Software and Hardware

Outline

1. Representing Information
2. Boolean Functions
3. Logic Circuits
4. Von Neumann Architecture

Representing Information

Decimal (base 10) system

$$135 = \quad \cdot 10^2 + \quad \cdot 10^1 + \quad \cdot 10^0$$

Decimal (base 10) system

$$135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

Representing Information

Decimal (base 10) system

$$135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

Binary (base 2) system

Decimal (base 10) system

$$135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

Binary (base 2) system

$$135 = \quad \cdot 2^7 + \quad \cdot 2^6 + \quad \cdot 2^5 + \quad \cdot 2^4 + \quad \cdot 2^3 + \quad \cdot 2^2 + \quad \cdot 2^1 + \quad \cdot 2^0$$

Representing Information

Decimal (base 10) system

$$135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

Binary (base 2) system

$$135 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Representing Information

Decimal (base 10) system

$$135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

Binary (base 2) system

$$135 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Octal (base 8) system

Decimal (base 10) system

$$135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

Binary (base 2) system

$$135 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Octal (base 8) system

$$135 = \quad \cdot 8^2 + \quad \cdot 8^1 + \quad \cdot 8^0$$

Representing Information

Decimal (base 10) system

$$135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

Binary (base 2) system

$$135 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Octal (base 8) system

$$135 = 2 \cdot 8^2 + 0 \cdot 8^1 + 7 \cdot 8^0$$

Decimal (base 10) system

$$135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

Binary (base 2) system

$$135 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Octal (base 8) system

$$135 = 2 \cdot 8^2 + 0 \cdot 8^1 + 7 \cdot 8^0$$

Hexadecimal (base 16) system

Decimal (base 10) system

$$135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

Binary (base 2) system

$$135 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Octal (base 8) system

$$135 = 2 \cdot 8^2 + 0 \cdot 8^1 + 7 \cdot 8^0$$

Hexadecimal (base 16) system

$$135 = \quad \cdot 16^1 + \quad \cdot 16^0$$

Representing Information

Decimal (base 10) system

$$135 = 1 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$$

Binary (base 2) system

$$135 = 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Octal (base 8) system

$$135 = 2 \cdot 8^2 + 0 \cdot 8^1 + 7 \cdot 8^0$$

Hexadecimal (base 16) system

$$135 = 8 \cdot 16^1 + 7 \cdot 16^0$$

Representing Information

Representing Information

Dec	Bin	Oct	Hex
0	00000	00	00
1	00001	01	01
2	00010	02	02
3	00011	03	03
4	00100	04	04
5	00101	05	05
6	00110	06	06
7	00111	07	07
8	01000	10	08
9	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F

Dec	Bin	Oct	Hex
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17
24	11000	30	18
25	11001	31	19
26	11010	32	1A
27	11011	33	1B
28	11100	34	1C
29	11101	35	1D
30	11110	36	1E
31	11111	37	1F

Representing Information

Arithmetic in any base is analogous to arithmetic in base 10

Arithmetic in any base is analogous to arithmetic in base 10

Example (addition in binary)

$$
\begin{array}{rrrr}
 & 1 & 1 & 1 \\
+ & 1 & 1 & 0 \\
\hline
\end{array}
$$

Arithmetic in any base is analogous to arithmetic in base 10

Example (addition in binary)

$$
\begin{array}{rccc}
 & 1 & 1 & 1 \\
+ & 1 & 1 & 0 \\
\hline
 & & & 1
\end{array}
$$

Arithmetic in any base is analogous to arithmetic in base 10

Example (addition in binary)

$$\begin{array}{rrrr} & 1 & & \\ & 1 & 1 & 1 \\ + & 1 & 1 & 0 \\ \hline & & 0 & 1 \end{array}$$

Arithmetic in any base is analogous to arithmetic in base 10

Example (addition in binary)

$$\begin{array}{cccccc} & & 1 & 1 & & \\ & & & 1 & 1 & 1 \\ + & & & 1 & 1 & 0 \\ \hline & & & 1 & 0 & 1 \end{array}$$

Arithmetic in any base is analogous to arithmetic in base 10

Example (addition in binary)

$$\begin{array}{rrrrr} & 1 & 1 & & \\ & & 1 & 1 & 1 \\ + & & 1 & 1 & 0 \\ \hline & 1 & 1 & 0 & 1 \end{array}$$

Representing Information

Two's complement method to compute $-x$:

Representing Information

Two's complement method to compute $-x$:

1. Represent x in binary

Two's complement method to compute $-x$:

1. Represent x in binary
2. Flip the bits of the result

Two's complement method to compute $-x$:

1. Represent x in binary
2. Flip the bits of the result
3. Add 1 to the result

Two's complement method to compute $-x$:

1. Represent x in binary
2. Flip the bits of the result
3. Add 1 to the result

Example (-3 on an 8-bit computer):

Representing Information

Two's complement method to compute $-x$:

1. Represent x in binary
2. Flip the bits of the result
3. Add 1 to the result

Example (-3 on an 8-bit computer):

- Represent 3 in binary as 00000011

Representing Information

Two's complement method to compute $-x$:

1. Represent x in binary
2. Flip the bits of the result
3. Add 1 to the result

Example (-3 on an 8-bit computer):

- Represent 3 in binary as 00000011
- Flip the bits of the result to obtain 11111100

Two's complement method to compute $-x$:

1. Represent x in binary
2. Flip the bits of the result
3. Add 1 to the result

Example (-3 on an 8-bit computer):

- Represent 3 in binary as 00000011
- Flip the bits of the result to obtain 11111100
- Add 1 to the result to obtain 11111101

Two's complement method to compute $-x$:

1. Represent x in binary
2. Flip the bits of the result
3. Add 1 to the result

Example (-3 on an 8-bit computer):

- Represent 3 in binary as 00000011
- Flip the bits of the result to obtain 11111100
- Add 1 to the result to obtain 11111101

  Note: just like how $3 + (-3) = 0$, we have $00000011 + 11111101 = 100000000$

Representing Information

Representing Information

Assuming we only have 10 decimal digits to represent a real number, we might use:

Assuming we only have 10 decimal digits to represent a real number, we might use:

- The first digit for the sign (0 for $+$ and 1 for $-$) of the fractional part

Representing Information

Assuming we only have 10 decimal digits to represent a real number, we might use:

- The first digit for the sign (0 for $+$ and 1 for $-$) of the fractional part
- The next six digits for the fractional part

Representing Information

Assuming we only have 10 decimal digits to represent a real number, we might use:

- The first digit for the sign (0 for $+$ and 1 for $-$) of the fractional part
- The next six digits for the fractional part
- The eigth digit for the sign (0 for $+$ and 1 for $-$) of the exponent

Representing Information

Assuming we only have 10 decimal digits to represent a real number, we might use:

- The first digit for the sign (0 for $+$ and 1 for $-$) of the fractional part
- The next six digits for the fractional part
- The eigth digit for the sign (0 for $+$ and 1 for $-$) of the exponent
- The last two digits for the exponent

Assuming we only have 10 decimal digits to represent a real number, we might use:

- The first digit for the sign (0 for $+$ and 1 for $-$) of the fractional part
- The next six digits for the fractional part
- The eigth digit for the sign (0 for $+$ and 1 for $-$) of the exponent
- The last two digits for the exponent

Example: the 10-digit number 0314159001 represents $0.314159 \times 10^1 = 3.14159$

Representing Information

Representing Information

ASCII (American Standard Code for Information Interchange) defines 8-bit encodings for letters and numbers in English, and a select set of special characters

Representing Information

ASCII (American Standard Code for Information Interchange) defines 8-bit encodings for letters and numbers in English, and a select set of special characters

Example: the numbers 65–90 encode upper-case letters A–Z, numbers 97–122 encode lower-case letters a–z, and numbers 48–57 encode digits 0–9

ASCII (American Standard Code for Information Interchange) defines 8-bit encodings for letters and numbers in English, and a select set of special characters

Example: the numbers 65–90 encode upper-case letters A–Z, numbers 97–122 encode lower-case letters a–z, and numbers 48–57 encode digits 0–9

The 16-bit Unicode system can represent every character in every known language, with room for more

Representing Information

Representing Information

A string is represented as a sequence of numbers, with a “length field” at the very beginning specifying the length of the string

Representing Information

A string is represented as a sequence of numbers, with a “length field” at the very beginning specifying the length of the string

Example: the string “Python” is represented in decimal as the sequence

006 080 121 116 104 111 110

or in binary as the sequence

00000110 01010000 01111001 01110100 01101000 01101111 01101110

Representing Information

Representing Information

We can represent any structured information as a sequence of numbers

Representing Information

We can represent any structured information as a sequence of numbers

Example (encoding pictures, sounds, and movies):

Representing Information

We can represent any structured information as a sequence of numbers

Example (encoding pictures, sounds, and movies):

- A picture as a sequence of triples, each containing the amount of red, green, and blue at a pixel

Representing Information

We can represent any structured information as a sequence of numbers

Example (encoding pictures, sounds, and movies):

- A picture as a sequence of triples, each containing the amount of red, green, and blue at a pixel
- A sound as a temporal sequence of “sound pressure levels”

Representing Information

We can represent any structured information as a sequence of numbers

Example (encoding pictures, sounds, and movies):

- A picture as a sequence of triples, each containing the amount of red, green, and blue at a pixel
- A sound as a temporal sequence of “sound pressure levels”
- A movie as a temporal sequence of pictures (usually 30 per second), along with a matching sound sequence

Boolean Functions

Boolean Functions

A boolean variable is a variable that has the value 1 (True) or 0 (False)

Boolean Functions

A boolean variable is a variable that has the value 1 (True) or 0 (False)

A boolean function is an algebraic expression consisting of boolean variables and logical operations

Boolean Functions

A boolean variable is a variable that has the value 1 (True) or 0 (False)

A boolean function is an algebraic expression consisting of boolean variables and logical operations

The three basic boolean functions: $\texttt{not}(x) = \bar{x}$, $\texttt{or}(x, y) = x + y$, and $\texttt{and}(x, y) = x \cdot y$

Boolean Functions

A boolean variable is a variable that has the value 1 (True) or 0 (False)

A boolean function is an algebraic expression consisting of boolean variables and logical operations

The three basic boolean functions: $\texttt{not}(x) = \bar{x}$, $\texttt{or}(x, y) = x + y$, and $\texttt{and}(x, y) = x \cdot y$

The truth table for a boolean function is a listing of all possible combinations of values of the input variables, together with the result produced by the function

Boolean Functions

A boolean variable is a variable that has the value 1 (True) or 0 (False)

A boolean function is an algebraic expression consisting of boolean variables and logical operations

The three basic boolean functions: $\texttt{not}(x) = \bar{x}$, $\texttt{or}(x, y) = x + y$, and $\texttt{and}(x, y) = x \cdot y$

The truth table for a boolean function is a listing of all possible combinations of values of the input variables, together with the result produced by the function

Truth tables for $\texttt{not}$, $\texttt{or}$, and $\texttt{and}$ functions

x	$\bar{x}$
0	1
1	0

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Functions

Boolean Functions

Any boolean function can be expressed in terms of the basic boolean functions

Any boolean function can be expressed in terms of the basic boolean functions

Minterm expansion algorithm:

Boolean Functions

Any boolean function can be expressed in terms of the basic boolean functions

Minterm expansion algorithm:

1. Write down the truth table for the boolean function

Any boolean function can be expressed in terms of the basic boolean functions

Minterm expansion algorithm:

1. Write down the truth table for the boolean function
2. Delete all rows from the truth table where the value of the function is 0

Boolean Functions

Any boolean function can be expressed in terms of the basic boolean functions

Minterm expansion algorithm:

1. Write down the truth table for the boolean function
2. Delete all rows from the truth table where the value of the function is 0
3. For each remaining row, create a "minterm" as follows:

Boolean Functions

Any boolean function can be expressed in terms of the basic boolean functions

Minterm expansion algorithm:

1. Write down the truth table for the boolean function
2. Delete all rows from the truth table where the value of the function is 0
3. For each remaining row, create a "minterm" as follows:
    a. For each variable x: if its value in that row is 1, write x; otherwise, write $\bar{x}$

Boolean Functions

Any boolean function can be expressed in terms of the basic boolean functions

Minterm expansion algorithm:

1. Write down the truth table for the boolean function
2. Delete all rows from the truth table where the value of the function is 0
3. For each remaining row, create a "minterm" as follows:
   a. For each variable x: if its value in that row is 1, write x; otherwise, write $\bar{x}$
   b. Combine all of the variables using $\cdot$

Boolean Functions

Any boolean function can be expressed in terms of the basic boolean functions

Minterm expansion algorithm:

1. Write down the truth table for the boolean function
2. Delete all rows from the truth table where the value of the function is 0
3. For each remaining row, create a "minterm" as follows:
   a. For each variable x: if its value in that row is 1, write x; otherwise, write $\bar{x}$
   b. Combine all of the variables using $\cdot$
4. Combine all of the minterms using $+$

Boolean Functions

Example (implication function): consider the proposition “if you score over 93% in this course, then you will get an A”

Boolean Functions

Example (implication function): consider the proposition "if you score over 93% in this course, then you will get an A"

The proposition is described by the `implication` function ($x \implies y$)

x	y	$x \implies y$	minterm
0	0	1	
0	1	1	
1	0	0	
1	1	1	

Example (implication function): consider the proposition "if you score over 93% in this course, then you will get an A"

The proposition is described by the `implication` function ($x \implies y$)

x	y	$x \implies y$	minterm
0	0	1	
0	1	1	
1	0	0	
1	1	1	

Example (implication function): consider the proposition "if you score over 93% in this course, then you will get an A"

The proposition is described by the `implication` function ($x \implies y$)

x	y	$x \implies y$	minterm
0	0	1	$\bar{x} \cdot \bar{y}$
0	1	1	
1	0	0	
1	1	1	

Example (implication function): consider the proposition "if you score over 93% in this course, then you will get an A"

The proposition is described by the `implication` function ($x \implies y$)

x	y	$x \implies y$	minterm
0	0	1	$\bar{x} \cdot \bar{y}$
0	1	1	$\bar{x} \cdot y$
1	0	0	
1	1	1	

Example (implication function): consider the proposition "if you score over 93% in this course, then you will get an A"

The proposition is described by the `implication` function ($x \implies y$)

x	y	$x \implies y$	minterm
0	0	1	$\bar{x} \cdot \bar{y}$
0	1	1	$\bar{x} \cdot y$
1	0	0	
1	1	1	$x \cdot y$

Example (implication function): consider the proposition "if you score over 93% in this course, then you will get an A"

The proposition is described by the `implication` function ($x \implies y$)

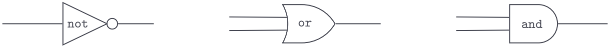
x	y	$x \implies y$	minterm
0	0	1	$\bar{x} \cdot \bar{y}$
0	1	1	$\bar{x} \cdot y$
1	0	0	
1	1	1	$x \cdot y$

Ergo, $\texttt{implication}(x, y) = \bar{x} \cdot \bar{y} + \bar{x} \cdot y + x \cdot y$

Logic Circuits

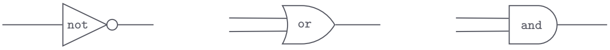
Logic Circuits

The logic gates that implement the `not`, `or`, and `and` functions

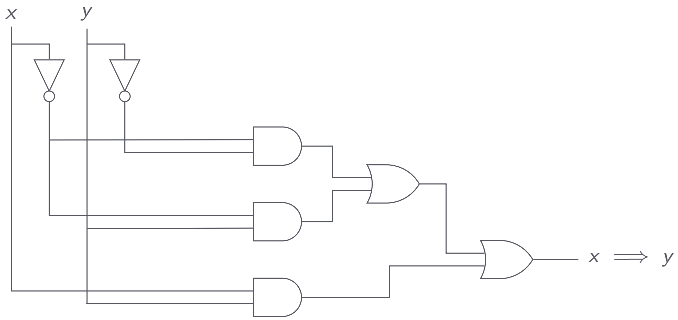


Logic Circuits

The logic gates that implement the `not`, `or`, and `and` functions

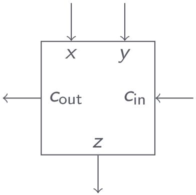


Logic circuit for the `implication` function $\bar{x} \cdot \bar{y} + \bar{x} \cdot y + x \cdot y$



Logic Circuits

A full adder (FA) circuit can add two 1-bit numbers (with carry) to produce a 2-bit result



x	y	c_{in}	z	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

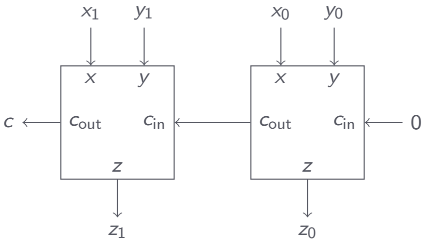
Logic Circuits

Logic Circuits

An n-bit ripple-carry adder is n FA circuits chained together to add two n-bit numbers

An n-bit ripple-carry adder is n FA circuits chained together to add two n-bit numbers

A 2-bit ripple-carry adder



Logic Circuits

Truth table for a nor gate (or followed by not)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

Logic Circuits

Truth table for a nor gate (or followed by not)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

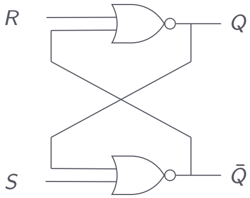
A 1-bit memory circuit, called a latch, built using two nor gates

Logic Circuits

Truth table for a nor gate (or followed by not)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

A 1-bit memory circuit, called a latch, built using two nor gates

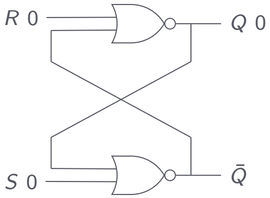


Logic Circuits

Truth table for a nor gate (or followed by not)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

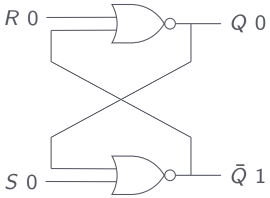
A 1-bit memory circuit, called a latch, built using two nor gates



Truth table for a `nor` gate (`or` followed by `not`)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

A 1-bit memory circuit, called a latch, built using two `nor` gates

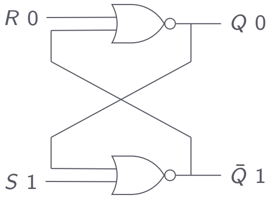


Logic Circuits

Truth table for a `nor` gate (`or` followed by `not`)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

A 1-bit memory circuit, called a latch, built using two `nor` gates

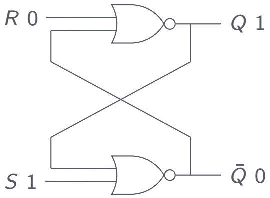


Logic Circuits

Truth table for a `nor` gate (`or` followed by `not`)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

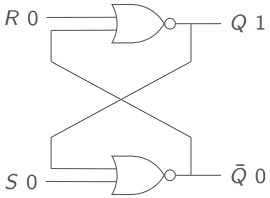
A 1-bit memory circuit, called a latch, built using two `nor` gates



Truth table for a nor gate (or followed by not)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

A 1-bit memory circuit, called a latch, built using two nor gates

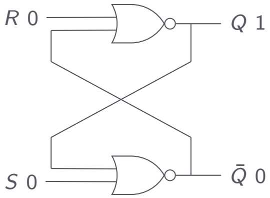


Logic Circuits

Truth table for a nor gate (or followed by not)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

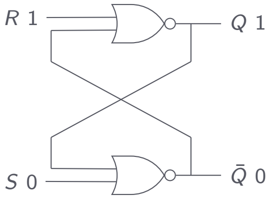
A 1-bit memory circuit, called a latch, built using two nor gates



Truth table for a nor gate (or followed by not)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

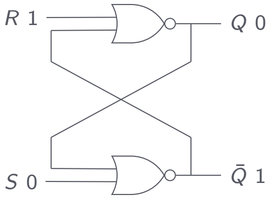
A 1-bit memory circuit, called a latch, built using two nor gates



Truth table for a nor gate (or followed by not)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

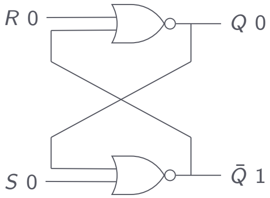
A 1-bit memory circuit, called a latch, built using two nor gates



Truth table for a nor gate (or followed by not)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

A 1-bit memory circuit, called a latch, built using two nor gates

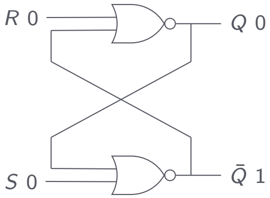


Logic Circuits

Truth table for a `nor` gate (`or` followed by `not`)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

A 1-bit memory circuit, called a latch, built using two `nor` gates

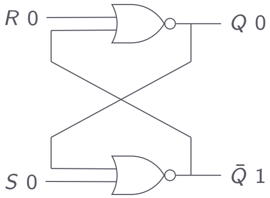


Logic Circuits

Truth table for a `nor` gate (`or` followed by `not`)

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0

A 1-bit memory circuit, called a latch, built using two `nor` gates



A billion latches can be combined together to produce a 1GB Random Access Memory (RAM) module

Von Neumann Architecture

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices such as ripple-carry adders for doing arithmetic, and a small amount of (scratch) memory called registers

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices such as ripple-carry adders for doing arithmetic, and a small amount of (scratch) memory called registers

The computer's main memory is separate from the CPU but connected to it by wires

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices such as ripple-carry adders for doing arithmetic, and a small amount of (scratch) memory called registers

The computer's main memory is separate from the CPU but connected to it by wires

A program, which is a long list of instructions, is stored in main memory and executed in the CPU, one instruction at a time

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices such as ripple-carry adders for doing arithmetic, and a small amount of (scratch) memory called registers

The computer's main memory is separate from the CPU but connected to it by wires

A program, which is a long list of instructions, is stored in main memory and executed in the CPU, one instruction at a time

The CPU has two special registers:

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices such as ripple-carry adders for doing arithmetic, and a small amount of (scratch) memory called registers

The computer's main memory is separate from the CPU but connected to it by wires

A program, which is a long list of instructions, is stored in main memory and executed in the CPU, one instruction at a time

The CPU has two special registers:

1. A program counter to track the next instruction to execute

Von Neumann Architecture

In a modern computer, the Central Processing Unit (CPU) is where all computation takes place

The CPU has devices such as ripple-carry adders for doing arithmetic, and a small amount of (scratch) memory called registers

The computer's main memory is separate from the CPU but connected to it by wires

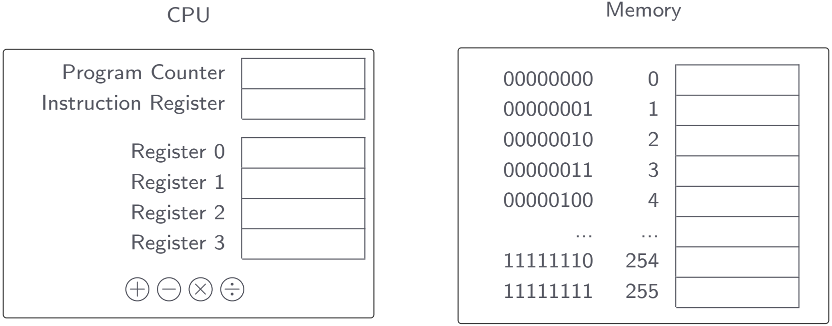
A program, which is a long list of instructions, is stored in main memory and executed in the CPU, one instruction at a time

The CPU has two special registers:

1. A program counter to track the next instruction to execute
2. An instruction register to store the next instruction for execution

Von Neumann Architecture

Example: an 8-bit computer with four operations (add, subtract, multiply, and divide), four registers (0 through 3), and 256 8-bit memory cells



Von Neumann Architecture

Von Neumann Architecture

Instructions, like data, can be encoded as numbers

Von Neumann Architecture

Instructions, like data, can be encoded as numbers

Example (our 8-bit computer revisited):

Von Neumann Architecture

Instructions, like data, can be encoded as numbers

Example (our 8-bit computer revisited):

- 2-bit operation encodings: add (00); subtract (01); multiply (10); divide (11)

Von Neumann Architecture

Instructions, like data, can be encoded as numbers

Example (our 8-bit computer revisited):

- 2-bit operation encodings: add (00); subtract (01); multiply (10); divide (11)
- 2-bit register encodings: register 0 (00); register 1 (01); register 2 (10); register 3 (11)

Von Neumann Architecture

Instructions, like data, can be encoded as numbers

Example (our 8-bit computer revisited):

- 2-bit operation encodings: add (00); subtract (01); multiply (10); divide (11)
- 2-bit register encodings: register 0 (00); register 1 (01); register 2 (10); register 3 (11)
- 8-bit instruction encoding: first two bits for the operation; next two bits for the result register; last four bits for the two input registers

Instructions, like data, can be encoded as numbers

Example (our 8-bit computer revisited):

- 2-bit operation encodings: add (00); subtract (01); multiply (10); divide (11)
- 2-bit register encodings: register 0 (00); register 1 (01); register 2 (10); register 3 (11)
- 8-bit instruction encoding: first two bits for the operation; next two bits for the result register; last four bits for the two input registers
- An assembly language program for computing the square of the sum of the values in registers 0 and 1, and storing the result in register 3

```
add 2 0 1
mul 3 2 2
```

Instructions, like data, can be encoded as numbers

Example (our 8-bit computer revisited):

- 2-bit operation encodings: add (00); subtract (01); multiply (10); divide (11)
- 2-bit register encodings: register 0 (00); register 1 (01); register 2 (10); register 3 (11)
- 8-bit instruction encoding: first two bits for the operation; next two bits for the result register; last four bits for the two input registers
- An assembly language program for computing the square of the sum of the values in registers 0 and 1, and storing the result in register 3

```
add 2 0 1
mul 3 2 2
```

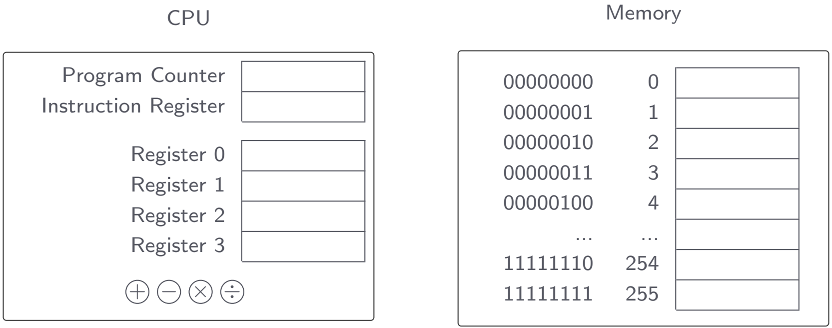
and the equivalent machine language program

```
00 10 00 01
10 11 10 10
```

Von Neumann Architecture

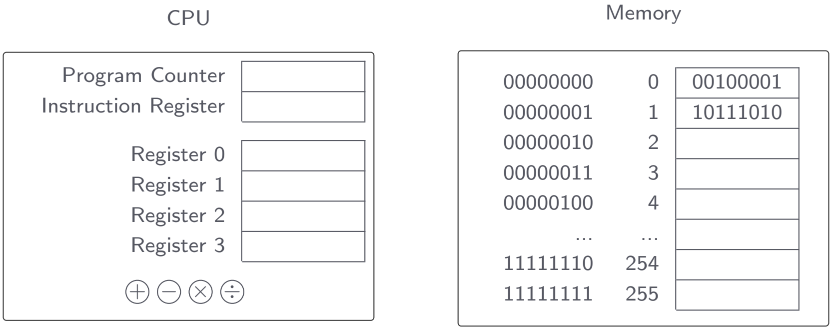
Von Neumann Architecture

Program execution



Von Neumann Architecture

Program execution



Program execution

CPU

Program Counter	
Instruction Register	
Register 0	00000100
Register 1	00000111
Register 2	
Register 3	

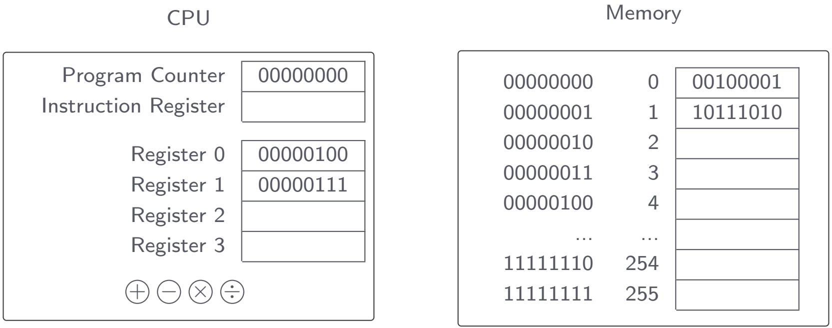
$\oplus$ $\ominus$ $\otimes$ $\div$

Memory

00000000	0	00100001
00000001	1	10111010
00000010	2	
00000011	3	
00000100	4	
...	...	
11111110	254	
11111111	255	

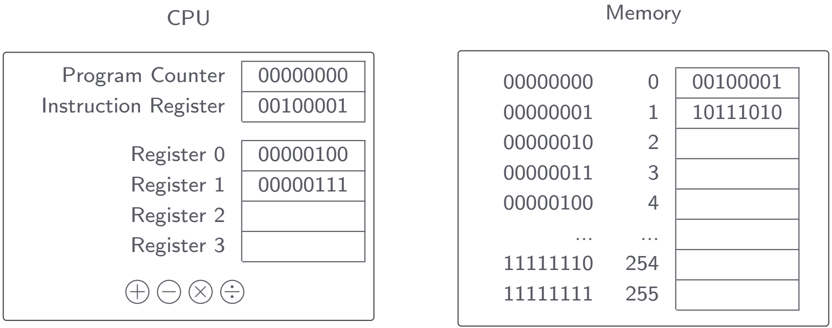
Von Neumann Architecture

Program execution



Von Neumann Architecture

Program execution



Von Neumann Architecture

Program execution

CPU

Program Counter	00000000
Instruction Register	00100001
Register 0	00000100
Register 1	00000111
Register 2	00001011
Register 3	

+ − × ÷

Memory

00000000	0	00100001
00000001	1	10111010
00000010	2	
00000011	3	
00000100	4	
...	...	
11111110	254	
11111111	255	

Von Neumann Architecture

Program execution

CPU

Program Counter	00000001
Instruction Register	00100001
Register 0	00000100
Register 1	00000111
Register 2	00001011
Register 3	

$\oplus \ominus \otimes \oslash$

Memory

00000000	0	00100001
00000001	1	10111010
00000010	2	
00000011	3	
00000100	4	
...	...	
11111110	254	
11111111	255	

Von Neumann Architecture

Program execution

CPU

Program Counter	00000001
Instruction Register	10111010
Register 0	00000100
Register 1	00000111
Register 2	00001011
Register 3	

$\oplus$ $\ominus$ $\otimes$ $\oslash$

Memory

00000000	0	00100001
00000001	1	10111010
00000010	2	
00000011	3	
00000100	4	
...	...	
11111110	254	
11111111	255	

Von Neumann Architecture

Program execution