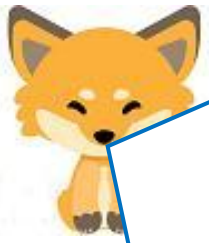
Episode 4 (Output device)

Kitsune, there is still output to the monitor, but the result [0000000000000001010] (+10) calculated by the arithmetic unit has been stored in [8009] (address) in the main memory, so we can output this and be done with it.

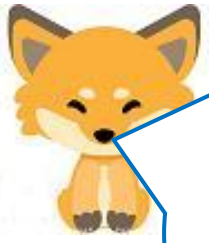
It's an acerbic concoction!
If you output it as is, as Tanuki said, you'll just get blanks on your monitor or garbage. If it's a printout, I guess nothing will be printed and it'll be like, what the heck is this? As I think I mentioned in Episode 2, to be displayed on the monitor, it has to be in character type, not numeric type. Plus, it has to be in zone format or it won't be displayed.

It's a lot of work just to get it to display.



In a high-level language, the print and write commands are enough to display and print, but in an assembler language, you have to convert a number to a character type, add a zone part to it, and make the code the same as the key on the keyboard. However, in the case of high-level languages, the programmer does not have to do the conversion work, but the same thing is done automatically during the compilation process. High-level languages, which use code similar to human language, are easy to handle, while assembler languages, which are similar to machine language, are difficult for humans to handle. In that sense, they are low-level languages. Assembler languages work directly with the CPU, so the processing speed was much faster, but nowadays the performance of computers has improved dramatically, so even if you use a high-level language, you no longer feel slow.

I understood that I had to add a zone part to the resultant number to get it to display, but then the next issue came up that I had to learn more about high-level languages, low-level languages, compilers, and so on.



Yes, but first, let's think about displaying the result of +10 with a zone part. This is a lot more difficult than it seems: we have to take the 1's and 0's out of the +10 and make them into [0000000001] and [000000000000]. If we add the zone part [0011000000], we get [00110001] and [0011000000], which are 31 and 30 in hexadecimal, the same codes as the keys [1] and [0] on the keyboard.

The code is the same as the key 1 and 0 on the keyboard, and can be displayed and printed.

Can you give me a concrete example, like 10 divided by 10?

It is easy to take out the first digit as 0 and the second digit as 1, but it takes a lot of knowledge to make a computer arithmetic unit do this. But the basic idea of dividing by 10 to take out each digit is the same.

It is important to understand that arithmetic units can only perform addition, subtraction, multiplication, division, and comparison operations. In addition, multiplication and division are a combination of shift operations and addition to produce the result. Therefore, you cannot understand multiplication and division without understanding the shift operation.

Let me explain the shift operation!

[Shift operation]

Multiplication is done by shifting left.
Let us calculate 7 x 5 = 35 by imagining a CPU arithmetic unit.
7 in 8-bit binary representation is as follows.

0	0	0	0	0	1	1	1

Shifting one bit to the left doubles the value to 14.

0	0	0	0	1	1	1	0

Shifting one bit further to the left increases the value by a factor of 4 to 28.

0	0	0	1	1	1	0	0

Shifting one bit further to the left would increase the value by a factor of 8 to 56. Here, the shift operation can no longer be used, and 7 must be added.

```
  00011100
+ 00000111
------------
  00100011
```

35 (decimal)

0	0	1	0	0	0	1	1

Division is done by right shift.
Let us calculate 7 ÷ 5 = 1･･･2 (remainder) by imagining a CPU arithmetic unit.
7 in 8-bit binary representation is as follows.

0	0	0	0	0	1	1	1

Shifting one bit to the right should be 1/2 times 3.5, though.

0	0	0	0	0	0	1	1

→ 1 A bit drop would be 3.

If you shift it one bit further to the right, it should be 1/4 times 1.75.

0	0	0	0	0	0	0	1

→ 1 A bit drop will result in 1.

If the division of an integer divisible by 2, the shift operation
but it is not likely to be possible to do 7 divided by 5 without combining it with other methods.
7 divided by 5 is not likely to be possible unless it is combined with other methods.
Tanuki, if you are interested in the details of the shift operation
you should study the details of the shift operation by yourself.

OK, I know what to study!

Now that I've explained the basic shift operation, I'll go back to the beginning and explain +10 ÷ 10 = 1･･･0!
I'll use the assembler language to illustrate the outline because it's complicated and confusing to explain.

[＋The concept of taking out the second digit 1 and the first digit 0 of +10]

Ignore sign

+１０ ÷ １０ ＝ １ (dealing)・・・・０ (over)

A B S M and place it on.

(initialization) Prepare registers in the CPU as follows.

A	0	0	0	0	1	0	1	0

S	0	0	0	0	0	0	0	0

Shifted 1 bit to the left so that A < B

B	0	0	0	1	0	1	0	0

M	0	0	0	0	0	0	0	0

L indicates digit to be operated, linked with B

L	0	0	0	0	0	0	1	0

(procedure①) If A ＜ B

・Shift B one bit to the right.

B	0	0	0	0	1	0	1	0

S	0	0	0	0	0	0	0	0

・L is also shifted one bit to the right.

L	0	0	0	0	0	0	0	1

M	0	0	0	0	0	0	0	0

A	0	0	0	0	1	0	1	0

(procedure②) If A >= B

・A － B

```
  00001010
- 00001010
----------
  00000000
```

・S ＋ L

```
  00000000
+ 00000001
----------
  00000001
```

A	0	0	0	0	0	0	0	0

S	0	0	0	0	0	0	0	1

B	0	0	0	0	1	0	1	0

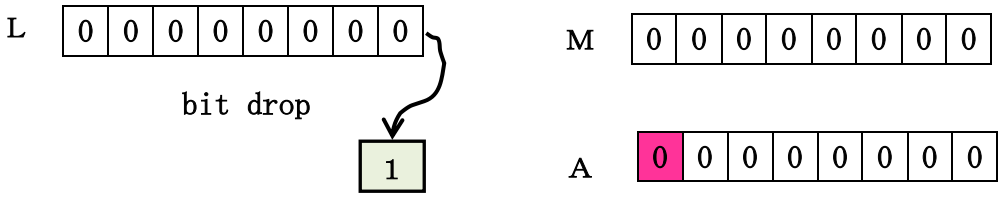
M	0	0	0	0	0	0	0	0

L	0	0	0	0	0	0	0	1

(procedure③) If A ＜ B

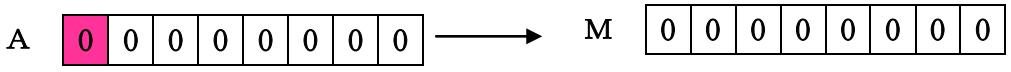
・Shift B one bit to the right.

・L is also shifted one bit to the right.



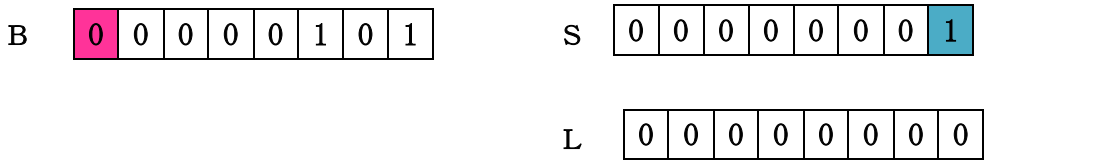
(procedure④) If L = 0

Extracting the first digit (remainder)



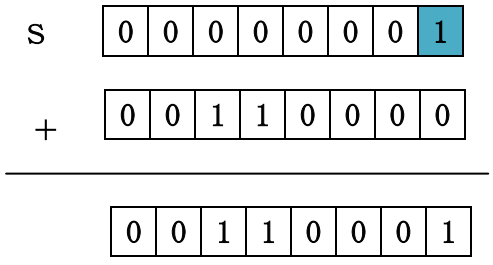
Transfer the value of A to M

Extracting the second digit (quotient)



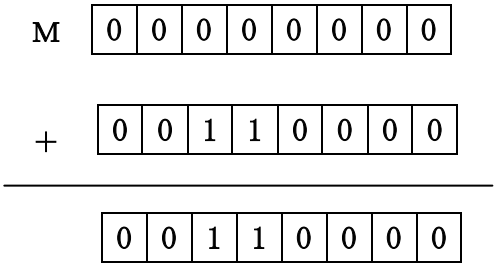
(procedure⑤) Addition of zone section [0011].

Extracting the second digit (quotient)



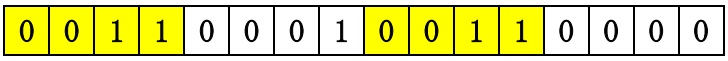
31 (hexadecimal key code)

Extracting the first digit (remainder)



30 (hexadecimal key code)

※ The output will be available at 3130.



I've explained it with the simplest example. See for yourself that "27 ÷ 10" can be used to separate 2 and 7 using the above method. This is just an exercise.
The result sent from the CPU to the memory device is sent to the output device like this. This is the basics.
As described above, being able to visualize the five major functions of a computer from input (data) to output (data) is the basis for learning about computers, and is the key to studying computers from now on. Once you have seen the forest, you can now go and see the trees (details).

I understand what Kitsune was trying to say. Some parts were difficult, but that's the basics. I was hoping to be able to make a fist fight game program soon.

If you don't understand the basics, even if you can program, you'll end up just attaching parts (libraries and classes) and losing the big picture. You will just enter the world of Chaplin's Modern Times.

Mr. Ponkichi, a professor of information at the University of Ponpoko, drew an arrow that outputs the calculation results directly from the CPU's arithmetic unit to the monitor without going through the main memory, but this is an error caused by understanding the memory unit only as a function that controls memory, and not understanding the important part of memory of the conversion process between pack and zone. This is a mistake because he does not understand the important part of memory, which is the process of converting between packs and zones. That's how important the basics are.

Now that we have the basics, let's move on to Episode 5. Let's move on to the tree part (in detail) from Episode 5.

Go to Episode 5

Translated at DeepL