Computer Science (Episode 3)

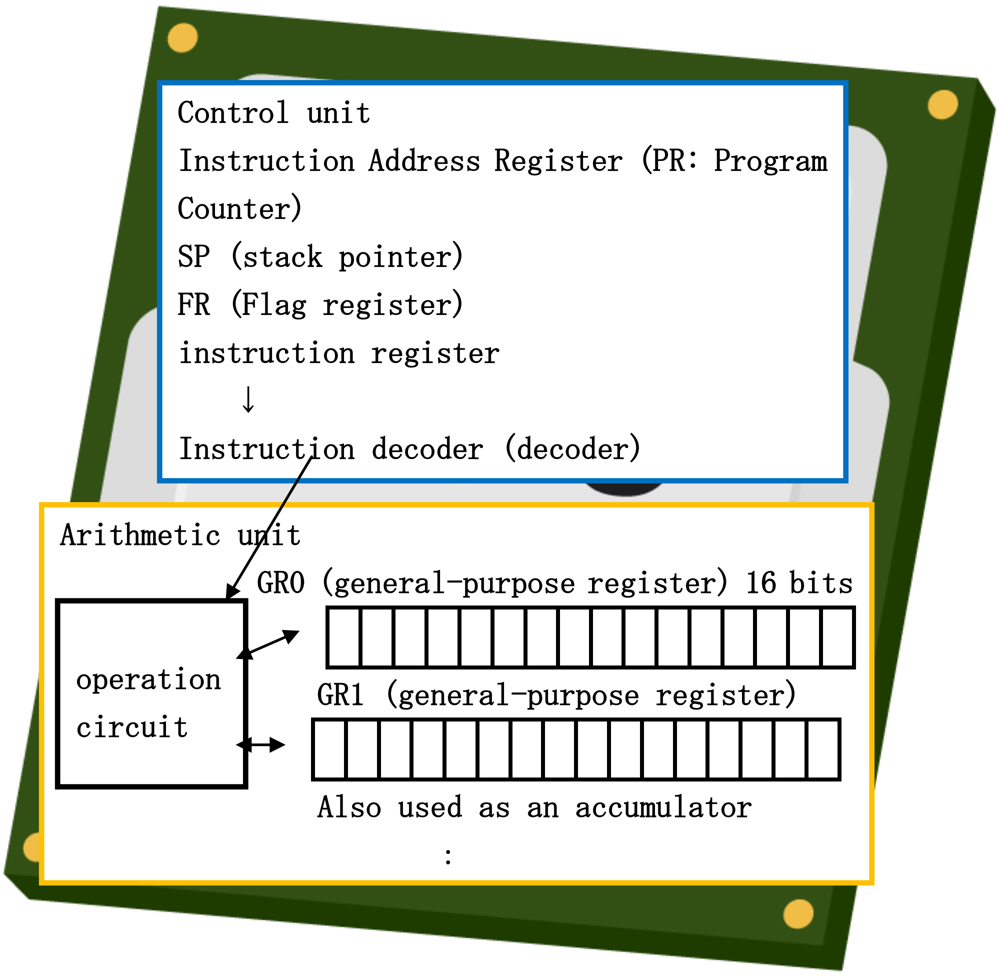
Episode 3 (Control and arithmetic devices)

Tanuki, next we're going to talk about CPUs (central processing units).
It's a tough one, so don't give up!

What is CPU?

It is the heart of the computer, the most important part of the five major devices, and does the most complex job.
Let us illustrate it below!

CPU (Central Processing Unit)

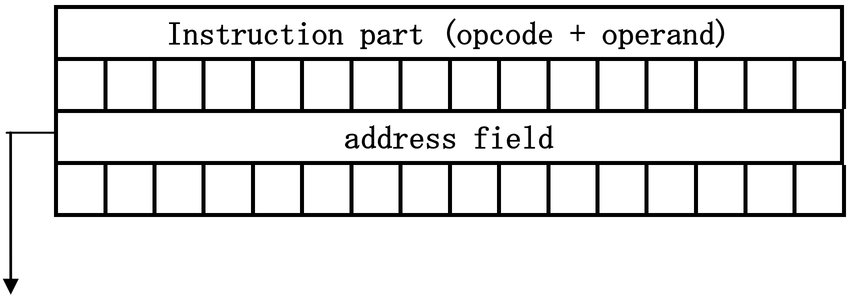


[Solution.]

A register is the name given to an area in the CPU that temporarily stores data. The register is a name given to an area in the CPU that temporarily stores data.

Control unit

- instruction address register
  A register that stores the address of the next program to be executed.
  Also called program counter (PR).
- Stack-pointer
  A register that stores the return address when control is transferred from the main program to a subprogram.
- Flag register (3 bits)
  OF (1 bit): 1 when the result of an operation no longer fits into the 16-bit area, 0 when it fits.
  Different for arithmetic and logical operations.
  SF (1 bit): 0 for a positive (+) operation result, 1 for a negative (-) result.
  ZF (1 bit): 1 for an arithmetic result of 0, 0 otherwise.
- instruction register
  Registers that store program instructions in the following format.



- Instruction decoder (decoder)
  The code in the instruction section of the instruction register is retrieved, interpreted, and operation instructions are conveyed to the arithmetic unit.

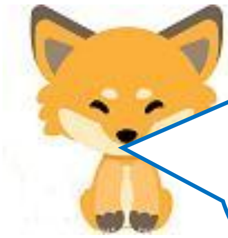
arithmetic unit

- general purpose register
  There are five registers from GR0 to GR4. (The number varies with specifications.)
  Stores numerical values used in calculations and saves calculation results.
  In particular, GR4 is used as an accumulator that stores the results of accumulations.

- operation circuit

  Performs arithmetic (addition, subtraction, multiplication, and division quadrature), logic, comparison, and shift operations.

There were so many terms I didn't know that it was making my head hurt.



Sure, maybe not for raccoons. I don't need to memorize the terms, I wish I could grasp the flow of the process. I hope you realize that there is a lot to learn in order to understand the terms used here.
For example, if you wonder what a stack pointer is, you need to learn about PUSH and POP of the stack area. This is what I call "seeing the forest and knowing the trees! I don't know if it's possible.

The arithmetic circuit shown above has an addition circuit and a subtraction circuit. When I explained complementary numbers before, I said that the addition circuit is simple and the subtraction circuit is a bit more complicated and slower.
Let me try to explain that here.

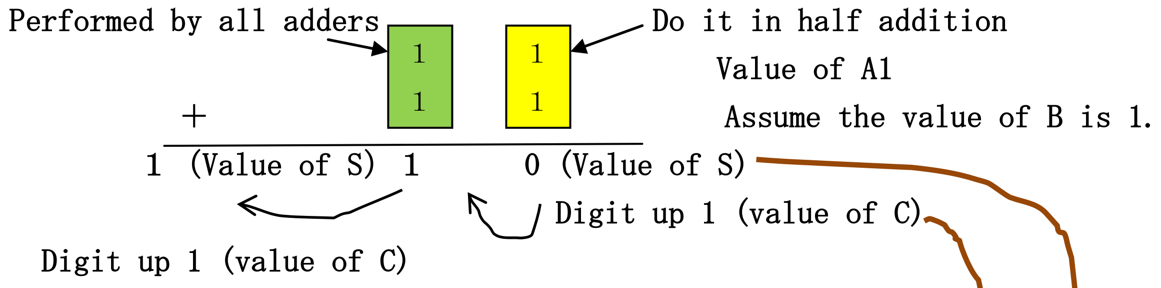
Kitsune, draw a diagram and explain it clearly.

The decimal number 3 + 3 = 6, which even a raccoon would understand. If we express this in binary, we get [11] + [11] = [110]. Let's run this through an adder. Let us mention that there are two types of adders: half adders, which are used to add the first digit, and full adders, which are used after the second digit corresponding to the carry-over.

Additive circuits for arithmetic circuits (half and full adders)

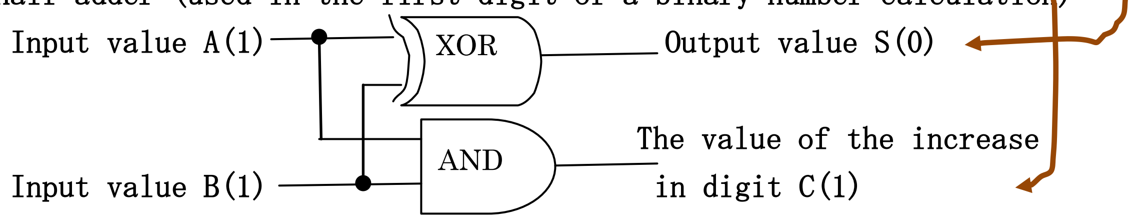
There are two types of adders: half adders that cannot carry carry digits and full adders that can carry digits.

(Calculation example) Let's calculate the following in binary.



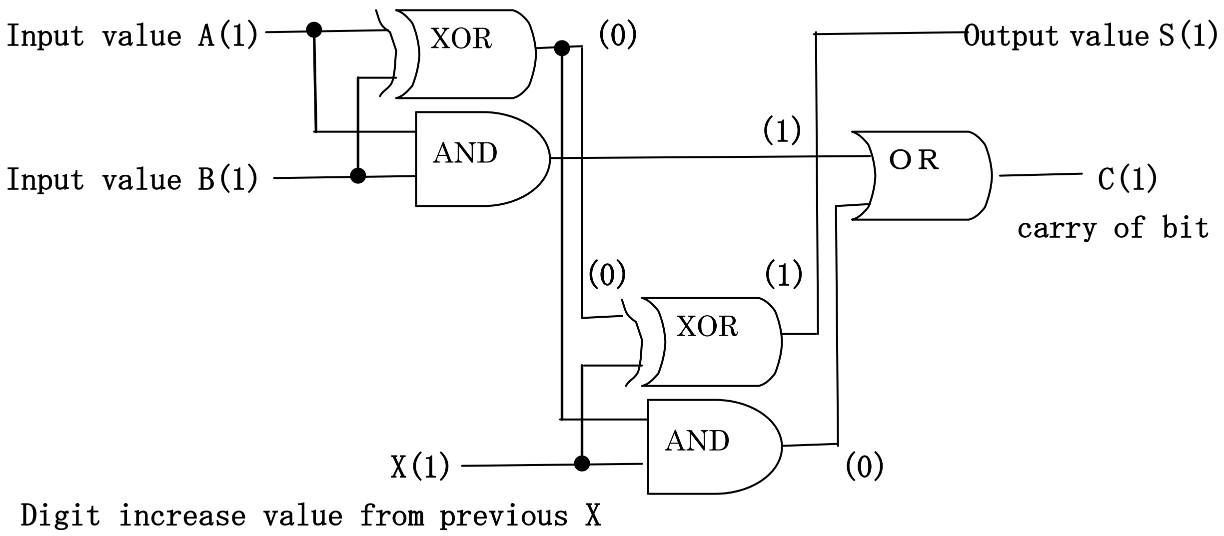
This is done with a half adder.

Half-adder (used in the first digit of a binary number calculation)



The next digit (the second digit) is made with the full adder.

All adders (used after the second digit of a binary number calculation)



Tanuki, I hope you understand. You can see that after you have reached this point, you must next study logical operators such as AND, OR, XOR, and so on. This is what it means to study. The more you know, the deeper you go.

Now, subtraction circuits also have half-subtractors and full-subtractors, and when subtracting, you have to borrow from the previous one, which is complicated, so I won't explain it here!

You may or may not know about the arithmetic circuits, Kitsune, but I don't see how those other registers and such work at all!

That's right. I don't get it, do I?

So, I thought, "Why don't I just run a simple program and look at it one by one? I thought it would be better to run a simple program and look at it one by one.

The programming language that the CPU can understand is machine language. Machine language is ultimately expressed in binary numbers, but that is too difficult to understand, so we use a language called assembler language. Converting a program written in assembler language into machine language is called assembly.

So the next step is to learn a programming language.

Programming languages include C, Python, Java ・・・・
and so on, but they all eventually get converted to machine language. Here, you've probably seen the need for a programming language.
I'm going to use assembler language for the explanation. Assembler language is a language whose specification is determined by the type of CPU.
Therefore, I will use the assembler language called COMET and CASL II, which are used in the Basic Information Technology Engineer Examination.

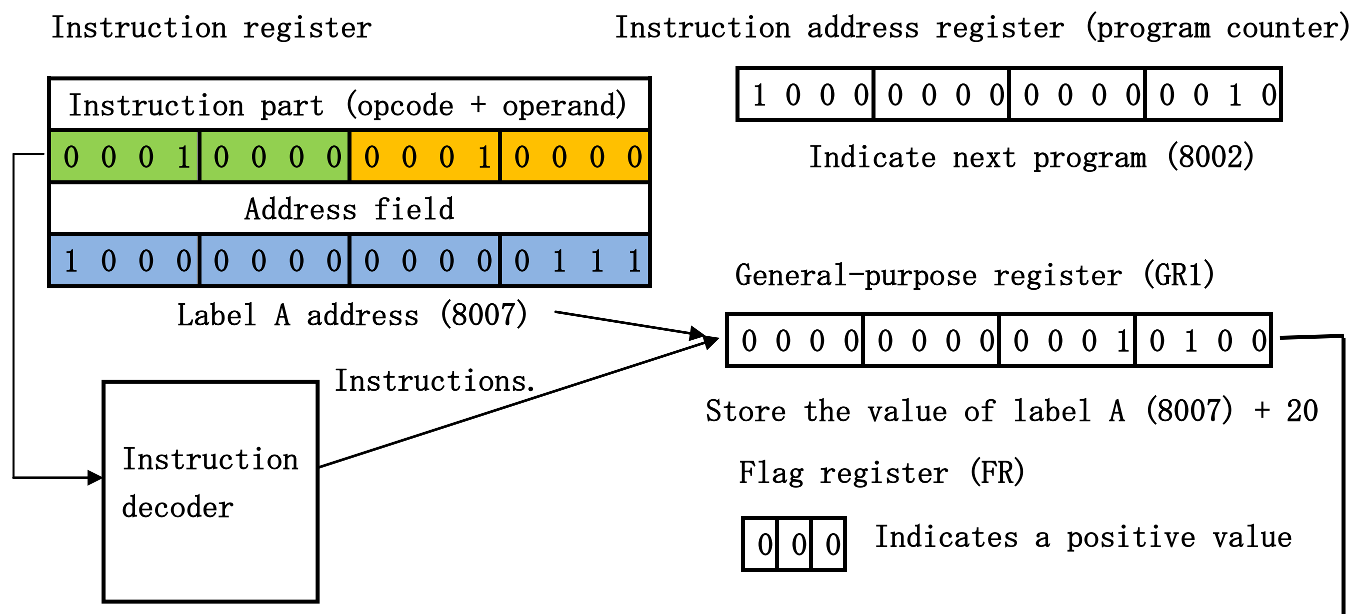
I came up with a program (asm1-7.cs2) that does the simplest calculation: 20 + (-10: converted to the complement) = 10.
The program is shown below, but even if you don't know the language, you should be able to visualize it. If you are interested, you can learn assembler language as well.

asm1-7.cs2

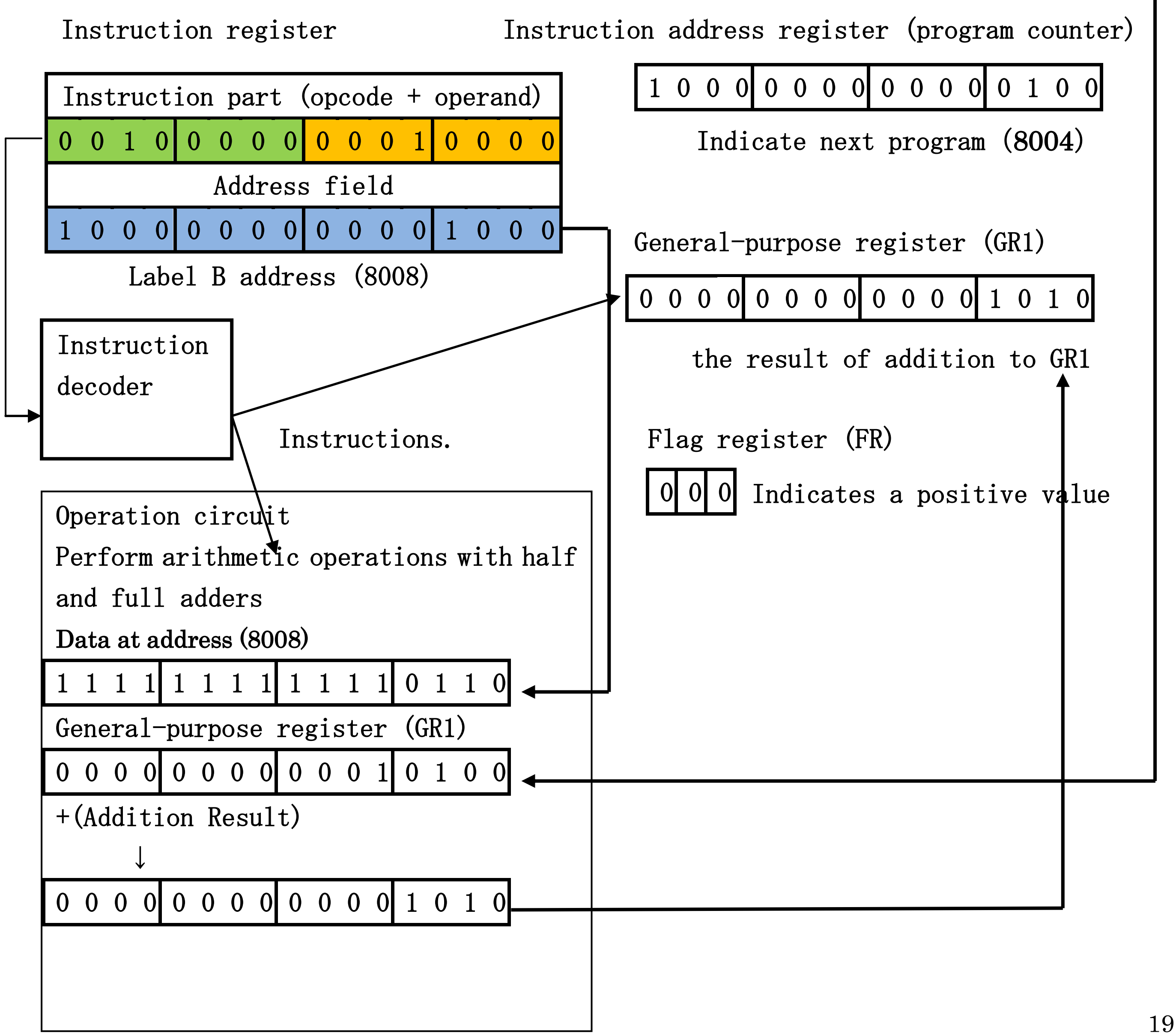
assembler source program	hexadecimal	Machine language (binary)
Labels. Inst. part (opcode + operand)		main storage
REIDAI START Explanation; Start of program		8000-8001 (address) program area
① LD GR1,A ;opcode operand、label ; Put the data of label A into the general purpose register (GR1).	1010 8007	0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
② ADDA GR1,B ; Add the data of label B to the value of GR1 and assign the result to GR1.	2010 8008	8002-8003 (address) 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
③ ST GR1,ANS ; Store the value of GR1 in label ANS.	1110 8009	8004-8005 (address) 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
④ RET ; Stopping the program.	8100	8006 (address) without operand 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
A DC 20 ; Data are 20	0014	8007 (address) data area 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
B DC -10 ; Data is -10, but complement conversion.	FFF6	8008 (address) Complement computed 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0
ANS DS 1 ; 1 word length (16 bits) area gua ranteed 1 word length (16 bits)	7FFF	8009 (address) Initial value for securing one word length 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
END ; End of Program		

I've attached an explanation of the opcode (shown in green) in the instruction section to the source on the left. The operand GR1 (shown in orange) is represented by [0001 0000], while GR2 is [0010 0000], distinguishing the general-purpose register to be used. The first one [10000000000000000111] (8007), displayed in blue, indicates the location (address) of the main memory in Label A. The -10 in the data displayed in red is converted to the binary 2's complement [11111111111111110110].
Next, let's look at the state of processing by the control unit & arithmetic unit for each instruction from (1) to (4).

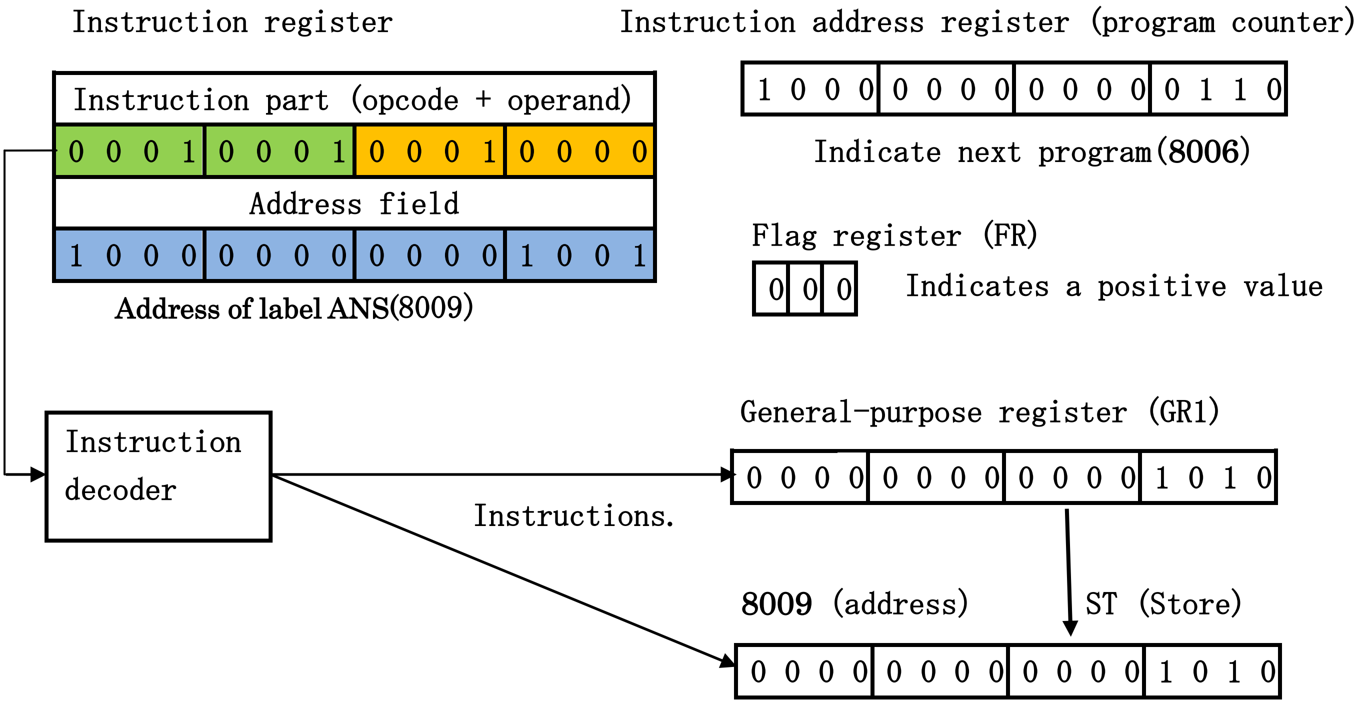
① LD GR1, A



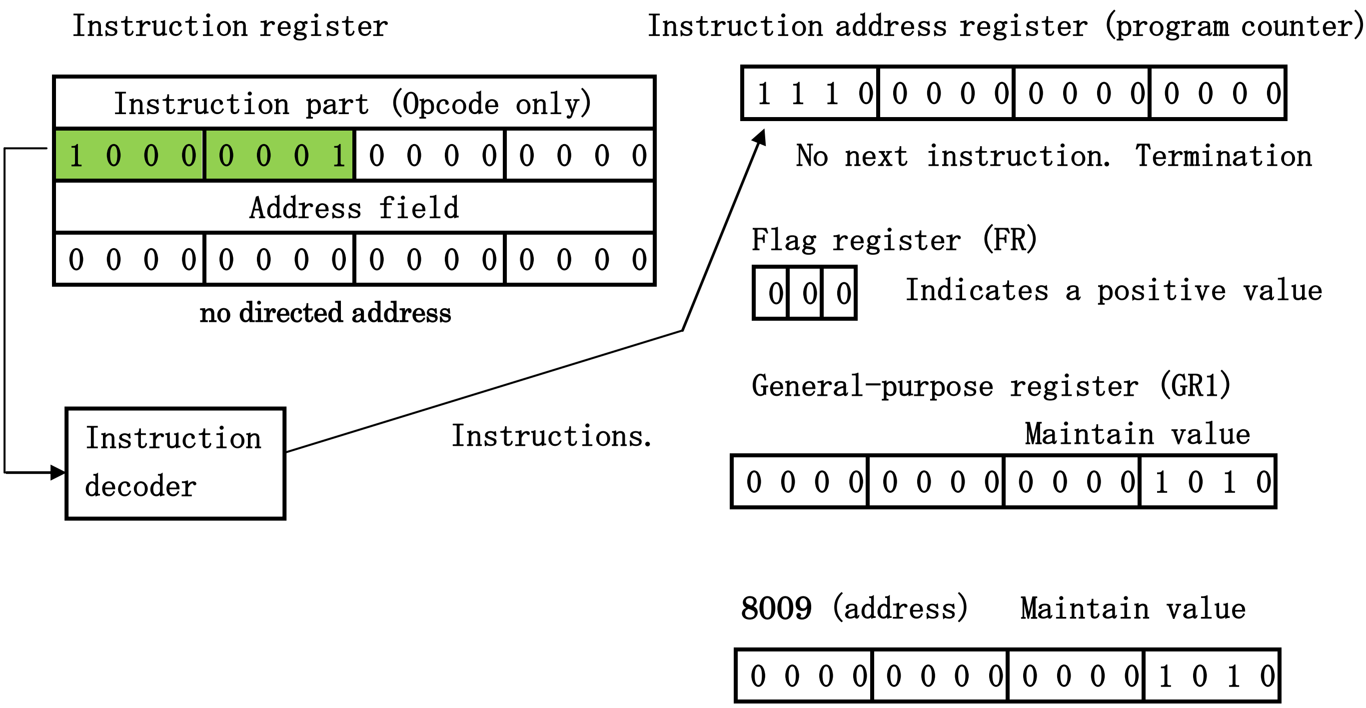
② ADDA GR1, B



③ ST GR1, ANS

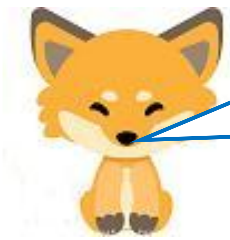


④ RET



It’s like this. Tanuki, do you understand?

I think I understand, but I don't know.
Well, I knew I was doing something complicated.



I haven’t studied the assembler language yet,
so I guess it’s not too much to ask.

Let’s move on to Episode 4.

Translated at DeepL