Machine Language&Assembly Language

-Now we have the computer: fully functional and powerful. But how come the machine knows what to do?

*-Remember the architecture of Hack computer? It adopts the Harvard Architecture,where ROM is where the instructions are stored,and RAM contains a set of registers. You may have found the **Reset** button on the surface of the machine which tells the computer to start from executing the first instruction in ROM. So put the instructions in ROM and press **Reset** and then there will be magic.*

*-Sounds great,but what on earth are the **instructions**?*

Instructions In The ROM

Back to the early days ROM can be plugged in and removed,and inside is the program we need. Let's think about it: what kind of code can the computer identify and execute?

Since you have built the CPU and known about its input,it is quite an easy question. It's the a string of 0s and 1s,and in this context it's 16 bits long.You may remember the input of the ALU,which carries out the arithematical and logical operations.It also takes a set of bits as input. Actually they are a part of the machine language,as ALU is part of the CPU.

The specification of Hack machine language is rather clear after you build the CPU,you may stop and think about that,do you think it can perform every thing?

The answer is *yes*,the machine code can perform calculation as well as jumping among codes. But also the answer is *no*,because not everything can be done by computer and therefore cannot be written as codes.What kind of problem cannot the computer solve?

A famous one is the **halting problem**,take in mind that some problems are unsolvable.Now let's continue thinking about the machine code,do you think they are easy to understand?

Not really,probably.0s and 1s are the characters computer identify,not humans.So to make it easier for people to read and write program,we need a kind of annotation to represent the underlying machine code.We called this *language* assembly language.

Assembly language

So now we decide to make an assembly language for Hack machine language,what should we consider for the design?The most simple way for this is to translate the machine code to human-readable code,one by one.In this way we can design the assembly language easily,and the amount of code in assembly equals to that of the machine code.

Sounds great,shall we adopt that?

There's no harm doing that,but it will just be another form of machine code,and nothing more will be provided for the programmer.If ,say,you are planning to writing a program for Hack,assembly makes it easy to write but still it's very annoying to do that.

So to avoid **part** of the annoyance,we decide to make the assembly language provide more useful utilities.For example,to achieve the goal of implementing a loop,we can define labels in the source code and refer to them afterwards and jump backwards or forwards.What else?

If we could define *variable* ,it will be much more convenient.But variables are more like high-level stuff and is really difficult to implement by machine code.So we choose to define alias for registers in RAM,which makes it easier to refer to a specific register there.