Memory

We can view memory as being composed of many registers, each at a specific location (known as an address).
This is not exactly true, but functionally/conceptually it makes sense.

We can picture a grid, where each location is an 8-bit register. The position in memory then is given by a row-column (or column-row) address.

R.A.M.

We call this Random Access Memory because it takes the same amount of time to access any arbitrary location.

Memory Address Register

There's then a Memory Address Register, which is usually about 32-bits, with 16 bits dedicated to storing row info, and 16 bits dedicated to column info. You can have 2^n different addresses, where n = 32 for a 32-bit machine.

Address Bus

Feeding into the memory address register is a 32-bit bus, which is 32 copper lines.

Memory Buffer Register

As well as the memory address register, there's another, smaller register (typically 8 bits), which is called the Memory Buffer Register.

Control lines

- write
    - 8 bits are put into the memory buffer register, and when the write line is activated, these 8 bits will

flow into the memory address specified in the memory address register

- read
  - for a read operation, the data flow is in the other direction, from a memory address into the memory buffer register
- enable

Data Bus

An 8-bit bus (8 wires) which takes data from the memory buffer register out, or into the buffer from elsewhere.

What is a Program?

A program is a sequence of instructions (usually designed in order to solve a problem).

What is an instruction?

An instruction is a sequence of bits designed to activate a specific functional component in the underlying machine (and to 'feed' these components with data on which they can operate).

Functional components

Functional components include:

- adders
- subtractors
- multipliers (repeated addition)
- logic functions (for ANDing/ORing/etc.)

These components are typically combined into an Arithmetic-Logic Unit (an ALU).

Other functional components include:

- registers

- memory
- etc. (not much else)

A.L.U.

Operands flow in, and results flow out.

Example:

An instruction might be:

```
ADD A, B, C
```

Where A, B, and C are identifications of registers. But there's no such thing as 'A' in the machine (or 'B' or 'C'), so there'll need to be some translation there.

The registers A and B hold the operands of the operation ADD. The result is put in C. In general, all operands need to be in registers before an instruction can take place.

Registers are identified by a number. So A might be 0, B might be 1, and C might be 2.

In "ADD A, B, C" the "ADD" is known as the operation code ("opcode" for short), and each opcode is also just a number.

So, "ADD A, B, C" might look like:

```
1010 000  001  010
ADD  A    B    C
```

The reading device (controller or whatever) is at first expecting an instruction. It looks at the ADD instruction and knows from that to read the next three things as memory addresses.