The Register Machine

I would like to demonstrate to you what ‘blind’ computation looks and feels like to the ‘outside’ observer - you. Fundamentally, computers are machines. Machines follow instructions without ‘knowing’ what they are doing. How can a machine be competent without comprehension?

The register machine was first introduced in 1957 by logician Hao Wang. Basically it is an idealised computer that is made up of a finite number of registers and a processing unit. Before I get into it here I should say that I got the idea about this machine from Daniel C. Dennett’s excellent book “Intuition Pumps and other Tools for Thinking”. Dennett is the Director for Cognitive Studies at Tufts University. He has also given us permission to use the “RodRego – Register Machine Simulation” resources at sites.tufts.edu/rodrego/. We will look at these a little later.

Now let me explain the register machine.

**Registers**

Think of registers as memory locations. You might consider these to be something like the registers in a CPU or memory slots in RAM. Each register has a unique address.

We could use any addressing scheme, but integers or letters of the alphabet will work as well as anything. So we may have labels like “Register 1”, Register 2”, “Register 3 and so on.

You might think of a register as something into which we can place countable objects. Each register may hold zero or more of these objects. The number of objects in a register will always be a non-negative integer.

**Processing unit**

Think of a processing unit as something like the control unit in a CPU. A register machine can follow instructions in a step-by-step (serial) manner. We call a sequence of these instructions a **program**. The register machine has only three instructions.

Here they are (and be mindful that x and y are just variables which represent any given integer):

1. **End**

(i.e. stop following instructions, shut down, end of program)

1. **Increment** register **x**, go to step **y**

(i.e. add an object to a given register x, and go to step y)

1. **Decrement** register **x**, go to step **y**

(i.e. remove one object from the register x and go to step y)

This defines the instruction set for the register machine.

**Register and Object Hardware**

A register could be represented by any kind of container – desk drawers, pockets, holes in the ground, bags – anything. We will use styrofoam cups, though any cup will do of course.

What are the objects to be processed? They can actually be anything but we will follow Dennett’s examples and use coffee beans. One object equals one coffee bean.

For example let’s label a cup (register) with the symbol “1” and place two beans in it then we can say that “Register 1 has 2 beans in it”. If we “Increment Register 1” then “Register 1 has 3 beans in it”. If we follow an instruction which says: “Decrement Register 1” then “Register 1 has 2 beans in it” again. Repeating this instruction again results in “Register 1 having 1 bean”. Repeating this instruction yet again results in “Register 1 having 0 beans”. Remember that “Register 1” is just a label for a register.

Since the contents of each register (cup) must be zero or more (a non-negative integer), it means we cannot decrement an empty register. You can’t remove a coffee bean from an empty cup, can you? Therefore a special action is reserved especially for case where empty registers are decremented.

By the way, can you see why register machines are sometimes called counting machines?

**Decrement is special**

If we try to decrement an empty register then we need an additional instruction defining what to do if the register is empty. The instruction would be like “decrement if you can, but if you can’t then **branch** (or go to) to another step”.

Rather than using the instruction **Decrement** we will modify it to a new one called **Decrement-or-Branch**. Its new definition is therefore:

**Decrement** register **x**, and go to step **y OR** if you can’t decrement register x then **branch (or go to)** to step **z.**

In other words “Decrement or Branch” can go to two diferent steps depending on the condition of whether it is empty or not.

**The final set of instructions**

|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Abbreviation | Example | Explanation |
| End | END | END | Stop the program. |
| Increment | INC | INC 2 1 | Increment register 2, then go to step 1 |
| Decrement or Branch | DEB | DEB 1 2 3 | Decrement register 1 and go to step 2 OR if register 1 is empty then go to step 3. |

**What can a register machine compute?**

Now, you might think that such a simple machine would not be able to do much. All you can do after all is put beans into a cup and remove them. In fact this machine can compute anything any computer can compute (just as a Turing Machine can compute anything any computer can compute).

Let’s have a go at a simple program to do addition. Keep in mind that we are computing with hardware here (as all computers must).

**Program to add two integers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Instruction | Register | Go to Step | Branch to step if register is empty |
| 1 | DEB | 1 | 2 | 3 |
| 2 | INC | 2 | 1 |  |
| 3 | END |  |  |  |

That’s it. We could also write it like this without a table:

1 DEB 1 2 3

2 INC 2 1

3 END

Dennett calls this programming language RAP (The Register Assembly Programming language).

**Setting up the hardware for addition of two integers**

We can set up many registers if we like, but this for this addition program we require only two.

Here is an image of the two registers (cups). Register 1 is labelled “1” and register 2 is labelled “2”.



Now we will initialise the registers with the two integers we want to add i.e 3 and 2. We place three coffee beans in register 1 and two coffee beans in register 2.



Note that the result of the computations will be found in register 2 when the program ends. In other words we should find the result of 3 + 2 presented in register two when the program ends.

Now we can begin the program. We start at step 1:

**1 DEB 1 2 3**

2 INC 2 1

3 END

This says “Decrement register 1 and then go to step 2 (unless register 1 is empty – which it is not)”. So we take one coffee bean from register 1.



After we remove the bean we end up at step 2, right?

1 DEB 1 2 3

**2 INC 2 1**

3 END

Step 2 says “Increment register 2 and then go to step 1”. So we add a bean to register 2.



After we add a bean to register 2 we go back to step 1.

**1 DEB 1 2 3**

2 INC 2 1

3 END

Again, we remove a bean from register 1 and go to step 2.



1 DEB 1 2 3

**2 INC 2 1**

3 END

And yet again we add a bean to register 2 and go to step 1. (Note that we appear to be in a loop between step 1 and 2.)

**1 DEB 1 2 3**

2 INC 2 1

3 END

There is one bean left in register 1. The instruction requires us to remove it and go to step 2. (Note that register 1 is now empty).



1 DEB 1 2 3

**2 INC 2 1**

3 END

Step 2 again tells us to add a bean to register 2 and go to step 1.



**1 DEB 1 2 3**

2 INC 2 1

3 END

We are now at step 1 yet again. But this time things are slightly different. There are no beans left in register 1. Recall that DEB means “decrement or branch if the register is empty”. If the register is empty (which it is) we must go to step 3.

Step 3 ends the program. We go no further.

1 DEB 1 2 3

2 INC 2 1

**3 END**

The result of the computation is in register 2. Count the beans. There are five beans, and this is the result of 3 added 2. The computations for adding two integers are finished.



**Blind mechanical computing**

There are a few important ideas involved with what we have just done. The first is that we have operated on data ‘mechanically’ by blindly following a sequence of steps. We really don’t have to think about what is happening during the process. We just have to follow the steps. We could interrupt the computations at any point in the program, insert a bookmark to remind us where we stopped, and then resume at any time in the future. It could be minutes, hours, months, years or more. But when we return to the program we merely continue from where we left off. Regardless of interruptions, the machine will always produce an exact result for the addition of two integers of any size.

The machine doesn’t understand what it is doing in any way. There is no intelligence to speak of. Yet the machine competently does the job. It doesn’t know what the numbers are. It doesn’t know what addition is. Dennett calls this “competence without comprehension”.

This machine and all similar machines (e.g. Turing Machines) are actually computers.

Note the things the machine was able to do.

* It was able to follow a **sequence** of instructions.
* It was able to **branch** to other steps.
* It was able to **loop** (or iterate).

These properties define exactly what computers can do and, importantly, no more. Programming languages, whether they are of low- or high-level can do no more than this because physical computers can do no more.

The main difference between the register machine and the computer you are using right now is the speed and capacity. Modern computers are electronic rather than mechanical, and they have a lot more registers. But in principle, a register machine can do whatever your current computer can do.

**Questions about the adding machine**

1. How many steps did it take to add 3 and 2?
2. Would it make any difference if two beans were placed in register 1, and three beans in register one? Would it take more or less steps?
3. What would happen if register 1 and 2 contained no beans (i.e. 0 + 0)? Do we get the expected result?

**Moving contents of registers**

Computers must be able to move or copy the contents of one register to another. They must also be able to ‘clean-out’ registers with content in them already. Registers need to be re-used from time to time. Here is another program which clears a register and then moves the contents of one register to another.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Instruction | Register | Go to Step | Branch to step if register is empty |
| 1 | DEB | 2 | 1 | 2 |
| 2 | DEB | 1 | 3 | 4 |
| 3 | INC | 2 | 2 |  |
| 4 | END |  |  |  |

The RAP instructions look like this:

1 DEB 2 1 2

2 DEB 1 3 4

3 INC 2 2

4 END

This program will move the contents of register 1 to register 2. Let’s assume that register 2 already has some beans in it from a previous set of compuations. It should be empty so we can move new contents to it. Therefore we want to first clear it of all beans.



As you can see from the image we have three beans in register 1. We want to move these to register 2. However we first want to clear register 2. The beans in register 2 may have been the result of some other computation. We don’t know why they are there. We don’t care. We just want them ‘cleared’.

We begin with step 1.

**1 DEB 2 1 2**

You should be able to see from the instruction that it is a loop. It decrements from register 2 and then goes to step 1 again. In other words one bean is removed each time until there are no beans left. When register 2 is empty it goes to (or branches) to step 2. The state of the machine now looks like this at the beginning of step 2:



1 DEB 2 1 2

**2 DEB 1 3 4**

3 INC 2 2

4 END

Step 2 tells us to decrement from register 1 and go to step 3 (unless it is empty, in which case it would go to step 4)



1 DEB 2 1 2

2 DEB 1 3 4

**3 INC 2 2**

4 END

Step 3 tells us to increment register 2 and go to step 2.



1 DEB 2 1 2

**2 DEB 1 3 4**

**3 INC 2 2**

4 END

Now we are back to step 2 again. We take one bean from register 1 and go to step 3 which in turn adds a bean to register 2. We did two steps here, ok?

Register 1 is now empty and we go back to step 2 for the final time. Since register 2 is empty we go to step 4 which ends the program.



As you can see the three beans from register 1 have been moved to register 2.

This program will work (blindly) every single time no matter how many beans need to be moved.

**Where to now?**

We could go on a create programs for copying, subtracting, multiplying, dividing, and so on. Indeed we could onto create all the operations required by a sophisticated operating system.

The register machine has served its purpose to demonstrate in a clear and simple way how computers compute.

Can you create a program to copy beans in Register 1 to Register 2? You may need additional registers for temporary storage.

Let’s now move onto a real computer program which you can install on your computer to run the RAP commands described earlier in this document. This may give you a clearer picture of what is happening with the instructions and the contents of the registers.

**Rodrego**

You will find Rodrego at sites.tufts.edu/rodrego (courtesy of Daniel C Dennett and Tufts University. You should also find it installed in your computer lab.

Once installed Rodrego looks like this:



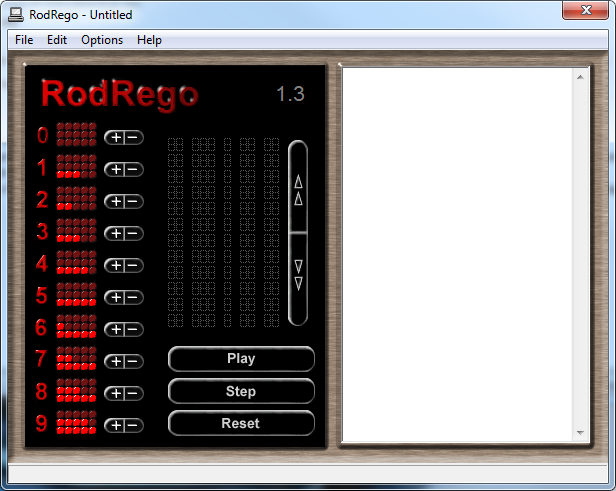
Note the two panels. The left-hand-side panel contains Registers 1-9. The bright red dots represent coffee beans. This represents the hardware (though only in a virtual way).

The right-hand-side represents the processor. This is where you type the instructions.

Lets’s go through the Addition of two integers program we did with cups and coffee beans earlier.

We will use Register 1 and Register 2 only, and ignore all the others.

First we will put three beans in Register 1 and two in Register 2. You can do this by clicking on adjacent beans or using the **+/-** button next to the given register. It should look like this:

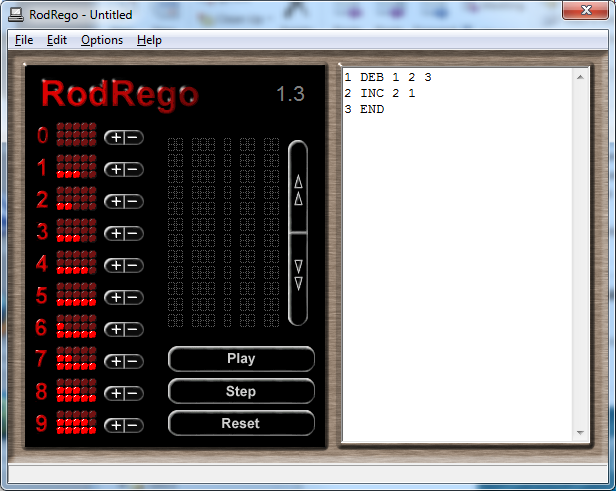


Second we type the RAP instructions into the right-hand-side panel. This example uses the exact instructions used earlier in this document.

1 DEB 1 2 3

2 INC 2 1

3 END



The final step is to execute the program. Click the “Play” button to execute the instructions. The instructions will commence executing and stop when the program ends.

After the program completes you need to “Reset” the processor. You will also need to allocate the correct number of beans again.

You may wish to also try the “Step” option which carries out each instruction according to when you click “Step”.

Try it.

Try using a different number of beans. Then try the “moving” program discussed earlier.

**Can you write a program to copy beans from register 1 to register 2? (You will need more than 2 registers for this one).**

What happens when too many beans are added to a register?

Register Machines are similar to Turing Machines but easier to understand and manipulate. Whereas Turing Machines are abstract mathematical devices, Register Machines are closer to the kind of programming we see with low-level hardware languages such as assembly. Assembly languages are very close to the hardware. They tell the hardware what to do – unambiguous step by unambiguous step. They are symbolic (written) instructions directly interpreted by hardware. High-level computer languages on the other hand (such C/C++, Java or Python) compile down (translate) to assembly-like languages often via intermediate languages and virtual machines. Assembly languages are machine languages.

I hope you have found the Register Machine interesting, and have gained some insights into what it means to compute at a fundamental level.

F. Klesch