SFU SIMON FRASER UNIVERSITY

SCIENCE ALIVE curiosity · creativity · confidence

What the Hack: Binary & ASCII | Episode 9 Challenges Solutions

Finish testing your knowledge? Take a look at some solutions to the challenges!

Challenge 1: Let's read the numbers from our computer's memory!

For the solution for this challenge, let's use the table below to guide us through the calculation!

The first row "Base" tells us what power is the base raised to; the second row "Value" shows us the actual numeric value of that base; the third row "Binary" is where we line up our binary numbers; and the fourth row "Switch" tells us whether the switch is on or off in our computer's memory.

Remember: if the switch is on, then we add the number; if the switch is off, then we don't!

1. 01010111

Base	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value	128	64	32	16	8	4	2	1
Binary	0	1	0	1	0	1	1	1
Switch	OFF	ON	OFF	ON	OFF	ON	ON	ON

In this table, we see that 64, 16, 4, 2, 1 are on. Let's add them up together:

64 + 16 + 4 + 2 + 1 = 87!

This binary number 01010111 represents the number 87 in our computer's memory!

2. 0101001010010101

This binary number is longer than 8 bits, but no problem! Doesn't matter how long the binary number is, from right to left, starting from the power of 0, you simply raise the base 2 to the power of the next higher integer! For a binary number that has 16 bits, the highest power that we raise the base to is 15.

Base	2_{15}	2_{14}	2_{13}	2_{12}	2_{11}	2_{10}	2_{9}	2_{8}
Value	32768	16384	8192	4096	2048	1024	512	256
Binary	0	1	0	1	0	0	1	0
Switch	OFF	ON	OFF	ON	OFF	OFF	ON	OFF

Base	2_{7}	2_{6}	2_{5}	2_{4}	2_{3}	2_{2}	2_{1}	2_{0}
Value	128	64	32	16	8	4	2	1
Binary	1	0	0	1	0	1	0	1
Switch	ON	OFF	OFF	ON	OFF	ON	OFF	ON

We see that the numbers 16384, 4096, 512, 128, 16, 4 and 1 are on. Let's add them up!

16384 + 4096 + 512 + 128 + 16 + 4 + 1 = 21,141

This is a big number! But it's no problem for our computer to store and represent as binary numbers!

3. 101001011010110110010100

This number is even bigger than the one in part 2! But now that you know the concept of how binary numbers work, you know that you can figure out what this number is by simply raising the base 2 to higher power! Let's take a look!

Base	2_{23}	2_{22}	2_{21}	2_{20}	2_{19}	2_{18}	2_{17}	2_{16}
Value	8388608	4194304	2097152	1048576	524288	262144	131072	65536
Binary	1	0	1	0	0	1	0	1
Switch	ON	OFF	ON	OFF	OFF	ON	OFF	ON

Base	2_{15}	2_{14}	2_{13}	2_{12}	2_{11}	2_{10}	2_{9}	2_{8}
Value	32768	16384	8192	4096	2048	1024	512	256
Binary	1	0	1	0	1	1	0	1
Switch	ON	OFF	ON	OFF	ON	ON	OFF	ON

Base	2_{7}	2_{6}	2_{5}	2_{4}	2_{3}	2_{2}	2_{1}	2_{0}
Value	128	64	32	16	8	4	2	1
Binary	1	0	0	1	0	1	0	0

Switch	ON	OFF	OFF	ON	OFF	ON	OFF	OFF

We've got 8388608, 2097152, 262144, 65536, 32768, 8192, 2048, 1024, 256, 128, 16 and 4 on!

That is a lot of numbers, let's add them up!

8388608 + 2097152 + 262144 + 65536 + 32768 + 8192 + 2048 + 1024 + 256 + 128 + 16 + 4 = 10,857,876

Now that is a super huge number! But our computer can store even larger number than this, by using more bits in its memory and raising 2 to the power of larger integer!

Challenge 2: Tell our computer a number!

For the solution of this challenge, let's utilize the table in challenge 1, but in a slightly different way!

The first row of the first table "Base" shows us what the base 2 is raised to the power of; the second row "Value" shows us the actual value when two is raised to the power of an integer, note that it is also the value inside the bit.

1. 27

Let's look through the values, and decide whether we will need the numbers in our calculation or not!

Base	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value	128	64	32	16	8	4	2	1

The number 128 is much much larger than our number 27, so we won't need it. Let's note down 0 for this bit; The number 64 and 32 are both larger than our number 27, so we will also note down 0 for those bits.

The number 16 is less than 27, which means we will need to use this number. Let's note down 1 for this bit. Before we continue to the next bit, we need to first subtract 16 from 27!

27 - 16 = 11

and we will continue to the next bit with the number 11.

The number 8 is less than our number 11, which means we will need it. Let's note down 1 for this bit! And do our subtraction before we continue!

11 - 8 = 3

The next bit has the number 4, which is larger than 3. We don't need it, so let's note down 0 for this bit!

The next bit is 2, which is less than 3, so let's note down 1 for this bit and do our subtraction!

3 - 2 = 1

and the next bit has exactly the number 1, so we will note down 1 for this bit too!

and with the table below, we see that 00011011 is the binary representation of the number 27!

Binary	0	0	0	1	1	0	1	1
Switch	OFF	OFF	OFF	ON	ON	OFF	ON	ON

2. 601

The number 601 is larger than the largest number that we have in our 8-bit memory, 128! What should we do? We simply use 8 more bits in our computer's memory for storing this number! and we will demonstrate that using an additional table.

Base	2_{15}	2_{14}	2_{13}	2_{12}	2_{11}	2_{10}	2_{9}	2_{8}
Value	32768	16384	8192	4096	2048	1024	512	256

Let's work off from this table first.

We see that the number 32768, 16384, 8192, 4096, 2048 and 1024 are all larger than our number 601. This means we don't need any of these bits, so let's note down 0 in those bits to turn them off!

We see that 512 is less than 601, so let's note down 1 in this bit and do our calculation!

601 - 512 = 89

With 89, we see that 256 is too big as well, so let's note down 0 in this bit, and continue to the next table!

Binary	0	0	0	0	0	0	1	0
Switch	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF

Base	2_7	2_6	2_5	2_4	2_3	2_2	2_1	2_0
Value	128	64	32	16	8	4	2	1

In this second table, we see that 128 is larger than our number 89. Let's note down 0 in this bit, and continue to the next bit!

The next bit has 64, which is less than 89. Let's note down 1 in this bit and do our subtraction.

89 - 64 = 25

With 25, we see that the next bit 32 is larger than what we have, so let's note down 0 and continue.

The next number is 16, which is less than 25. Let's note down 1 and do our subtraction!

25 - 16 = 9

and with 9, we see that it's larger than the next bit, which means we will also need the next bit. Let's note down 1 and again do our subtraction!

9 - 8 = 1.

With 1, we see that the following 2 bits are too big, so let's note down 0 for them.

The last bit on the table has exactly 1, so let's note down 1, and we are done with our calculation!

Combining the binary numbers in the two tables, we see that 0000001001011001 is the binary representation of our number 61!

Binary	0	1	0	1	1	0	0	1
Switch	OFF	ON	OFF	ON	ON	OFF	OFF	ON

3. 7474

The number 7474 seems much larger than the number 601 in part 2, but for our computer, it is actually not much longer to write in binary representation!

Base	2_{15}	2_{14}	2_{13}	2_{12}	2_{11}	2_{10}	2_{9}	2_{8}
Value	32768	16384	8192	4096	2048	1024	512	256

Because our number 7474 isn't larger than the largest number in this 16-bits memory, we don't need a 3rd table that contains larger values! We can begin with this table.

We see that 32768, 16384 and 8192 are all too big for our number 7474, so let's note down 0 to turn these bits off.

We see that the next bit has 4096, which is less than 7474, so let's note down 1 and do our calculation!

7474 - 4096 = 3378

With 3378, we see that we also need the next bit 2048, so let's note down 1 and do the calculation again!

3378 - 2048 = 1330

1330 is again larger than our next bit 1024, so let's note down 1 again and do the calculation!

1330 - 1024 = 306

With 306, we see that the next bit is too large, so we will note down 0 and continue to the next bit.

The next bit has 256, which is less than 306. Let's note down 1 and do our calculation!

306 - 256 = 50

And with 50, we will continue to our next table!

Binary	0	0	0	1	1	1	0	1
Switch	OFF	OFF	OFF	ON	ON	ON	OFF	ON

Base	2_7	2_6	2_5	2_4	2_3	2_2	2_1	2_0
Value	128	64	32	16	8	4	2	1

With our number 50, we see that both 128 and 64 are too big, we don't need them. So let's note down 0 and continue to the next bit!

The next bit has 32, which is less than 50. Let's note down 1 and do our calculation!

50 - 32 = 18

and with 18, we will need the next bit because it is 16. Let's note down 1 again and do the calculation!

18 - 16 = 2

with 2, we see that we don't need any other bits except the second last bit, which has exactly the number 2. Let's note down 1 for the second last bit and 0 for the other bits!

And that is the end of our calculation. Our number 7474 converted to binary numbers is 0001110100110010. This number looks nowhere close to 7474 in our mind, but our computer will be able to store it in its memory in this way!

Binary	0	0	1	1	0	0	1	0
Switch	OFF	OFF	ON	ON	OFF	OFF	OFF	OFF

Challenge 3: Tell our computer a message!

As we are now converting characters into binary, we need to make use of the ASCII table!

1. “Hello”

Looking through our ASCII table, we can find that uppercase H is associated with the integer 72, lowercase e is associated with the integer 101, lowercase l is associated with the integer 108, and lowercase o is associated with the integer 111!

Now let’s convert all these integers into binary numbers!

Using our table in challenge 2, we can see that the integer 72 in binary representation is 01001000; and the integer 101 in binary is 01100101; the integer 108 in binary is 01101100; and the integer 111 in binary is 01101111.

Character	ASCII integer (key)	Binary Number
H	72	01001000

e	101	01100101
l	108	01101100
l	108	01101100
o	111	01101111

If we put all these binary numbers together, we get:

01001000 01100101 01101100 01101100 01101111

This is how the message “Hello” will get stored in our computer’s memory as binary numbers!

2. “What The Hack.”

Look through the ASCII table, and you will find the integer key associated with each character.

Using our binary and switches table in challenge 2, we can convert these integers into binary numbers.

The table below lists the characters, their unique ASCII integer keys and the binary numbers of those integers! Remember that empty space and period punctuation are also characters, and they also have specific ASCII integer keys associated with them!

Character	ASCII integer (key)	Binary Number
W	87	01010111
h	104	01101000
a	97	01100001
t	116	01110100
(space)	32	00100000

T	84	01010100
h	104	01101000
e	111	01100101
(space)	32	00100000
H	72	01001000
a	97	01100001
c	99	01100011
k	107	01101011
. (period punctuation)	46	00101110

Putting all the binary numbers together, we have :

01010111 01101000 01100001 01110100 00100000 01010100 01101000 01100101 00100000 01001000 01100001 01100011 01101011 00101110

And this is how the name of our series “What The Hack.” will be stored in our computer’s memory!

3. “Thank you!”

Look through the ASCII table, and you will find the integer key associated with each character.

Using our binary and switches table in challenge 2, we can convert these integers into binary numbers.

The table below lists the characters, their unique ASCII integer keys and the binary numbers of those integers! Remember that empty space and period punctuation are also characters, and they also have specific ASCII integer keys associated with them!

Character	ASCII integer (key)	Binary Number
T	84	01010100
h	104	01101000
a	97	01100001
n	110	01101110
k	107	01101011
(space)	32	00100000

y	121	01111001
o	111	01101111
u	117	01110101
! (exclamation mark)	33	00100001

Putting all the binary numbers together, we will get:

01010100 01101000 01100001 01101110 01101011 00100000 01111001 01101111 01110101 00100001

And this is how “Thank you!” will be stored in our computer’s memory! Next time when your computer helps you do something, maybe say thank you to it in binary numbers :) !

Challenge 4: Let’s read the messages from our computer’s memory!

When working with characters using the ASCII table, we will need to separate binary numbers into bytes (8-bits), and convert each byte into an integer, and translate the integer back to the character using the ASCII table.

1. 010010000110100100100001

Let’s first break down this string of binary numbers into bytes:

01001000 01101001 00100001

Then, let’s convert them back into decimal numbers, and then use these numbers to look up the characters in the ASCII table!

The table below shows the binary numbers, what decimal numbers they represent, and what characters those decimal numbers are associated with in the ASCII table.

Binary Number	Decimal Number (ASCII key)	Character
01001000	72	H
01101001	105	i
00100001	33	!

Putting all the characters together, we have :

Hi!

The binary numbers 01001000 01101001 00100001 represent the message “Hi!” in our computer’s memory!

2. 0100000101110111011001010111001101101111011010110101100101

Let’s first break down this string of binary numbers into bytes:

01000001 01110111 01100101 01110011 01101111 01101101 01100101

Then, let’s convert them back into decimal numbers, and then use these numbers to look up the characters in the ASCII table!

The table below shows the binary numbers, what decimal numbers they represent, and what characters those decimal numbers are associated with in the ASCII table.

Binary Number	Decimal Number (ASCII key)	Character
01000001	65	A
01110111	119	w
01100101	101	e
01110011	115	s
01101111	111	o

01101101	109	m
01100101	101	e

Putting the characters together, we have :

Awesome

This string of binary numbers 01000001 01110111 01100101 01110011 01101111 01101101 01100101 is how our computer will store the word Awesome!

3. 0101100101101111011101010010000001100001011100100110010100100000011101110110010101100011000110110111101101011001010010000

Let's first break down this string of binary numbers into bytes:

01011001 01101111 01110101 00100000 01100001 01110010 01100101 00100000 01110111 01100101 01101100 01100011 01101111 01101101 01100101 00100001

Then, let's convert them back into decimal numbers, and then use these numbers to look up the characters in the ASCII table!

The table below shows the binary numbers, what decimal numbers they represent, and what characters those decimal numbers are associated with in the ASCII table.

Binary Number	Decimal Number (ASCII key)	Character
01011001	89	Y
01101111	111	o
01110101	117	u
00100000	32	(space)
01100001	97	a

01110010	114	r
01100101	101	e
00100000	32	(space)
01110111	119	w
01100101	101	e
01101100	108	l
01100011	99	c
01101111	111	o
01101101	109	m
01100101	101	e
00100001	33	! (exclamation mark)

Putting all the characters together, you will see :

You are welcome!

this long string of binary numbers
0101100101101111011101010010000001100001011100100110010100100000011101110110010101101100011000110110111101101101011001010010000100001 is how our computer represents the message “You are welcome!” Maybe when you say “Thank you!” to your computer in binary numbers, this is what it will respond back to you! :)