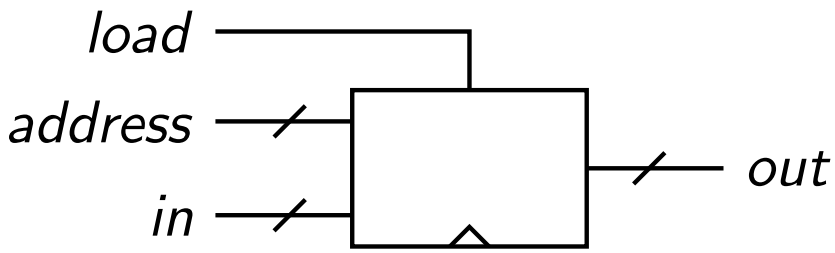
From registers to RAM
COMSM1302 Overview of Computer Architecture

John Lapinskas, University of Bristol

What is memory?

Memory behaves like a collection of registers. Each register has an **address** and stores one binary **word** of a set length (e.g. 16 bits for Hack).

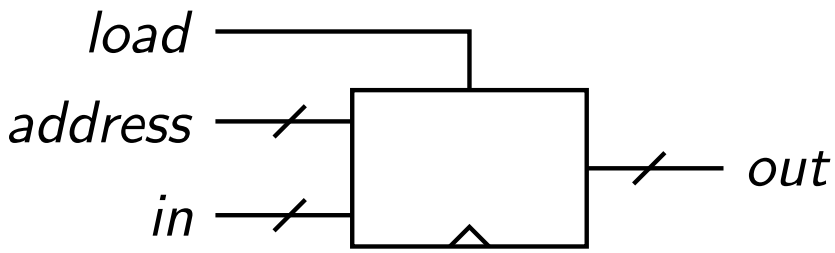


The output *out* is always set to the value stored in the register with address *address*. This happens immediately, not on a clock tick.

If *load* is high, then on a rising clock edge, the value stored in the register with address *address* is set to *in*. If *load* is low, it doesn't change.

What is memory?

Memory behaves like a collection of registers. Each register has an **address** and stores one binary **word** of a set length (e.g. 16 bits for Hack).



The output *out* is always set to the value stored in the register with address *address*. This happens immediately, not on a clock tick.

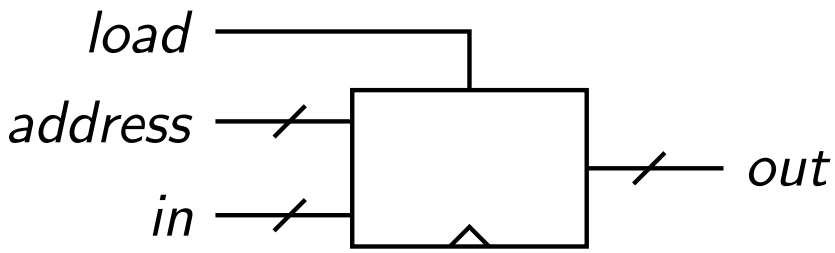
If *load* is high, then on a rising clock edge, the value stored in the register with address *address* is set to *in*. If *load* is low, it doesn't change.

Key terminology:

- The size of each register is called the **word size**.
- The set of valid addresses is called the **address space**.
- Each 1-bit register (inside the larger registers) is called a **cell**.

What is memory?

Memory behaves like a collection of registers. Each register has an **address** and stores one binary **word** of a set length (e.g. 16 bits for Hack).



The output *out* is always set to the value stored in the register with address *address*. This happens immediately, not on a clock tick.

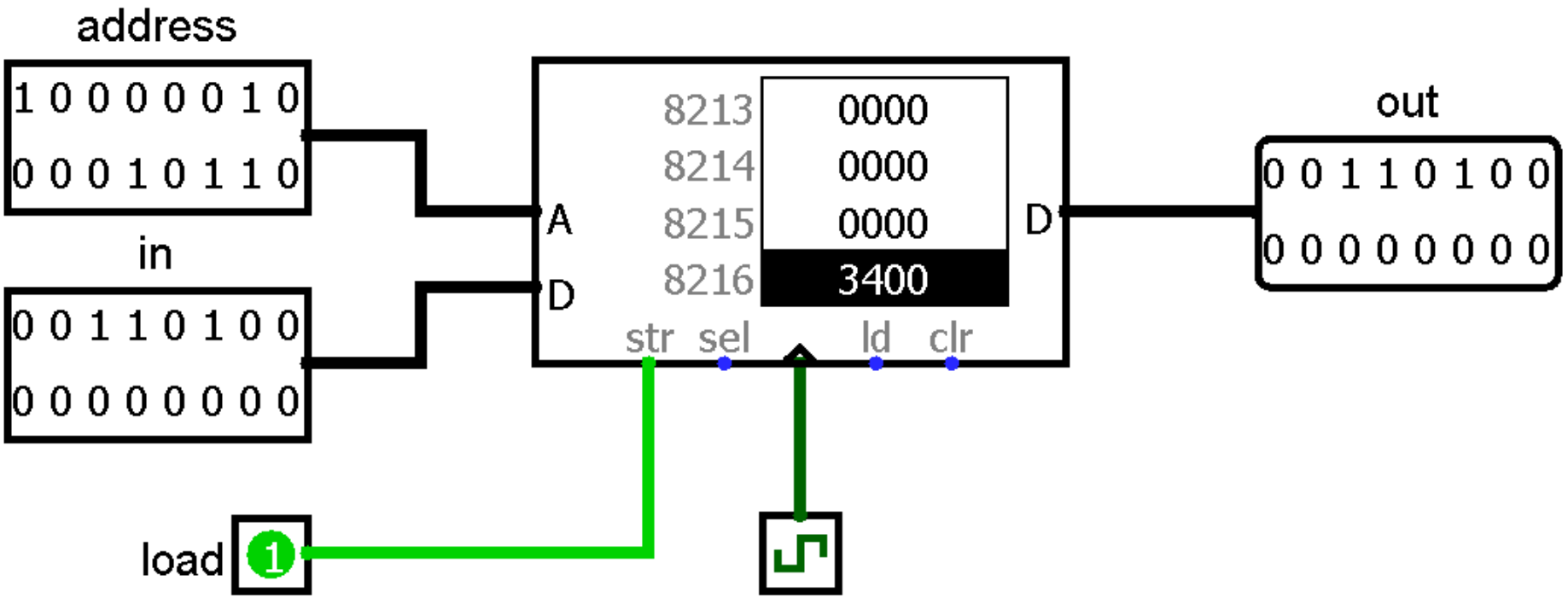
If *load* is high, then on a rising clock edge, the value stored in the register with address *address* is set to *in*. If *load* is low, it doesn't change.

Key terminology:

- The size of each register is called the **word size**.
- The set of valid addresses is called the **address space**.
- Each 1-bit register (inside the larger registers) is called a **cell**.

How to build memory? You should be able to do this — see assignment!

Demonstration



[See video for a demonstration of the built-in RAM component in Logisim.]

RAM and ROM

The memory we've built out of flip-flops is **volatile**: it stores data only as long as it's kept supplied with power. Volatile memory is also called **RAM** (**Random Access Memory**) for historical reasons that no longer apply.

Non-volatile storage, by contrast, stores data even while un-powered.

ROM (**Read-Only Memory**) is a type of non-volatile memory which is only written to once, physically altering the chip. The advantage is low cost and faster access times — it might be used for e.g. non-upgradable firmware.

The Hack CPU uses 16KB RAM and 32KB ROM for ease of implementation.

Most modern non-volatile memory is **EEPROM** (**Electrically Erasable Programmable ROM**) including flash memory, which can be slowly (on CPU timescales!) erased and rewritten.

RAM and ROM

The memory we've built out of flip-flops is **volatile**: it stores data only as long as it's kept supplied with power. Volatile memory is also called **RAM** (**Random Access Memory**) for historical reasons that no longer apply.

Non-volatile storage, by contrast, stores data even while un-powered.

ROM (**Read-Only Memory**) is a type of non-volatile memory which is only written to once, physically altering the chip. The advantage is low cost and faster access times — it might be used for e.g. non-upgradable firmware.

The Hack CPU uses 16KB RAM and 32KB ROM for ease of implementation.

Most modern non-volatile memory is **EEPROM** (**Electrically Erasable Programmable ROM**) including flash memory, which can be slowly (on CPU timescales!) erased and rewritten.

So e.g. flash drives and SSDs use flash memory, which is EEPROM, but they definitely aren't read-only. But like all modern memory they *are* random-access! The terminology is all very sensible. 😭 😭 😭

SRAM and DRAM

The RAM we've built is similar to **SRAM** (**Static RAM**).

SRAM uses a highly efficient design that only needs 4–6 transistors per bit, where ours would need ∼40. But both work on similar principles, and both hold their data as long as they're supplied with power.

DRAM (**Dynamic RAM**) works totally differently, storing data in a way that needs to be manually refreshed every few ms and on each read. Compared to SRAM it's cheaper and higher-capacity, but much slower.

Modern computers use both DRAM and SRAM — see later in the unit!

SDRAM (**Synchronised DRAM**) is DRAM driven by a clock. All modern DRAM is SDRAM; this has absolutely nothing to do with SRAM.

DDR (**Double Data Rate**) 1 through 5 are a family of protocols for data transfer to and from DRAM, not fundamentally different technologies.

Addendum: SI units for storage

Memory size "naturally" comes in powers of two, so we normally use these alternative SI units, where e.g. 1KB means 1024 bytes rather than 1000:

Base 10	Symbol	Name	Base 2	Symbol	Name
	⋮			⋮	
10^{15}	P	Peta-	2^{50}	P or Pi	Peta- or pebi-
10^{12}	T	Tera-	2^{40}	T or Ti	Tera- or tebi-
10^{9}	G	Giga-	2^{30}	G or Gi	Giga- or gibi-
10^{6}	M	Mega-	2^{20}	M or Mi	Mega- or mebi-
10^{3}	**k**	Kilo-	2^{10}	**K** or Ki	Kilo- or kibi-
10^{0}	N/A	N/A	2^{0}	N/A	N/A

This allows for e.g. two 16GB sticks of RAM to be combined into 32GB total RAM by just adding one bit to the address space.

Everyone agrees on this except non-volatile storage manufacturers, who quite like being able to advertise e.g. a "1TB drive" while only providing a 10^{12}-byte drive. And they have lawyers, so the usual P/T/G/M prefixes are often ambiguous...

In this unit, we will always use base 2 units for bytes.

Addendum: A refresher(?) on logarithms

$\log_b a$ ("Log base b of a") is the number such that $b^{\log_b a} = a$.

For example, $\log_2 16 = 4$, $\log_2 256 = 8$, and $\log_2 1 = 0$.

Logarithms are occasionally useful to us as a tool: for example, if we want to know how many bits we need to represent x as an (unsigned) binary number, or to store an address for an x-word memory, the answer will be $\log_2 x$ rounded up.

Addendum: A refresher(?) on logarithms

$\log_b a$ ("Log base b of a") is the number such that $b^{\log_b a} = a$.

For example, $\log_2 16 = 4$, $\log_2 256 = 8$, and $\log_2 1 = 0$.

Logarithms are occasionally useful to us as a tool: for example, if we want to know how many bits we need to represent x as an (unsigned) binary number, or to store an address for an x-word memory, the answer will be $\log_2 x$ rounded up.

For historical reasons, your calculator probably uses logarithms base 10 ("log") or base $e \approx 2.718$ ("ln"). We can still calculate $\log_2 x$, using the fact that for any base b, we have $\log_2 x = (\log_b x)/(\log_b 2)$. This is because

$$2^{(\log_b x)/(\log_b 2)} = (b^{\log_b 2})^{(\log_b x)/(\log_b 2)} = b^{\log_b x} = x.$$

Addendum: A refresher(?) on logarithms

$\log_b a$ ("Log base b of a") is the number such that $b^{\log_b a} = a$.

For example, $\log_2 16 = 4$, $\log_2 256 = 8$, and $\log_2 1 = 0$.

Logarithms are occasionally useful to us as a tool: for example, if we want to know how many bits we need to represent x as an (unsigned) binary number, or to store an address for an x-word memory, the answer will be $\log_2 x$ rounded up.

For historical reasons, your calculator probably uses logarithms base 10 ("log") or base $e \approx 2.718$ ("ln"). We can still calculate $\log_2 x$, using the fact that for any base b, we have $\log_2 x = (\log_b x)/(\log_b 2)$. This is because

$$2^{(\log_b x)/(\log_b 2)} = (b^{\log_b 2})^{(\log_b x)/(\log_b 2)} = b^{\log_b x} = x.$$

Also, $\log_b x$ grows very very slowly compared to e.g. x. In Programming in C, you'll see algorithms which run on inputs of size n in time that grows with $\log_2 n$. If $\log_2 n \geq 50$ then $n \geq 1\text{PB}$! So you can think of $\log_2 n$ as effectively being a large-ish constant, to be traded off against other constant factors.

Addendum: A refresher(?) on logarithms

$\log_b a$ ("Log base b of a") is the number such that $b^{\log_b a} = a$.

For example, $\log_2 16 = 4$, $\log_2 256 = 8$, and $\log_2 1 = 0$.

Logarithms are occasionally useful to us as a tool: for example, if we want to know how many bits we need to represent x as an (unsigned) binary number, or to store an address for an x-word memory, the answer will be $\log_2 x$ rounded up.

For historical reasons, your calculator probably uses logarithms base 10 ("log") or base $e \approx 2.718$ ("ln"). We can still calculate $\log_2 x$, using the fact that for any base b, we have $\log_2 x = (\log_b x)/(\log_b 2)$. This is because

$$2^{(\log_b x)/(\log_b 2)} = (b^{\log_b 2})^{(\log_b x)/(\log_b 2)} = b^{\log_b x} = x.$$

Also, $\log_b x$ grows very very slowly compared to e.g. x. In Programming in C, you'll see algorithms which run on inputs of size n in time that grows with $\log_2 n$. If $\log_2 n \geq 50$ then $n \geq 1\text{PB}$! So you can think of $\log_2 n$ as effectively being a large-ish constant, to be traded off against other constant factors.

That's all you'll ever need to know about logarithms (for this degree)!