TI-83+ Z80 ASM for the Absolute Beginner

LESSON TWO:

- *Of Mice and Men*

OF MICE AND MEN

We're not doing any programming today, but take out a pencil and some paper. We're going to do a little brain teaser, aka a brain warm-up, and I ask that you do it without using any calculator. Do all the work on paper, every single step. If you understood my introduction to this course, you understand that nothing I ask you to do is without reason. Doing these problems by hand is important to understanding this next lesson I have for you.

So, solve the following problems by hand, showing all your work. The answers are on the next page!

1. 3861 * 2534
2. 1011 + 3267 + 9219 + 4235 + 6712
3. 6917 – 3211 – 1992
4. 241164 / 252

Answers:

1. 9783774
2. 24444
3. 1714
4. 252

Excellent! Pat yourself on the back, grab a chocolate bar, tear that paper to shreds, and throw it away. I mean it. Go ahead, take a break from the tutorial and toss that paper in the trash. Maybe you can score points by crumbling it into a ball and throwing it in the trash can as if you're tossing a basketball.

Finished? Now let me ask you: how many computations did it take you to solve this problem, no matter how small? **MORE THAN 100.** Can you remember any of them, without running to the trash can? I'll give you a hint: for the second problem, when you had to add the first line, the ones place, you added 1 + 7 + 9 + 5 + 2. In other words, you had to add 1 + 7, then add 8 + 9, then add 17 + 5, then add 22 + 2. That's 4 computations.

But can you remember ALL of them? If you grab your shredded pieces of paper and try to put them together to answer this question, you're missing the point. I'm hoping your answer is "No, I can't remember all of them." And there's no problem with that. Is it easy for you to remember all these minor computations? No. Did I ask you to count all of these computations? No, I only asked you to solve four math problems, and you did so.

But were those computations important in solving the problems? Yes, Yes, YES!! Yet, once you solved the problem, you didn't remember any

of those computations. They were forgotten. They were only needed for solving the problem, so your brain didn't waste any storage space. You only needed permanent brain space for storing the answer to the four problems.

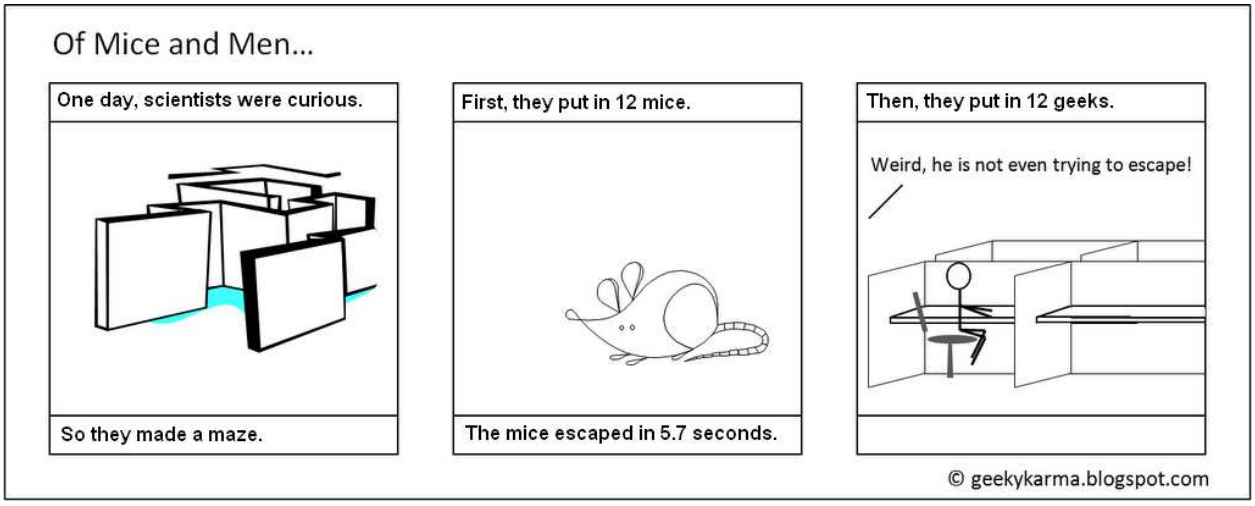
I repeat, your brain did not store any of these 100 computations that you needed. Instead, they were temporary. When your brain was processing the problem that you were solving, it had a temporary storage area—called working memory—needed to solve a particular problem. Working memory holds and relates a variety of stored information during tasks such as talking, reading, math problems, and recognizing objects. The brain cannot solve the problem in long-term memory, as the memory is meant for storage, not for solving problems.

READ THIS NEXT PARAGRAPH CAREFULLY UNTIL YOU UNDERSTAND THIS. When you solved 1+7, you needed to remember that it equals 8, and so your brain temporarily remembered the number "8" in its working memory. If you did not remember that 1 + 7 = 8, you could not solve the next step in the problem, 1 + 7 + 9 = 8 + 9. Thus, your brain remembers the number 8. Then it helped you take the number "8" and add 9 to it. Afterwards, you had to remember the number "17", and so on and so forth. However, when you discovered that 1 + 7 + 9 + 5 + 2 = 24, and when you wrote down "4" and carried the "2", you no longer needed to remember 1 + 7, 8 + 9, etc. and your brain focused on the next aspect of the problem in its working memory.

Finally, when you solved all of problem 2, your brain returned 24444, and if this problem was needed for a test, you would have been able to get the answer instantly: 24444 was *permanently* stored in your brain as a solution.

So, just to review, you saw problem 2. As you solved this problem, your brain did around 18 computations to solve the problem. The computations were put in "working memory," temporary memory. You needed these computations to solve the problem, but you did not need to remember them. When you got the answer, which was important to remember, it was not stored in working memory. It was stored more permanently.

So why do I call this "Of Mice and Men?"



Because now it is time to take about a very important part of ASM programming. It's just a cute and attention-attracting way of saying "Of RAM and Registers."

Assuming you know Ti-Basic, you know what RAM is. No need to remind you as the reader that the calculator has 24 KB of RAM you use for storage. This stores pictures, programs, variables, anything you want it to store. And it's permanent, until you take the batteries out or until you need to reset the RAM.

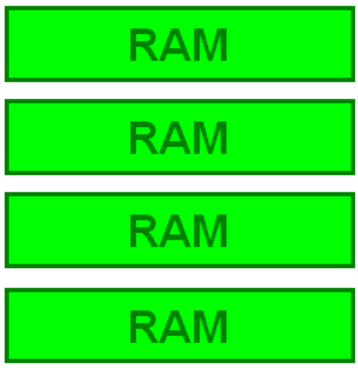
But when you run a program, or even solve a simple problem, it is not solved in RAM. There is very little that the calculator's CPU can do directly with RAM. Just like the brain's long-term memory is meant for storage and not for solving problems, the calculator's RAM is meant for storage and not for solving problems. The CPU cannot use RAM to solve multiplication problems, plot a graph, or draw a picture; it can only save the answers to these problems in RAM.

So then, the question to ask is, where DOES the CPU solve the problem? However does it remember the computations it needs to solve the first part of problem 2, 1 + 7 + 9 + 5 + 2? The calculator's processor has its own "working memory," its own temporary storage area. The processor has some very special ram used for holding the numbers it needs to solve the problem. As you'll see in the following diagram, these special pieces of ram that make up the "working memory" are called **registers**.

Let's suppose the calculator is asked to solved 1 + 7 + 9 + 5 + 2. The diagram on the next page shows that the process is not unlike that of the brain. The final answer, 24, is stored in ram for long term storage, since we are asking the calculator to solve 1 + 7 + 9 + 5 + 2.

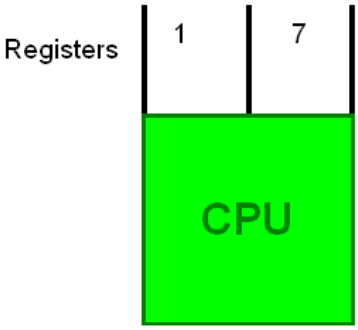
1. The processor reads the problem

1 + 7 + 9 + 5 + 2

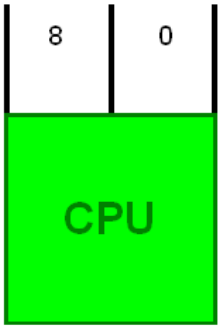


2. The processor stores the number "1" into one of its registers.

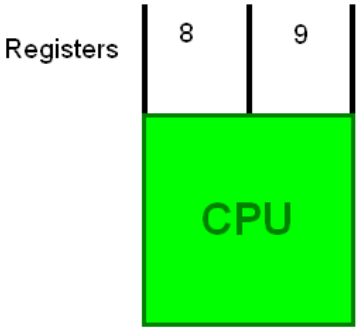
Also, the number "7" is put in another register.



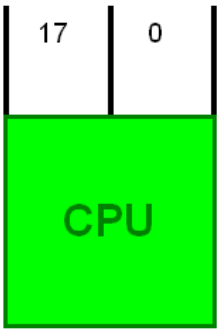
3. Just like your brain needs to remember that 1 + 7 is 8 to continue the rest of the problem, the CPU adds 7 to 1 in its register, so it can remember the number "8" to add the next number, "9", to.

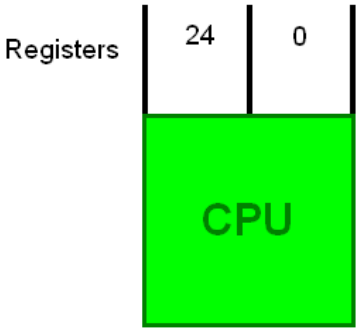


4. The processor stores the number "9" into one of its registers.



5. The calculator adds 8 and 9. It remembers the number 17, needed it to continue the problem.





6. Continuing, the calculator recieves the final solution, 24. Having completed the answer, the solution is stored in RAM.

As you can see, just like your brain's working memory, the registers hold numbers temporarily so that the CPU can operate on them to arrive at the final solution. However, it only holds them temporarily. The registers of a CPU are meant for solving a problem and carrying out a task. They are NOT meant for permanent storage. When you solve 8 + 3, your calculator's RAM remembers the answer "11." When you solve 6 + 10, your calculator's RAM remembers the answer "16." Thus, your calculator remembers both the answer 11 and 16. However, the registers do not remember the numbers 8, 3 and 11. The registers hold the number 16, part of the last problem you solved. Now, let's say you solve an equation. Then the registers will not remember anything related to the problem 6 + 10.

This is one of the most important lessons you will learn in ASM. You use RAM to hold more permanent storage, but you cannot use it to solve a problem. The processor solves the problem, but it uses registers to hold the memory it needs to solve the problem. Then the answer leaves the CPU and is stored in RAM. If you don't understand this, be sure read the tutorial again.

With this in mind, in ASM programming, you'll be working directly with the processor. Therefore, you'll be working with the registers. Some of the ASM programming you'll be doing is storing to and retrieving data from RAM, but most of the work you'll be doing is computational and functional work inside the processor itself. For example, if you work on a game where you control a ship that moves X and Y forward, you'll place the values from RAM into the registers, tell the processor to increase the values in the registers to create a new X, Y position (since you cannot tell the processor to do this to RAM directly), and then store the new positions in RAM. I cannot begin to overstate how much faster this will run in ASM than in Ti-Basic.

Next tutorial, we'll take a trip to Sesame Street—can you tell me how to get to Sesame Street—and then we'll begin programming in Tutorial #4.