



Episode 2 (Zone and Pack)



Hey, hey, fox!
If I press the keys **A** and **1** from the keyboard, what happens then?



When the raccoon dog presses **1**, the code [0011 0001] is transferred to the memory, and when **A** is pressed, the code [0100 0001] is transferred to the memory.
When the raccoon dog presses the key, the code [0011 0001] is sent to the memory device. [The [0011] in [0011 0001] is called the zone part and is used to treat numbers as characters.
Removing the zone [0011] and changing it to [0001] is called packing, and the pressed **1** can be used as a numerical value in calculations.



What spell is [0011 0001] or [0100 0001]!



This is called a binary number, and since computers are ON (1) and OFF (0) circuits, everything is expressed in binary. [0011 0001] is 49 in decimal, 31 in hexadecimal.
You have to learn binary and hexadecimal numbers by yourself. I will only teach you the outline, so you have to learn it by yourself.
Study is hard work, so it's not always easy!



I should also add that when outputting numerical data to the monitor, the data must be marked with a zone [0011] in order to be displayed.

Adding a zone section to a pucker is also called unpacking.



I understand that characters are stored as the code assigned to the key, but am I correct in assuming that numbers are stored in a packed format?



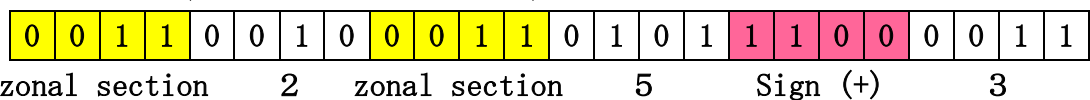
Tanuki is right about how letters are stored, but numbers are a bit more complicated. There are positive and negative numbers, and then there are integers, and then there are real numbers with a decimal point.

The way numbers are stored in the memory is important and I will comment on it now.

How numbers are stored in the memory device

Positive integer (e.g., +253 for)

Zone decimal (treated as characters)



packing

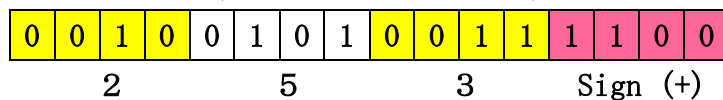


conversion



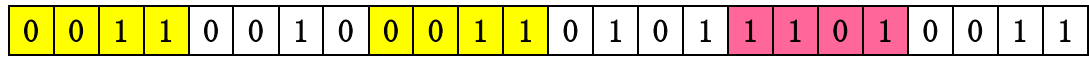
unpacking

Packed decimal (treated as a number)

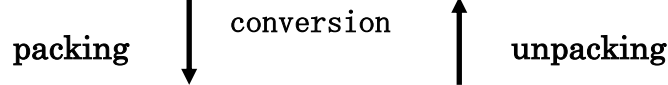


Negative integer (e.g., -253 for)

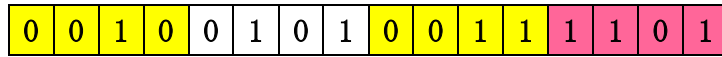
Zone decimal (treated as characters)



zonal section 2 zonal section 5 Sign (-) 3



Packed decimal (treated as a number)



2 5 3 Sign (-)



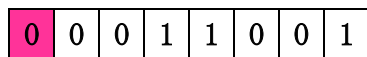
The numbers entered from the keyboard are stored in zone decimal or packed decimal as needed, but may also be stored in integer or real (fixed or floating point) format as directed by the program.

This is explained next.

How numbers are stored in the memory device according to the program's instructions (type)

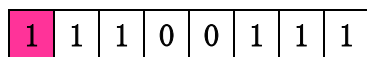
Integer (int) type

How a positive integer (e.g. +25) is stored (8 bits of storage capacity = 1 byte)



↑
Sign bit (+)

How negative integers (e.g., -25) are stored (2's complement)



↑
Sign bit (-)



Choi wait!
 What's the 2's complement to memorize
 -25? You're omitting too much of the
 explanation.



I was going to explain complement later because
 it's tedious, but I'll explain complement in
 decimal instead of binary in a nutshell. If you make
 a negative number the complement, you can make a
 subtraction into an addition. There are two types
 of arithmetic units: addition and subtraction
 circuits, but the addition circuit is simpler, less
 expensive, and faster.

**Now, let's get an idea of complementary numbers.
 25 + (-25) = 0 becomes 25 + 75 = 00 when the
 complementary number is calculated.**

Decimal (-25) complement calculation

$$\begin{array}{r}
 99 \text{ (Maximum number of 2 digits)} \\
 - 25 \text{ (positive decimal)} \\
 \hline
 74 \text{ (1's complement in decimal)} \\
 + 1 \\
 \hline
 75 \text{ (2's complement in decimal)}
 \end{array}$$

**Two's complement is the addition of one to the
 one's complement; adding one in advance to
 the largest two-digit number to make three
 digits gives the two's complement.**

$$\begin{array}{r}
 100 \text{ (99 plus 1 first plus 3 digits)} \\
 - 25 \\
 \hline
 75 \text{ (2's complement in decimal)}
 \end{array}$$

$$25 + 75 = \cancel{1}00$$

Since this is a two-digit calculation, removing the up-digit 1 will
 result in 00 (i.e., 0).



(25-25) is 0. If we consider that a calculation using the two's complement of a decimal number (25 + 75) cannot store the third digit of a digit increase, the result is 0, right? This is what I mean by being able to handle subtraction with addition. I've misled you a bit, but did you notice the raccoon dog? Now, let's try the same thing with the two's complement of a binary number.

Binary 2's complement calculation

00011001 (Decimal +25)

11111111 (Maximum number of 8 digits)
 -00011001 (Binary positive number)

11100110 (Binary 1's complement)
 + 1

 11100111 (Binary 2's complement)

00011001 (+ 25)
 +11100111 (- 25)

~~10000000 (0)~~

Since this is an 8-bit storage area, the ninth bit of the digit is dropped.



You are using a subtraction circuit to find the 1's complement of a binary number (11100110). This can't be handled by the addition circuit alone. You are trying to fool me!



Tanuki, you are perceptive!

In computers, the 1's complement of a binary number (11100110) is found by inverting (00011001), not subtracting. Since it then adds 1, it is impossible to change the sequence of finding the 2's complement via the 1's complement.

By the way, there is no way to explain decimal numbers. After the complement calculation, 100 is obtained by the complement, so the correct answer is $100 - 100 = 0$, since the complement is undone.



When I went to Hawaii, I bought some nut chocolates as a souvenir for \$2 and gave her a \$10 bill. The lady at the counter asked me to open my hand, and when I did, she placed eight one-dollar bills in my palm: three, four, five, six, seven, eight, nine, and ten dollars. At that time, I was impressed that this lady had converted subtraction into addition.



Fox!

I hope you didn't give him a \$10 bill for a leaf, and he didn't get a \$2 nut chocolate and an \$8 bill.



Tanuki, I don't think too much.
Next, let's consider addition in the arithmetic unit.
Before we do that, there is one important thing to remember.
Remember that a storage device has two separate parts: the part that stores the program and the part that stores the data that the program uses.

Okay, then. **Episode 3.**

Area for storing programs in a storage device

Area for storing data (to be used) in a storage device