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 the magic.*

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

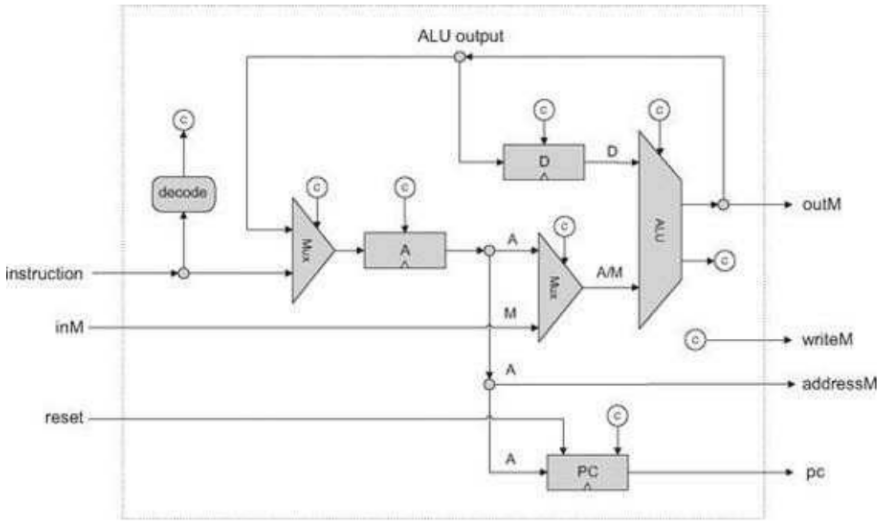
Instructions In The ROM

Language in accordance with the hardware

Back to the early days ROM can be plugged in and removed,and inside it 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 an easy question. It's 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 *whole* machine language contains also the code to jump - **by changing the content stored in PC**.

The specification of Hack machine language is rather clear after you have built the CPU and the whole Hack computer,you may **stop and think about that**,and here is the graph to make your memory come back.



You should get your hand dirty about the binary code,know about which part is manipulated by ALU and which part is used for PC.**As you can see,the underlying architecture of Hack has already proposed a kind of machine language**. We call it a *language*,so can this language express all the ideas we intend to tell the computer?

The answer is *no*,because not everything can be done by computer and therefore cannot be written down as codes. What kind of problem cannot the computer solve?A famous one is the **halting problem**,which you should take in mind as there are some unsolvable problems.

Let's summarize a bit here:Machine code is powerful for opearating the computer to carry out either arithmetical or logical operations.It takes control of the computer we built in *chapter 5* and acts as a software abstraction layer over the underlying hardware.

Why do we need assembly?

The machine language can do various things as we stated earlier,we saw in chapter 1 that **nand gate** only can build the whole hardware needed,and in chapter 2 we saw that **Add** only can be used to construct all sequential logic.**What is turing complete for a language?**

`mov` is Turing-complete

Stephen Dolan
Computer Laboratory, University of Cambridge
stephen.dolan@cl.cam.ac.uk

Stephen Dolan wrote the paper **mov is Turing-complete**,which is not a component in machine language.We have to build a more high-level and user-friendly language over machine language,because there is **no single machine code that can be used to represent the rest**.And from now on,you will start the journey through **software**.

Assembly language

Overview and Design

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.

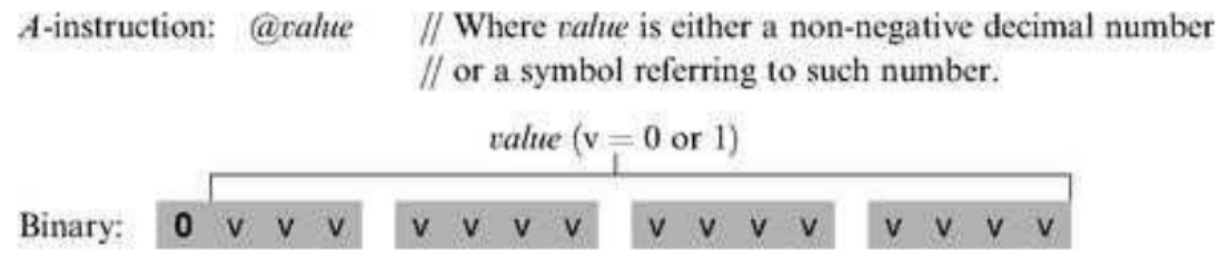
So to avoid **part** of the annoyance,we decide to make the assembly language provide more useful utilities.For example,to achieve the goal of implementing a loop,we can define labels in the source code and refer to them afterwards and jump backwards or forwards.What else?

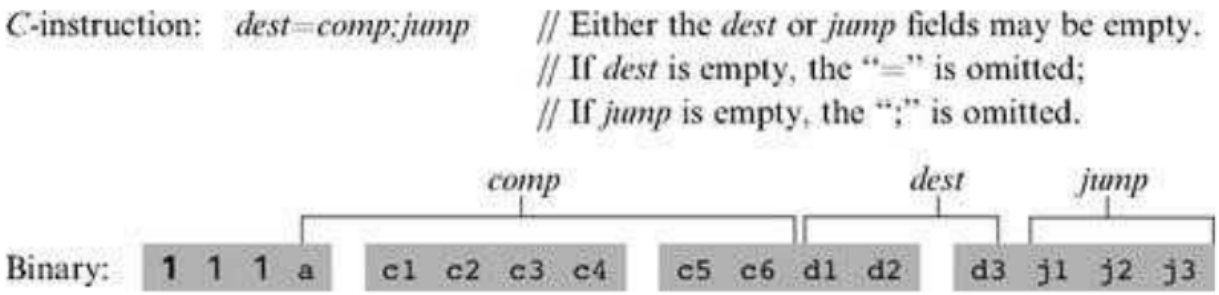
If we could define *variable* ,it will be much more convenient.But variables are more like high-level stuff and is really difficult to implement by machine code.So we choose to define alias for registers in RAM,which makes it easier to refer to a specific register there.

Adding more features is completely ok as it is what modern assembly does,but for simplicity we limit our footsteps here.

Specification

See the textbook.....





Logic Model

Background	**The general-purpose Hack computer we built earlier, and the interface designed as the ALU's input.**
Goal	Consider again about the ALU and its input,which define a set of instructions and a machine language.Design an assembly language corresponding to this set and implement the compiler,making our first step towards software layer.
Input	Content:Whole Hack Computer and the ALU I/O specification.Test assembly language scripts
Process	Review the input bit set of ALU in Hack implementation
Output	Programs written inHack assembly language
Effect	The Hack Computer can take input as assembly language and act accordingly,which leads to later design of software.