# Nand_Game

August 22, 2019

## 1 NAND Game

The Nand Game is inspired by the amazing course From NAND to Tetris - Building a Modern Computer From First Principles which is highly recommended.

### 1.1 Introduction

The Nand Game takes you though building a working computer, starting from the most basic components. It does not require any prerequisites, in particular it does not require any previous knowledge about computer architecture or software, and does not require math skills beyond addition and subtraction. It does require some patience—some of the tasks might take a while to solve.

The game consists of a series of levels. In each level you are tasked with building a component that behaves according to a specification. This component can then be used as a building-block in the next level.

All components are specified through what input should lead to what output. How exactly you build the component is up to you, as long as input/output conforms to the specification. The game doesn't care whether or not you have found the simplest or the most efficient design. It only cares if it works correctly.

The first challenge is to build an inverter component.

The inverter has a single input and a single output, and the specification look like this:

The Nand gate

The only component available in the first level is the nand gate. The nand gate is a fundamental building block which all other components can be built from.

A nand gate has two inputs and one output, and the specification is like this:

| In1 | In2 | Out |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

## 1.2 Logic Gates

### 1.2.1 Invert

An inv-component has a single input and a single output. The output should be the opposite of the input, so 0 for 1 and vice versa. Components are typically specified with a table showing inputs and outputs, like this:

$$\neg a = \overline{a}$$

| A | Q |
|---|---|
| 0 | 1 |
| 1 | 0 |

**Solution** From a Nand erase the two middle entries.

| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| ~~1~~ | ~~0~~ | ~~1~~ |
| ~~0~~ | ~~1~~ | ~~1~~ |
| 1 | 1 | 0 |


### 1.2.2 And

An and gate output is 1 when both inputs are 1:

$$a \wedge b = ab$$

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

**Solution** It is just an invertet NAND



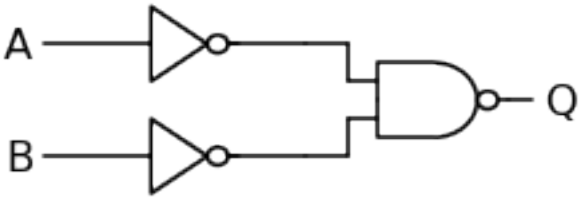
### 1.2.3 Or

An or gate output is 1 when at least one input is 1:

$$a \vee b = a + b$$

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

**Solution** Invert Inputs of a NAND

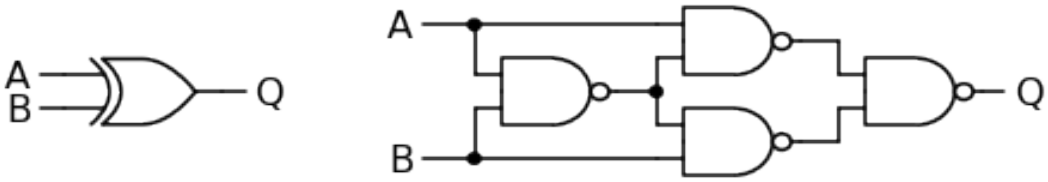

### 1.2.4 Xor

An xor gate output is 1 when the two inputs are different:

$$a \oplus b = a\overline{b} + \overline{a}b$$

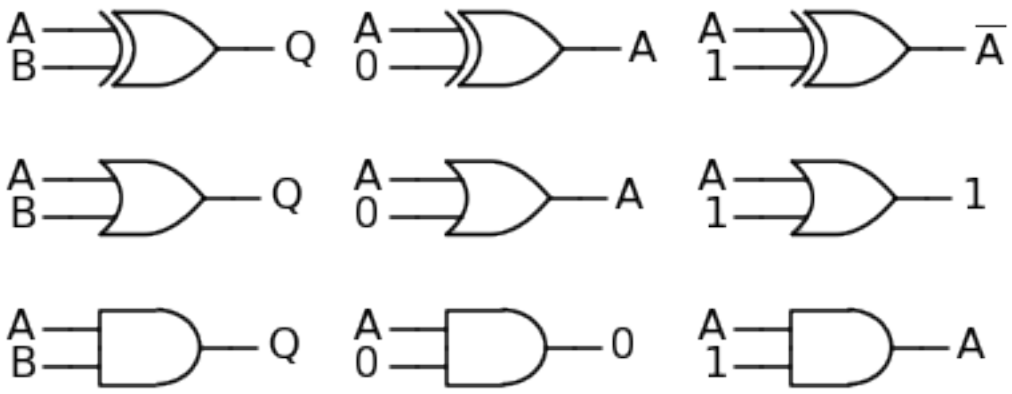
| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

**Solution** a not(b) or b not (a)




### 1.2.5 Logic Gates Appendium

- `AND` is used to let through or set to `'0'`
- `OR` is used to let through or set to `'1'`
- `XOR` is used to let through or invert signal



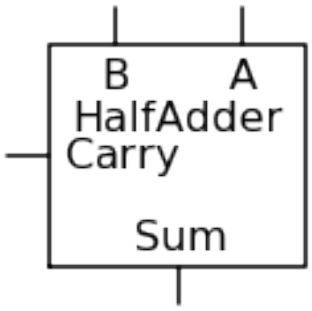

## 1.3 Arithmetics

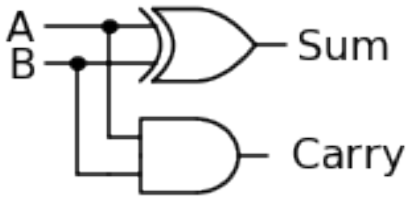
### 1.3.1 Half Adder

Half Adder

An add component which adds two bits. The output is a two-bit value.
The h output is the high bit, the l is the low bit.

| A | B | Carry | Sum |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |



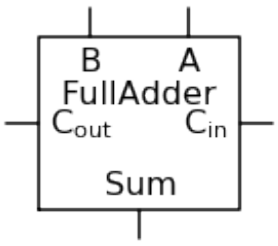

### 1.3.2 Full Adder

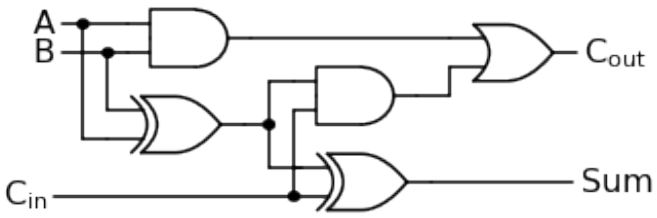
An add component which adds three bits: a, b, and c.

The output is a two-bit value. Carry and Sum with is the HSb and LSb.

| A | B | $C_{in}$ | $C_{out}$ | Sum |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**Solution**: * Two Half-Adder can be linked to create a Full-Substractor combines with an Or

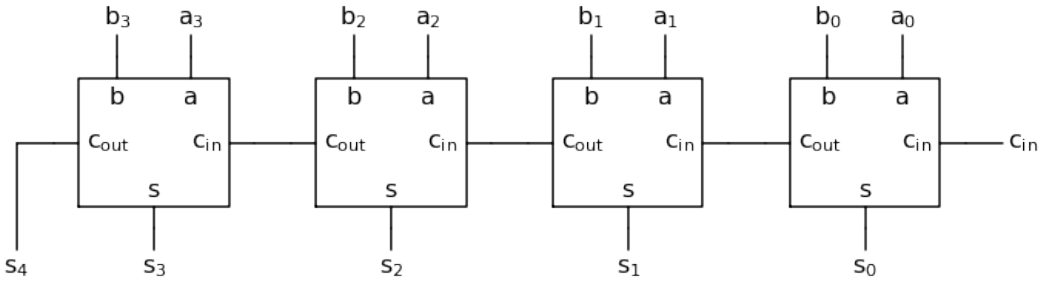



### 1.3.3 Multi-Bit Adder

Create 4Bit-Adder

**Solution**: * Chain Full-Adders together

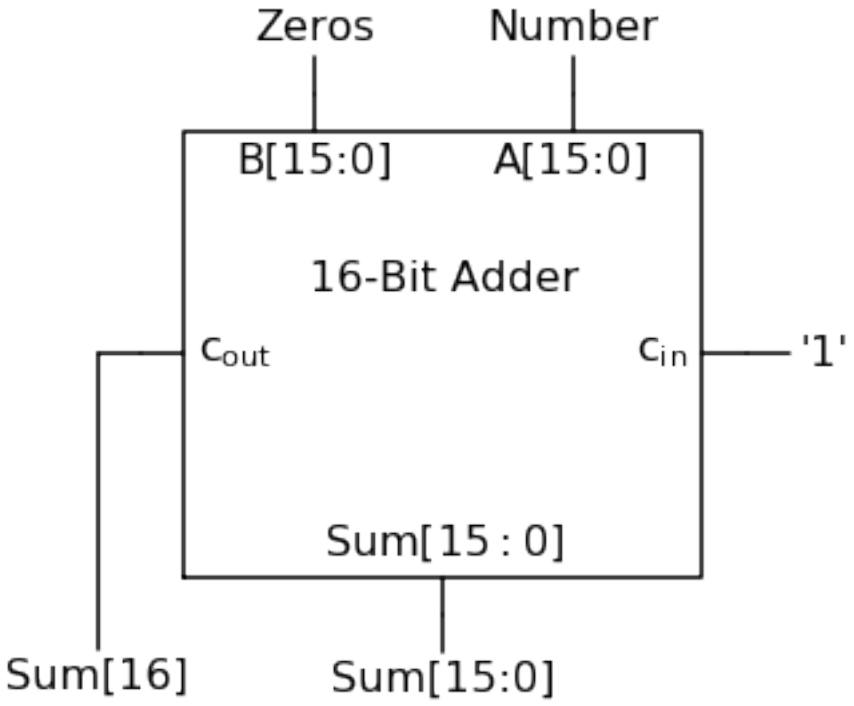



### 1.3.4 Increment

Add 1 to a 16-bit number.

Ignore the carry if the result is larger than 16 bits

**Solution** * Chain Full-Adders and set A to the number and B to Zero. The +1 comes from the cin which is set to '1'.



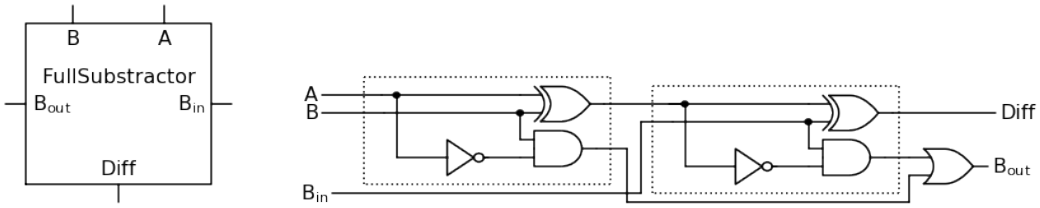

### 1.3.5 Subtraction

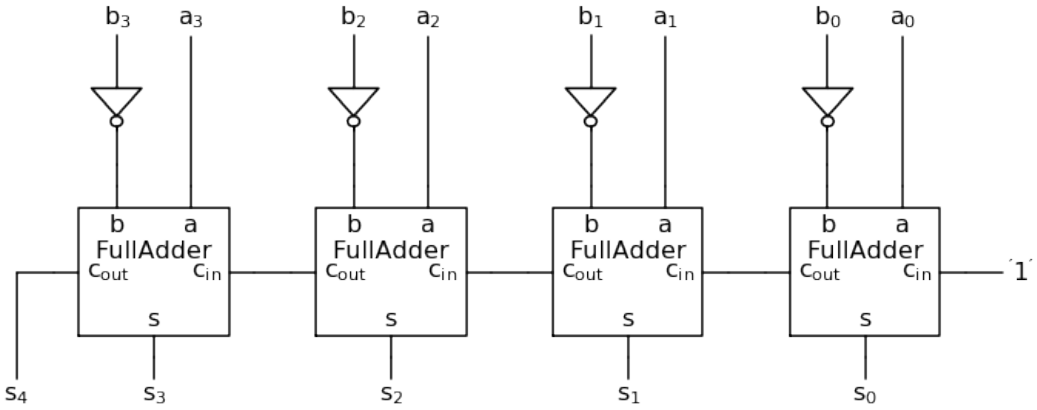
Outputs A minus B as a 16-bit number.

If the result is less than zero it is represented as 65536 plus the result.
Examples:

| Result | 16-bit Binary | unsigned decimal |
|---|---|---|
| 1 | 0000000000000001 | 1 |
| 0 | 0000000000000000 | 0 |
| -1 | 1111111111111111 | 65535 |
| -2 | 1111111111111110 | 65534 |
| -3 | 1111111111111101 | 65533 |

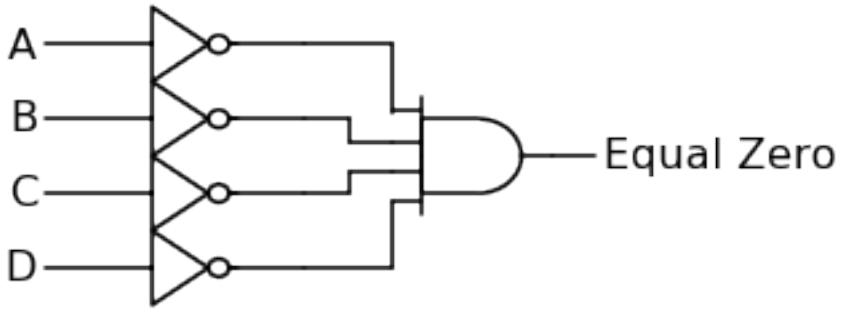
(This is equivalent to two's complement representation)
**Solution**: * Two Half-Substractor can be linked to create a Full-Substractor




### 1.3.6 Equal to Zero

Should output 1 if and only if all bits in the input are 0.




#### 1.3.7 Less than Zero

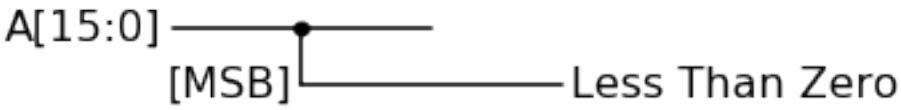
Outputs 1 if the input as a 16-bit number is negative

Specification:

| In | Out |
|---|---|
| In >= 0 | 0 |
| In < 0 | 1 |

A number is considered less than zero if bit 15 is 1.

Bit numbering Bits are numbered from right to left, starting with 0 as the rightmost bit. So bit 15 is the leftmost bit in a 16-bit word.




### 1.4 Plumbing

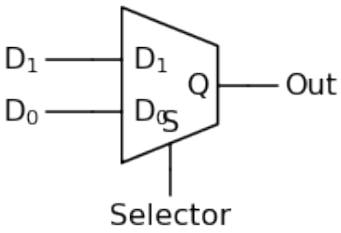
#### 1.4.1 Selector / Demuxliplexer

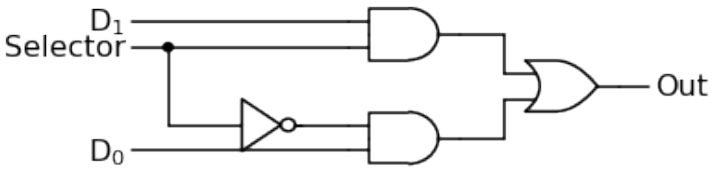
A select-component selects one out of two input bits for output.

The s (select) bit indicates which input is selected: If 0, d0 is selected, if 1, d1 is selected.

| S | D1 | D0 | Output |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |

| S | D1 | D0 | Output |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

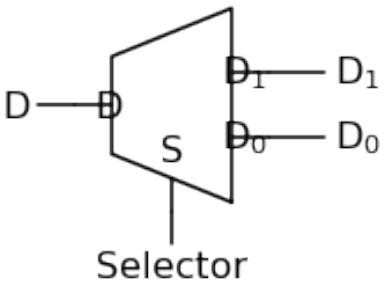


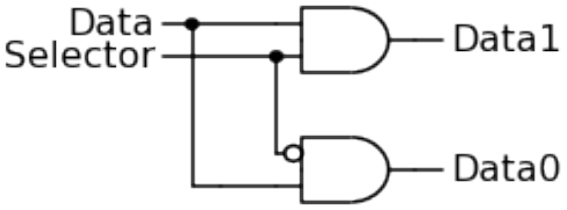




### 1.4.2 Switch /Multiplexer

A switch component channels a data bit through one of two output channels.
s (selector) determines if the d (data) bit is dispatched through c1 or c0.

| S | D | D1 | D0 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 |



## 1.5 Memory

### 1.5.1 Latch

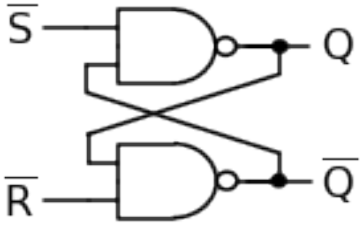
$\overline{SR} - Latch$ and $SR - Latch$

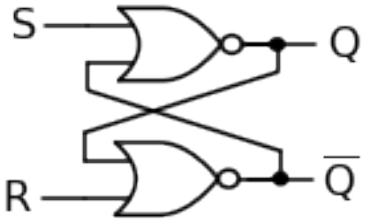
A latch component stores and outputs a single bit

To describe this in an input/output table, we introduce a variable, out, which can be assigned a bit value and keep it:

| $\overline{Set}$ | $\overline{Reset}$ | Q | State |
|---|---|---|---|
| 0 | 0 | ? | Forbidden |
| 0 | 1 | 1 | Set |
| 1 | 0 | 0 | Reset |
| 1 | 1 | Previous State | out |

| Set | Reset | Q | State |
|---|---|---|---|
| 0 | 0 | Previous State | Store |
| 0 | 1 | 0 | Reset |
| 1 | 0 | 1 | Set |
| 1 | 1 | ? | Forbidden |




### 1.5.2 Data Latch

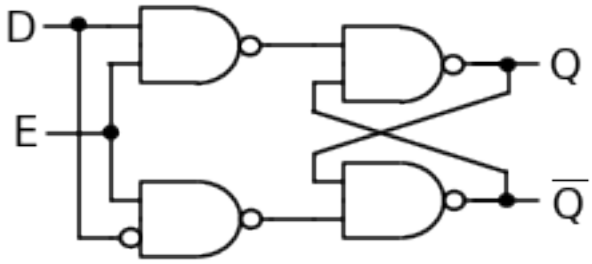
A latch component stores and outputs a single bit

When st (store) is `1`, the value on d is stored and emitted.

When st is `0`, the value of d is ignored, and the previously stored value is still emitted.

To describe this in an input/output table, we introduce a variable, out, which can be assigned a bit value and keep it:

| E | D | Effect | Output |
|---|---|---|---|
| 1 | 0 | set out to 0 | out |
| 1 | 1 | set out to 1 | out |
| 0 | 1 | - | out |
| 0 | 0 | - | out |




### 1.5.3 Data Flip-Flop

A DFF (Data Flip-Flop) component stores and outputs a bit, but only change the output when the clock signal change from `0` to `1`.

When st (store) is `1` and clk (clock signal) is `0` the value on d is stored. But the previous value is still emitted.

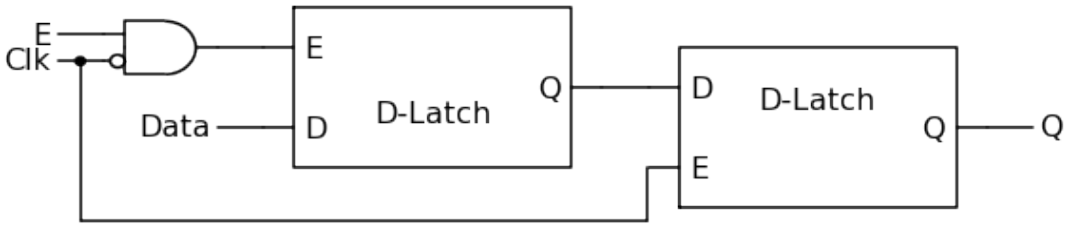
When the clock signal changes to `1`, the flip-flop starts emitting the new value.

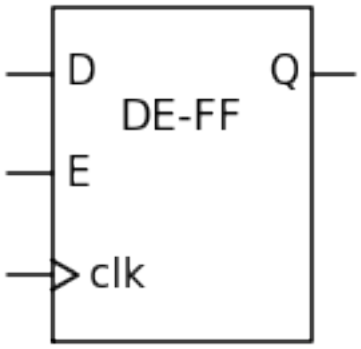
When st is `0`, the value of d does not have any effect.

When clk is `1`, the value of st and d does not have any effect.

To describe this in a table requires two variables, in and out:

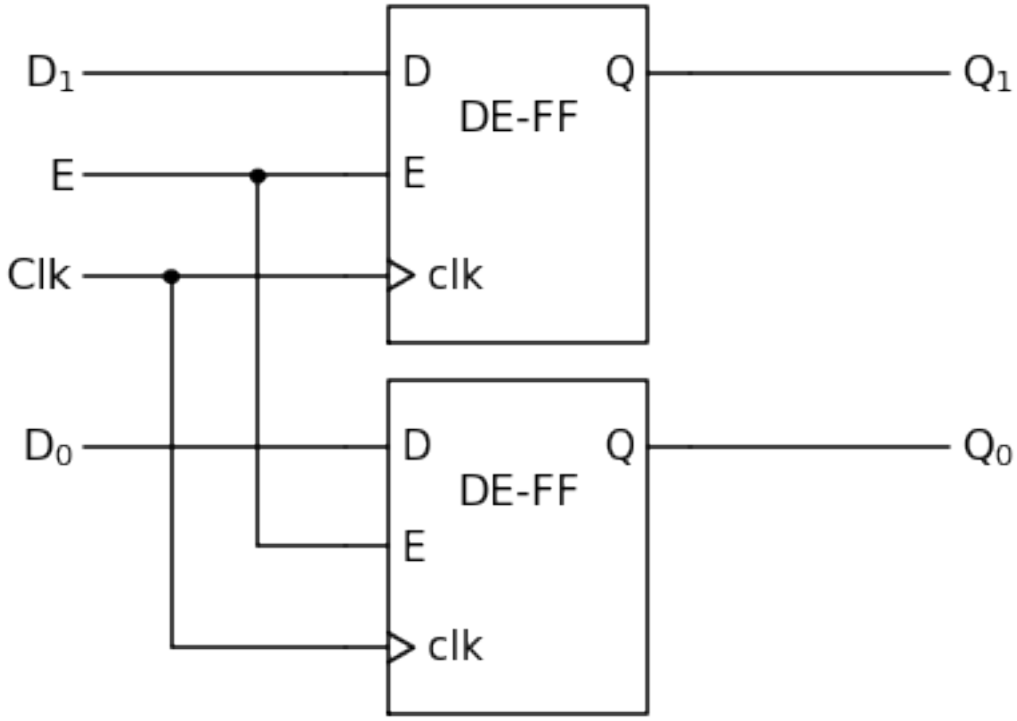
| E | D | Clk | Effect | Out |
|---|---|---|---|---|
| 1 | 0 | 0 | set in to 0 | out |
| 1 | 1 | 0 | set in to 1 | out |
| 0 | - | 0 | - | out |
| - | - | 1 | set out to in | out |






### 1.5.4 Register

A 2-bit DFF component works like a data flip-flop, except two bits (d1 and d0) are stored and emitted instead of one.



### 1.5.5 Counter

### 1.5.6 RAM

## 1.6 Arithmetic Logic Unit

### 1.6.1 Unary ALU

### 1.6.2 ALU

### 1.6.3 Condition

## 1.7 Processor

### 1.7.1 Combined Memory

### 1.7.2 Instruction Decoder

### 1.7.3 Control Unit

### 1.7.4 Program Engine

### 1.7.5 Computer

### 1.7.6 Input and Output