

"How does a computer work?"

"How does a computer work?"

- "Easy: just turn it on, and type stuff."
- "Electricity."
- "There's a chip."
- "Depends if you have a Windows or a Google."

Application software



Code

```
static void
on_open_image (GtkButton* button, gpointer user_data)
{
    GtkWidget *image = GTK_WIDGET (user_data);
    GtkWidget *toplevel = gtk_widget_get_toplevel (image);
    GtkFileFilter *filter = gtk_file_filter_new ();
    GtkWidget *dialog = gtk_file_chooser_dialog_new (_("Open image"),
                                                    GTK_WINDOW (toplevel),
                                                    GTK_FILE_CHOOSER_ACTION_OPEN,
                                                    GTK_STOCK_OK, GTK_RESPONSE_ACCEPT,
                                                    GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
                                                    NULL);

    gtk_file_filter_add_pixbuf_formats (filter);
    gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dialog),
                                filter);
    switch (gtk_dialog_run (GTK_DIALOG (dialog)))
    {
        case GTK_RESPONSE_ACCEPT:
        {
            gchar *filename =
                gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog));
            gtk_image_set_from_file (GTK_IMAGE (image), filename);
            break;
        }
        default:
            break;
    }
    gtk_widget_destroy (dialog);
}
```

Assembly language

```
#include <stdio.h>
int main() {
printf("Hello world\n");
return 0;
}
```

compiler



```
.file "helloworld.c"
.text
.section .rodata
.LC0:
.string "Hello world"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
leaq .LC0(%rip), %rdi
call puts@PLT
movl $0, %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (Debian 8.3.0-6) 8.3.0"
.section .note.GNU-stack,"",@progbits
```

Machine language

```
.file    "helloworld.c"
.text
.section .rodata
.LC0:
.string  "Hello world"
.text
.globl   main
.type    main, @function
main:
.LFB0:
.cfi_startproc
pushq    %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
leaq .LC0(%rip), %rdi
call puts@PLT
movl $0, %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size    main, .-main
.ident   "GCC: (Debian 8.3.0-6) 8.3.0"
.section .note.GNU-stack,"",@progbits
```

assembler



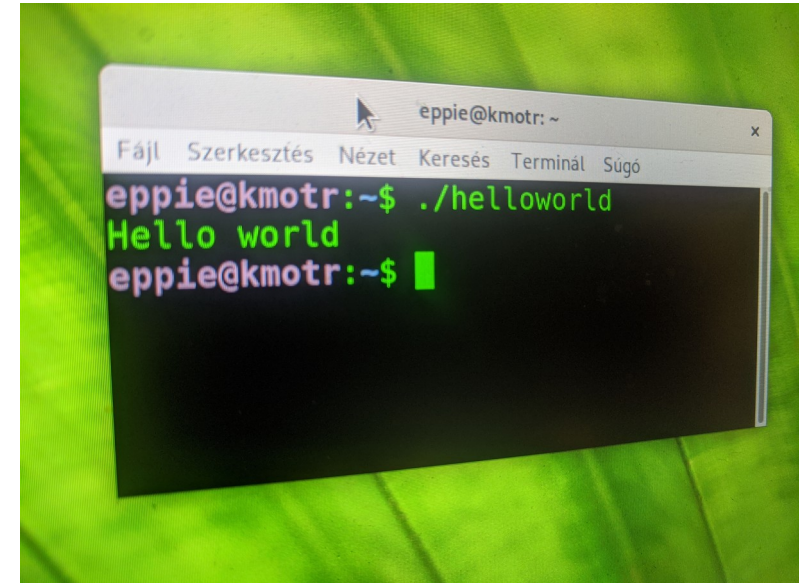
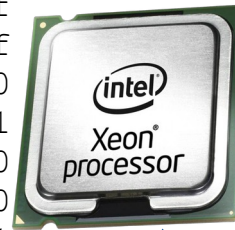
helloworld.exe

01	00	02	00	48	65	6c	6c	6f	20	77	6f	72	6c	64	00
01	1b	03	3b	3c	00	00	00	06	00	00	00	10	f0	ff	ff
88	00	00	00	30	f0	ff	ff	b0	00	00	00	40	f0	ff	ff
58	00	00	00	25	f1	ff	ff	c8	00	00	00	40	f1	ff	ff
e8	00	00	00	a0	f1	ff	ff	30	01	00	00	00	00	00	00
14	00	00	00	00	00	00	00	01	7a	52	00	01	78	10	01
1b	0c	07	08	90	01	07	10	14	00	00	00	1c	00	00	00
e0	ef	ff	ff	2b	00	00	00	00	00	00	00	00	00	00	00
14	00	00	00	00	00	00	00	01	7a	52	00	01	78	10	01
1b	0c	07	08	90	01	00	00	24	00	00	00	1c	00	00	00
80	ef	ff	ff	20	00	00	00	00	0e	10	46	0e	18	4a	0f
0b	77	08	80	00	3f	1a	3b	2a	33	24	22	00	00	00	00
14	00	00	00	44	00	00	00	78	ef	ff	ff	08	00	00	00
00	00	00	00	00	00	00	00	1c	00	00	00	5c	00	00	00
55	f0	ff	ff	17	00	00	00	00	41	0e	10	86	02	43	0d
06	52	0c	07	08	00	00	00	44	00	00	00	7c	00	00	00
50	f0	ff	ff	5d	00	00	00	00	42	0e	10	8f	02	45	0e
18	8e	03	45	0e	20	8d	04	45	0e	28	8c	05	48	0e	30
86	06	48	0e	38	83	07	47	0e	40	6a	0e	38	41	0e	30
41	0e	28	42	0e	20	42	0e	18	42	0e	10	42	0e	08	00

Execution

helloworld.exe

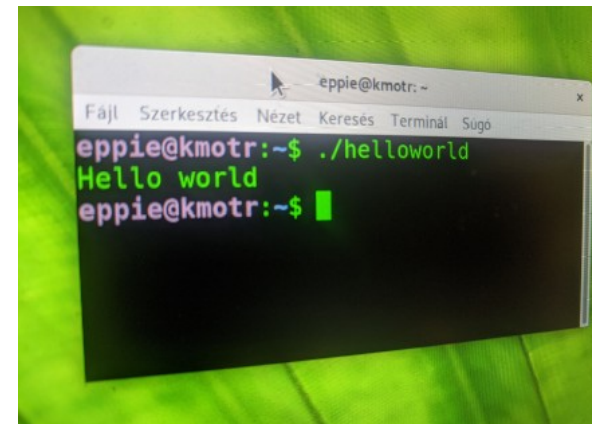
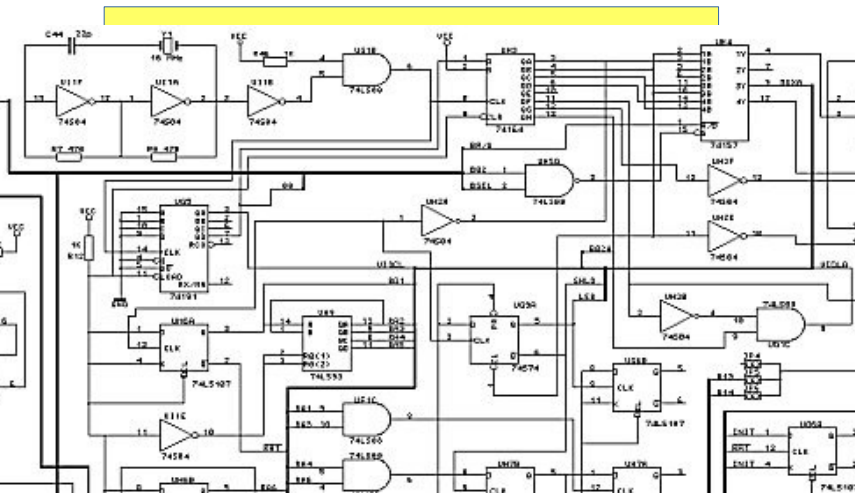
```
01 00 02 00 48 65 6c 6c 6f 20 77 6f 72 6c 64 00
01 1b 03 3b 3c 00 00 00 06 00 00 00 10 f0 ff ff
88 00 00 00 30 f0 ff ff b0 00 00 00 40 f0 ff ff
58 00 00 00 25 f1 ff ff c8 00 00 00 40 f1 ff ff
e8 00 00 00 a0 f1 ff ff 30 01 00 00 00 00 00 00
14 00 00 00 00 00 00 00 01 7a 52 00 01 78 10 01
1b 0c 07 08 90 01 07 10 14 00 00 00 1c 00 00 00
e0 ef ff ff 2b 00 00 00 00 00 00 00 00 00 00
14 00 00 00 00 00 00 00 01 7a 52 00 01 78 10 01
1b 0c 07 08 90 01 00 00 24 00 00 00 1c 00 00 00
80 ef ff ff 20 00 00 00 00 0e 10 46 0e 18 4a 0f
0b 77 08 80 00 3f 1a 3b 2a 33 24 22 00 00 00 00
14 00 00 00 44 00 00 00 78 ef ff ff 08 00 00 00
00 00 00 00 00 00 00 00 1c 00 00 00 5c 00 00 00
55 f0 ff ff 17 00 00 00 00 41 0e 10 86 02 43 0d
06 52 0c 07 08 00 00 00 44 00 00 00 7c 00 00 00
50 f0 ff ff 5d 00 00 00 00 42 0e 10 8f 02 45 0e
18 8e 03 45 0e 20 8d 04 45 0e 28 8c 05 48 0e 30
86 06 48 0e 38 83 07 47 0e 40 6a 0e 38 41 0e 30
41 0e 28 42 0e 20 42 0e 18 42 0e 10 42 0e 08 00
```



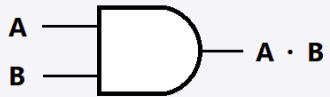
Execution

helloworld.exe

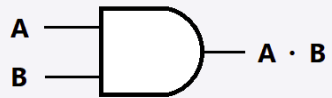
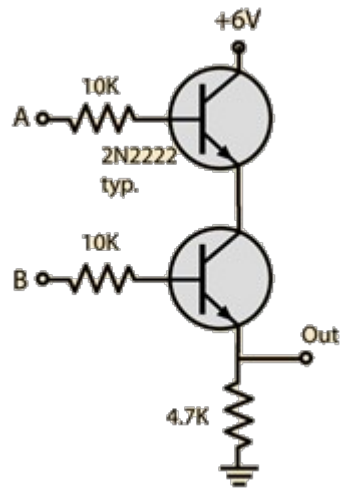
```
01 00 02 00 48 65 6c 6c 6f 2
01 1b 03 3b 3c 00 00 00 06 0
88 00 00 00 30 f0 ff ff b0 0
58 00 00 00 25 f1 ff ff c8 0
e8 00 00 00 a0 f1 ff ff 30 0
14 00 00 00 00 00 00 00 01 7
1b 0c 07 08 90 01 07 10 14 0
e0 ef ff ff 2b 00 00 00 00 0
14 00 00 00 00 00 00 00 01 7
1b 0c 07 08 90 01 00 00 24 0
80 ef ff ff 20 00 00 00 00 0
0b 77 08 80 00 3f 1a 3b 2a 3
14 00 00 00 44 00 00 00 78 e
00 00 00 00 00 00 00 00 1c 0
55 f0 ff ff 17 00 00 00 00 4
06 52 0c 07 08 00 00 00 44 0
50 f0 ff ff 5d 00 00 00 00 4
18 e0 03 45 0e 20 8d 04 45 0
86 06 48 0e 38 83 07 47 0e 4
41 0e 28 42 0e 20 42 0e 18 4
```



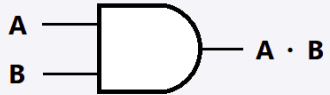
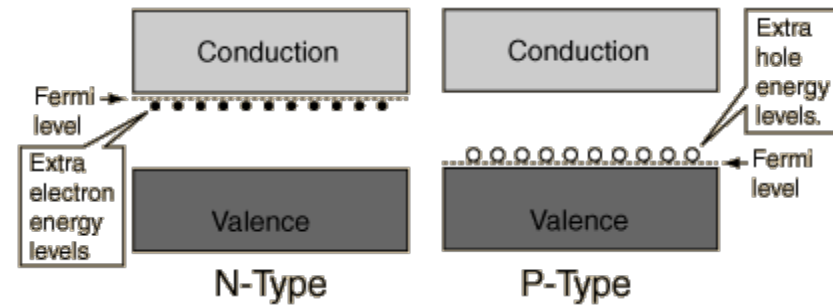
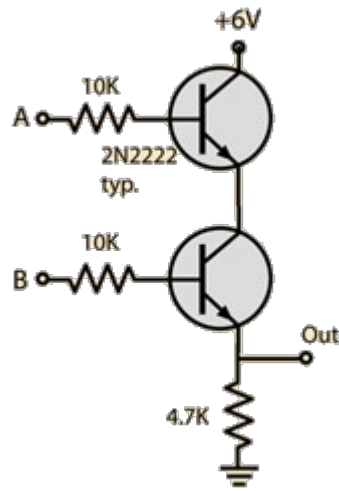
Gates, transistors, semiconductors, silicon



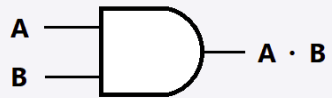
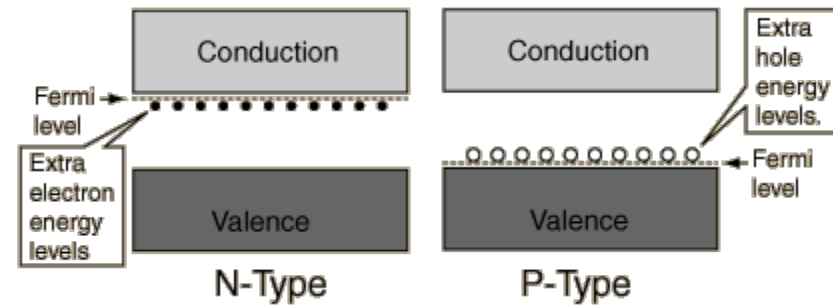
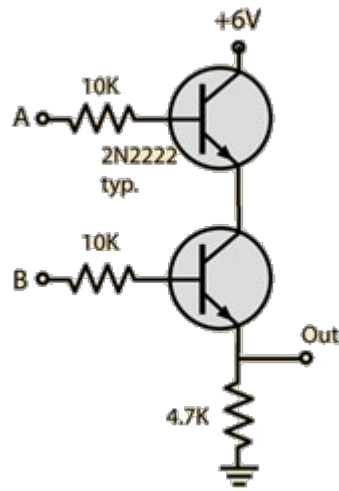
Gates, transistors, semiconductors, silicon

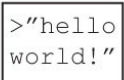


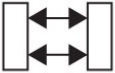
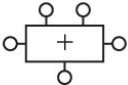

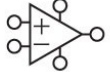




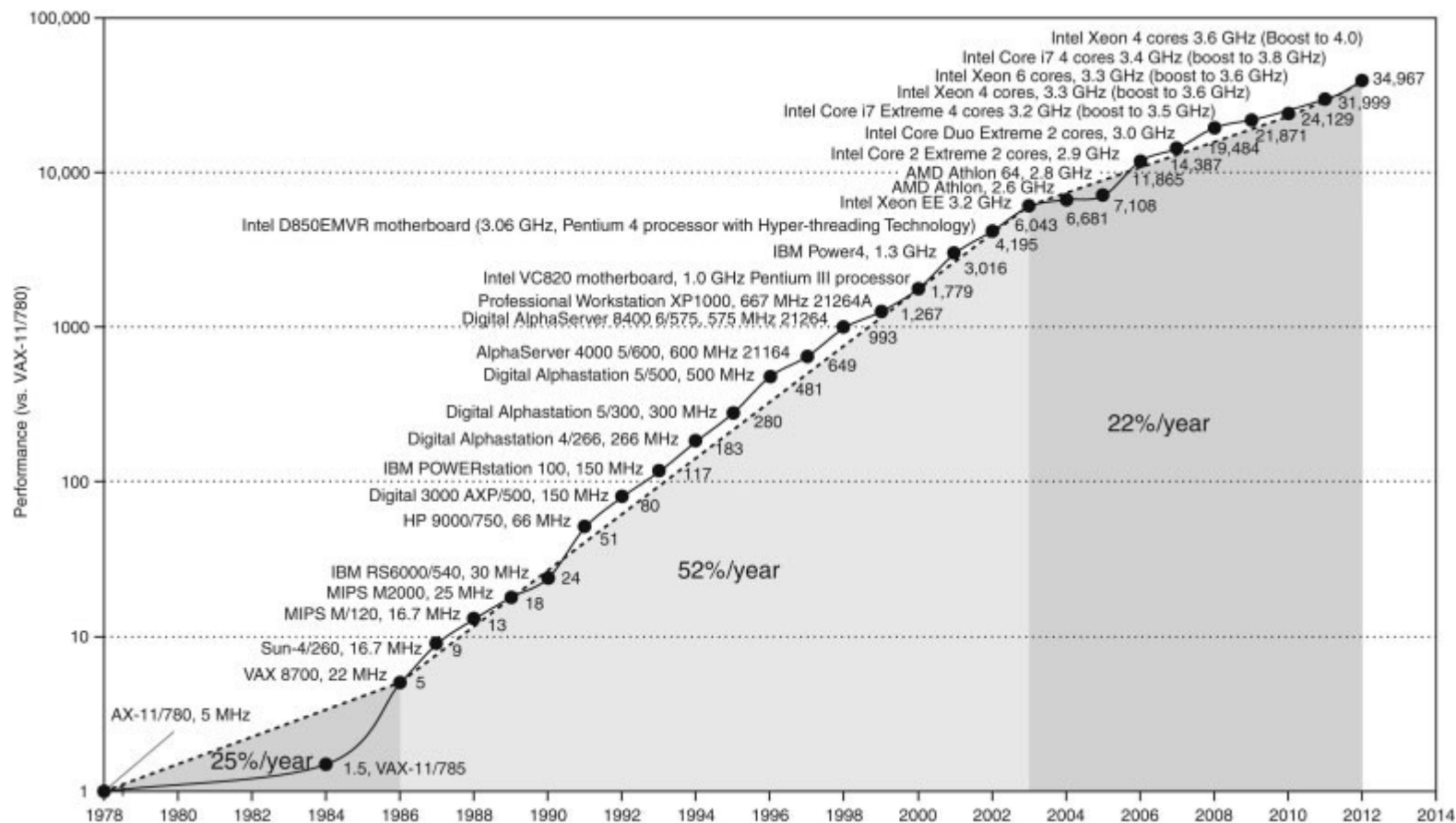
Gates, transistors, semiconductors, silicon

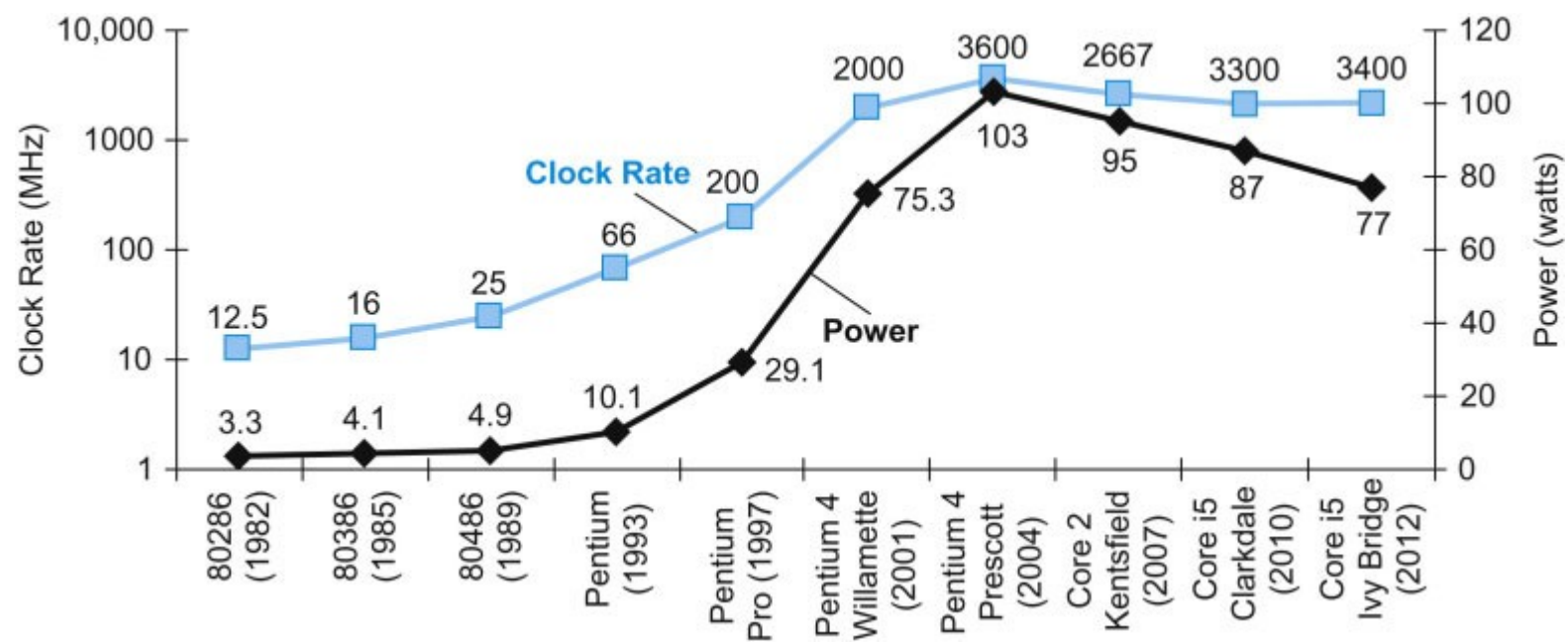


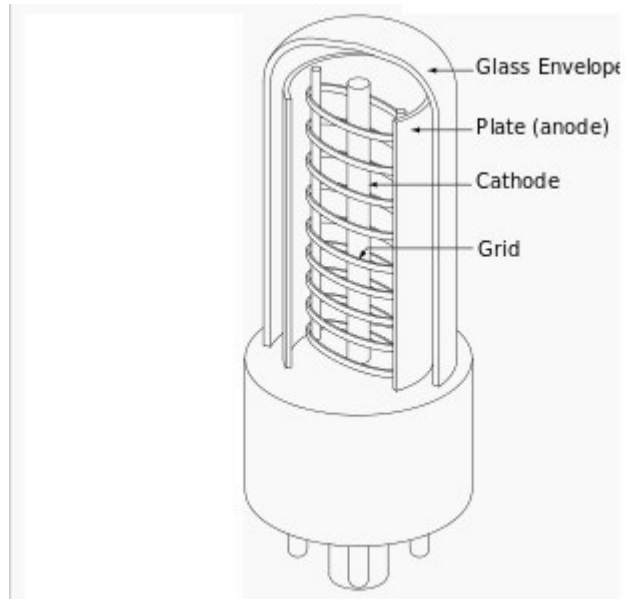
Gates, transistors, semiconductors, silicon

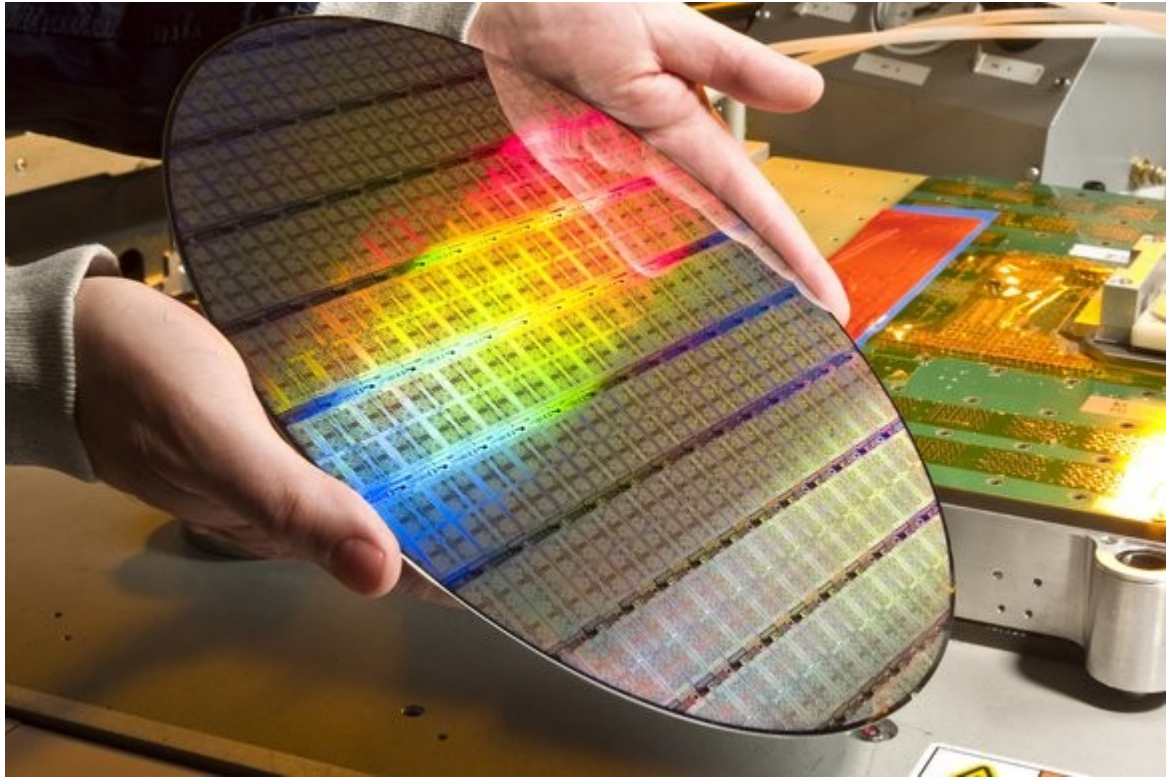


Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons



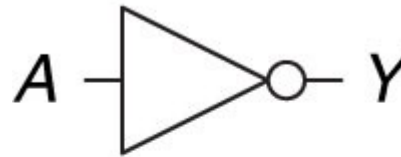






NOT

Gate symbol



Boolean equation

$$Y = \bar{A} \quad \begin{array}{l} Y = A' \\ Y = \neg A \end{array}$$

Truth table

A	Y
0	1
1	0

AND



$$Y = AB$$

$$Y = A \cdot B$$

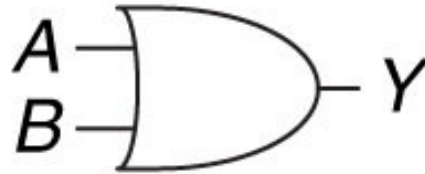
$$Y = A \times B$$

$$Y = A \wedge B$$

<i>A</i>	<i>B</i>	<i>Y</i>
0	0	0
0	1	0
1	0	0
1	1	1

AND gate

OR



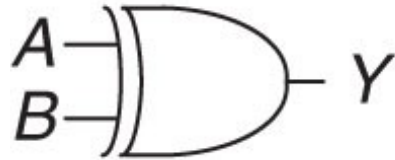
$$Y = A + B$$

$$Y = A \vee B$$

<i>A</i>	<i>B</i>	<i>Y</i>
0	0	0
0	1	1
1	0	1
1	1	1

OR gate

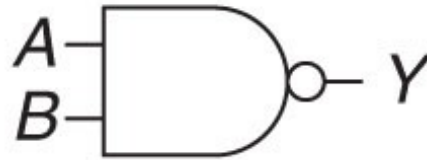
XOR



$$Y = A \oplus B$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

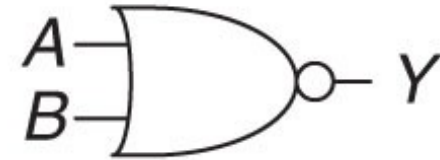
NAND



$$Y = \overline{AB}$$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR

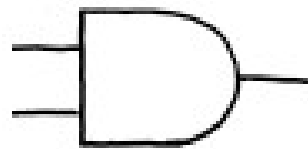


$$Y = \overline{A + B}$$

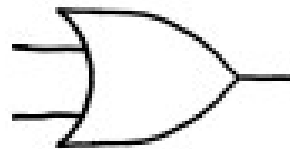
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

More two-input logic gates

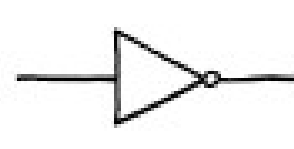
COMMON LOGIC GATE SYMBOLS



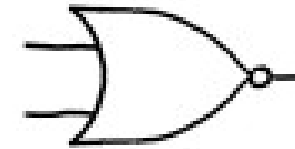
AND GATE



OR GATE



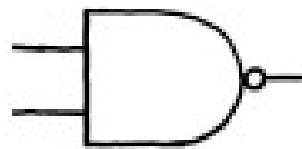
NOT GATE



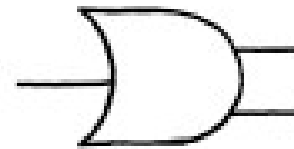
NOR GATE



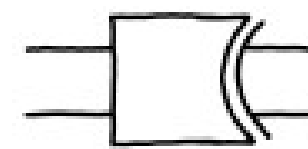
XOR GATE



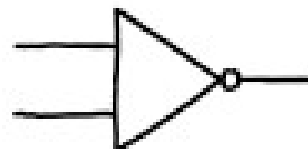
NAND GATE



NORX GATE



GAND ATE



XAND GORT



NORG XORT



ANDORX GANT

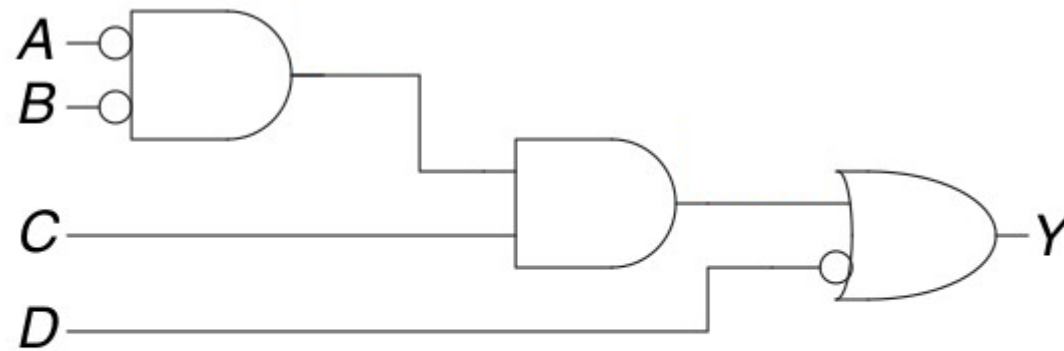


NORXONDOR
GORGONAX

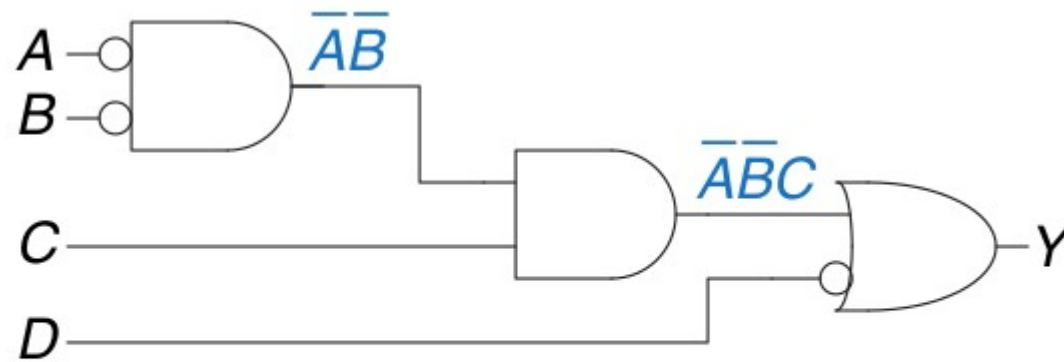
Please note that this slide is an attempt at humor. Many of these gates do not exist.

Some common algebraic notation for logical operations

not A	$\sim A$	$\neg A$	\bar{A}	
A or B	A B	A v B	A + B	
A and B	A & B	A ^ B	A · B	AB
A xor B	A ^ B		A ⊕ B	



Write Y as an algebraic expression in term of A, B C, D. Then write a truth table for this circuit.

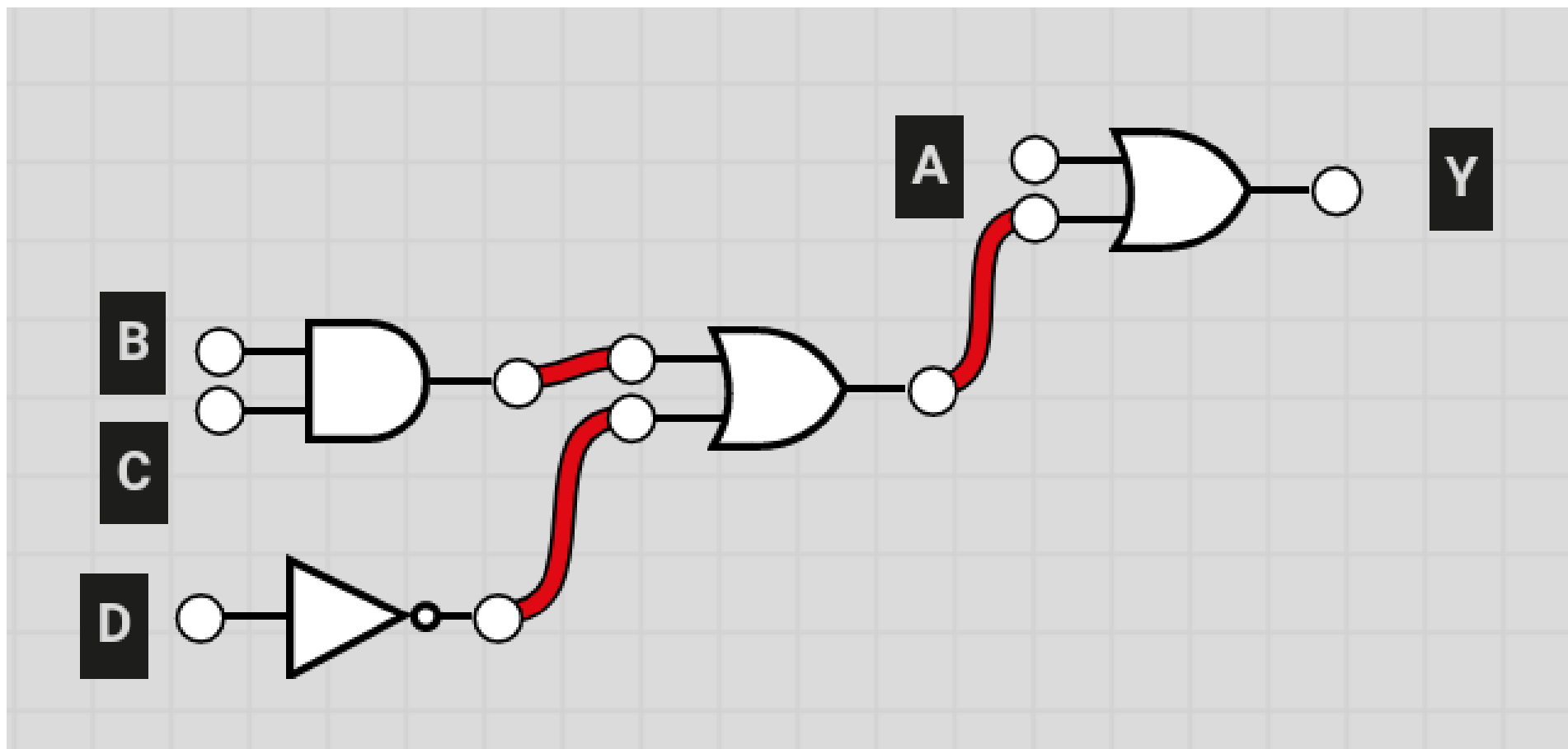


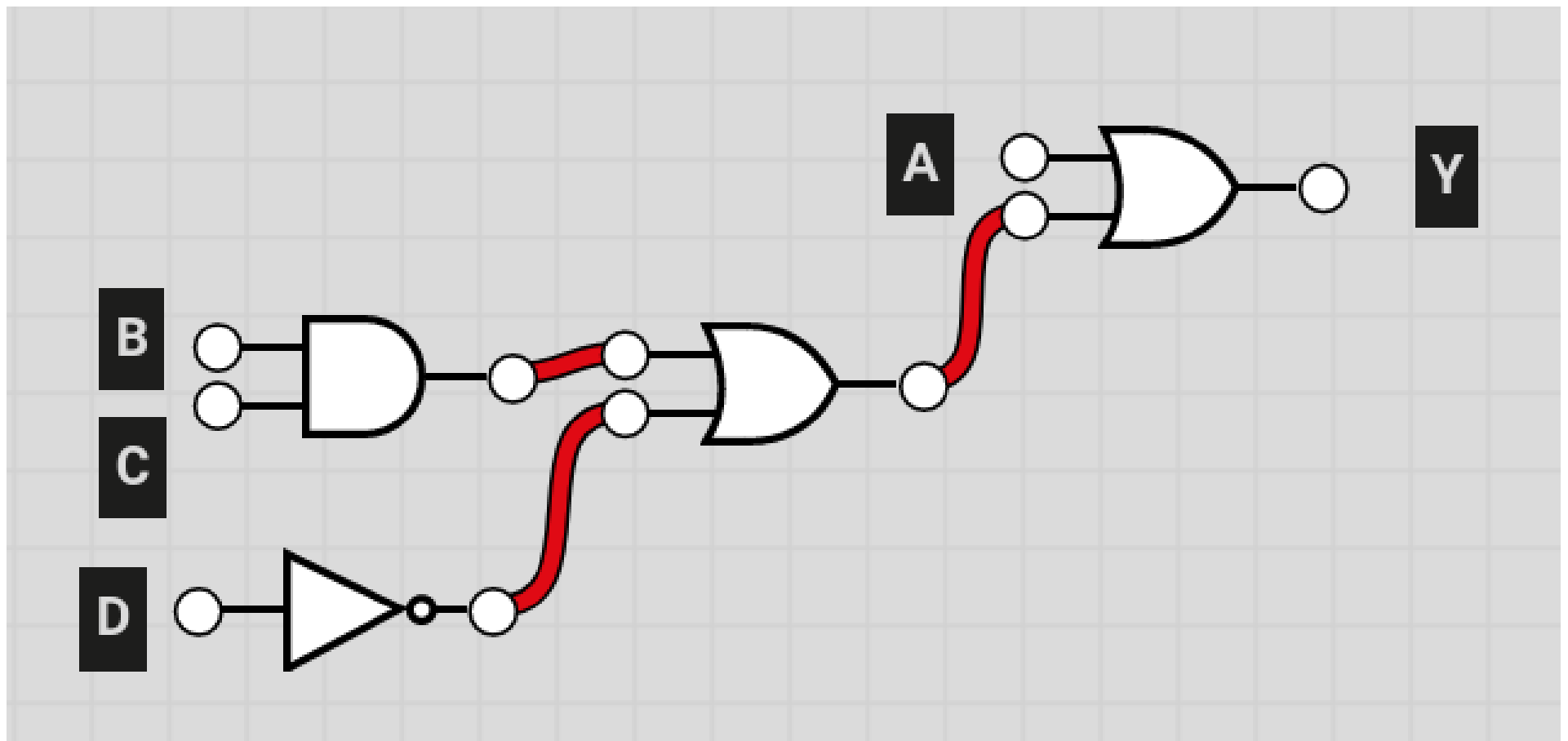
Write Y as an algebraic expression in term of A, B C, D. Then write a truth table for this circuit.

$$Y = (\sim A \ \& \ \sim B \ \& \ C) \mid \sim D$$

$$Y = \overline{A}\overline{B}C + \overline{D}$$

A	B	C	D	$((\neg A \wedge \neg B) \wedge C) \vee \neg D$
F	F	F	F	T
F	F	F	T	F
F	F	T	F	T
F	F	T	T	T
F	T	F	F	T
F	T	F	T	F
F	T	T	F	T
F	T	T	T	F
T	F	F	F	T
T	F	F	T	F
T	F	T	F	T
T	F	T	T	F
T	T	F	F	T
T	T	F	T	F
T	T	T	F	T
T	T	T	T	F





$$Y = A \mid ((B \& C) \mid \sim D)$$

Let's build a circuit in Verilog equivalent to this expression.

$$Y = \overline{A}\overline{B}C + \overline{D}$$

Let's build a circuit in Verilog equivalent to this expression.

$$Y = \overline{A}\overline{B}C + \overline{D}$$

```
module MyCircuit(A, B, C, D, Y);  
  
endmodule
```

A module represents a physical component. It interacts with the outside world via named *ports*, which can carry signals in or out. Here, we list all the ports.

Let's build a circuit in Verilog equivalent to this expression.

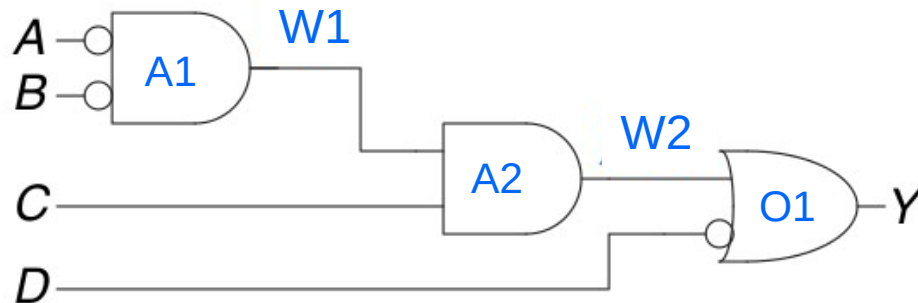
$$Y = \overline{A}\overline{B}C + \overline{D}$$

```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
endmodule
```

We have to specify which ports are for input and which for output. Y is our result, it's where our circuit sends its value. The others are input ports.

Let's build a circuit in Verilog equivalent to this expression.

$$Y = \overline{A}\overline{B}C + \overline{D}$$



mod

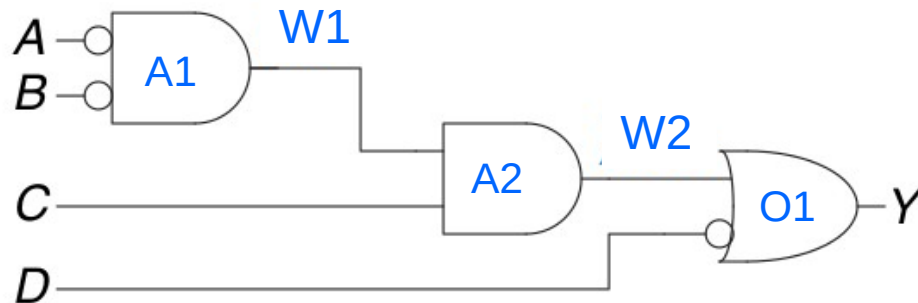
We want to describe the module in terms of the components it uses. It has two AND gates, A1 and A2; one OR gate, O1; and some wires connecting them.

end

The names I've chosen are intended to be descriptive but can be arbitrary.

Let's build a circuit in Verilog equivalent to this expression.

$$Y = \overline{A}\overline{B}C + \overline{D}$$

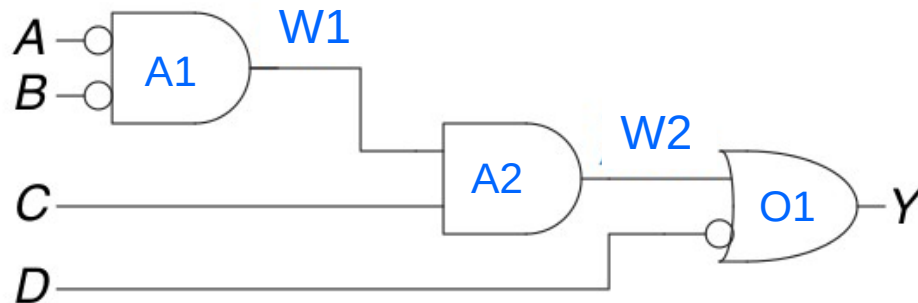


```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
    wire W1, W2;  
  
endmodule
```

Let's define the wires.
These aren't ports, because
they connect only to components
within this module.

Let's build a circuit in Verilog equivalent to this expression.

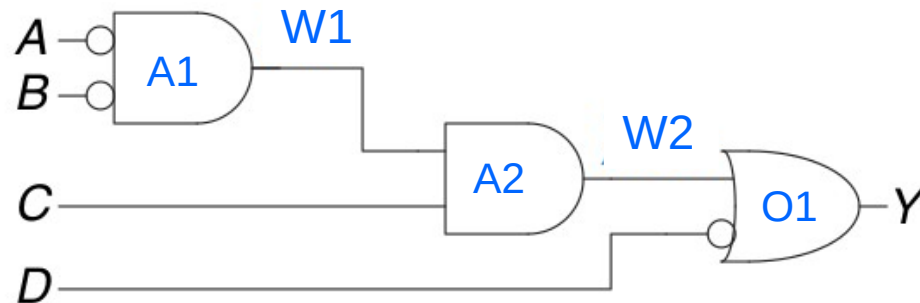
$$Y = \overline{A}\overline{B}C + \overline{D}$$



```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
    wire W1, W2;  
  
    or O1(Y, W2, ~D);  
endmodule
```

Here we define a submodule. The syntax is similar to C++ syntax for creating a class instance: **or** is the type of module, **O1** is the name of this instance.

Let's build a circuit in Verilog equivalent to this expression.



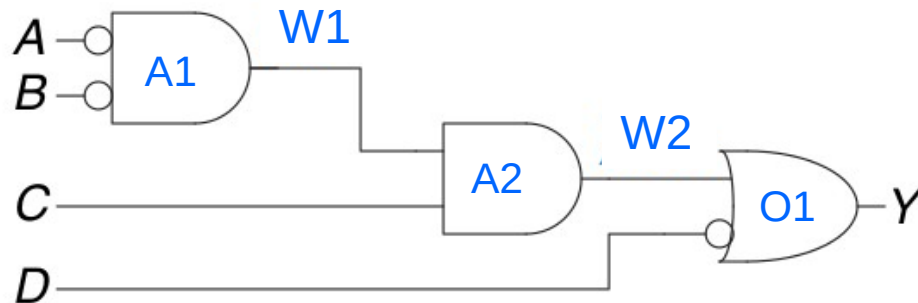
$$Y = \overline{A}\overline{B}C + \overline{D}$$

```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
    wire W1, W2;  
  
    or O1(Y, W2, ~D);  
endmodule
```

We're creating an OR gate named O1 within our module. The parameters connect it to other defined components: the OR gate's output is connected to Y, its inputs come from NOT D and W2.

Let's build a circuit in Verilog equivalent to this expression.

$$Y = \overline{A}\overline{B}C + \overline{D}$$

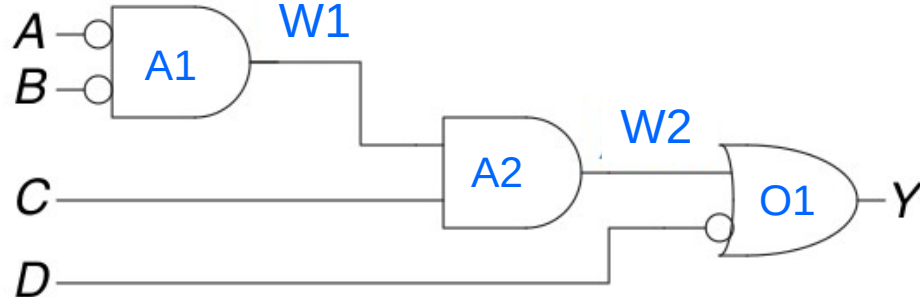


```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
    wire W1, W2;  
  
    or O1(Y, W2, ~D);  
    and A2(W2, W1, C);  
endmodule
```

Now we create an AND gate with similar syntax. The AND gate's output W2 becomes an input to the OR gate.

Let's build a circuit in Verilog equivalent to this expression.

$$Y = \overline{A}\overline{B}C + \overline{D}$$



```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
    wire W1, W2;  
  
    or O1(Y, W2, ~D);  
    and A2(W2, W1, C);  
    and A1(W1, ~A, ~B);  
endmodule
```

Finally we provide the last AND gate.

Let's build a circuit in Verilog equivalent to this expression.

$$Y = \overline{A}\overline{B}C + \overline{D}$$

```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
    assign Y = (~A & ~B & C) | ~D;  
endmodule
```

There's an easier way!
We can also write the circuit
using conventional expression
syntax. This is *continuous
assignment syntax*.

But it's important that you know
that both syntaxes are equivalent:
we are describing the connections
between components in our module.

The equation can be directly
translated using C++-like
syntax from the boolean expression
above.

We are connecting inputs ports
into gates which in turn connect
to the output port.

Synthesis

Having described the circuit in Verilog, we want to produce a physical device. This is *synthesis*.

```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
    assign Y = (~A & ~B & C) | ~D;  
endmodule
```

Synthesis

```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
    assign Y = (~A & ~B & C) | ~D;  
endmodule
```

yosys and other synthesis frameworks convert Verilog code into an RTL (register-transfer level) form, which one could theoretically take to a factory and use as the basis for manufacture. Alternatively, we can visualize the physical circuit design.



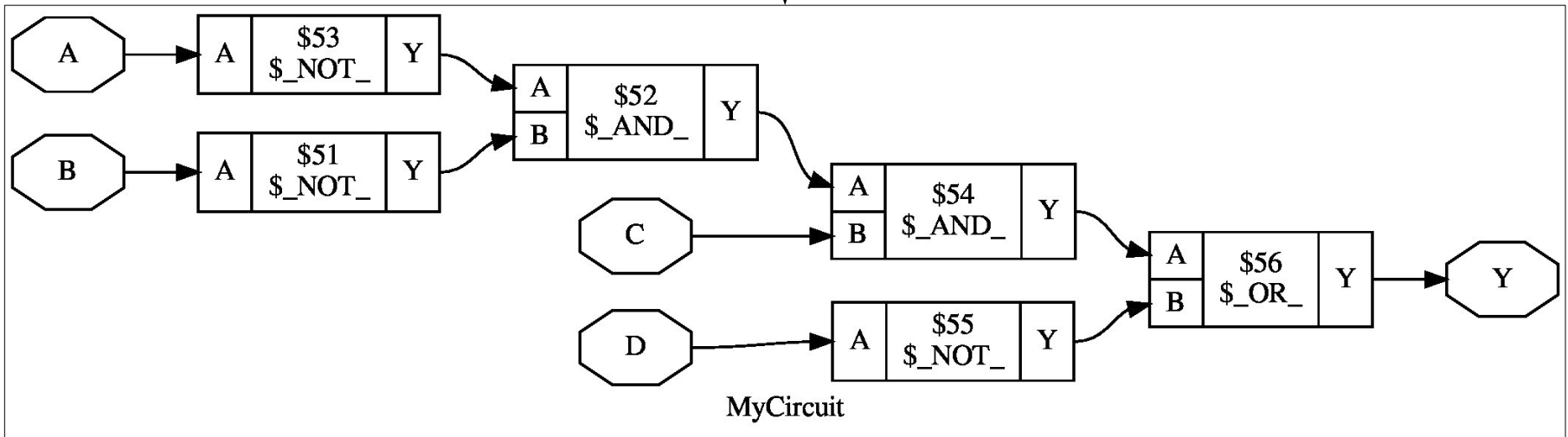
yosys

Synthesis

```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
    assign Y = (~A & ~B & C) | ~D;  
endmodule
```

yosys outputs a diagram and RTL showing how the Verilog code can be implemented as lower-level components, in this case gates.

yosys



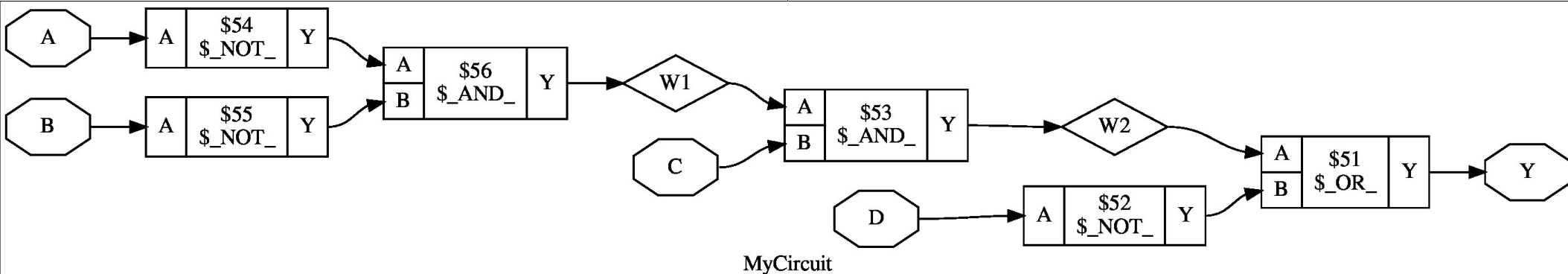
Synthesis

```
module MyCircuit(A, B, C, D, Y);  
    input A, B, C, D;  
    output Y;  
  
    wire W1, W2;  
  
    or O1(Y, W2, ~D);  
    and A2(W2, W1, C);  
    and A1(W1, ~A, ~B);  
endmodule
```

Equivalent Verilog code will produce an equivalent diagram.

Here, we use more explicit syntax to describe the same circuit.

yosys

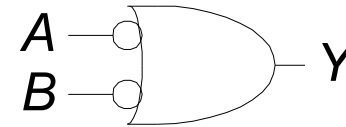
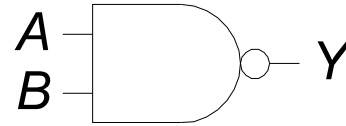


$$(AB)+C'$$

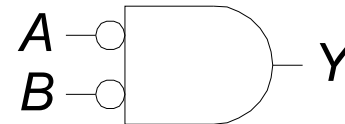
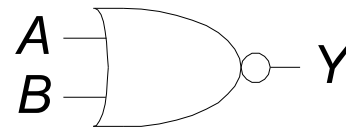
A	B	C	$((A \wedge B) \vee \neg C)$
F	F	F	T
F	F	T	F
F	T	F	T
F	T	T	F
T	F	F	T
T	F	T	F
T	T	F	T
T	T	T	T

DeMorgan's Theorem

- $Y = \overline{AB} = \overline{A} + \overline{B}$



- $Y = \overline{A + B} = \overline{A} \cdot \overline{B}$



Expressing numbers in binary

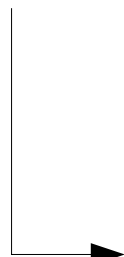
We know how to express numbers in binary.

$$101010_2 = ???_{10}$$

Expressing numbers in binary

We know how to express numbers in binary.

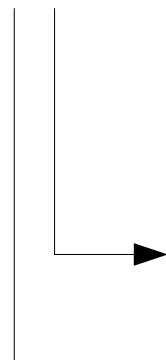
$$101010_2 = ???_{10}$$


$$\rightarrow 0 * 1 = 0$$

Expressing numbers in binary

We know how to express numbers in binary.

$$101010_2 = ???_{10}$$


$$\begin{array}{l} \rightarrow 0 * 1 = 0 \\ \rightarrow 1 * 2 = 2 \end{array}$$

Expressing numbers in binary

We know how to express numbers in binary.

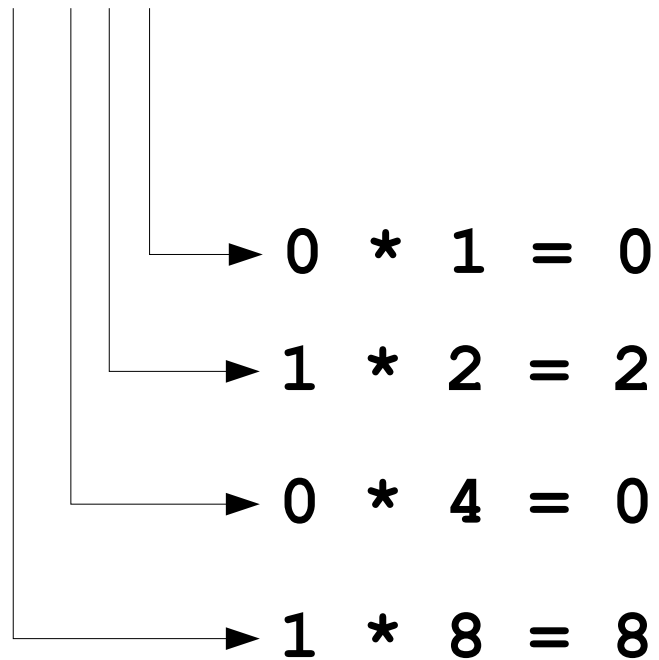
$$101010_2 = ???_{10}$$

$$\begin{array}{l} \rightarrow 0 * 1 = 0 \\ \rightarrow 1 * 2 = 2 \\ \rightarrow 0 * 4 = 0 \end{array}$$

Expressing numbers in binary

We know how to express numbers in binary.

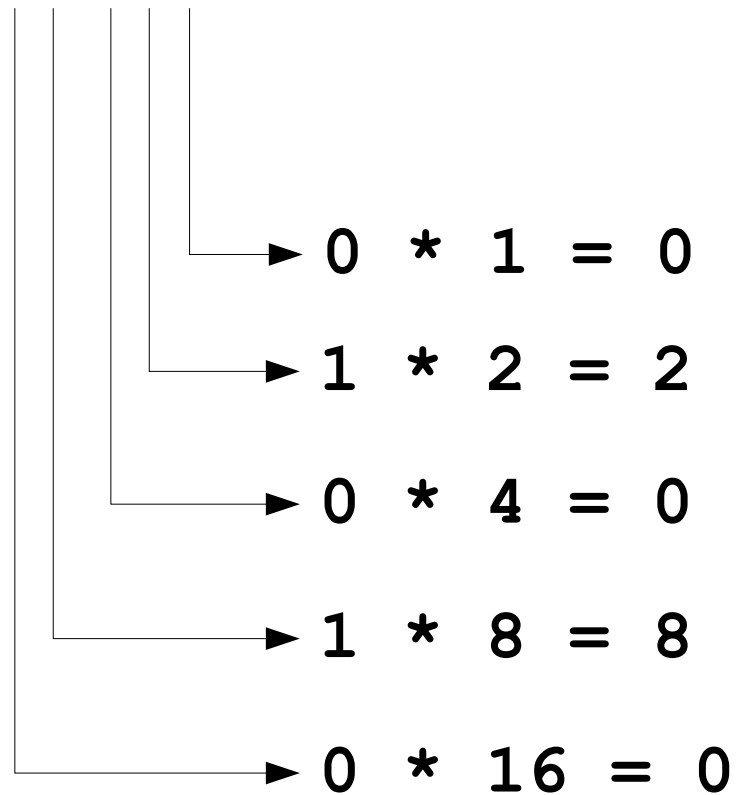
$$101010_2 = ???_{10}$$



Expressing numbers in binary

We know how to express numbers in binary.

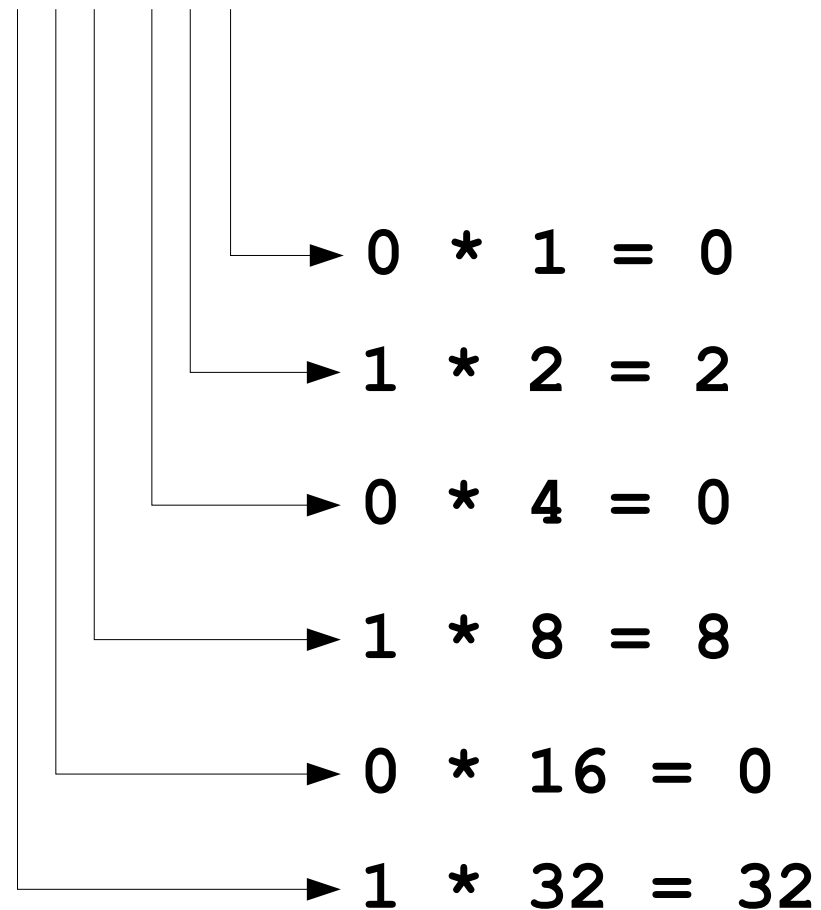
$$101010_2 = ???_{10}$$



Expressing numbers in binary

We know how to express numbers in binary.

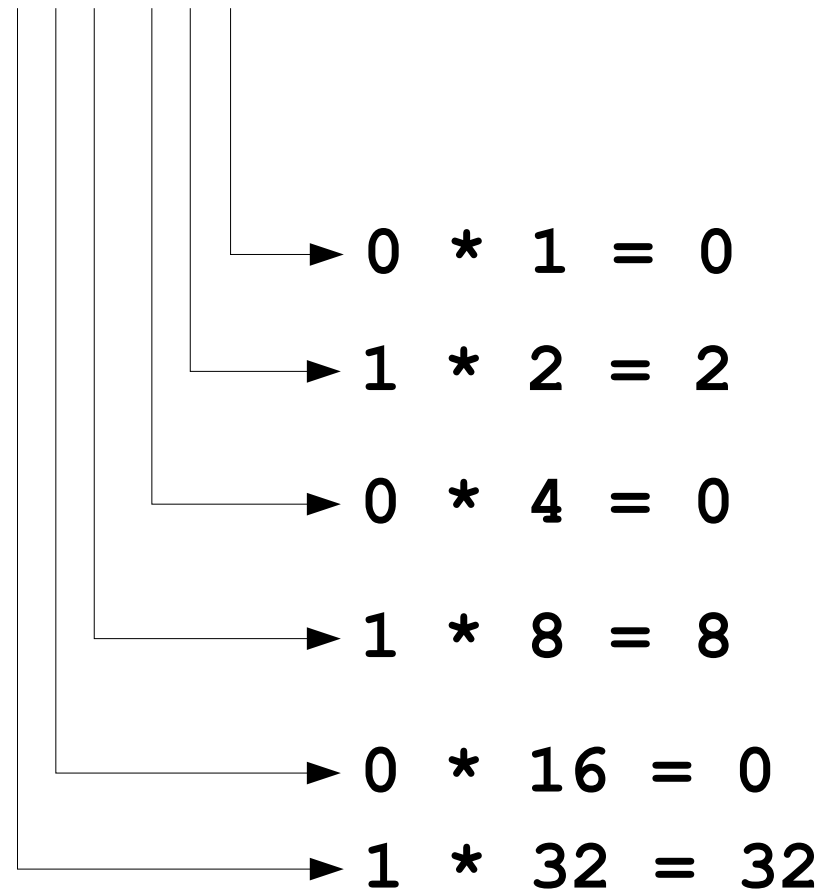
$$101010_2 = ???_{10}$$



Expressing numbers in binary

We know how to express numbers in binary.

$$101010_2 = 42_{10}$$



$$0 + 2 + 0 + 8 + 0 + 32 = 42$$

Bitwise operations

1 & 0 == 0

1 | 0 == 1

10 & 6 == ???

10 | 6 == ???

Bitwise operations

Decimal	Binary
10	1010
6	0110

10 & 6 == ???

10 | 6 == ???

Bitwise operations

10 & 6 == ???

Can we calculate the
AND of decimal
10 and 6?

$$\begin{array}{r} 1010 \\ \& 0110 \end{array}$$

Bitwise operations

10 & 6 == ???

	1	0	1	0
&	0	1	1	0
<hr/>				

Bitwise operations

10 & 6 == ???

	1	0	1	0
&	0	1	1	0
<hr/>				

0 & 0 == 0

Bitwise operations

10 & 6 == ???

	1	0	1	0
&	0	1	1	0
<hr/>				
			0	

0 & 0 == 0

Bitwise operations

`10 & 6 == ???`

	1	0	1	0
&	0	1	1	0
<hr/>				
		1	0	

`1 & 1 == 1`

Bitwise operations

10 & 6 == ???

	1	0	1	0
&	0	1	1	0
<hr/>				
	0	1	0	

0 & 1 == 0

Bitwise operations

10 & 6 == ???

	1	0	1	0
&	0	1	1	0
<hr/>				
	0	0	1	0

1 & 0 == 0

Bitwise operations

$$10 \ \& \ 6 == 2$$

1010	==10
& 0110	==6
0010	==2

Bitwise operations

10 | 6 == ???

Can we calculate the
OR of decimal
10 and 6?

	1	0	1	0
	0	1	1	0
<hr/>				

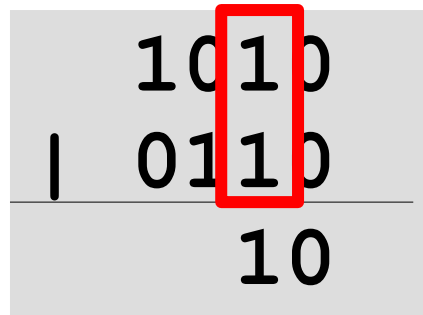
Bitwise operations

10 | 6 == ???

	1	0	1	0
	0	1	1	0
<hr/>				
			0	

Bitwise operations

10 | 6 == ???



A diagram illustrating the bitwise OR operation between the decimal numbers 10 and 6. The numbers are represented in binary: 10 is 1010 and 6 is 0110. The OR operation is shown with a vertical bar between the two numbers. A red rectangle highlights the third bit from the right, which is 1 in both numbers, resulting in 1. The final result, 10 (1010), is shown below a horizontal line.

```
  1010
| 0110
-----
  1010
```

Bitwise operations

10 | 6 == ???

	1	0	1	0
	0	1	1	0
<hr/>				
	1	1	0	

Bitwise operations

$$10 \mid 6 == 14$$

1010	==10
0110	==6
1110	==14

Bitwise operations

$\sim 10 == ???$

Can we calculate the
NOT of decimal
10?

~ 1010

Bitwise operations

$\sim 10 == ???$

Can we calculate the
NOT of decimal
10?

\sim	1	0	1	0
<hr/>				
				1

Bitwise operations

$\sim 10 == ???$

Can we calculate the
NOT of decimal
10?

\sim	1	0	1	0
<hr/>				
			0	1

Bitwise operations

$\sim 10 == ???$

Can we calculate the NOT of decimal 10 (as a 4-bit unsigned integer)?

\sim	1010
<hr/>	
	101

Bitwise operations

$$\sim 10 == 5$$

\sim	1010	==10
<hr/>		
	0101	==5

Bitwise operations

Can you use *only* bitwise operators to determine if a number is odd or even?

For example, `is_odd(3) == true`, but `is_odd(50) == false`.

```
bool is_odd(int x) {  
    return ??????  
}
```

Bitwise operations

Can you use *only* bitwise operators to determine if a number is odd or even?

```
bool is_odd(int x) {  
    return x & 1 == 1;  
}
```

We test if the least-significant bit of the number is one. If it is, it must be odd.

Bitwise operations

Can you use *only* bitwise operators to round a non-negative integer down to the nearest multiple of four?

For example, `align_to_4(10) == 8`, and `align_to_4(20) == 20`.

```
int align_to_4(int x) {  
    return ??????;  
}
```

Bitwise operations

Can you use *only* bitwise operators to round a non-negative integer down to the nearest multiple of four?

For example, `align_to_4(10) == 8`, and `align_to_4(20) == 20`.

```
int align_to_4(int x) {  
    return ??????;  
}
```

Hint:

```
  00001010  
& 11111100  
-----
```

```
  00010100  
& 11111100  
-----
```

Bitwise operations

Can you use *only* bitwise operators to round a non-negative integer down to the nearest multiple of four?

For example, `align_to_4(10) == 8`, and `align_to_4(20) == 20`.

```
int align_to_4(int x) {  
    return x & ~3;  
}
```

We are simply turning off the two least-significant bits of the input value.

Recall that 3 in binary is 11. Therefore, `~3` will have all bits on except the two least significant, e.g.: 11111100 (depending on how many bits are in an int).

ANDing that value with our input will keep all bits except the two least significant.

Any number with zero in the two least-significant bits is guaranteed to be a multiple of four.