solid blocks, and either hoppers (only in Java Edition) or Droppers (in any edition). Each observer powers the dropper/hopper above it through the solid block, which changes its block state, activating the observer above it. The blocks can be removed to make a wire that is slower but does not power anything next to itself. Any of these can go horizontally or downward as well. There is also a more efficient downward method that goes (in downward order) observer, solid block, redstone dust, solid block, hopper/dropper.

Observers with walls/scaffolding

Starting from Java Edition 1.16, a Wall with a tall center post (due to a protruding wall block over it) causes all walls directly below it to also have a tall center post. In Java Edition, this change can be detected by observers, allowing for instantaneous downward transmission by moving a block over or away from the topmost wall.

Scaffolding provides the upward counterpart: Each scaffolding block tracks how far it is "overhanging", that is the distance from a column of scaffolding which is actually resting on a block. A trapdoor counts as support only when it is closed. Place a column of scaffolding atop a trapdoor, and also next to another scaffolding block that can support it while the trapdoor is open. When the trapdoor is closed, the column isn't overhanging at all, when the trapdoor is open it is overhanging 1 more space than its support scaffold. The difference propagates up the column, and can be detected by an observer.

(Credit for scaffolding technique: ianxofour via Youtube)

Observers with water

Water streams can carry information up and down. For a slow downward signal, a dispenser can place and pick up a water block (or even a lava block), and an observer can detect the changes in the flow downstream. For upward signals, a column of water source blocks can have the solid block below it swapped out by pistons; switching to a magma or soul sand block instantly propagates a bubble

column up the water column, while switching to a normal block removes the bubbles; an observer watching one of the water blocks detects these changes.

Falling items

A dropper can toss items down a shaft or into a water stream; the items can then land on a wooden pressure plate, or be picked up by a hopper and then detected by a comparator. It is also possible to launch items upward by various means, but that is generally less reliable, as controlling their exact path can be tricky.

Diode

Another important aspect of signal transmission is making sure a signal doesn't go the wrong way. A "diode" is a redstone component or circuit that allows signals through in one direction but not the other.

Component diode

Component diode

1×1×2 (2 block volume)

1-wide, flat, silent

circuit delay: 1 tick

Both the redstone repeater and the redstone comparator transmit signals in only one direction, but add 1 tick of delay.

Block diode

Block diode

1×2×2 (4 block volume)

1-wide, flat, silent

circuit delay: 1 tick

By strongly-powering a block, a signal can transmit in only one direction. None of the output lines can affect each other.

Transparent diode

Transparent diode

1×2×3 (6 block volume)

1-wide, instant, silent

circuit delay: none

Some transparent blocks can support redstone dust: hoppers, glowstone, upside-down slabs, and upside-down stairs. These blocks have the property that

redstone dust on them can propagate signals diagonally upward, but not diagonally downward (transparent blocks that cannot support redstone dust cannot be used for this purpose). Thus, simply jumping the signal up one block to one of these transparent blocks creates a diode circuit.

Upside-down slabs are the transparent block most commonly used for this purpose, but glowstone is used where light would be useful (to suppress mob spawning, etc.), upside-down stairs can be used where a full-side solid surface is required without light (for example, alongside a water channel transporting items over ice), and hoppers may be used in this way where they are already being used for item transport.

To get the output back to the same level as the input, run the line over an opaque block before dropping it.

Repeater

To "repeat" a signal means to boost it back up to full strength. When redstone signals are transmitted through redstone dust, their signal strength fades and must be repeated after 15 blocks. Repeater components and circuits keep signals going over long distances.

Redstone repeater

Redstone repeater

1×1×2 (2 block volume)

1-wide, flat, silent

circuit delay: 1 to 4 ticks

The most basic and common method of repeating a signal is to use a redstone repeater.

When transmitting signals over long distances, it is more efficient to use a block before and after the repeater – this method of repeating a signal averages 18 redstone used per 18 blocks (15 redstone dust, and 3 redstone per repeater) and 1 tick delay per 18 blocks.

Piston repeater

Piston repeater

1×3×2 (6 block volume)

1-wide, instant (falling edge)

circuit delay: 1 tick (rising edge), 0 ticks (falling edge)

A sticky piston can push a block into position to power the output.

Because of the differences in rising and falling edge delays, pulses are shortened by 1 tick per piston repeater (possibly erasing short pulses). On the other hand, zero falling edge delay makes this repeater attractive for applications that need instantaneous falling-edge transmission but do not care about rising-edge delay— for example, a distant circuit that activates on a falling edge.

A piston repeater cannot handle pulses shorter than 1.5 ticks [*Java Edition only*]; with shorter pulses, the block is left behind (not retracted by the sticky piston) and continues to power the output until a later input pulse ends (Note: A piston repeater *can* handle pulses shorter than 1.5 ticks in *Bedrock Edition*.)

Variations: When transmitting signals over long distances, it is more efficient to place a block before the piston. This method of repeating a signal averages 17 redstone used per 19 blocks (1 for the piston, 1 for the torch, and 15 redstone dust) and 1 tick delay per 19 blocks.

The moving block can be replaced with a block of redstone, which allows the removal of the lower block and redstone torch, reducing the circuit size to a 1-high 1×3×1 (3 block volume).

Double-torch repeater

Double-torch repeater

1×3×2 (6 block volume)

1-wide, silent

circuit delay: 2 ticks

The double-torch repeater was the standard repeater circuit used before redstone repeater blocks were added to *Minecraft*.

In transmission lines, one double-torch repeater is required every 18 blocks (the 3-block circuit, plus 15 blocks of redstone dust), using 18 redstone per 18 blocks and adding 2 ticks delay per 18 blocks.

Single-torch repeater

Single-torch repeater

1×2×1 (2 block volume)

1-high, 1-wide, flat, silent

circuit delay: 1 tick

When crossing long distances, redstone torches can be used singly, simply allowing the signal to be inverted an even number of times during its journey. Single-torch repeaters use slightly less redstone than redstone repeaters (16 redstone per 17 blocks) but are slightly slower (1 tick delay per 17 blocks).

Instant repeater

An **instant repeater** is a circuit that repeats a redstone signal change with no delay.

A sequence of instant repeaters and redstone dust lines is known as **instawire** (or "instant wire").

See also: Instant Two-Way Repeater (below) and Tutorials/Instant repeaters

Insta-drop instant repeater

Insta-drop instant repeater

1×3×2 (6 block volume)

1-wide, instant

circuit delay: none

Behavior (rising-edge): While the input is off, the block of redstone keeps the lower sticky piston activated by connectivity. When the input turns on, the upper sticky piston begins to extend the block of redstone. The instant the block of redstone starts moving, the lower sticky piston deactivates and begins to retract block A, the reason the upper piston is extending — this turns the upper sticky piston's extension into a 0-tick extension/retraction (the "insta-drop": the sticky piston "drops" its grip on the block and leaves it behind when it retracts), leaving the block of redstone above the lower sticky piston and powering the output. All of this happens instantly (in the same redstone tick), effectively allowing a rising edge to pass through the circuit with no delay. Now that the block of redstone is above the lower sticky piston, the lower sticky piston extends again, and two ticks later block A is back in position causing the upper sticky piston to extend again, ready to retract block A when the signal turns off.

Behavior (falling-edge): When the input turns off, the upper sticky piston begins to retract the block of redstone, immediately cutting off power to the output, effectively allowing the falling edge to pass through the circuit with no delay. While the block of redstone is moving, the lower sticky piston deactivates, but then activates again when the block of redstone stops moving and can activate the lower sticky piston by connectivity again.

Earliest known publication: February 14, 2013[1]

Dust-cut instant repeater

Dust-cut instant repeater

The space under the first piston prevents the block of redstone from activating its own piston.

1×5×4 (20 block volume)

1-wide, instant

circuit delay: none

Behavior (rising-edge): When the input turns on, the lower sticky piston begins to extend, causing the upper sticky piston to retract, allowing the powered redstone dust below block A to connect to the output. All of this happens instantly (in the same redstone tick), effectively allowing a rising edge to pass through the circuit with no delay. The moving block of redstone also instantly depowers the dust below it, but by the time that turns off the repeater's output, the block of redstone has arrived to continue powering the output.

Behavior (falling-edge): When the input turns off, the lower sticky piston begins retracting the block of redstone, immediately cutting off power to the output, effectively allowing the falling edge to pass through the circuit with no delay. The block of redstone then arrives at its retracted state and tries to power the output dust again, but it also powers the piston above it and block A arrives to cut the output before the repeater can output the signal from the block of redstone.

Variation (2-wide): The two upper levels (including the dust on top of the block the repeater is facing) can be moved one block over and down, and the last block on the lower level and its dust removed, to make a 2-wide version that is shorter in height and length (but larger in volume: 2×4×3, 24 block volume). In this version, to reduce the amount of redstone used, the block of redstone can be replaced with a regular block if redstone torches are placed under both its extended and retracted position.

Earliest known publication: January 3, 2013[2]

observer-rail instant repeater

observer-rail instant repeater

1×12×2 (24 block volume)

1-wide, silent, instant

circuit delay: none

Behavior: When the input changes at all, the observer powers the rail and instantly triggers the next observer, potentially triggering more rails. All of this happens in the same game tick. A shorter version will work as a diode. Inputs can be added by powering any rail in the chain, and outputs added by adding an observer anywhere.

Earliest known publication: 2016, using a piston BUD instead[3]

Two-way repeater

A **two-way repeater** (aka "2WR", "bi-directional repeater") is a circuit that can repeat a signal in either direction.

Two-way repeaters have two inputs that also act as outputs.

Typically the problem to be solved in design is repeating the signal in either direction without repeating the signal back into the same input, which could create a clock, a RS latch or a permanently-powered repeater loop.

Each circuit description below lists a **transmission speed**, the rate at which multiple circuits can transmit signals when placed at maximum distance from each other. Most circuits have their inputs offset from each other by one or two blocks – moving the wires in-line with each other reduces the transmission speed (because the signal has to move sideways to get to the correct input).

Current designs also have a **two-way reset time** – when input from one side is turned on, and then input from the other side is also turned on, then the first input turned back off, there is a short time while the transmission on the first side remains off until the circuit can reset itself to allow the second input through. Thus, the reset time can be seen as a spurious off pulse in a line that should be on.

Schematic Gallery: Two-Way Repeater [show] [edit]

Comparison two-way repeater

Comparison two-way repeater – [schematic]

2×5×2 (20 block volume)

flat, silent

transmission speed: 8 blocks/tick

circuit delay: 2 ticks

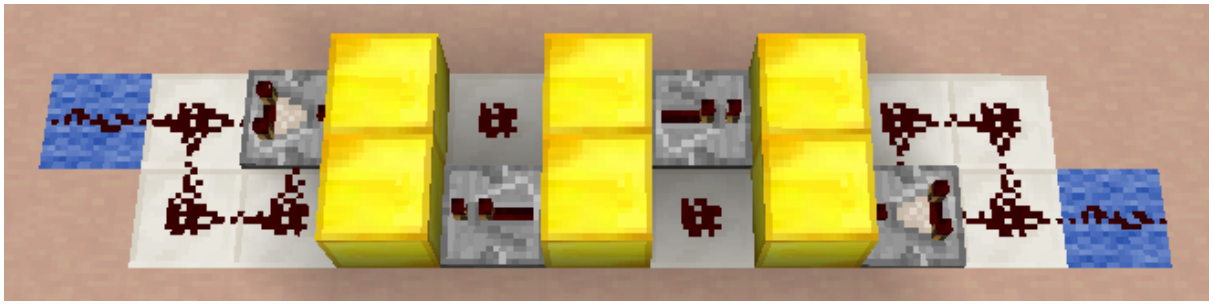
fastest clock signal: 2-clock

two-way reset time: 4 ticks

When a signal comes in from either side, it blocks the other input by providing a strength 15 signal to its comparator's side.

It is possible to override or block this circuit with additional inputs from the comparators' other sides.

Variations: Transmission speed can be increased by lengthening the circuit. Possibilities include placing opaque blocks before and after the repeaters, adding a segment of analog comparator wire before the repeaters, or using slab diodes to allow placing blocks before the comparators.

- 
- **Comparison 2WR**
  2×7×2 (28 block volume)
  9 blocks/tick
- 
- **Comparison 2WR**
  2×9×2 (36 block volume)
  10 blocks/tick
- 
- **Comparison 2WR**
  2×13×3 (78 block volume)
  10.5 blocks/tick

Earliest known publication: February 16, 2013[4]

CodeCrafted's two-way repeater

CodeCrafted's two-way repeater – [schematic]

2×6×3 (36 block volume)

silent

transmission speed: 9.5 blocks/tick

circuit delay: 2 ticks

fastest clock signal: 3-clock

two-way reset time: 3 ticks

The output on each side is produced by the redstone torch under the block, held unpowered by the input torch from the other side. When the other input signal turns on, the output torch turns on – this also turns off the input torch holding the other output torch off, but each output torch also holds the other output torch off, keeping the circuit from becoming permanently powered.

Variations: If it's not necessary to get the signal back to the lowest level (such as if this is built in a 1-deep hole), then this circuit can be considered to be 2×4×3 (24 block volume) and thus only four blocks long.

Earliest known publication: August 9, 2012[5]

Instant two-way repeater

Instant two-way repeater – There is redstone dust under the blocks of diamond, and 1-tick repeaters under the sticky pistons facing away from the bottom torches. [schematic]

4×4×3 (48 block volume)

instant

transmission speed: instant

circuit delay: 0 ticks

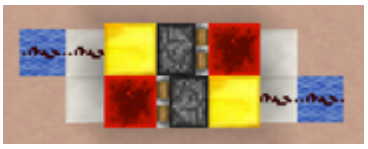
fastest clock signal: 2-clock

two-way reset time: 2.5 ticks

When an input turns on, it (a) turns off the torch on the side of the block and (b) powers the block in front of the input, activating the sticky piston on the other side. When the piston starts moving its block, this instantly allows the powered dust underneath to connect to the output. By the time the power from the torch and repeater have turned off, the block has arrived at its extended position where it connects the power from the other torch and repeater to the output.

Earliest known publication: February 18, 2013[6]

Moved-block two-way repeater



Moved-block two-way repeater – [schematic]

2×5×2 (20 block volume)

flat

transmission speed: 12 blocks/tick (18 blocks per 1.5 ticks)

circuit delay: 1.5 ticks (rising edge) and 0 ticks (falling edge)

fastest clock signal: 3-clock (but shortens pulses)

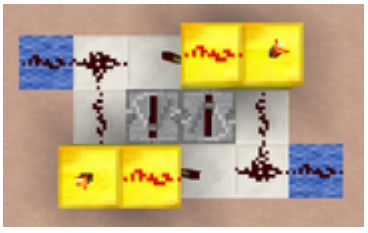
two-way reset time: 1.5 ticks

When an input turns on, a sticky piston pushes a block of redstone into position to power the other line, but that also reconfigures the dust on the other side to prevent it from powering the other sticky piston.

Because of the difference in rising and falling edge delays, pulses is shortened by 1.5 ticks per two-way repeater.

Earliest known publication: September 8, 2013[7]

Classic two-way repeater



Classic two-way repeater – [schematic]

3×4×3 (36 block volume)

silent

transmission speed: 8 blocks/tick

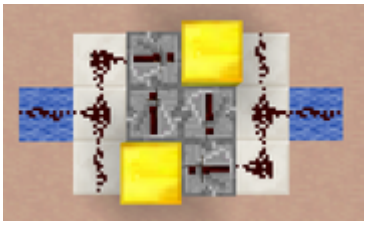
circuit delay: 2 ticks

fastest clock signal: 3-clock

two-way reset time: 4 ticks

This design offers few advantages over the other designs, but may be of historical interest.

Locked-repeater two-way repeater



Locked-repeater two-way repeater – [schematic]

3×4×2 (24 block volume)

flat, silent

transmission speed: 15 blocks/tick

circuit delay: 1 tick

fastest clock signal: 1-clock

two-way reset time: 3 ticks

When a signal comes in from either side, it blocks the other input with a repeater lock.

This circuit is locked in a permanent powered state if signals enter from both sides simultaneously.

Variation (offset input): The circuit shown in the schematic to the right keeps the transmission lines in-line with each other, but reduces the signal strength by 1 in side movement in both input and output before continuing the transmission, so the circuits must be placed with only 11 dust between them to work. Placing a block behind each of the input repeaters and moving the input/output lines closer to the repeaters' outputs means that signal strength is lost only in side movement at input, allowing an additional dust between the circuits (and thus a more

efficient transmission), but requires that the transmission lines alternate which side they run on.

Earliest known publication: December 21, 2012[8]

observer-rail instant two-way repeater

observer-rail instant two-way repeater

1×12×4 (48 block volume)

1-wide, silent, instant

transmission speed: instant

circuit delay: 0 ticks

fastest clock signal: 2-clock

two-way reset time: ?

Behavior: The powered rails initially hold powered. When any input is given, the rails act as a BUD and depowers, spreading the pulse both ways. All of this happens in the same game tick. Inputs can be added by creating a block update anywhere in the chain, and outputs added by adding an observer anywhere.

Variations: Can be coupled to a line of droppers for an instant item line (Inspector Talon, 2022).

Earliest known publication: Aug 26, 2021[9]

Transmission encoding

For simple redstone structures, digital ("on/off") transmission is sufficient.

For complex redstone structures, with banks of inputs or outputs, more sophisticated forms of transmission may be required, such as analog, binary, or unary transmission.

When numbers are represented by different types of transmission, they are said to be **encoded**.

Analog

An **analog** encoding in *Minecraft* (a.k.a. **hexadecimal wire** or simply **hex wire**) is a transmission that outputs the same signal strength it receives as input. Because

power levels can vary from 0 to 15, an analog transmission can convey 16 states in a single wire.

Analog vs. Digital in Real-Life

"Analog" means "continuously variable". This doesn't match *Minecraft* analog wires that have 16 discrete values (for example, a signal strength of 6.89 can't be in Minecraft). But a term was needed to differentiate between signal strength transmissions and on/off transmissions, and the real-life distinction between digital electronics (which generally transmit either a high voltage or low voltage) and analog electronics (which operate on continuously varying voltage levels) was a close fit, so the two terms were adapted for use by the *Minecraft* community.

Analog comparator wire (ACW)

Analog comparator wire

flat, silent

circuit delay: 1 tick per 4 blocks

The simplest analog wire is a line of redstone comparators. However, like redstone repeaters, comparators can draw a signal from an opaque block and push a signal into an opaque block, thus it is usually more efficient in resources and in signal delay to place comparators every four blocks. Hence, it is the best option for short distances and tricky turns.

The signal strength of an analog comparator wire (ACW) can be reduced or suppressed at some point along its length by feeding another signal into one of the comparators in subtraction mode. The signal can be overridden by feeding a stronger signal into one of the opaque blocks.

Because the redstone dust is not adjacent to any power or transmission components, only opaque blocks, it does not configure itself to point in any particular direction. This also causes the dust to power any opaque blocks or mechanism components to the side of the analog wire. Transmission components should not be placed adjacent to the wire's dust because that would cause the dust to configure itself in a way where it doesn't power the rest of the analog wire.

Earliest known publication: January 9, 2013[10]

Analog repeater wire

Analog repeater wire — One segment of analog repeater wire. The signal strength output at B is the same as the signal strength input at A.

flat, silent

circuit delay: 1 tick per 14 blocks

Signal strength can also be retained by using repeaters to repeat every possible signal strength at the correct distance from the output to convey the correct signal strength. It is the fastest option for long distances.

A single segment of analog repeater wire (ARW) consists of exactly 15 repeaters connecting an input line to an output line. To connect multiple segments together without additional comparators, the segments must be arranged so that the output dust of the last repeater is the same as the input dust of the next segment (i.e., block B of the previous segment is block A of the next segment). This causes the segments to overlap in distance by one block and causes each segment to be offset to the side from the previous segment by two blocks.

Variations: To keep the segments in-line, or to turn against the direction the repeaters are facing, raise the final repeater by one block and drop the next segment underneath it.

Another option is to use a comparator and an opaque block between the segments, and alternate the direction the repeaters are facing. This keeps the height to 2 blocks but increases the circuit delay to 2 ticks per 17 block.

Earliest known publication: November 21, 2012[11]

Analog subtraction wire

Analog Subtraction Wire — *Shown:* 10-state ASW (aka "decimal wire"). For a 10-state wire, the input signal strength must be between 6 and 15 (so should never actually be off). Chests are completely full with any items. The signal strength output at B is the same as the signal strength input at A.

flat, silent

circuit delay: 1 tick per (18-N) blocks (see below for N)

If fewer than 15 states need to be transmitted (for example, output from a picture frame, composter or cauldron), it may be more efficient to encode those N states in the higher levels of signal strength, and then repeatedly subtract the transmitted value from 15 after (17-N) dust, an even number of times. However, it is complicated and infrequently useful.

Variations: The chests can be replaced with any other full container. The chests can also be replaced with regular power components (redstone torches, powered levers, etc.) if the redstone dust next to them is raised or lowered by one block, or if the subtraction comparator and its power source are moved so that the redstone dust runs straight into the comparator's side with the comparator perpendicular to the line still facing into the same block.

Tap-anywhere analog comparator wire

Tap-anywhere analog comparator wire — A fragment of the line. The signal strength at every output is the same as the highest signal strength at any input.

flat, silent

circuit delay: 1 tick per 2 blocks + 2 ticks always

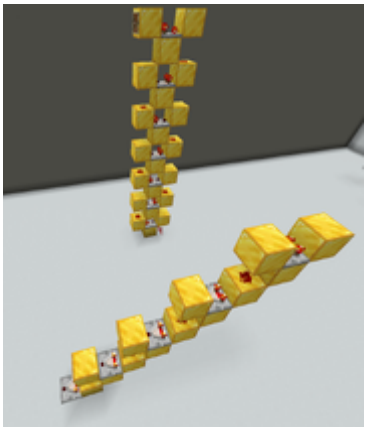
The tap-anywhere ACW is helpful with 1-tileable devices, both all requiring the same input, or collecting output from them (e.g. readout from a memory bank), sustaining the strongest of the inputs all the way "downstream" from it, on all the outputs.

Vertical analog transmission

The vertical options for analog transmission are similar to the horizontal options.

Vertical ACW

Vertical ACW



Vertical comparator wire diagonally and straight downward.

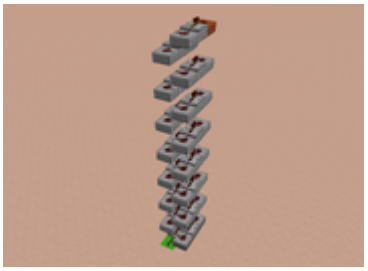
silent

circuit delay: 1 tick per 1 vertical block (up), 1 tick per 2 vertical blocks (down)

A redstone comparator can power a block with dust on it, and that dust can power another comparator at its level, etc. Vertical ACW travels two blocks sideways for every 1 block moved upward (or three blocks with an additional block between the dust and the comparator), but can also be bent at each block into a 3×3 "circular staircase".

The downward variant can go two blocks down for two block sideways, and double back, for a 1x3 tower.

Vertical ARW



Vertical ARW

silent

circuit delay: 1 tick per 14 vertical blocks

Vertical ARW is an analog repeater wire built on redstone ladders. It transmits signals only upward and in segments of 14 vertical blocks (use vertical ACW to close any gaps). Like horizontal ARW, the last dust of the previous segment must be the first dust of the next segment unless a short run of vertical ACW is used to connect the two segments.

Horizontal ARW built on a 3-wide staircase can be used to transmit analog redstone signal diagonally downward.

Vertical ASW basically just consists of redstone staircases or ladders with occasional breaks for subtraction.

Analog inverter

Analog inverter

An analog inverter is a circuit that inverts the signal strength, for example a signal with the strength of 6 becomes 9, and a full strength signal becomes 0. This can be achieved by placing a redstone block at the back of a redstone comparator and the input signal at the side of the comparator, with the output signal at front of the comparator, or by making an analog repeater wire, but instead of the wire in front of the repeater, place a block with a redstone torch attached to the front. place a wire in front of every torch, and the bottom-left redstone dust is the output.

Binary

A **binary** encoding consists of multiple digital lines run in parallel, with each line representing a different digit in a single binary number. For example, three lines might individually represent binary 001 (decimal 1), binary 010 (decimal 2), and

binary 100 (decimal 4), allowing them together to represent any value from decimal 0 to 7 (by summing the represented values of the powered lines). An individual digital line of a binary transmission is referred to by the value it can add to the total number (for example, the 1-line, the 2-line, the 4-line, the 8-line, the 16-line, etc.)

When a binary encoding is intended to carry only decimal numbers (encoding only values 0 to 9), it is known as binary-coded decimal (BCD).

4-bit binary encoding
: A 4-bit binary encoding contains the same amount of information as an analog line, i.e. 16 states. Compared to the analog version, binary lines are easier to transmit since no concerns about singal strenth is needed. They can also be handled with logic circuits.

  Converting between digital and analog, i.e. building a digital-analog converter (DAC), can be complicated.

8-bit and 16-bit
: 8-bit (a.k.a. "byte bus") and 16-bit binary encodings are also used in the construction of computer recreations.

One-hot

A **one-hot** encoding consists of multiple digital lines run in parallel, where a value is represented by which line is on (for example, the number 5 might be represented by having only the fifth line on, and all other lines off). A common alternative to one-hot encoding is **one-cold** encoding, where a number is represented by which line is *off* instead of which line is on.

While a strict one-hot encoding would always have precisely one line hot, a useful alternative is to eliminate the "0" line and represent the number 0 with no lines hot.

One-hot and one-cold encoding are rarely used for transmitting values over distance, but may be used for inputs (e.g., which button is pressed) or outputs (e.g., which dispenser is activated), with conversion to or from a more efficient transmission encoding in between. They are also used as a transitional encoding — for example,

it is relatively complicated to decode from analog to binary directly, but relatively simple to first decode from analog to one-cold, then encode from one-cold to binary.

Unary

A **unary** encoding consists of multiple digital lines run in parallel, where a value is represented by the number of lines on (for example, the number 5 might be represented by having 5 of 16 lines powered). A unary encoding might use a convention that lines must be powered starting from one side (allowing values to be determined from the transition from powered lines to unpowered lines) or that they can be powered in any combination (requiring logic circuits or calculations to determine the represented value).

Unary encoding is rarely used for transmitting values over distance, but may be used for inputs (e.g., the number of players standing on pressure plates) or outputs (e.g., the number of doors opened along a passageway).

Unary in Real-Life

"Unary" means "having only one element". Among other uses, the term is used for the unary numeral system, a way of representing numbers using only one digit, and for unary coding, a way of representing numbers using two digits but the second digit is used only to terminate numerals. In *Minecraft*, "unary" doesn't match either of these real-life uses exactly, but the term is still widespread.

Wireless transmission

Note: This section contains circuits built from command blocks, which cannot be obtained legitimately in Survival mode. These circuits are intended for server ops and adventure map builds.

Command blocks allow redstone signals to be transmitted to any loaded chunk, without a direct connection.

Setblock transmission

Setblock transmission works by using the setblock command to create and remove power components at a receiver.

For the setblock transmitters below, the two command blocks should be given commands to create and remove power components at the receiver location. For example, if X, Y, and Z are the absolute or relative locations of the setblock receiver:

- "on": `setblock X Y Z redstone_block`
- "off": `setblock X Y Z stone`

Other power components can be used to activate the receiver, but most require additional data to specify their orientation (for example, to specify the direction a lever is attached). Additionally, any non-power component can be used to deactivate the receiver (even air). Avoid using blocks that are transparent or produce light (like redstone torches) as that can increase lag due to block light calculations in up to hundreds of blocks around the receiver.

When a setblock command is executed, the affected block does not change until a half a redstone tick later (one game tick). Thus, the minimum circuit delay for setblock transmission is 0.5 ticks.

Setblock transmitter, redstone Torch

Redstone torch setblock transmitter

1×3×1 (3 block volume)

1-high, 1-wide, flat, silent

circuit delay: 0.5 ticks (rising edge) and 1.5 ticks (falling edge)

fastest clock: 3-clock

Because of the difference in rising and falling edge delays, on-pulses are lengthened by 1 tick and off-pulses are shortened by 1 tick.

Variations: Many other arrangements of the torch and command blocks are possible.

Setblock transmitter, repeater-torch

Repeater-torch setblock transmitter

1×3×3 (9 block volume)

1-wide, silent, tileable

circuit delay: 1.5 ticks

fastest clock: 3-clock

Unlike the redstone torch setblock transmitter, this transmitter doesn't change the lengths of input pulses.

Variations: The torch can be moved to the side of the input block, and the command blocks moved to the side of the repeater and the block it's facing, to make the circuit 2-wide but flat.

Setblock transmitter, piston

Piston setblock transmitter

1×3×2 (6 block volume)

1-wide, tileable

circuit delay: 2 ticks

fastest clock: 2-clock

Noisy, but small and can run on a faster input clock.

Variations: The command blocks can also be moved above or to the side of the block of redstone's positions. The piston can also be pointed downward (but not upward), with the command blocks alongside the block of redstone's positions.

Setblock transmitter, repeater-comparator

The hopper contains a single stackable item.

2×4×2 (16 block volume)

flat, silent

circuit delay: 1.5 ticks

fastest clock: 2-clock

Larger, but can handle a faster input clock without noise.

Setblock receiver

The stone is replaced by a block of redstone when activated.

1×1×1 (1 block volume)

1-high, 1-wide, flat, silent, tileable

A setblock receiver is simply a single block of space for a transmitter to create or remove a power component.

Scoreboard transmission

Scoreboard transmission works by setting values for scoreboard objectives. Scoreboard transmission can be used to transmit simple binary values (as shown below), but scoreboard objectives can store values between -2,147,483,648 and 2,147,483,647 (inclusive) and multiple scoreboard objectives can be active at once (though transmitting and receiving many values requires arrays of transmitters and receivers). A single scoreboard transmitter can activate multiple receivers at once and different transmitters can set the scoreboard objective to different values, activating specific sets of receivers and simultaneously deactivating all other receivers. Scoreboard receivers can also respond to *ranges* of values, instead of just specific values.

Scoreboard transmission requires the creation of a dummy scoreboard objective to store the transmission's current value ("WirelessBus01" is an example and can be anything):

```
/scoreboard objectives add WirelessBus01 dummy
```

Binary scoreboard transmitters are similar to setblock transmitters, except they require different commands (WirelessBusFakePlayer is an example and can be anything, but only one fake player is required for all wireless buses):

- "on": `scoreboard players set WirelessBusFakePlayer WirelessBus01 1`
- "off": `scoreboard players set WirelessBusFakePlayer WirelessBus01 0`

Unlike setblock receivers, scoreboard receivers must be run on clock circuits.

Scoreboard receiver, setblock clock

Setblock clock scoreboard receiver

1×3×3 (9 block volume)

1-wide, silent

circuit delay: 1 tick

The scoreboard receiver uses a fast clock to test the objective's value. The command blocks should have the following commands:

- R: `setblock ~ ~-1 ~ redstone_block`
- S: `setblock ~ ~1 ~ stone`
- W: `scoreboard players test WirelessBusFakePlayer WirelessBus01 1 1`

Variations: The setblock clock can be replaced with any other fast clock.

Summon transmission

Summon transmission works by summoning an item onto a wooden pressure plate.

Unlike setblock and scoreboard transmission, summon transmission doesn't require an "off" command block, depending only on the summoned item's despawn time to deactivate the receiver.

Redstone circuits/Pulse

A **pulse circuit** is a redstone circuit which generates, modifies, detects, or otherwise operates on redstone pulses.

Pulses

A **pulse** is a temporary change in redstone power that eventually reverts to its original state.

An **on-pulse** is when a redstone signal turns on, then off again. On-pulses are usually just called "pulses" unless there is a need to differentiate them from off-pulses.

An **off-pulse** is when a redstone signal turns off, then on again.

The **pulse length** of a pulse is how long it lasts. Short pulses are described in redstone ticks (for example, a "3-tick pulse" for a pulse that turns off 0.3 seconds after it turns on) while longer pulses are measured in any convenient unit of time (for example, a "3-second pulse").

The **rising edge** of a pulse is when the power turns on – the beginning of an on-pulse or the end of an off-pulse.

The **falling edge** of a pulse is when the power turns off – the end of an on-pulse or the beginning of an off-pulse.

Pulse logic

Pulse logic is a different approach to binary logic than standard redstone power binary (power present = 1, power absent = 0). In pulse logic, the pulse is a toggle of logic level of the contraption: (first pulse = 1, second pulse = 0). This approach allows implementing computational logic that operates not only on redstone signal, but also on block updates, and block positions; in particular implementation of mobile logical circuits in flying machines, and significant reduction of server-side lag through avoiding redstone dust, transporting signals through block updates instead - e.g. over Powered Rail. In many cases use of pulse logic also results in more compact circuitry, and allows building 1-tileable modules where classic redstone power would "spill" to the neighbor modules.

Conversion from classic redstone binary to pulse logic is performed through dual edge detectors, (usually just an Observer observing redstone dust or other power components), and conversion back is performed through T flip-flop circuits, in particular the block-dropping behavior of sticky pistons. That behavior is also utilized

as memory storage in pulse logic, position of the block encoding state of memory cell.

Pulse interactions

Some redstone components react differently to short pulses:

- In *Java Edition*, a piston or sticky piston usually takes 1.5 ticks to extend. If the activation pulse ends before this (because it's only 0.5 ticks or 1 tick long), the piston or sticky piston "aborts" – it places the pushed blocks at their pushed position and return to its retracted state instantly. This can cause sticky pistons to "drop" their block – they push a block and then return to their retracted state without pulling it back.
- A redstone comparator does not always activate when given a pulse of 1 ticks or less.
- A redstone lamp can be deactivated only by an off-pulse of minimum 2 ticks.
- A redstone repeater does increase the length of pulses which are shorter than its delay to match its delay (for example, a 4-tick repeater changes any pulse shorter than 4 ticks into a 4-tick pulse).
- In *Java Edition*, a redstone torch cannot be activated by pulses shorter than 1.5 ticks.

Pulse analysis

When building circuits, it can sometimes be helpful to observe the pulses being produced to confirm their duration or spacing.

Oscilloscope

1×N×2, flat, silent

An oscilloscope allows you to watch pulses as they move through the repeaters.

A pulse can be measured with 1-tick precision with an **oscilloscope** (see schematic, right).

An oscilloscope simply consists of a line of 1-tick repeaters (aka a "racetrack"). An oscilloscope should be constructed to be at least as long as the expected pulse, plus a few extra repeaters (the more repeaters, the easier it is to time capturing a pulse).

For periodic pulses (as from clock circuits), an oscilloscope should be at least as long as the clock period (both the on and off parts of the pulse).

An oscilloscope can be frozen to aid reading by:

- using an oak sign next to the design.
- positioning the oscilloscope on the screen so that it can be viewed when the player pauses the game, or
- taking a screenshot with F2, or
- running repeaters into the side of the oscilloscope and powering them simultaneously to lock the repeaters of the oscilloscope.

An oscilloscope is not capable of displaying fractional-tick pulses directly (0.5-tick pulses, 1.5-tick pulses, etc.), but for fractional-tick pulses greater than 1 tick, the pulse length may appear to change as it moves through the oscilloscope. For example, a 3.5-tick pulse may sometimes power 3 repeaters and sometimes 4 repeaters.

Half-tick pulses do not vary between powering 0 or 1 repeaters (they just look like 1-tick pulses), but half-tick and 1-tick pulses can be differentiated with a redstone comparator – a 1-tick pulse can activate a comparator, but a half-tick pulse cannot in most cases.

Multiple oscilloscopes can be laid in parallel to compare different pulses. For example, you can determine a circuit's delay by putting the circuit's input signal through one oscilloscope and the circuit's output through another and counting the difference between the input and output signal edges.

Oscilloscopes are useful but sometimes require you to be in an inconvenient position to observe them. If you just need to observe the simultaneity of multiple pulses it can be useful to use pistons or note blocks and observe their movement or note particles from any angle. Redstone lamps are less useful for this purpose because they take 2 ticks to turn off.

Monostable circuit

A circuit is **monostable** if it has only one stable output state ("mono-" means "one", so "monostable" means "one stable state").

A circuit's output can be powered or unpowered. If an output stays in the same state until the circuit is triggered again, that output state is called "stable". An output state that changes without the input being triggered is not stable (that doesn't necessarily mean it's random – it may be an intentional change after a designed delay).

If a circuit has only *one* stable output state then the circuit is called "monostable". For example, if a powered state inevitably reverts to the unpowered state, but the unpowered state doesn't change until the input is triggered.

When someone says "monostable circuit" in *Minecraft*, they usually mean a pulse generator or a pulse limiter. However, any redstone circuit which produces a finite number of pulses is technically a monostable circuit (all the circuits in this article, in fact, as well as some others), so instead of saying monostable circuit, it can be helpful to be more specific:

- A pulse generator generates a pulse
- A pulse limiter reduces the duration of long pulses
- A pulse extender increases the length of short pulses
- A pulse multiplier produces multiple output pulses in response to a single input pulse
- A pulse divider produces an output pulse after a specific number of input pulses
- An edge detector produces an output pulse when it detects a specific edge of an input pulse
- A pulse length detector produces an output pulse when it detects an input pulse of a specific length
- A block update detector produces an output pulse when a specific block is updated (for example, stone is mined, water turns to ice, etc.)
- A comparator update detector produces an output pulse when a specific comparator is updated by an inventory update

Clock circuits also produce pulses, but they aren't monostable because they have *no* stable output states (they are "astable") unless forced into one by external interference (for example, when they're turned off). Logic and memory circuits aren't monostable because *both* of their output states are stable (they are "bistable") – they don't change unless triggered by their input.

See also: Wikipedia:Monostable

Pulse generator

A **pulse generator** creates an output pulse when triggered.

Most pulse generators consist of an input and a pulse limiter. A pulse extender can be added on to generate a longer pulse.

Schematic Gallery: Pulse Generator [show] [edit]

On-pulse Generator

Circuit Breaker Pulse Generator

Circuit Breaker Pulse Generator – *Left:* Sticky piston. *Right:* Regular piston. [schematic]

1×3×3 (9 block volume), 1-wide

circuit delay: 1 tick

output pulse: 1 tick

The circuit breaker is one of the most commonly used pulse generator due to its small size and adjustable output.

Variations: The output repeater may be set to any delay, which also lengthens the output pulse to equal the delay. When oriented north-south, the output repeater may be replaced by any mechanism component, causing the mechanism component to receive a 0 tick pulse.

Observer Pulse generator

common observer pulse generator

1×1×3 (3 block volume), 1-wide, 1 high, tileable

circuit delay: 2 ticks

output pulse: 1 tick

The observer pulse generator is one of the most common pulse generators due to its adaptability. It can be oriented in almost any direction, and the observer can be oriented in almost any direction, allowing for lots of flexibility. And depending on where the output is taken from, it can be a rising or falling edge pulse generator. The observer can also be updated by other circuitry to send more pulses from the output.

> *Variations:* The piston base can be oriented in any way; the same is true for the observer except for facing the piston itself. Output can be taken from either the extended or retracted position to change which edge it activates on.

Dust-Cut Pulse Generator

Dust-Cut Pulse Generator – [schematic]

1×4×3 (12 block volume), 1-wide

circuit delay: 0 ticks

output pulse: 1.5 ticks when the output is a piston, 1 tick for everything else

A dust-cut pulse generator limits the output pulse by moving a block so that it cuts the output dust line.

NOR-Gate Pulse Generator

NOR-Gate Pulse Generator – [schematic]

1×4×3 (12 block volume), 1-wide, silent

circuit delay: 2 ticks

output pulse: 1 tick

A NOR-gate pulse generator compares the current power to the power from 2 ticks ago – if the current power is on and the previous power was off, the output torch flashes on briefly.

This design uses a trick to limit the output pulse to a single tick. A redstone torch cannot be activated by a 1-tick pulse from exterior sources, but a torch activated by a 2-tick exterior pulse can short-circuit itself into a 1-tick pulse. To increase the output pulse to 2 ticks, remove the block over the output torch. To then increase it to 3 ticks, increase the delay on the repeater to 4 ticks.

Locked-Repeater Pulse Generator

Locked-Repeater Pulse Generator – [schematic]

2×3×2 (12 block volume), flat, silent

circuit delay: 2 ticks

output pulse: 1 tick

When the lever is turned *off*, the locked repeater allows a pulse through.

Variations: The locked repeater and the input repeaters can be set to any delay. This increases the output pulse length, but also the circuit delay.

Comparator-Repeater Pulse Generator

Comparator-Repeater Pulse Generator - [schematic]

2×4×2 (15 block volume), flat, silent

circut delay: 1 tick

output pulse: 2 ticks

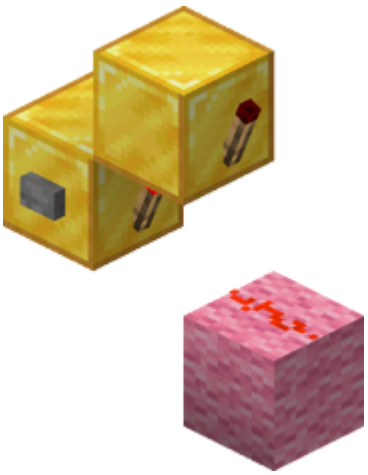
The dust first powers the comparator, turning on the output, then the delayed pulse (with the repeater) shuts off the output.

Variations: The repeater can be set to any number of ticks, increasing only the output pulse length.

Off-pulse Generator

An **off-pulse generator** has an output which is usually on, but generates an off-pulse when triggered.

OR-Gate Off-Pulse Generator



OR-Gate Off-Pulse Generator – [schematic]

1×3×3 (9 block volume), 1-wide, silent

circuit delay: 1 tick

output pulse: 1 tick (off)

When triggered, the bottom torch turns off, but the top torch doesn't turn on until 1 tick later, allowing a 1-tick off-pulse output.

Pulse length limiter

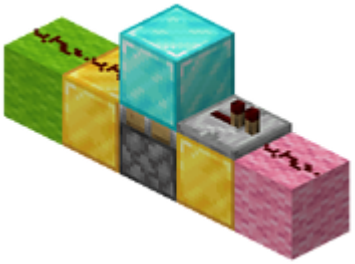
A **pulse limiter** (aka "pulse shortener") reduces the length of a long pulse.

An **ideal pulse limiter** would allow shorter pulses through unchanged, but in practice the range of input pulse can often be determined (or guessed) and it is sufficient to use a circuit which produces a specific pulse shorter than expected input pulses.

Any rising edge detector can also be used as a pulse limiter.

Schematic Gallery: Pulse Limiter [show] [edit]

Circuit Breaker Pulse Limiter



Circuit Breaker Pulse Limiter – [schematic]

1×3×3 (9 block volume), 1-wide

circuit delay: 1 tick

output pulse: 1 tick

The circuit breaker is the most commonly used pulse limiter due to its small size and adjustable output.

Variations: The output repeater may be set to any delay, which also lengthens the output pulse to equal the delay. The output repeater may be replaced by any mechanism component, causing the mechanism component to receive a 0.5-tick activation pulse.

Dust-cut pulse limiter

Dust-Cut Pulse Limiter – [schematic]

1×5×3 (15 block volume), 1-wide, instant

circuit delay: 0 ticks

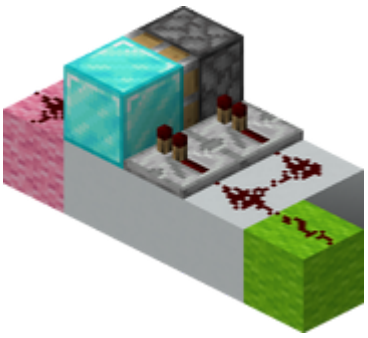
output pulse: 1.5 ticks

A dust-cut pulse limiter limits the output pulse by moving a block so that it cuts the output dust line.

The dust-cut pulse limiter doesn't "repeat" its input (boost it back up to full power), so a repeater may be needed before or after it (adding delay).

The dust-cut pulse limiter is an "ideal" pulse limiter (see above). Pulses shorter than 1.5 ticks (its maximum output pulse) are allowed through unchanged.

Moved-block pulse limiter



Moved-Block Pulse Limiter – [schematic]

3×3×2 (12 block volume), flat

circuit delay: 1 tick

output pulse: 1 tick

Uses the same principle as the circuit breaker pulse limiter – power the output through a block, then remove the block to keep the output pulse short.

Variations: The bottom repeater can be set to a longer delay to produce output pulses of 2 or 3 ticks. The repeater powering the piston can be replaced with a comparator to generate a 0-tick pulse

NOR-gate pulse limiter

NOR-Gate Pulse Limiter – (1-wide) [schematic]

NOR-Gate Pulse Limiter – *Top:* 1-tick. *Bottom:* Flat. [schematic]

features vary (see schematics)

A NOR-gate pulse limiter compares the current power to the power from 2 ticks ago – if the current power is on and the previous power was off, the output torch flashes on briefly.

The "1-wide" and "1-tick" designs use a trick to limit the output pulse to a single tick. A redstone torch cannot be activated by a 1-tick pulse from exterior sources, but a torch activated by a 2-tick exterior pulse can short-circuit itself into a 1-tick pulse. Remove the block over an output torch to increase the output pulse to 2 ticks.

Locked-repeater pulse limiter

Locked-Repeater Pulse Limiter – [schematic]

2×4×2 (16 block volume), flat, silent

circuit delay: 3 ticks

output pulse: 1 tick

Uses repeater locking to shut pulses off after 1 tick.

Variations: The output repeater can set to any delay. This increases the output pulse, but also increases the circuit delay.

If the input doesn't have to be at the same height as the output, you can move the torch so that it's attached to the top of the block it's currently above, and run the input into that block (making the circuit only 2×3×2).

Dropper-hopper pulse limiter

Dropper-Hopper Pulse Limiter – [schematic]

1×4×2 (8 block volume), 1-wide, flat, silent

circuit delay: 3 ticks

output pulse: 3.5 ticks

When the input turns on, the dropper pushes an item into the hopper, activating the comparator until the hopper pushes the item back.

The initial block is required to activate the dropper without powering it (which would deactivate the adjacent hopper, preventing it from returning the item to turn off the output pulse).

Because the output comes from a comparator used as an inventory counter, the output power level is 1 (with a stackable item) or 3 (with a non-stackable item) – add a repeater for a higher power level output.

Variations: If the input and output don't need to be at the same height, you can reduce the size of the circuit by putting the hopper on top of the dropper (making the circuit 1×3×2).

Off-pulse limiter

An **off-pulse limiter** (aka "inverted pulse limiter") has an output which is usually on, but which shortens the length of long off-pulses.

Any inverted falling edge detector can also be used as an off-pulse limiter.

OR-gate off-pulse limiter

OR-Gate Off-Pulse Limiter – *Top:* 1-tick. *Bottom:* Flat. [schematic]

OR-Gate Off-Pulse Limiter – Instant. [schematic]

features vary (see schematics)

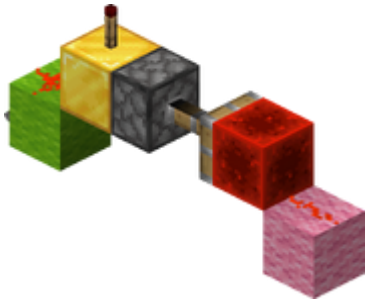
An or-gate off-pulse limiter combines the input with a delayed inverted input to limit off-pulses.

The "instant" version doesn't repeat its input (boost it back up to full power), so a repeater may be needed before or after it (adding delay).

Variations: The bottom repeater of the flat version can be adjusted to any delay, increasing the length of the off-pulse to match the repeater's delay (this doesn't actually increase the circuit delay).

The bottom redstone dust in the "instant" version can be replaced with a repeater to increase the length of its off-pulse.

Moving-block off-pulse limiter



Moving-Block Off-Pulse Limiter – [schematic]

1×4×2 (8 block volume), 1-wide, instant

circuit delay: 0 ticks

output pulse: 2.5 ticks (off, 3 if the output is a piston)

When the input turns off, the piston begins to retract. 1 tick later, the torch turns on, which re-activates the sticky piston by quasi-connectivity, causing it to extend again.

Pulse extender

A **pulse extender** (a.k.a. "pulse sustainer", "pulse lengthener") increases the duration of a pulse.

The most compact options are:

- Up to 4 ticks: Repeater
- Up to 4 ticks per repeater: Repeater-Line Pulse Extender
- 1 second to 4 minutes: Dropper-Latch Pulse Extender or Hopper-Clock Pulse Extender
- 5 minutes to 81 hours: MHDC Pulse Extender

Schematic Gallery: Pulse Extender [show] [edit]

Redstone repeater

1×1×2 (2 block volume)

1-wide, flat, silent

circuit delay: 1 to 4 ticks

output pulse: 1 to 4 ticks

For any input pulse shorter than its delay, a redstone repeater increases the duration of the pulse to match its delay. For example, a 3-tick repeater turns a 1-tick pulse or a 2-tick pulse into a 3-tick pulse.

Additional repeaters only delay the pulse, not extend it (but see the repeater-line pulse extender below).

Repeater-line pulse Extender

Repeater-Line Pulse Extender – *Top:* Delayed (1.4 second). *Bottom:* Instant (1 second). [schematic]

2×N×2

flat, silent, instant

circuit delay: 0 ticks (instant) or 4 ticks (delayed)

output pulse: up to 4 ticks per repeater

For the instant version, the input must be a pulse at least as long as the longest-delay repeater in the line (usually 4 ticks) – if not, use the delayed version.

Dropper-latch pulse extender

Dropper-latch pulse extender – [schematic]

2×6×2 (24 block volume)

flat, silent

circuit delay: 5 ticks

output pulse: 5 ticks to 256 seconds

Each stackable item, 16-stackable item and unstackable item in the middle hopper adds 8 ticks (0.8 seconds), 32 ticks or 256 ticks to the output pulse respectively. The output pulse can be fine-tuned by increasing the delay on the 1-tick repeater by up to 3 ticks, decreasing the delay on the 4-tick repeater by up to 3 ticks, or by replacing the 4-tick repeater with a block to decrease the delay by 4 ticks (these adjustments affect the *total* pulse duration, not per item, allowing pulse durations of any tick amount from 5 ticks to 256 seconds).

Variations: If the input pulse might be longer than half the output pulse, add a block before the dropper to keep it from deactivating the hopper. A 1-wide version is possible by using two droppers (adjustable only in increments of 8 ticks):

1-wide dropper-latch pulse extender

1×7×3 (21 block volume)
1-wide
circuit delay: 4 ticks
output pulse: 4 ticks to 256 seconds

The left dropper contains a single item and the left hopper contains one to 320 items.

Hopper-clock pulse extender

Hopper-Clock Pulse Extender – *Top:* 1-wide. *Bottom:* Flat. In both, the left piston is sticky and the right is regular. [schematic]

features vary (see schematics)

circuit delay: 1 tick

output pulse: 4 ticks to 256 seconds

A **hopper-clock pulse extender** is a hopper clock with one of the sticky pistons replaced with a regular piston so that it doesn't pull the block of redstone back, but instead wait for the input to trigger a new clock cycle.

A hopper-clock pulse extender with a single item in its hoppers produces a 4-tick output pulse. Each additional item adds 8 ticks to the output pulse (unlike the dropper-latch pulse extender, the output of a hopper-clock pulse extender can be adjusted only in 8-tick increments).

While waiting for the input to turn on, the sticky piston is actually in a state where it is powered but doesn't know it (like a stuck-piston BUD circuit) until "woken up" by the input changing its power level. This works as long as the input power level

is different than the resting output of the powered comparator (unintuitively, it even works if the input power level is *less* than the comparator output).

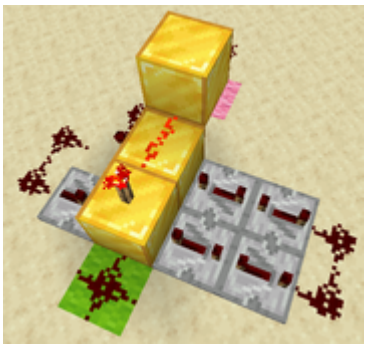
Caveats:

- Any block or redstone update near the powered & stuck sticky piston can trigger it, so care should be taken to keep other circuit activity away from the sticky piston.

- The timer/counter part starts after the rising edge of the input pulse. -> At worst the output pulse is only "*input_pulse + extender_time/2*" not "*input_pulse + extender_time*". For "*input_pulse < extender_time/2*" it's always just "*extender_time*".

Earliest known publication: 4 May 2013 CodeCrafted: "Minecraft QASI: Compact adjustable pulse extender" (based on the ethonian hopper clock)

RS latch pulse extender



RS NOR Latch Pulse Extender (3 seconds) – There is redstone dust under the raised block. [schematic]

features vary (see schematics)

output pulse: up to 8 ticks per repeater

An **RS latch pulse extender** works by setting the output on with a latch, then resetting the latch after some delay.

Both of the circuits below use a trick to double the delay produced by the repeaters, by first powering the output from the latch, then from the repeaters. This means that any 1-tick adjustment to the repeater loop produces a 2-tick adjustment in the output pulse.

Fader pulse extender

Fader Pulse Extender (6 seconds) – [schematic]

2×N×2

flat, silent

circuit delay: 0 ticks

output pulse: up to 14 ticks per comparator

The delay depends on the input's signal strength – for input signal strength S, the delay is (S-1) ticks per comparator. The signal strength of the output gradually decays, so should usually be boosted with a repeater. Because this uses comparators, this pulse extender does not work with most 1 tick or shorter pulses.

MHC pulse extender

MHC Pulse Extender – All pistons are sticky. [schematic]

6×6×2 (72 block volume)

flat

circuit delay: 3 ticks

output pulse: up to 22 hours

"MHC" stands for "multiplicative hopper clock" (a hopper counter multiplies the clock period of a hopper clock).

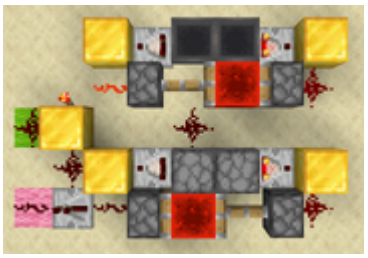
When the input turns on, the torch turns off, allowing both clocks to cycle into a state where the bottom clock continues to hold the torch off until it's completed one full cycle. The number of items in the top hoppers determines the top clock's cycle period, and its block of redstone moves every half-cycle, allowing the bottom clock to move one item.

The half-cycle is equal to the number of items in the top hoppers times 4 ticks (or 0.4 seconds per item) – up to 128 seconds for 320 items. The bottom clock keeps the output on for a number of half-cycles equal to twice the number of items in the bottom hoppers, minus 1. Thus, the output pulse equals 0.4 seconds × <top items> × (2 × <bottom items> - 1).

[show]

Items Required for Useful Output Pulses

MHDC pulse extender



MHDC Pulse Extender – All pistons are sticky. [schematic]

5×7×2 (70 block volume)

flat

circuit delay: 5 ticks

output pulse: up to 81 hours

"MHDC" stands for "multiplicative hopper-dropper clock" (a dropper counter multiplies the clock period of a hopper clock).

When the input turns on, the torch turns off, allowing both clocks to cycle into a state where the bottom clock continues to hold the torch off until it's completed one full cycle. The hoppers can hold up to 320 items (X) and the droppers can hold up to 576 items (Y). The duration of the output pulse is X × (2Y-1) × 0.8 seconds.

[show]

Items Required for Useful Output Pulses

Cooldown pulse extender

Note: This circuit uses command blocks which cannot be obtained legitimately in survival mode. This circuit is intended for server ops and adventure map builds.

Cooldown pulse extender — The dropper contains a single item.

1×4×2 (8 block volume)

circuit delay: 3 ticks

output pulse: up to 27 minutes

This pulse extender uses a command block to slow the hopper transfer rate. The exact command depends on the direction the pulse extender is facing, but for a

pulse extender facing the positive X direction it look something like this: `/data modify block ~2 ~ ~ TransferCooldown set value X`, where X is the number of game ticks (up to 32,767) to hold the item in the hopper (20 game ticks = 1 second, lag permitting).

When the command block is powered directly it activates the adjacent dropper, pushing the item into the hopper to power the output, and simultaneously changes the hopper's cooldown time to delay when it pushes the item back to the dropper.

Pulse multiplier

A **pulse multiplier** turns one input pulse into multiple output pulses.

There are three primary strategies for designing pulse multipliers:

- Split the input pulse into multiple paths that arrive at the output at different times
- Enable a clock to run while the input pulse is on
- Trigger a clock that runs for a finite number of cycles, independent of the input pulse length

In case the player requires only the pulse frequency to be doubled, usually a simple

dual edge detector is often sufficient:

Observer pulse doubler

Observer pulse doubler

1×1×1 (1 block), flat, silent, 1-tileable

circuit delay: 1 tick

output pulses: 2 1-tick pulses the length of input pulse apart.

An observer watching the input signal (redstone dust, button, repeater set to 1 tick, etc) produces a pulse on each of the edges of the input, producing two 1-tick pulses on each edge of the input pulse, providing the input pulse is sufficiently long (3 redstone ticks minimum). If the pulse is shorter than this, a redstone lamp can be put infront of the observer to remedy this issue.

Schematic Gallery: Pulse Multiplier [show] [edit]

Split-path pulse multiplier

A **split-path pulse multiplier** produces multiple pulses by splitting the input signal into multiple paths and having them arrive at the output at different times. This usually requires first reducing the length of the input pulse with a pulse limiter to reduce the delay required between each output pulse.

Dispenser double-pulser

Dispenser Double-Pulser – [schematic]

> *1×6×3 (18 blocks), 1-wide*
>
> *circuit delay: 1 tick*
>
> *output pulses: 1 tick and 2 ticks*

This circuit is useful for double-pulsing a dispenser, to quickly dispense then retract water or lava. First it powers a block on one side of the dispenser, then the other side.

Enabled-clock pulse multiplier

An **enabled-clock pulse multiplier** runs a clock for as long as the input stays on, thus producing a number of pulses relative to the input pulse length.

Subtraction 1-clock pulse multiplier

Subtraction 1-Clock Pulse Multiplier – [schematic]

> *2×3×2 (12 blocks), flat, silent*
>
> *circuit delay: 1 tick*
>
> *output pulses: 1 tick*

This pulse multiplier does not repeat its input signal, so may need a repeater before or after (increasing the circuit delay).

This circuit produces 5 pulses when enabled with a stone button, or 7 pulses when enabled with a wooden button. For other number of pulses, consider a pulse extender to lengthen the input pulse.

Subtraction N-clock pulse multiplier

Subtraction N-Clock Pulse Multiplier – [schematic]

2×3×2 (12 blocks), flat, silent

circuit delay: 1 tick

output pulses: 2+ ticks

The output pulses are 1 tick longer than the delay set on the repeater (so, 2 to 5-tick output pulses). For even longer pulses, replace the dust next to the repeater with another repeater.

This pulse multiplier does not repeat its input signal, so may need a repeater before or after (increasing the circuit delay).

The table below shows the number of output pulses produced with various combinations of button inputs and repeater delays (for more pulses, consider a pulse extender to lengthen the input pulse):

Repeater Delay	Stone Button	Wooden Button
1 tick	3 pulses	4 pulses
2 ticks	2 pulses	3 pulses

3 ticks	2 pulses	2 pulses
4 ticks	1 pulse	2 pulses

Torch-repeater N-clock pulse multiplier

Torch-Repeater N-Clock Pulse Multiplier – [schematic]

2×4×2 (16 blocks), flat, silent

circuit delay: 2 ticks

output pulses: 3+ ticks

The output pulses are 1 tick longer than the delay set on the repeater (so, 3 to 5-tick output pulses). The repeater can't be set to a 1-tick delay or the right torch burns out (which could be useful for limiting the number of pulses to 8 maximum).

Triggered-clock pulse multiplier

A **triggered-clock pulse multiplier** consists of a clock circuit that is allowed to run for a specific number of cycles once triggered. Strategies for designing a triggered-clock pulse multiplier include using a latch to turn the clock on and have the clock itself reset the latch back off after one or one-half clock cycles, or using a pulse extender to run a clock.

Dropper-latch 2-clock pulse multiplier

Dropper-Latch 2-Clock Pulse Multiplier – The top dropper contains a single item. The bottom dropper contains a number of items equal to the desired pulse count. [schematic]

3×4×2 (24 blocks), flat, silent

circuit delay: 3 ticks

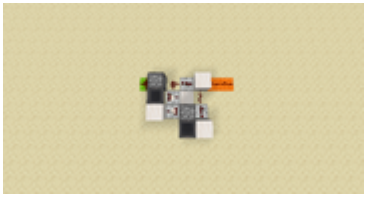
output pulses: 1 to 320 2-tick pulses

This pulse multiplier produce one 2-tick pulse for every item placed in the bottom dropper (with a 2-tick off-pulse between each on-pulse).

After it has finished its pulses, it requires a reset time equal to 0.4 seconds × pulse count. If it is reactivated during this time, it produces fewer pulses.

If the input pulse is longer than the output pulses, the powered dropper prevents the clock from turning off because the disabled hopper can't push its item back. If a long input pulse is possible, place a solid block between the input and the dropper so that it activates without being powered.

Earliest known publication: 4 September 2013[1]



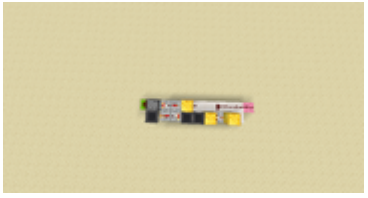
Dropper-Latch 2-Clock Pulse Multiplier (Updated) Added a repeater for the lower hopper to compensate and lock the items while active

As of 1.11, it the lower hopper needs a longer pulse from the clock.

To compensate, we add a repeater facing down to a block next to the, now below the dropper, hopper, and set it to 3 ticks.

If you want a longer clock, use the formula: $2n - 1$ where n is the clock pulse, for the delay of the lower repeater

Dropper-latch 1-clock pulse multiplier



Dropper-Latch 1-Clock Pulse Multiplier – The dropper contains a single item. The middle hopper contains one or more items depending on the desired pulse count (the first and last items should be non-stackable items). [schematic]

2×9×2 (36 blocks), flat, silent

circuit delay: 5 ticks

output pulses: 2 to 777 1-tick pulses

This pulse multiplier allows a wide range of pulses, with no reset time required.

The first and last items placed in the middle hopper should be non-stackable items (to give the output enough signal strength to run the subtraction clock). Up to three stacks of stackable items may be placed between the two non-stackable items.

The circuit produces four 1-tick pulses for every item placed in the middle hopper (with a 1-tick off-pulse between each on-pulse). The total number of pulses may be reduced by 1 by changing the 4-tick repeater to 2 ticks, or reduced by 2 by replacing the 4-tick repeater with a block, or increased by 1 by changing the 1-tick repeater to 3 ticks.

If the input pulse is longer than the output pulses, the powered dropper prevents the clock from turning off because the disabled hopper can't push its item back. If a long input pulse is possible, place a solid block between the input and the dropper so that it activates without being powered.

Pulse divider

A **pulse divider** (a.k.a. "pulse counter") produces an output pulse after a specific number of input pulses – in other words, it turns multiple input pulses into one output pulse.

Because a pulse divider must count the input pulses to know when to produce an output pulse, it has some similarity to a ring counter (an *n*-state memory circuit with only one state on). The difference is that a ring counter's output state changes only when its internal count is changed by an input trigger, while a pulse divider produces an output pulse and then returns to the same unpowered output it had before its count was reached (in other words, a pulse divider is monostable but a ring counter is bistable). Any ring counter can be converted into a pulse divider just by adding a pulse limiter to its output (making it monostable).

In addition to the circuits here, a clock multiplier can function as a pulse divider (or a ring counter, for that matter); unlike these circuits, its output remains ON until the next input pulse turns it off.

Schematic Gallery: Pulse Divider [show] [edit]

Hopper-Loop Pulse Divider – [schematic]

Hopper-loop pulse divider

2×(3 + pulse count/2)×3

output pulse: 3 ticks

This is a hopper-loop ring counter with an incorporated pulse limiter on the output.

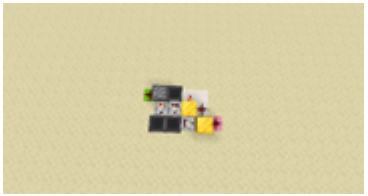
Each input pulse turns the redstone dust off for 1 tick, allowing the item to move to the next hopper. When the item reaches the dropper it turns on the output briefly, until the redstone dust turning back on activates the dropper to push the item to the next hopper.

To count an even number of pulses, replace another hopper with a dropper. Putting the second dropper right before the first dropper changes the output pulse to 6 ticks.

The output is signal strength 1 or 3 (with a stackable or non-stackable item in the hoppers) so may need to be boosted with a repeater.

Variations: Removing the dust from on top of the dropper and replacing the dropper with a hopper increases the output pulse to 4 ticks but makes the entire circuit silent.

Dropper-hopper pulse divider



Dropper-Hopper Pulse Divider – The dropper contains a number of items equal to the pulse count. The bottom-left hopper contains a single item. [schematic]

3×4×2 (24 block volume)

flat

output pulse: (4 × pulse count) ticks

The dropper-hopper pulse divider can count up to 320 pulses.

Each input pulse pushes an item from the dropper to the hopper next to it. When the dropper is finally emptied, its comparator turns off, allowing the item in the bottom-left hopper to move to the right, starting the reset process. When the top hopper has finished moving items back to the dropper, the item in the bottom hoppers moves back to the left, ending the reset process.

Once it has begun its output pulse, the pulse divider goes through a reset period of (4 × pulse count) ticks (the same length as the output pulse). Any new input pulses during the reset period is not counted, but only extends the reset period. Because of this reset period, this pulse divider is best when the typical interval between input pulses is greater than the reset period, or you can run a line back from the output to suppress inputs while it is resetting.

The output is signal strength 1 or 3 (with a stackable or non-stackable item in the bottom hoppers) so may need to be boosted with a repeater. The output pulse length is also proportional to the pulse count, so may need to be shortened with a pulse limiter.

Dropper-dropper pulse divider

Dropper-Dropper Pulse Divider – The left dropper contains a number of items equal to the pulse count. The left hopper contains a single non-stackable item. [schematic]

3×6×2 (36 block volume)

flat

output pulse: (2 × pulse count) ticks

The dropper-dropper pulse divider can count up to 576 pulses.

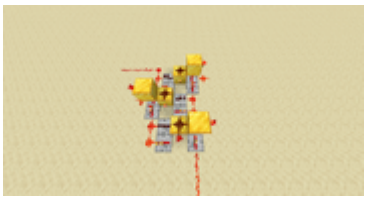
Each input pulse pushes an item from the left dropper to the right dropper. When the left dropper is finally emptied, its comparator turns off, allowing the item in the bottom-left hopper to move to the right, starting the subtraction 1-clock driving the reset process (although the subtraction clock pulses the dropper, the circuit's output alternates only in signal strength, staying on the whole time – subtraction clocks can be tricky that way!). When the right dropper has finished moving items

back to the left dropper, the item in the bottom hoppers moves back to the left, ending the reset process.

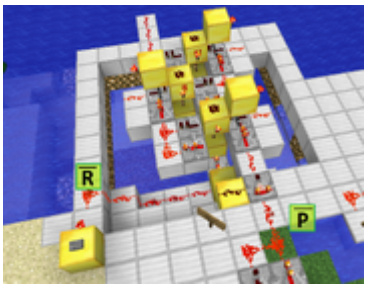
Once it has begun its output pulse, the pulse divider goes through a reset period of (2 × pulse count) ticks (the same length as the output pulse). Any new input pulses during the reset period is not counted, but only extends the reset period. Because of this reset period, this pulse divider is best when the typical interval between input pulses is greater than the reset period, or you can run a line back from the output to suppress inputs while it is resetting.

The output alternates between signal strength 1 and 3 so may need to be boosted with a repeater. The output pulse length is also proportional to the pulse count, so may need to be shortened with a pulse limiter.

Inverted binary divider or counter



Binary Counter (Tall) – Three dividers stacked to make an 8-counter. [schematic]



Binary Counter with Reset [schematic]

3×5×2 (30 block volume)

flat, silent, 3-wide stackable (alternating)

input: 2 off-ticks, use a pulse limiter if necessary

output pulse: 2 off-ticks

delay: 3 ticks (per unit in stack)

The inverted binary divider or counter uses the latching feature of redstone repeaters to create a two-state (binary) counter. Multiple counters can be stacked to construct an *n*-bit counter, giving 2^n input pulses per output pulse. It is called 'inverted' because it counts the number of *off* pulses, rather than on-pulses. Note that it triggers every two off-ticks, so holding the input low causes it to count

multiple times then burn out a redstone torch. You may want to use a pulse limiter on the input signal to prevent this.

Used purely as a pulse divider or counter this circuit is somewhat inefficient, since it would have to be stacked nine times to be able to count almost as many pulses (512) as the dropper-dropper divider. However, the stacking binary design means that the pulse count value can be easily read out by simply taking an output line from each stack element. In combination with OR or NOR gates, this can be used to trigger an output after an arbitrary number of pulses, or to create a divider for any number when combined with the reset circuit below.

'Tall' binary counter

2×5×3 (30 block volume)

silent, 2-wide stackable (alternating)

Functionally the same as the flat (3×5×2) binary counter, but takes one extra vertical block and one less horizontally, which may be an advantage when stacking them together. Requires an extra torch compared to the flat circuit.

Binary counter reset circuit

Adding this to the binary counter circuit allows it to be reset at any time; this can be used to create a counter for any desired number, or even a programmable counter (with extra circuits to select the number). This can be applied to either version, though the schematic shows it connected to the 'tall' version.

Like the counter itself, the reset circuit is *active low*; it requires at least three off-ticks to perform the reset, although the actual reset does not take place until the rising edge (end) of the off-pulse. (A standard button followed by an inverter works fine, as seen in the screenshot.)

1-tick binary counter/divider [*Java Edition only*]

1×3×2n+1 (1-tick input) or 1×3×2n+3 (for input longer than 1 tick)

1-wide, tileable

2^n divider

output pulse: 1-4 ticks

Binary 1-tick pulse divider (1/32 divider, 3 tick output example)

A cheap, noisy option to output 1 out of 2^n pulses (1 in 2, 4, 8, 16, 32 etc.), indefinitely extensible - each next module (repeater-piston pair) doubling the divider. Depends on the Java Edition's quirk of sticky pistons 'spitting out' their payload when activated with 1-tick pulses and quasi-connectivity. If the input pulse is longer than 1 tick, the first module acts as pulse limiter instead of a 'memory cell', so the only modification needed for this sort of input is adding one

more module vs 1-ticked input (e.g. from an observer). The output pulse can be extended up to 4 ticks by increasing the tick count on the last repeater.

Use as a binary counter requires reading position of the blocks moved by the pistons, e.g. through repeaters one block above the 'rest' position.

If the input has mixed length pulses, both 1-tick and longer, set the first repeater to 2 ticks and treat the first piston as pulse limiter, not counter module.

Edge detector

	Circuit	
Rising Edge Detector	On-pulse	n/a
Falling Edge Detector	n/a	On-pulse
Dual Edge Detector	On-pulse	On-pulse
Inverted Rising Edge Detector	off-pulse	n/a
Inverted Falling Edge Detector	n/a	off-pulse
Inverted Dual Edge Detector	off-pulse	off-pulse

An **edge detector** outputs a pulse when it detects a specific change in its input.

- A rising edge detector outputs a pulse when the input turns on.
- A falling edge detector outputs a pulse when the input turns off.
- A dual edge detector outputs a pulse when the input changes.

An **inverted edge detector** is usually on, but outputs an off-pulse (it turns off, then back on again) when it detects a specific change in its input.

- An inverted rising edge detector outputs an off-pulse when the input turns on.

- An inverted falling edge detector outputs an off-pulse when the input turns off.
- An inverted dual edge detector outputs an off-pulse when the input changes.

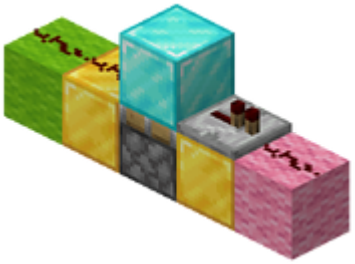
Rising edge detector

A **rising edge detector** (RED) outputs a pulse when its input turns on (the *rising edge* of the input).

Any rising edge detector can also be used as a pulse generator or pulse limiter.

Schematic Gallery: Rising Edge Detector [show] [edit]

Circuit breaker



Circuit Breaker – [schematic]

1×3×3 (9 block volume)

1-wide

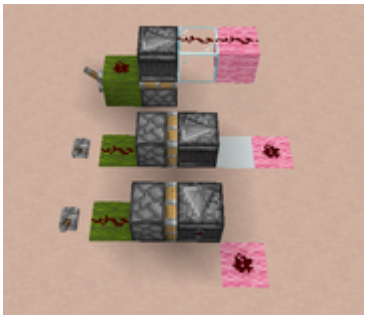
circuit delay: 1 tick

output pulse: 1 tick

The circuit breaker is one of the most commonly used rising edge detector due to its small size and adjustable output.

Variations: The output repeater may be set to any delay, which also lengthens the output pulse to equal the delay. When oriented north-south, the output repeater may be replaced by any mechanism component, causing the mechanism component to receive a 0-tick activation pulse.

Moved observer RED



Moved Observer RED variants (vertical, straight, angled)

1x1x3, 1x1x1, 1x2x2

1-wide, 1-wide flat, flat

circuit delay: Java: 2 ticks, Bedrock: 4 ticks

output pulse: 1 tick

This observer pulse edge detector is one of the most common edge detectors due to its modifyablity. It can be oriented in almost any direction, and the observer can be oriented in almost any direction, allowing for lots of flexibilty. And depending on where the output is taken from, it can be a rising or falling edge pulse generator. The observer can also be updated by other circiutry to send more pulses from the output.

Variations: The piston base can be oriented in any way, the observer can be oriented in any way except for facing the piston. Output can be taken from either the extended or retracted postion to change which edge it activates on.

Works both with standard binary and pulse logic. [*Java Edition only*]

Dust-cut rising edge detector

Dust-Cut RED (Unrepeated) – [schematic]

Dust-Cut RED (Repeated) – [schematic]

1×5×3 (15 block volume)

1-wide, instant

circuit delay: 0 ticks ("Unrepeated") or 1 tick ("Repeated")

output pulse: 1 tick, 1.5 ticks if the output is a piston

A dust-cut rising edge detector works by moving a block so that it cuts the output dust line after only one tick.

Because of the output's fractional length, a 1-tick repeater may be needed to force a sticky piston to drop its block.

Subtraction rising edge detector

Subtraction RED (Unrepeated) – [schematic]

Subtraction RED (Repeated) – [schematic]

2×4×2 (16 block volume)

flat, silent

circuit delay: 1 tick ("Unrepeated") or 2 ticks ("Repeated")

output pulse: 1 tick

A subtraction rising edge detector works by using the subtraction mode of a redstone comparator to shut off the output pulse.

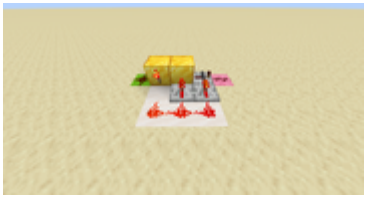
This design uses a trick to limit the output pulse to a single tick. A comparator can't produce a 1-tick pulse by subtraction from an exterior source (such as if the repeater was set to a 1-tick delay), but if the external source would usually produce a 2-tick pulse or more, the comparator can short-circuit itself into a 1-tick pulse by incorporating it into a subtraction 1-clock (the block and parallel dust after the comparator), but allowing the clock to run for only one cycle.

Variations: Remove the final block and dust to increase the output pulse to 2 ticks. Then increase the delay on the subtraction repeater to increase the output pulse length further.

Earliest known publication: 7 January 2013 (basic concept)[2] and 3 May 2013 (1-tick output refinement)[3]

Locked-repeater rising edge detector

Locked-Repeater RED (Corner) – [schematic]



Locked-Repeater RED (In-line) – [schematic]

2×4×2 (16 block volume)

flat, silent

circuit delay: 3 ticks

output pulse: 1 tick

Uses repeater locking to shut pulses off after 1 tick.

Variations: If the input doesn't have to be at the same height as the output, you can move the torch so that it's attached to the top of the block it's currently above, and run the input into that block.

Dropper-hopper rising edge detector

Dropper-Hopper RED – [schematic]

1×4×2 (8 block volume)

1-wide, silent

circuit delay: 3 ticks

output pulse: 3.5 ticks

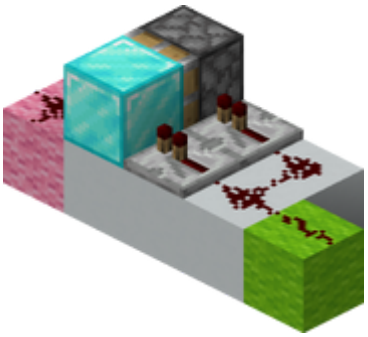
When the input turns on, the dropper pushes an item into the hopper, activating the comparator until the hopper pushes the item back.

The initial block is required to activate the dropper without powering it (which would deactivate the adjacent hopper, preventing it from returning the item to turn off the output pulse).

Because the output comes from a comparator used as an inventory counter, the output power level is 1 (with a stackable item) or 3 (with a non-stackable item) – add a repeater for a higher power level output.

Variations: You can reduce the size of the circuit by putting the hopper on top of the dropper.

Moved-block rising edge detector



Moved-Block RED – [schematic]

3×3×2 (18 block volume)

flat

circuit delay: 1 tick

output pulse: 1 tick

Uses the same principle as the circuit breaker – power the output through a block, then remove the block to keep the output pulse short.

Variations: To increase the output pulse length, increase the delay on the repeater powering the piston. To get a 0-tick pulse, replace the repeater powering the piston with a comparator

Other variations start with the piston powered. The output of the "offset" variation is weakly-powered and requires a repeater or comparator to do anything other than activate a mechanism component.

- 

  Moved-Block RED (In-line)
- 

  Moved-Block RED (Offset)

Earliest known publication: 14 March 2013[4] and 29 March 2013[5]

NOR-gate rising edge detector

NOR-Gate RED – [schematic]

1×4×3 (12 block volume)

1-wide, silent

circuit delay: 2 ticks

output pulse: 1 tick

A NOR-gate rising edge detector compares the current power to the power from 2 ticks ago – if the current power is on and the previous power was off, the output torch flashes on briefly.

All of these designs use a trick to limit the output pulse to a single tick. A redstone torch cannot be activated by a 1-tick pulse from exterior sources, but a torch activated by a 2-tick exterior pulse can short-circuit itself into a 1-tick pulse. Remove the block over an output torch to increase the output pulse to 2 ticks.

Falling edge detector

A **falling edge detector** (FED) outputs a pulse when its input turns off (the *falling edge* of the input).

Schematic Gallery: Falling Edge Detector [show] [edit]

Dust-cut falling edge detector

Dust-Cut FED – [schematic]

1×4×3 (12 block volume)

1-wide, instant

circuit delay: 0 ticks

output pulse: 2 ticks

When the input turns off, the piston immediately retracts the block, allowing the still-powered repeater to output a signal for 2 ticks. When the input turns on again, the piston cuts the connection before the signal can get through the repeater.

Moved-block falling edge Detector

Moved-Block FED – [schematic]

1×3×3 (9 block volume)

1-wide

circuit delay: 1 ticks

output pulse: 1 ticks

For some directions and input methods, the repeater may be needed to be set to 3 ticks to operate mechanism components.

Earliest known publication: 27 May 2013[6]

Moved-observer FED

Moved Observer FED - 1-wide

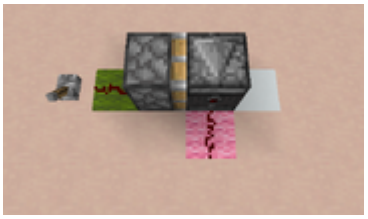
1×2×3 (6 block volume)

1-wide

circuit delay: Java: 2 ticks, Bedrock: 4 ticks

output pulse: 1 tick

This circuit uses a sticky piston and an observer to separate the rising from the falling edge of a signal. The rising edge powers the piston, lifting the observer above the redstone where it has no effect. Then, at the falling edge of the input signal, the piston retracts and the observer sends a 1-tick pulse via the redstone on the glass block. Note that the glass block is required to prevent this from turning into a clock.



Moved Observer FED - flat

Variations: The piston base can be oriented in any way, the observer can be oriented in any way except for facing the piston. Output can be taken from either the extended or retracted postion to change which edge it activates on.

Locked-hopper falling edge detector

Locked-Hopper FED – [schematic]

1×4×2 (8 block volume)

1-wide, silent

circuit delay: 1 tick

output pulse: 4 ticks

When the input turns off, it takes 1 tick for the torch to turn back on, giving hopper A a chance to push its item to the right and activate the output.

This circuit requires time to reset (to push the item back into hopper A), so the fastest input clock it can handle is a 4-clock.

Because the output comes from a comparator used as an inventory counter, the output power level is 1 (with a stackable item) or 3 (with a non-stackable item). Add a repeater for a higher power level output.

Variations: This circuit can be snaked around in many different ways as long as the input dust is able to deactivate the first hopper.

Earliest known publication: 22 May 2013[7]

Locked-repeater falling edge detector

Locked-Repeater FED – [schematic]

2×3×2 (12 block volume)

flat, silent

circuit delay: 2 ticks

output pulse: 1 tick

When the input turns on, the output repeater is locked before it can be powered by the block behind it. When the input turns off, the output repeater is unlocked and is briefly powered by the block behind it, producing a 1-tick output pulse.

Variations: Increase the delay on the output repeater to increase the output pulse length (up to 4 ticks), but also the circuit delay.

Subtraction falling edge detector

Subtraction FED – [schematic]

2×5×2 (20 block volume)

flat, silent

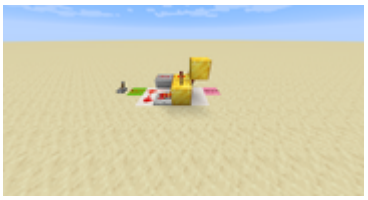
circuit delay: 1 tick

output pulse: 1 tick

This design uses a trick to limit the output pulse to a single tick. A comparator can't produce a 1-tick pulse by subtraction from an exterior source (such as if the repeater was set to a 1-tick delay), but if the external source would usually produce a 2-tick pulse or more, the comparator can short-circuit itself into a 1-tick pulse by incorporating it into a subtraction 1-clock (the block and parallel dust after the comparator), but allowing the clock to run for only one cycle.

Variations: Remove the final block and the dust next to it for a 2-tick pulse, then increase the delay on the repeater for a 3 or 4-tick pulse.

NOR-gate falling edge detector



NOR-Gate FED – [schematic]

2×4×3 (24 block volume)

silent

circuit delay: 1 tick

output pulse: 1 tick

This circuit compares the current power to the power from 2 ticks ago – if the current power is off and the previous power was on, the output torch flashes on briefly.

This designs uses a trick to limit the output pulse to a single tick. A redstone torch cannot be activated by a 1-tick pulse from exterior sources, but a torch activated by a 2-tick exterior pulse can short-circuit itself into a 1-tick pulse.

Variations: Remove the block over the output torch to increase the output pulse to 2 ticks, then increase the delay on the repeater to increase the output pulse further.

Observer FED / RED

Observer FED - flat – [schematic]

Observer FED - 1-wide

Observer FED - tileable

2x4x2, 1×4×3 or 1x5x3 (16, 12 or 15 block volume)

1-wide or flat, silent (tileable)

circuit delay: 2 or 5 rs-tick (afaik)

output pulse: 1 rs-tick

For some reason the observer in this circuit (only) triggers when the input turns off. It works in Java version 1.17.1 thou I'm not sure why (Redstone torch and repeater have the same delay, don't they?).

The "trick" is to keep the same distance between the observer to the repeaters output **and** the torch. Thus the power level in front of the observer always stays the same and yet the observer triggers on one of the two input changes.

Variations:

- Both flat and 1-wide versions allow for four different observer orientations.

- Increase the repeater delay of the non-tileable variants to turn this FED into a RED.

- The blue block in the tileable version is an inverted input to turn it into a RED.

(by Nilbadimo)

Dual edge detector

A **dual edge detector** (DED) outputs a pulse when its input changes (at either the rising edge *or* the falling edge of the input). The simplest way to do is using an observer.

Schematic Gallery: Dual Edge Detector [show] [edit]

Moving-block dual edge detector

The block of redstone moves when the signal turns on and when it turns off. While it is moving it cannot power the redstone dust, so the output torch turns on until the block of redstone stops moving.

In the 1-wide version the block over the output torch short-circuits it into a 1-tick pulse – remove the block and take the output directly from the torch to increase the output pulse to 1.5 ticks. To get an output on the same side as the input, the torch can be placed on the other side of the bottom blocks (but without the block above it, which would clock the piston). The piston and block of redstone can be moved to the side of the dust, rather than on top of the dust, producing a shorter but wider circuit.

Earliest known publication: 28 January 2013[8]

Dust-cut dual edge detector

features vary (see schematics)

The simple version splits the difference between a rising edge detector and a falling edge detector to produce an output of 1 tick on each edge. The instant version adds an unrepeated rising edge detector to reduce the rising edge circuit delay to 0 ticks.

Locked-repeater dual edge detector

features vary (see schematics)

A locked-repeater dual edge detector uses the timing of repeater locking to detect signal edges.

The nor-gate design uses a trick to limit the output pulse to a single tick. A redstone torch cannot be activated by a 1-tick pulse from exterior sources, but a torch activated by a 2-tick exterior pulse can short-circuit itself into a 1-tick pulse.

Remove the block over the output torch (and the dust on the block it's attached to) to increase the output pulse to 3 ticks.

Earliest known publication: 16 April 2013 (NOR-gate locked-repeater FED)[9] and 1 May 2013 (OR-gate locked-repeater FED)[10]

Piston OR-gate dual edge detector

3×4×2 (24 block volume)

flat

circuit delay: 1.5 ticks

output pulse: 1.5 ticks

A piston OR-gate dual edge detector moves a block between repeaters that change states shortly after the piston moves. This causes a pulse to be sent to a wire behind the moving block.

Subtraction dual edge detector

features vary (see schematics)

A subtraction dual edge detector powers a comparator with an ABBA circuit, cutting the pulse short with subtraction.

Earliest known publication: 3 August 2013[11]

Twin NOR-gate dual edge detector

The most trivial way to build a dual edge detector is to OR the outputs of a NOR-gate rising edge detector and a NOR-gate falling edge detector. A useful feature of this approach is that you get the rising- and falling-only pulses for free if you need them. If rsource or space usage is more important than timing, parts of the components of the 2 single edge detectors can be shared (the middle row of the example in the Schematic Gallery: Dual Edge Detector). Again, the blocks above the torches limit the output pulse to 1 tick.

Inverted rising edge detector

An **inverted rising edge detector** (IRED) is a circuit whose output is usually *on*, but which outputs an off-pulse on the input's rising edge.

Schematic Gallery: Inverted Rising Edge Detector [show] [edit]

OR-gate inverted rising edge detector

OR-Gate IRED – [schematic]

1×3×3 (9 block volume)

1-wide, silent

circuit delay: 1 tick

output pulse: 1 to 3 ticks (off-pulse)

An OR-gate inverted rising edge detector compares the current and previous input – if the current input is on and the previous input was off, the output turns off for a brief period.

Variations: The "adjustable" version takes up the same space, but its output pulse can be adjusted from 1 to 3 ticks. The "flat" version can also be adjusted from 1 to 3 ticks.

- 

  OR-Gate IRED (Adjustable)
- 

  OR-Gate IRED (Flat)

Earliest known publication: 1 June 2013[12]

Moving-block inverted rising edge detector

Moving-Block IRED – [schematic]

1×4×3 (12 block volume)

1-wide, instant

circuit delay: 0.5 ticks

output pulse: 1 tick (off-pulse)

This is a moving-block inverted dual edge detector with a repeater added to suppress the output on the falling edge.

Dropper-hopper inverted rising edge detector

Dropper-hopper IRED – The dropper contains a single item.

1×3×3 (9 block volume)

1-wide, silent

circuit delay: 3 ticks

output pulse: 4 ticks (off-pulse)

When the input turns on, the dropper pushes the item up into the hopper, deactivating the comparator until the hopper pushes the item back down.

The initial block is required to activate the dropper without powering it (which would deactivate the adjacent hopper, preventing it from returning the item to turn the output pulse back on).

Because the output comes from a comparator used to measure inventory, the output power level is 1 (with a stackable item) or 2 (with a non-stackable item) – add a repeater for a higher power level output.

Variations: The input block can be moved to the side of or underneath the dropper, and the hopper can be moved to the side of the dropper.

Inverted falling edge detector

An **inverted falling edge detector** (IFED) is a circuit whose output is usually *on*, but which outputs an off-pulse on the input's falling edge.

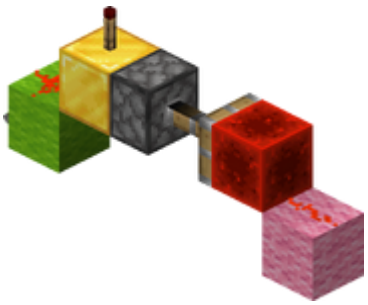
Schematic Gallery: Inverted Falling Edge Detector [show] [edit]

OR-gate inverted falling edge detector

features vary (see schematics below)

The input has two paths to the output, timed so that the output blinks off briefly when the input turns off.

Moved-block inverted falling edge detector



Moved-Block IFED – [schematic]

1×4×2 (8 block volume), 1-wide, instant

circuit delay: 0 ticks, output pulse: 2.5 ticks (off-pulse)

Earliest known publication: 4 June 2013[13]

Locked-repeater inverted falling edge detector

2×3×2 (12 block volume), flat, silent

circuit delay: 2 ticks, output pulse: 1 tick (off-pulse)

When the input turns on, the output repeater is locked before it can turn off. When the input turns off, the output repeater is unlocked and is briefly un-powered by the block behind it, producing a 1-tick output off-pulse.

Inverted dual edge detector

An **inverted dual edge detector** (IDED) is a circuit whose output is usually *on*, but which outputs an off-pulse when its input changes.

Schematic Gallery: Inverted Dual Edge Detector [show] [edit]

Moving-block inverted dual edge detector

1×3×3 (9 block volume), 1-wide, instant

circuit delay: 0 ticks, output pulse: 1.5 ticks (off-pulse)

Variations: The piston and block of redstone can be moved to the side of the dust, rather than on top of the dust, producing a flat 2-wide circuit.

The sticky piston can be oriented vertically if the redstone dust is run around the side in a 2×2×4 configuration.

OR-gate inverted dual edge detector

3×4×2 (24 block volume), flat, silent

circuit delay: 2 ticks, output pulse: 3 ticks (off-pulse)

Uses the timing of repeater locking to detect pulse edges.

Slime BUD inverted dual edge detector

1×3×4 (12 block volume)

circuit delay: instant, output pulse: 1 tick (off-pulse)

The Slime BUD made possible by Minecraft 1.8 works great as an instant inverted dual-edge detector. Simply put a block of obsidian, a hopper, afurnace, etc. right next to the slime block, and run redstone from its top to your output, and put a piece of redstone dust on the same plane as the piston, with one block space between. That's your input.

Variations: move the obsidian (or whatever you used) -- and the redstone on top of it -- up one block to get a normal (non-inverted) dual edge detector, but with 1.5 ticks delay.

Pulse length detector

Sometimes it is useful to be able to detect the length of a pulse generated by another circuit, and specifically whether it is longer or shorter than a given value. This has many uses, such as special combination locks (where the player have to hold down the button), or detecting Morse code.

Long pulse detector

Long pulse detector

2×6×3 (36 block volume)

silent

To test for a long pulse, we use an AND gate between the beginning and end of a line of redstone repeaters. These allow the signal to pass through only if it has a signal length longer than the delay of the repeaters. A pulse that does get through is *shortened* by the delay amount, possibly down to 1 tick.

Long pulse detector

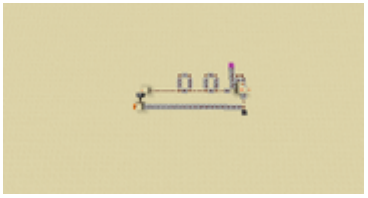
Long pulse detector

2×5×2 (20 block volume)

flat

Similar to the design above, but using a piston-based AND gate which shuts off the output as soon as the input turns off.

Pulse length differentiator



Input at gray wool, short output at orange wool, long output at purple wool.

A pulse length differentiator has two outputs and one input. Long pulses go through one output, while short pulses go to the other. It also keeps the tick length of the signals, which is why all the repeaters are set to one tick (i.e., a 1-tick signal remains a 1-tick signal). This is useful in a telegraph machine, in order to split up dashes and dots.

Transports and Logic gates implemented in Pulse logic

Some basic circuits exploiting the pulse logic. See the reference link for more advanced use of pulse logic circuitry.[14]

Rail update transport

Transport line in pulse logic

1-tileable

Typically, in pulse logic circuitry, signal is sent over Powered Rail or Activator Rail. Since the two don't propagate the updates to each other, this allows for tight tiling of modules.

NOT gate

NOT gate in pulse logic

1-tileable

Negation of signal depends only on initial position of blocks, or often - only on interpretation of the signals by the creator.

AND gate

AND gate in pulse logic

1-tileable

OR gate

OR gate in pulse logic

1-tileable

The OR gate in pulse logic differs from AND gate only by initial positions of the blocks.

XOR gate

XOR gate in pulse logic

1-tileable

Generic redstone OR in pulse logic acts as XOR.

Leaf block update transport

Leaf block update transport

1-wide

The "greenstone" or "leafstone" transport depends on updates of leaf blocks depending on changing distance from the nearest log block. This transport is particularly helpful in transporting signal upward and downward. Updates do propagate to neighboring blocks though, and take 1 game tick to progress to next block. It makes it useful in creating 1 gametick resolution timing source though.

Scaffolding block update transport

Scaffolding update transport

The Scaffolding propagates updates containing distance from supported scaffolding block. By moving a block under a suspended section of scaffolding, the player can send a signal an arbitrary distance upward and up to six blocks horizontally in any direction. The signal propagates at 1 block per redstone tick.

Wall block update transport

Wall update transport (side view)

1 tick regardless of distance, 1-tileable (see caveat)

Wall blocks (cobblestone wall etc.) instantly transmit signal arbitrary distance down by turning themselves and all wall block below from smooth wall segment to a pillar segment if certain blocks are placed on them or attached from a side. To form a smooth segment, a wall needs two other wall blocks or other blocks wall can attach to, adjacent to it from two opposite sides. If they are other wall blocks though, it doesn't matter if they are smooth or pillars - so the solution is 1-tileable, but requires uninterrupted columns of full blocks (or wall) on far ends. Probably the most practical way to toggle a wall between these states is a redstone-controlled trapdoor. The readout through an observer is possible only from below though, as the wall connects to an observer from a side.

Redstone circuits/Memory

Note: This page uses many schematics, which are loaded individually for performance reasons. [Schematic Help]

Latches and flip-flops are effectively 1-bit memory cells. They allow circuits to store data and deliver it at a later time, rather than acting only on the inputs at the time they are given. As a result of this, they can turn an impulse into a constant signal, "turning a button into a lever".

Devices using latches can be built to give different outputs each time a circuit is activated, even if the same inputs are used, and so circuits using them are referred to as "sequential logic". They allow for the design of counters, long-term clocks, and complex memory systems, which cannot be created with combinatorial logic gates alone. Latches are also used when a device needs to behave differently depending on *previous* inputs.

There are several basic categories of latches, distinguished by how they are controlled. For all types, the input lines are labeled according to their purpose (**S**et, **R**eset, **T**oggle, **D**ata, **C**lock). There are also more arbitrary labels: The output is commonly labeled for historical reasons. Sometimes there is also an "inverse output" , which is always ON when is OFF and vice versa. If both and are available, we say the circuit has "dual outputs". Most of the following types can be built as a "latch" that responds to the *level* of a signal, or as a "flip-flop" triggered by a *change* in the signal.

- A RS latch has separate control lines to **s**et (turn on) or **r**eset (turn off) the latch. Many also have dual outputs. The oldest form of RS latch in *Minecraft* is the RS-NOR latch, which forms the heart of many other latch and flip-flop designs.
- A T latch has only one input, the toggle. Whenever the toggle is triggered, the latch changes its state from OFF to ON or *vice versa*.

        - There are also SRT latches, combining the inputs and abilities of the RS and T latches.
    - A D latch has a **d**ata input and a **c**lock input. When the clock is triggered, the data input is copied to the output, then held until the clock is triggered again.
    - A JK latch has three inputs: A **c**lock input, and the **j**ump and **k**ill inputs. When the clock is triggered, the latch's output can be set, reset, toggled, or left as is, depending on the combination of J and K. While these are common in real-world electronics, in *Minecraft* they tend to be bulky and impractical — most players would use an SRT latch instead.

RS latches

An RS latch has two inputs, and . The output is conventionally labeled , and there is often an optional "inverse output" . (Having both and is called "dual outputs"). When a signal comes into , is **s**et on and stays on until a similar signal comes into , upon which is **r**eset to "off". indicates the opposite of — when is high, is low, and vice versa. Where a output is available, the player can often save a NOT gate by using it instead of .

Note that the proper name for this category of latch is "SR latch". However, in real-world electronics as in Minecraft, the classic implementation of such latches starts by inverting the inputs; such a latch is the proper "RS latch", but they're so common that the term is commonly used also for what "should" be called SR latches.

Typical uses include an alarm system in which a warning light stays on after a pressure plate is activated until a reset button is pushed, or a rail T-junction being set and reset by different detector rails. RS latches are common parts of other circuits, including other sorts of latches.

Setting both inputs high simultaneously is a "forbidden" condition, generally something to avoid. In the truth table, S=1, R=1 breaks the inverse relationship

between and . If this happens, the player will get "undefined behavior" — various designs can do different things, and especially and can be high or low at the same time. If the forbidden state is co-opted to *toggle* the output, the circuit becomes a JK latch, described in its own section. If there is instead a third input which **t**oggles the output, the circuit becomes an "RST latch".

Any RS latch with dual outputs is functionally symmetrical: pulsing each input turns on "its" output, and turns off the other one. Thus and are interchangeable, if the outputs is swapped: Which input players pick as chooses which of the outputs is , then the other input will be R and the other output will be . (If the original circuit only had a output, then swapping the inputs will turn it into .) In several designs (A, B, C, D, E, F, I) the functional symmetry is reflected by the circuit's physical symmetry, with each input energizing the torch it leads to, while turning off the other.

RS latches can be built in a number of ways:

- Two NOR gates can be linked so that whichever is lit, the other will be off. The RS NOR latch is the "original" RS latch, and still among the smallest memory devices that can be made in vanilla *Minecraft*. While they can be built with just torches and redstone dust, repeaters can also be used. Many of these designs have "duplex I/O"—the same locations can be used to read or set the latch state.
- It is also possible to construct an RS NAND latch, using NAND gates instead of NOR gates. These will be larger and more complex than an RS NOR latch, but may be useful for specialized purposes. Their inputs are inverted (see below for details).
- Other RS latches can be created by fitting an "input sustaining circuit" with a reset switch, say by adding a pair of NOT gates or a piston, placed so as to interrupt the circuit when triggered. Such a construction can be nearly as compact as an RS NOR latch (and often with better I/O isolation and/or timing), but they will usually not have a natural output.

- Other devices can also be involved. Pistons can be used to physically toggle a block's location, while hoppers or droppers can pass around an item entity. These circuits can be very fast and small, with little redstone dust.

1	1	0	0	Undefined	Undefined
1	0	0	1	1	0
0	1	1	0	0	1
0	0	1	1	Keep state	Keep state

RS-NOR latches

Basic RS-NOR Latches [show] [edit]

Designs **A** and **B** are the most fundamental RS-NOR latches. In both cases, their inputs and outputs are "duplex"—the latch state can be read ( ) or set ( ) on one side of the circuit, while on the other side, the latch can be reset ( ), or the inverse output read ( ). If separate lines for input and output are needed, opposite ends of **B** can be used, or **A** can be elaborated into **A'** with separate locations for all four lines.

Isolated RS-NOR Latches [show] [edit]

These can be modified to provide separate, even isolated, input and output. **C** and **D** use torches and repeaters respectively to isolate the outputs, though the inputs can still be read. **E** expands the circuit slightly to isolate all four I/O lines.

Vertical RS-NOR Latches [show] [edit]

Design **F** provides a vertical (1-wide) option; again, the I/O is duplex, though isolated outputs can be taken at alternate locations.

Design **G** takes up more room than **F**, but may be preferable, as both the set and reset are on the same side. Also, be sure to compensate for the extra tick on ( ), caused by the last torch.

Design **H** is smaller than design **F** in term of height, input and output are on the same height, but it is longer and a bit slower due to the repeater.

Furthermore, it is easily stacked vertically and horizontally (with a shift of 2 blocks on the Y axis).

Design **I** is similar to design **G** as it has both set and reset on the same side,but takes up less space. The I/O is duplex, though isolated outputs can be taken at alternate locations.

Design **J** is similar to design **G** as it has both set and reset on the same side, but has no slowness due to not having any extra repeaters or torches. This may be more preferable to **G**, although the outputs (/) are not level with the inputs (R/S).

RS NAND latches

An RS latch can also be designed using NAND gates. In *Minecraft*, these are less efficient than the RS NOR latch, because a single Redstone torch acts as a NOR gate, whereas several torches are required to create a NAND gate. However, they can still be useful for specialized purposes.

Such an "RS NAND latch" is equivalent to an RS NOR, but with inverters applied to all the inputs and outputs. The RS NAND is logically equivalent to the RS NOR, as

the same R and S inputs give the same output. However, these designs take *inverse* R and S ($\overline{R}$, $\overline{S}$) as inputs. When $\overline{S}$ and $\overline{R}$ are both off, and are on. When $\overline{S}$ is on, but $\overline{R}$ is off, will be on. When $\overline{R}$ is on, but $\overline{S}$ is off, will be on. When $\overline{S}$ and $\overline{R}$ are both on, it does not change and . They will be the same as they were before $\overline{S}$ and $\overline{R}$ were both turned on.

RS-NAND Latches [show] [edit]

RS-Latch summary table 1

This table summarizes the resources and features of the RS latches which use only redstone dust, torches, and repeaters.

Design	A	B	A'	C	D	E	F	G	H
Size	4×2×3	3×2×3	4×4×3	2×3×3	2×3×2	2×4×2	3×**1**×4	5×3×3	6×3×3
Torches	2	2	2	2	2	2	2	4	6
Redstone wire	6	4	10	4	0	4	3	6	8
Repeaters	0	0	0	0	2	0	0	0	0
Inputs isolated?	Duplex	Duplex	Duplex	Duplex	Yes	Yes	Duplex	Yes	Yes

Outputs isolated?	Duplex	Duplex	Duplex	Yes	Yes	Yes	Duplex/Yes	No	Yes
Input orientation	opposite	adjacent	opposite	opposite	opposite	opposite	opposite	perpendicular	perpendicular

Analog RS latch

Analog RS latch

This latch will maintain the highest signal level that arrived from input **S** if **R** is off, and fade (reduce memorized signal strength) by strength of **R** every two redstone ticks. For maximum strength (15) signals it behaves like any other RS latch, but it can also memorize intermediate signal levels, and since 2-tick pulses on **R** will subtract their strength from its memorized state, it makes a nice element of counter or countdown circuits.

Input stabilization with reset

Input Stabilization Circuit

An "Input-Stabilizing Circuit" responds to an input pulse by turning its input on and leaving it on. This can be built up into an RS Latch by adding a means to turn it off.

These circuits usually don't offer a "natural" output. Design **J** adds a pair of NOT gates, with the reset going to the second torch. (The NOT gates can also be added to the upper redstone loop.) Design **K** uses its piston to block the circuit where it goes up onto the solid block. Design **L** shows the reverse approach, breaking the circuit by withdrawing a power-carrying block.

RS-ISR Latches [show] [edit]

Pistons and other devices

Other RS Latches [show] [edit]

A pair of non-sticky pistons can be used to physically push a block back and forth. This can make or break a circuit from a torch, producing an RS latch with no inverse output (**M**). If the block being pushed is a block of redstone, the circuit can be even smaller, with dual outputs (**N**). Both of these have isolated inputs and outputs. Putting *two* blocks between the pistons produces an SRT latch **O**, with an extra input to toggle the latch state. And droppers can also be pressed into service, as in design **P**: Small, tileable, but it does require a comparator.

Variations

- Expand an RS latch easily into a monostable circuit, which automatically disables itself some time after activation. To do this, split the output redstone path into two parts. The new path should run through some repeaters, and in to the reset input. When players turn on the latch, redstone feeds a signal through the delay before turning off the latch.

  This works not only for and R, but for and S as well. A more complex delay mechanism, such as a water clock, can replace the repeaters.
- An "Enable/Disable RS latch" can be made by adding a pair of AND gates in front of the inputs, testing each of them against a third input, E. Now if E is true, the memory cell works as normal. If E is false, the memory cell will not change state. That is, E latches (or equivalently, clocks) the RS latch itself. Note that for design **Q**, the outputs are *not* isolated, and a signal to them can set the latch regardless of E. Alternatively, repeaters could be used to latch the inputs, but this costs more and saves no space.
- As noted above, if it is possible to add a "toggle" input, the RS latch becomes an RST latch. If the "forbidden" state is used for the toggle, then it's a JK latch.

Dropper SR latch

Allows a lot of flexibility in geometry — the droppers can be read from 3 sides each and activated from 5 sides each; can be oriented vertically too and content can be read with comparators through solid blocks. However, always power it through an

adjacent block; if players power the dropper directly, they will activate the other dropper too and the order is unpredictable. Activates on rising edge, meaning they can apply S even while R is still active or vice versa.

Enable/Disable RS Latch [show] [edit]

RS latch summary table 2

Design	J	K	L	M	N	O	P	Q
Size	2×3×3	4×3×3	4×4×2	4×3×2	4×**1**×**1**	5×3×3	3×**1**×2	5×5×3
Torches	2	0	1	1	0	1	0	7
Dust	7	4	6	0	9	4	0	7
Repeaters	1	1	1	1	0	1	0	0
Other devices	--	1 sticky piston	1 sticky piston	2 normal pistons	2 normal pistons	2 normal pistons	2 droppers, 2 comparators	N/A
Inputs isolated?	Yes,	No	No	Yes	Yes	No	Yes	Yes
Outputs isolated?	Yes	No	No	Yes	Yes	Yes	Yes	No

available?	No	No	No	No	Yes	No	Yes	Yes
Input orientation	Perpendicular	Perpendicular	Adjacent	Opposite	Opposite	Opposite	Adjacent	Adjacent

D latches and flip-flops

A D ("data") flip-flop or latch has two inputs: The data line D, and the "clock" input C. When triggered by C, the circuits set their output ( ) to D, then hold that output state between triggers. The latch form, a "gated D latch", is level triggered. It can be high- or low-triggered; either way, while the clock is in the trigger state, the output will change to match D. When the clock is in the other state, the latch will hold its current state until triggered again. A D flip-flop is edge triggered; it sets the output to D only when its clock input changes from "off" to "on" (rising edge) or *vice versa* (falling edge), according to the circuit. An edge trigger can turn a gated D latch into a D flip-flop.

Building these devices with torches is fairly unwieldy, though some older designs are given below. Repeaters have a special latching ability, which drastically simplifies the problem. Now a gated D latch can be made with two repeaters, and a D flip-flop with four repeaters and a torch:

Design **G** uses the repeater's latching feature, which is added to the game in Java Edition 1.4.2. It holds its state while the clock is high, and is by far the most compact of the D latch designs. Design **H** combines two such latches, one high and one low triggered, to create a rising edge-triggered D flip-flop. The block and redstone torch

can be reversed for a falling edge-triggered design. The design is based on a real life implementation of an edge-triggered D flip-flop called a "Master-Slave" configuration.

Modern Gated D Latch (G)

(High level)

Modern D Flip-flop (H)

(rising edge)

Analog D latch

Analog D latch (J) (Low level)

6×4×2 (48 block volume)
flat, silent
circuit delay: 3 ticks
Earliest Known Publication: May 26, 2018[1]

Design **J** is an analog version of a low-triggered D latch. The signal strength of the output is the same as input D when the latch is triggered.

For maximum strength (15) signals for D, this latch behaves like a normal (digital) low-triggered D latch.

Torch-based designs

For historical interest, here are several older designs, not dependent on latched repeaters, along with a table of their resource needs and other characteristics. A few of these designs also have the additional inputs and inverse output of an RS latch.

This basic level-triggered gated D latch (design **A**) sets the output to D as long as the clock is set to OFF, and ignores changes in D as long as the clock is ON. However, on a rising clock edge, if D is low, the output will pulse high for 1 tick, before latching low.

Design **B** includes a rising-edge trigger and it will set the output to D only when the clock goes from OFF to ON. The torch-based edge trigger could also be replaced with one of the designs from the Pulse circuit page.

These are RS latch-based circuits with appropriately set front-ends. Directly trigger the RS latch using the R and S inputs to override the clock, forcing a certain output state. Sending signals into the and lines works similarly, because the output is not isolated. To isolate the outputs, add inverters and swap the labels.

D Latch A [show] [edit]

D Latch B [show] [edit]

Design **C** is a one block wide vertical version of **A**, except for using a non-inverted clock. It sets the output to D while the clock is ON (turning the torch off). This design can be repeated in parallel every other block, giving it a much smaller footprint, equal to the minimum spacing of parallel data lines. A clock signal can be distributed to all of them with a wire running perpendicularly under the data lines, allowing multiple flip-flops to share a single edge-trigger if desired. The output is most easily accessed in the reverse direction, toward the source of input. As in design **A**, the un-isolated and wires can do double duty as R and S inputs. can be inverted or repeated to isolate the latch's Set line.

D Latch C [show] [edit]

D Latch D [show] [edit]

Design **E** provides a more compact (but more complex) version of **A**, while still affording the same ceiling requirement. **E'** allows the latch to act on a high input.

Design **F** holds its state while the clock is high, and switches to D when the clock falls low. The repeater serves to synchronize the signals that switch out the loop and switch in D. It must be set to 1 to match the effect of the torch.

D Latch E [show] [edit]

D Latch F [show] [edit]

Design	A	B	C	D	E	E'	F	G	H
Size	7×3×3	7×7×3	6×**1**×5	5×2×6	5×3×3	5×3×3	5×3×3	2×1×2	3×2×2
Torches	4	8	5	6	4	5	4	0	1
Redstone wire	11	18	5	6	10	9	7	0	0
Repeaters	0	0	0	0	0	0	1	2	4
Trigger	Low Level	Rising Edge	High Level	High Level	Low Level	High Level	Low Level	High Level	Rising Edge
Output isolated?	No	No	No	No	No	No	Yes	Yes	Yes
Input isolated?	Yes	Yes	C Only	C Only	Yes	Yes	No	Yes	Yes

BUD-based D flip-flop

BUD-based D Flip-flop (I) (low level)

Piston BUD based D Flip-flop

Design **I** represents an entirely different form of the D flip-flop, based on the principle of the block update detector. This flip-flop is small so it can be used multiple times at large integrated redstone circuits. Note that no blocks that are adjacent to the piston can be used as circuit components except flip-flop itself.

The lever in the screenshot shown is the D input. The redstone wire in the middle is trigger signal input. The trapdoor is part of the BUD – it need be replaced by a note block, an activator rail, etc.

JK flip-flops and latches

A JK flip-flop is another memory element which, like the D flip-flop, will only change its output state when triggered by a clock signal C. They can be edge-triggered (designs **A**, **D**, **E**) or level-triggered (**C**). Either way, the two inputs are called J and K. These names are arbitrary, and somewhat interchangeable: if a output is available, swapping J and K will also swap and .

J	K	Q_{next}
0	0	
0	1	0

1	0	1
1	1	

When the flip-flop is triggered the effect on the output will depend on the values of the two inputs:

- If the input J = 1 and the input K = 0, the output Q = 1.
- When J = 0 and K = 1, the output Q = 0.
- If both J and K are 0, then the JK flip-flop maintains its previous state.
- If both are 1, the output will complement itself — i.e., if before the clock trigger, afterwards.

The table summarizes these states — note that is the new state after the trigger, while represents the state before the trigger.

The JK flip-flop's complement function (when J and K are 1) is only meaningful with edge-triggered JK flip-flops, as it is an instantaneous trigger condition. With level-triggered flip-flops (e.g. design C), maintaining the clock signal at 1 for too long causes a race condition on the output. Although this race condition is not fast enough to cause the torches to burn out, it makes the complement function unreliable for level-triggered flip-flops.

The JK flip-flip is a "universal flip-flop", as it can be converted to any of the other types: It's already an RS latch, with the "forbidden" input used for toggling. To make it a T flip flop, set **J = K = T**, and to make it a D flip-flop, set K to the inverse of J, that is **J = $\overline{K}$ = D**. In the real world, mass production makes JK latches useful and common: a single circuit to produce in bulk, that can be used as any other sort of latch. In *Minecraft*, however, JK latches are generally larger and more complex than the other types, and using their toggle function is awkward. It's almost always easier

to build the specific latch type needed. Notably, an SRT Latch has all the same abilities, but gets the toggle function from a separate input.

Design **E** is a vertical JK Flip-Flop from the basis of design A.

Aside from these redstone designs, it is also possible to make a JK flip-flop by modifying a rail toggle, or with newer components such as hoppers and droppers.

JK Latch A [show] [edit]

JK Latch C [show] [edit]

JK Latch D [show] [edit]

JK Latch E [show] [edit]

Design table

Design	A	C	D	E
Size	9×2×11	7×4×5	5×2×7	14×10×1
Torches	12	11	8	10
Redstone	30	23	16	24
Repeaters	0	0	6	6
Accessible ?	No	Yes	Yes	No
Trigger	Edge	Level	Edge	Edge

T flip-flop

T flip-flops are also known as "toggles". Whenever T changes from OFF to ON, the output will toggle its state. A useful way to use T flip-flops in Minecraft could, for example, be a button connected to the input. When players press the button the output toggles (a door opens or closes), and does not toggle back when the button pops out. These are also the core of all binary counters and clocks, as they function as a "period doubler", releasing one pulse for every two received.

There are many ways to build a T flip-flop, ranging from torches and dust through pistons to more exotic devices. Many designs depend on a quirk in sticky-piston behavior, namely that after pushing a block, a sticky piston will let go of it if the activating pulse was 1 tick or less. This allows short pulses to toggle the position of a block, which is very useful here.

Best in class TFF designs

These are designs which seem markedly superior in various categories.

T Latch L3 [show] [edit]

T Flip-flop L4 [show] [edit]

T Flip-flop L5 [show] [edit]

T Flip-flop L6 [show] [edit]

T Flip-flop L7 [show] [edit]

L3 is a latch, which responds to a high level. Like most T latches, if the toggle line is held high too long, it will "oscillate", toggling repeatedly. A stone button will produce a single pulse, while wooden button's pulse is long enough to cause oscillation.

L5 is a true flip-flop with the same footprint as the **L3**(but higher), which triggers on a rising edge. Both are extremely compact, thanks to the use of latched repeaters.

L6 is a compact 1-high adaptation of D flip-flop **H**. The video shows **L6** and a similar T flip-flop.

L4 and **L7** are basically two opposite halves of the same machine — both are extremely compact and customizable tick-wise but **L4** is made for off-pulses with durations ranging from 2 to 8 redstone ticks while **L7** is made for on-pulses with durations that are 9+ redstone ticks long which includes the 10-tick stone button. Customizing each requires changing the repeater delay or adding repeaters to match the trigger duration.

To customize **L4** for your use, adjust the top most repeater according to the duration of your trigger, as shown in the table below:

Off-pulse Duration (Redstone Ticks)	Recommended Setting for Output Repeater
1	N/A - Use a Sticky Piston
2	1
3	2
4	2
5	3

6	3
7	4
8	4
9+	N/A - Use TFF O or L7

L6 and another TFF (view on YouTube) [show]

Piston TFF designs

This design doesn't use the quasi-connectivity effect, so it works in both *Bedrock* and *Java* editions. It uses a pulse generator that feeds into repeaters that power the piston through a solid block and an underground redstone dust patch. The redstone block position is the output value of the TFF. This design requires one sticky piston (for the repeater) and two non-sticky pistons, and a 6×6 area, which is linear-tilable so that the output of one TFF feeds into the next TFF.

Bedrock Edition Piston TFF [show] [edit]

The following designs work in *Java Edition* but may present difficulties in *Bedrock Edition*.

Linear tilable TFF M [show] [edit]

3×3 piston TFF N [show] [edit]

2 piston QC TFF O [show] [edit]

TFF R [show] [edit]

Design **M** is a 1-wide dual-piston design, which can be tiled adjacent to each other for compact circuitry. (If they *don't* have to be right next to each other, dust can be used instead of the input and output repeaters.) The hidden piston forms a simple monostable circuit that cuts off the button signal (10 ticks or so) as soon as a 1-tick signal has passed through to the second repeater. Due to the piston quirk mentioned above, this 1-tick signal lets the main piston toggle the position of its mobile block, to set or unset the latch and the output. It can be made more compact by removing the last block, the repeater and the torch and replacing the block in front of the last piston with a redstone block.

That linear design can also be bent into a 3×3 square, as **N**. (The "any" blocks can be air, and that torch can just as well be on the ground.) Tiling design **N** is a little tricker, but it can be done in either horizontal direction, by mirroring adjacent copies. Note that the output can be taken from whichever side of that corner is free, but the player will need repeaters to keep adjacent outputs from cross-connecting.

Design **O**, based on the quasi-connectivity effect that works only in *Java Edition*, uses a redstone block that swaps positions when the top dust receives a signal; it is a dual piston design that uses only two pistons, two torches, two dust, and two solid blocks and a redstone block. While one of the most compact designs; using only 10 blocks of space before inputs and outputs in addition to being 1 wide and vertical, it also requires no slime balls and uses few resources aside from the redstone block while allowing for four areas to input and 4 areas to output (if repeaters are used for the output, 2 if not), in addition it can be built in the air since it doesn't have any redstone or repeaters that require placement on the ground. The design toggles on a falling edge.

Design **R** is a variation of design O, and it adds the ability to reset the output to 0, using the input R.

Observer TFF designs (*Java Edition*)

T Flip-flop O1 [show] [edit]

T Flip-flop O2 [show] [edit]

T Flip-flop O3 [show] [edit]

Those designs make use of observers and the quasi-connectivity effect. Designs **O1** and **O2** work for a rising signal, while the **O3** toggles on a falling signal.

Design table

Design	O1 horizontal	O1 vertical	O2	O3
Size	6x1x1	3×4×1	5×2×1	4x2x1
Observers	1			
Redstone blocks	1			
Sticky pistons	2			
Trigger	rising			falling
Delay	2			

Other conventional TFF designs

This section needs cleanup to comply with the style guide. [discuss]

Please help improve this page. The talk page may contain suggestions.

Reason: There are many designs here which are not well documented, and some may be redundant or broken. Any help in describing or testing circuits would be appreciated.

T Flip-flop A [show] [edit]

T Flip-flop B [show] [edit]

T Latch D [show] [edit]

T Flip-flop E [show] [edit]

T Flip-flop J [show] [edit]

T Flip-flop K [show] [edit]

T Flip-flop P [show] [edit]

Design **A** demonstrates that a TFF can be made solely with redstone dust and torches, but it sprawls over 9×7×3 blocks. Design **B** is slightly unreliable for very long pulses; while the input is on, the piston will toggle every time the block below the piston arm is updated.

Design **D** (another torches-and-dust design, but vertical) does not have an incorporated edge trigger and will toggle multiple times unless the input is passed through one first. Design **E** adds such a trigger (and a repeater).

Designs **J** and **K** make more use of repeaters, but not as latches, and they are still quite large.

T Flip-flop L1 [show] [edit]

T Flip-flop L2 [show] [edit]

Design **L2**, (also **L3**, **L4**, and **L5** above) relies on the redstone repeater locking mechanic introduced in Java Edition 1.4.2. **L4** is the smallest, but requires a piston and activates on a falling edge.

T Flip-flop Z3 [show] [edit]

T Flip-flop Z4 [show] [edit]

T Flip-flop Z5 [show] [edit]

TFF summary table

These tables are incomplete, and need more data.

Design	A	B	D	E	J	K	M	~~O~~	P	R
Size	7×9×3	5×6×3	**1**×7×6	**1**×11×7	3×7×3	3×7×3	**1**×7×3	3×4×4	5x5x2	4×5×4
Redstone wire	28	14	9	13	11	9	0	2	8	8
Torches	10	4	7	12	5	5	1	3	7	4
Repeaters	0	0	0	1	3	2	3	0	0	1
Other Devices	none	1 SP	none	none	none	none	2 SP	2 P	none	3 SP
Input isolated?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No

Output(s) isolated?	No	No	No	No	only	No	Yes	No	No	No
available?	No	No	No	No	Yes	No	No	No	Yes	No
Trigger	rising	rising	rising	rising	rising	rising	rising	falling	rising	falling for both T and R
Delay	4				3	4	3	1		3 for R, 1 for T
Cycle time										
Other		BUD					tilable			

Design	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**
Size	3×6×3	3×5×2	3×4×2	2×3×1	3×4×3	4×4×2
Redstone wire	4	6	2	2	4	4
Torches	4	2	2	0	2	2
Repeaters	4	3	3	3	4	4

Other devices	1 SP	none	none	none	none	none
Input isolated?	Yes	Yes	Yes	No	Yes	Yes
Output(s) isolated?	Yes	Yes	Yes	Yes	only	No
available?	Yes	No	No	No	Yes	Yes
Trigger	rising	rising	high	falling	rising	rising
Delay		3	5	1/2 Trigger Duration	4 ( )	4
Cycle time				Trigger Duration	6	

Design	~~**Z1**~~	~~**Z2**~~	**Z3**	**Z4**	**Z5**
Size	3×3×3	3×5×3	**1**×6×5	3×5×3	**1**×5×4
Redstone wire	4	4	4	4	2
Torches	2	3	3	3	2
Repeaters	1	2	2	2	2
Other devices	1 SP	1 SP	1 SP	1 SP	1 SP
Input isolated?	Yes	Yes	Yes	Yes	Yes

Output(s) isolated?	Yes	Yes	Yes	Yes	Yes
available?	No	No	No	No	No
Trigger					
Delay					
Cycle time					

Size
"In a void", that includes required blocks supporting redstone.

Delay
The number of ticks from the trigger to switching the output.

Cycle time
How often the latch can toggle, including any recovery time. This is the *period* of the fastest clock that can drive it.

Other Devices
P == normal piston, **SP** == sticky piston, **C** == comparator, **H** == hopper, **D** == dropper.

Trigger
rising edge (the usual), **falling** edge, **high** or **low** level. Level-triggered TFFs oscillate on long pulses.

Rail and exotic TFFs

Pressure-Plate Rail TFF (B) [show] [edit]

Basic Rail TFF (A) [show] [edit]

The rail T flip-flop is a T flip-flop which uses rails and redstone. The general design uses a length of track that is stopped by a block at both ends. When the T flip-flop is in a stable state, the minecart is at either end of the track (depending on the state).

An input pulse turns on powered rails at both ends of the track, causing the minecart to move to the other end.

Along the track, there are two separate detector elements (e.g. detector rails). These two detectors are each connected to an input of an RS NOR latch, and hence serve to translate minecart motion into a state transition. When the minecart moves, depending on its direction of motion, one detector will turn on (and off) before the other; the second detector to be hit is what determines which input of the RS NOR latch stays on last and hence what the new state of the RS NOR latch is.

Design **A** uses detector rails, while design **B** uses pressure plates. (A minecart triggers a pressure plate on the inside of a turn, including diagonals.) Note that for B, the other side of the latch isn't a true , as the passage of the cart turns on before actually switching the latch.

This type of T flip-flop is slower than traditional redstone-only circuits, but this may be desirable in certain situations. With T flip-flop designs that are level-triggered (as opposed to clocked or edge-triggered), a long input pulse will cause the flip-flop to continuously switch state (oscillate) while the pulse is present. In pure redstone circuits, this is only limited by the redstone circuit delays, and hence a relatively short input pulse can cause several state transitions. Pure redstone T flip-flops usually include an edge-trigger or pulse-limiting circuit to the design, since the input pulse usually can't be guaranteed to be short enough without the use of that kind of circuit.

With rail-based designs, the speed at which the output can flip is limited by the time needed for the cart to move from one end of its rail to the other, which allows for a much longer pulse to be applied to a level-triggered input without needing an edge-trigger or pulse limiter circuit. However, the delay between the input pulse and the output transition is also longer.

Grizdale's T flip-flop

Grizdale's Compact TFF [show] [edit]

This hopper/dropper design is not only compact, but tileable in three dimensions. The only hitch (for survival mode) is that the player needs access to nether quartz for the comparator.

The **A** variant has a size of 1×2×3. The **B** variant puts the input and output inline, but changes the footprint to 2×2×2, or 4×2×2 if players want fully powered input and output. The **B** design can also be tiled in line, side by side, vertically (by reversing alternate rows), or all three at once.

Once built, place a single item inside any of the containers and it will work as a T flip-flop, with the item cycling between the two droppers. The core has a 1 tick delay between input and turning off or on, but the optional repeaters would raise this to 3.

This T Flip Flop can be turned into an SRT latch by only powering the bottom dropper to set, and the top to reset. However, it won't be as tileable as the original TFF.

Obsolete T flip-flops

T Flip-flop Z1 [show] [edit]

T Flip-flop Z2 [show] [edit]

Designs **Z1** and **Z2** do not work as of Java Edition 1.5.2 — in both cases, their pulse generator does not cause the piston to toggle its block as apparently intended.

Redstone circuits/Logic

A logic gate can be thought of as a simple device that will return a number of *outputs*, determined by the pattern of *inputs and rules* that the logic gate follows. For

example, if both inputs in an AND gate are in the 'true'/'on'/'powered'/'1' state, then the gate will return 'true'/'on'/'powered'/'1'.

There are many different kinds of logic gates, each of which can be implemented with many different designs. Each design has various advantages and disadvantages, such as size, complexity, speed, maintenance overhead, or cost. The various sections will give many different designs for each gate type.

Concepts

The output of each logic circuit reflects the state of its inputs at all times (though possibly with some delay incurred by the circuit).

Swapping inputs

For most of these gates, A and B can be swapped without changing the output.

Swapping the inputs of the IMPLIES gate *will* affect its output, and the NOT gate has only one input.

Stacking inputs

The AND, OR, and XOR gates can each be used in arrays to perform their operation on more than two inputs, by combining two inputs at a time, then combining the results with each other and/or other inputs. For these gates, the order in which the inputs are combined doesn't matter.

When an XOR gate is combined in this way, its output is on when an *odd* number of inputs is on.

Choosing a logic gate

When unsure which logic gate to use, try building a table like the one down below but with just one row of outputs. List the known inputs coming in and the possible combinations of power, and for each combination write down what the output should be for the structure to work. Then compare that to the table on the right and see which gate matches the desired outputs.

If the the output needs to change when the input is stable, or needs to be remembered after the input has ended, the player may also need to look at pulse circuits or memory circuits.

A	Question Answered

B					
A AND B	ON	off	off	off	Is A and B on?
NOT (A IMPLIES B)	off	ON	off	off	Is A on and B off?
NOT (B IMPLIES A)	off	off	ON	off	Is B on and A off?
A NOR B	off	off	off	ON	Are both inputs off?
A	ON	ON	off	off	Is A on?
A XOR B	off	ON	ON	off	Are the inputs different?
NOT A	off	off	ON	ON	Is A off?
A XNOR B	ON	off	off	ON	Are the inputs the same?
B	ON	off	ON	off	Is B on?
NOT B	off	ON	off	ON	Is B off?
A NAND B	off	ON	ON	ON	Is either input off?
A IMPLIES B	ON	off	ON	ON	If A is on, is B also on?
B IMPLIES A	ON	ON	off	ON	If B is on, is A also on?

A OR B	ON	ON	ON	off	Is either input on?

Logic gate

A **logic gate** is a basic logic circuit.

NOT gate

A

NOT A	off	ON

A **NOT gate** ( ), also known as an inverter, is a gate used when an opposite output is wanted from the input given. For instance, when the switch, or input, is set to "on", the output will be toggled to "off", and when the switch is toggled to "off", the output will be toggled to "on".

Torch Inverter

1-wide, flat (horizontal only), silent, tileable

circuit delay: 1 tick

The torch inverter is the most commonly used NOT gate, due to its small size, versatility, and easy construction.

One drawback of the torch inverter is that it will "burn out" if run on a clock cycle faster than a 3-clock (3 ticks on, 3 ticks off). A burnt out torch inverter will turn back on after it receives a block update.

Subtraction Inverter

flat, silent

circuit delay: 1 tick

The subtraction inverter offers little advantage over the torch inverter except that it can run on a 2-clock cycle without burning out. Faster clocks will not work though — the comparator simply won't react to them.

Variations: The powered lever can be replaced with another always-on power component (e.g., redstone torch, block of redstone), or with a full container if a power component would be inconvenient in that location.

The repeater is required to ensure the input signal is strong enough to overcome the comparator's rear source, but can be removed in a number of ways. If the input power level is known (because the circuit design is fixed, so it can be calculated), the repeater can be removed by replacing the powered lever with a container which will produce the same power level. Alternatively, the repeater can be removed if the output continues to a length of redstone wire which will reduce the subtracted signal enough that the signal is inverted eventually.

Instant Inverter

instant

circuit delay: 0 ticks

The instant inverter is a basic building block of larger instant circuits.

The "ground" version has the largest volume, but is shorter and fits easily with flatter circuits. The 1-wide version is the smaller in total volume and 2-tileable.

Behavior (i.e., how it works): An instant inverter has two functional elements, and a piston, or pistons, that activate them:

1. a constant power source with output that can be instantly powered off (but powering it on takes time): either a redstone block that ceases to provide power as soon as piston starts moving it (within the same tick the piston receives or loses power) or a solid block in front of a powered repeater or comparator, powering redstone dust; as soon as the block starts moving the dust is unpowered.
2. a signal line with output that can be instantly powered on but not necessarily off, its input delayed by and sustained for 2 ticks. The "instant on" is achieved by the dust-cut technique: a solid block placed against edge of a block over which a redstone line is running, disconnects that line from line below. Start of motion of that block instantly reconnects the line and provides power. The delay is achieved by running the input through a 2 tick repeater, two torches or similar means. That means, when power appears on input, the block moved by piston is able to cut the line before signal passes through the delay. With input unpowered, the output is instantly activated and the line still provides power "stored" in the repeater for two ticks, which time is sufficient to reactivate the constant power source from previous point.
3. Piston, or pistons, to move the block/blocks activating the elements from point 1 or 2.

Schematic gallery: NOT gate [show] [edit]

OR gate

A

B

A OR B	ON	ON	ON	off

An **OR gate** ( ) is a gate which uses two or more inputs and whenever any input is "on", the output is also "on". The only time the output is "off" is when all inputs are "off". Note that since the OR operation is associative and commutative, OR gates can be combined freely: The player can compare huge numbers of inputs by using small OR gates to collect groups of inputs, then comparing their results with more OR gates. The result will not depend on the arrangement of the inputs, or on which ones were combined first.

The simplest version of the OR gate is design **A**: merely a wire connecting all inputs and outputs. However, this causes the inputs to become "compromised", so that they can only be used in this OR gate. The introduction's example, using a solid block instead of wire, does not suffer the same hazard.

If players need to use the inputs elsewhere, the inputs need to be "isolated", by passing them through a block as above, or a device such as a torch or repeater. Torches yield version **B**. Note that this is in fact a NOR gate with an inverter on the output.

Version **C** isolates the inputs with repeaters. It can be expanded horizontally up to 15 inputs. Besides the isolated inputs, it is one tick faster than **B**. Additional repeaters can be used to add new groups of inputs, or to strengthen the output signal. This design is more expensive, as each repeater costs 3 redstone dust to craft (along with smooth stone).

Version **D** is a 1-wide version designed for vertical use, such as in walls. The repeater serves to isolate the outputs from the inputs. This version can only take two inputs, though of course the inputs can be stacked with multiple gates.

Version **E** utilizes the properties of light-transparent blocks: glowstone, and upside-down stairs or slabs. These send signals up, but not down. It is expandable, like design **C**.

Schematic gallery: OR gate [show] [edit]

NOR gate

A

B

A NOR B				
	of f	of f	of f	O N

A **NOR gate** (or) is the opposite of the OR gate. Whenever at least one switch is toggled to "on", the output is toggled to "off". The only time the output is "on" is when all inputs are toggled to "off". This gate also uses two or more inputs.

All logic gates can be made from some combinations of the NOR gate.

In *Minecraft*, NOR is a basic logic gate, implemented by a torch with two or more inputs. (A torch with 1 input is the NOT gate, and with no inputs is the TRUE gate, that is, a power source.)

A torch can easily accommodate 3 mutually isolated inputs, as in design **A**. Design **B** goes to greater lengths to squeeze in a fourth input. If more inputs are necessary, it is simplest to use OR gates to combine them, then use an inverter (NOT) at the end.

It is also possible to combine OR and NOR gates, by using the inversion of OR gates as inputs to NOR gates.

For inverted output when A is OFF, use redstone torch for B and result is:

	A			
	B			
A NOR B	off	off	off	ON

Schematic gallery: NOR gate [show] [edit]

AND gate

	A			
	B			
A AND B	ON	off	off	off

An **AND gate** ( ) is used with two or more switches or other inputs. The output is toggled to "on" only when all inputs are "on". Otherwise, the output will remain "off".

In reality, the usual implementation is a NOR gate with inverted inputs ( ). Taking the inputs and , the first two torches (at the top and bottom of Schematic (A) below) invert them into and . The redstone wire between these torches serves as an OR gate, and is therefore in state , which can be interpreted as

by De Morgan's Law. Finally, the third torch (the center-right one) applies a NOT to that statement; thus it becomes .

A 3-input AND gate is shown, but, like OR gates, AND gates can be freely "ganged", combining groups of inputs and then combining the results.

A typical use for an AND gate would be to build a locking mechanism for a door, requiring both the activating button and the lock (typically a lever) to be on.

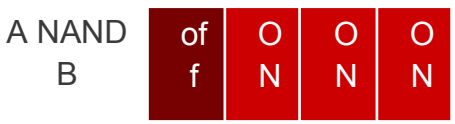
Piston AND gates act similarly to a "tri-state buffer", in which input B acts like a switch, connecting or disconnecting input A from the rest of the circuit. Such designs have one input feeding a circuit, which is opened or closed by a sticky piston driven by the other input. The difference from real-life tri-state buffers is that one cannot drive a low current in Minecraft.

Schematic gallery: AND gate [show] [edit]

NAND gate

A

B



A **NAND gate** (or) turns the output off only when both inputs are on, the reverse of an AND gate. All logic gates can be made from NAND gates. As with NOR, large numbers of inputs are probably best handled by stacking up AND gates, then inverting the result. By De Morgan's Law, is identical to .

All logic gates can be made from some combinations of the NAND gate.

Schematic gallery: NAND gate [show] [edit]

XOR gate

A

B

A XOR B				
	of f	O N	O N	o ff

An **XOR gate** (, or) is a gate that uses two inputs and the output is toggled to "on" when one switch is "on" and one switch is "off". XOR is pronounced "zor" or "exor", a shortening of "exclusive or", because each input is mutually exclusive with the output. It is useful for controlling a mechanism from multiple locations. Because of these properties, XOR gates are commonly found in complex redstone circuits. In some cases, it is possible to get an OR gate output and an AND gate output on different channels. Design F is composed of AND gates, OR gates and NOT gates. The whole circuit is , which can be further simplified into (or, equivalently,).

A useful feature is that an XOR (or XNOR) gate will always change its output when one of its inputs changes, hence it is useful for controlling a mechanism from multiple locations. When controls (such as levers) are combined in an XOR gate, toggling any control will toggle the XOR gate's output (like a light bulb controlled by two light switches — players can flip *either* one to turn the light on or off, or *either* of which can always open or close a door, or turn some other device on or off.

Like AND and OR gates, XOR gates can freely be "stacked" together, with gates gathering groups of inputs and their outputs being gathered in turn. The result of XORing more than two inputs is called "parity" — the result is 1 if and only if an odd number of inputs are 1.

Design **D** is tiny, but only useful if players want the levers to be fixed to the circuit. The shaded block indicates the block the levers and the lit torch are attached to, along with the block that one is resting on.

Design **F** is the most widely used of the torch-only designs, but newer components can do much better. Design **H** uses pistons, and is both faster and more compact.

Beyond torches and pistons, various diodes can be used to produce fairly compact and cheap XOR gates. Design **I** can have its input repeaters coming in from either side or underneath, changing its size accordingly to fit tight spaces. Design **J** uses transparent blocks for a cheaper option.

Schematic gallery: XOR gate [show] [edit]

The introduction of the comparator allows for several variations of a new design, the "subtraction XOR gate", which is flat, fast and silent (also easy to remember). The cons in Survival mode is that making comparators requires the access to the Nether to obtain nether quartz.

Each input is the same distance to the rear and side of the comparator closest to it, so will suppress its own signal there, but travels farther to get to the side of the further comparator, so won't suppress its signal in the further comparator. Only if both inputs are on will both comparators get suppressed by a side input.

However, that is only true if the inputs are the same power level (or at least not different by more than 1), otherwise one signal could overwhelm the other's attempt

to suppress its signal. If this circuit is sure to receive inputs of the same power level (because the system it's part of has been designed that way), then the "basic" version can be used. Otherwise, some method should be used to ensure the inputs are equal — for example, with repeaters (the "repeated" version) or with torches (the "inverted" version).

Schematic gallery: subtraction XOR gate [show] [edit]

XNOR gate

A

B

A XNOR B	ON	off	off	ON

An **XNOR gate** (or) is the opposite of an XOR gate. This is commonly referred to as "if and only if" ("iff" [*sic*] for short), "bi-conditional", or "equivalence". It uses two inputs. When both switches are in the same state (both switches are "on" or both switches are "off"), then the output is toggled to "on". Otherwise, if the switches differ, the output is toggled to "off". Similar to the XOR gate, when either input changes, the output changes.

An XNOR gate can be built by inverting either the output, or *one* input, of an XOR gate.

Design **A** is a pure-torch design. If external input isn't needed, the back-facing torches can be replaced with levers, yielding **B**. Design **F** is larger but highlights the logic, while **I** is an inverted variant of XOR gate **H**. Note that the output inverter can also be placed in line with the rest of the gate, or even in a pit attached to one of the output redstone's support blocks.

Schematic gallery: XNOR gate [show] [edit]

IMPLY gate

A

B

A IMPLIES B

An **IMPLY gate** (A → B) turns on either if both inputs are on, or if the first input is off. Unlike the other gates here, the inputs are not interchangeable; it is not commutative. This represents material implication or a conditional statement, "if *A* then *B*", or "*A* implies *B*". The output is off only if the antecedent *A* is true, but the consequent *B* is false. It is the logical equivalent of B ∨ ¬A, and the mathematical equivalent of A ≤ B.

Design **C** has a speed of 2 ticks if output is 1, but 1 tick if the output is 0. Similarly, the other designs take 1 tick if the output is 0, but are immediate (and not isolated) if the output is 1. If the player must synchronize (or isolate) the output, consider placing a 1-tick repeater in front of the "fast" input (input A for **C**, input B for the others).