Simpletron Computer Simulator

Reference Manual

Version 1.10
Updated January 20, 2018

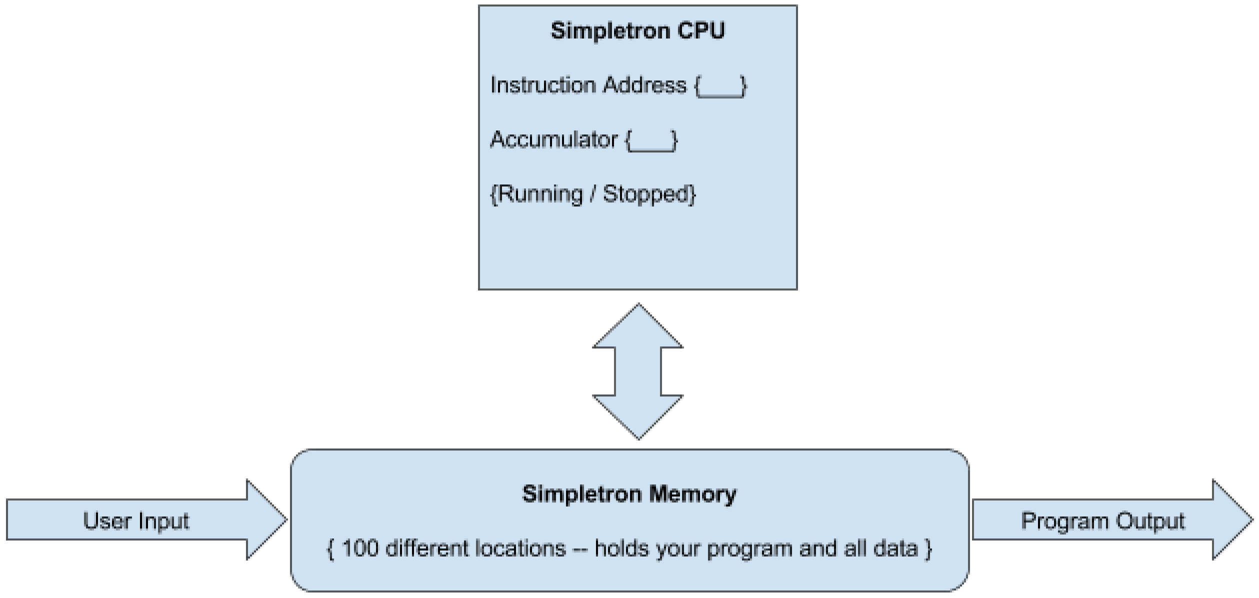
Table of Contents

Introduction

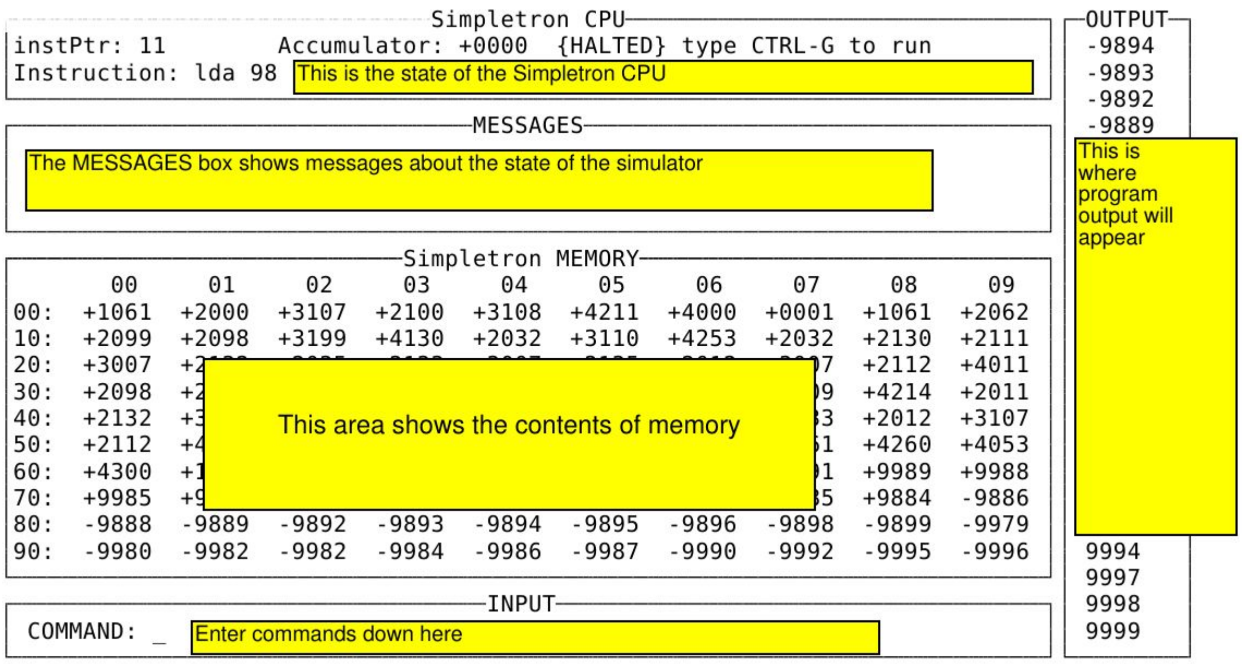
The Simpletron is a very simple computer. This program simulates the Simpletron computer on your machine. You are able to enter and run Simpletron programs using the SML (Simpletron Machine Language) and SAL (Simpletron Assembly Language). There are also special commands that you can use to control the simulator itself, which can change how the Simpletron runs.

Your basic Simpletron has a few important parts.

1. CPU: This is the “brain” of the Simpletron and runs the programs. It also does all the calculations. This keeps track of what location of memory the next command is located at. It also keeps track of all the results of any mathematical operations that it is asked to do. It can be “Running” or “Stopped”.
2. Memory: This is a special area holding 100 different numbers. The numbers can range from -9999 to 9999. This is a very important part of the Simpletron. Your program and all its data will be stored here.
3. User Input: This is a way to get a number from the person using the Simpletron. The number is stored in a memory location.
4. Program Output: The contents of a memory location is displayed to the person using the Simpletron.



The Simpletron Simulator Screen



```
-------------------------------Simpletron CPU-----------------------------   --OUTPUT--
instPtr: 11        Accumulator: +0000  {HALTED} type CTRL-G to run            -9894
Instruction: lda 98                                                           -9893
                                                                              -9892
----------------------------------MESSAGES--------------------------------    -9889



-----------------------------Simpletron MEMORY----------------------------
       00     01     02     03     04     05     06     07     08     09
00:  +1061  +2000  +3107  +2100  +3108  +4211  +4000  +0001  +1061  +2062
10:  +2099  +2098  +3199  +4130  +2032  +3110  +4253  +2032  +2130  +2111
20:  +3007  +2                                            7  +2112  +4011
30:  +2098  +2                                            9  +4214  +2011
40:  +2132  +3                                            3  +2012  +3107
50:  +2112  +4                                            1  +4260  +4053
60:  +4300  +1                                            1  +9989  +9988
70:  +9985  +9                                            5  +9884  -9886
80:  -9888  -9889  -9892  -9893  -9894  -9895  -9896  -9898  -9899  -9979
90:  -9980  -9982  -9982  -9984  -9986  -9987  -9990  -9992  -9995  -9996   9994
                                                                              9997
------------------------------------INPUT---------------------------------    9998
  COMMAND: _                                                                  9999
```

This is the state of the Simpletron CPU
The MESSAGES box shows messages about the state of the simulator
This area shows the contents of memory
This is where program output will appear
Enter commands down here

Example of what the Simpletron looks like when running

There are 5 parts of the Simpletron Simulator display.

- Simpletron CPU
  - This box shows the Simpletron CPU.
    - <u>instPtr</u>: The memory location for the next Simpletron command.
    - <u>Accumulator</u>: The result of the last mathematical operation.
    - <u>HALTED</u> / <u>RUNNING</u>: Says if the Simpletron is running or stopped (halted).
    - <u>Instruction</u>: The actual command the Simpletron will run next.
    - <u>Version Number</u>: This is the version number of the simulator.
- Messages
  - This box shows messages from the simulator. These can be error messages. This is where you can see if something went wrong.
- Simpletron MEMORY
  - This shows you the contents of the Simpletron Memory. If you want to know what your program looks like, to the Simpletron CPU, this is will show you. You can also see where a program is storing data and how the memory is used by the program when it runs.
- INPUT
  - This box is where you enter commands or program input.
    - If the Simpletron is running, this will show “INPUT“ as a prompt. You can still enter simulator commands, but the program might be waiting for a number from the user.
    - If the Simpletron is not running, it will show “COMMAND“. You can’t enter input numbers in this state. But, you can change memory values and control the simulator.
- OUTPUT
  - This box will show the last few numbers that were output by the program. The newest number will be on the bottom.

The Simpleton CPU and Memory

Simpletron Math and the Accumulator

The accumulator is a special spot inside the CPU where the answer to the last math operation is kept. It can hold numbers from -9999 to 9999. It can also be used to control how your program works. Your program can change the next instruction based on the value of the accumulator. You can't show the value in the accumulator directly to the user (although this simulator will have it visible to you). You will need to send it to memory before it can be output to the user.

The accumulator is set to 0000, when the simulator is started.

Watch out! If you want to save the value of the accumulator, you should store it in a memory location before you do any other math or load any values to it. If you don't save it to memory, there is no way to get the old value back.

All Simpletron math is integer math. This means there are no fractions or decimal points. When you divide on the Simpletron, the remainder is always thrown out.

- 8 ÷ 3 = 2
- 7 ÷ 3 = 2
- 6 ÷ 3 = 2
- 5 ÷ 3 = 1

As you can see, the remainder is always thrown out. It doesn't matter how big it is. The whole number part of the quotient is all that remains in the accumulator.

Math operations never change the value in memory. The result of the operation is stored in the accumulator. If the value is bigger than 9999, it restarts counting at zero. 9999 + 1 = 0, 9999 + 2 = 1. If the value is smaller, the same thing happens. -9999 - 1 = 0. -9999 - 2 = -1. The same thing happens with multiplication. It is not possible for division to cause the numbers to go higher or lower than the limits.

Watch out! You can't divide by zero. It will crash the Simpletron.

Good idea! If you are dividing by a number that could be zero (maybe the user entered the number or maybe it is the result of another math operation), you should check the number first. You might choose to halt the machine or use a different number for the division instead.

The Simpletron Instruction Pointer

The Simpletron Instruction Pointer is a very important part of the CPU. It tells you where the CPU is looking in the memory. The lowest number it can hold is 00. The highest number it takes is 99. Your programs won't be able to know what value the instruction pointer has, but they can change the value by using jump, jneg, or jzero. If you do not use one of those commands, the instruction pointer increases by 1 after every command.

The instruction pointer is set to 00 when the simulator starts.

The Simpletron Memory

The Simpletron has 100 words of memory. They start at 00 and go up to 99. Each word of memory can hold values from -9999 to 9999. When a user enters a number, it will be stored in a memory location. If you want to show a value to your user, it must be in a memory location before you try and output it. Your program must also be stored in the memory. You might also need some memory to hold the results of calculations your program does. It is very important, but also very limited.

All of the memory addresses start at 0000 when the simulator starts.

Simpletron Assembly Language

How to program the simulator

The Simpletron takes very simple commands. Each command has two parts. The first part is a word that describes what the command does. The second part is a memory address that the command will work with. For example: "add 19" tells the simpletron that we are going to add whatever is at memory address 19 to the current value in the accumulator.[1]

The following is a very simple program. It will read two numbers from the user, add them together, and print the sum.

Read 99	Get a number from the user and save it to memory location 99
Read 98	Get a number from the user and save it to memory location 98
LDA 99	Load the Accumulator with the value from memory location 99
Add 98	Add what is in memory location 98 to the accumulator
STA 97	Store the Accumulator's value in memory location 97
Write 97	Write the contents of memory location 97 in the output window
Halt	Halt the Simpletron CPU, because we are done

Watch out! The Simpletron commands always take a memory location and not a number. Add 1 does not add the number 1 to the accumulator. It will add whatever number is currently saved at memory location 01.

Good idea! Most programs start at location 00 and they fill up the bottom locations in memory. It is a good idea to save user input to the highest location (99) first and then move down in memory locations. This will prevent you from accidentally having a user erase one of your commands.

[1] The truth is, the Simpletron, doesn't understand any words at all. When you enter a command, it is turned into a number that tells the Simpletron what to do. The number 3000 tells the Simpletron that you want it to add. The address is then included in the number. So 3012 means the same thing as "add 12".   If you watch the memory window, carefully, when you enter commands, you will start to see the patterns for how the Simpletron commands are stored.

Complete Command Set[2]

- ★ Add {location}
  - ○ Adds the value saved at {location} to the accumulator
- ★ Sub {location}
  - ○ Subtracts the value saved at {location} from the accumulator
- ★ Mult {location}
  - ○ Multiplies the accumulator by the value saved at {location}
- ★ Div {location}
  - ○ Divides the accumulator by the value saved at {location}
- ★ Jump[3] {location}
  - ○ Go to {location} before getting the next command
- ★ Jneg {location}
  - ○ If the accumulator is less than 0, go to {location} before getting the next command. If the accumulator is 0 or positive, don't do anything
- ★ Jzero {location}
  - ○ If the accumulator is 0, go to {location} before getting the next command. If the accumulator is anything else, don't do anything
- ★ Halt
  - ○ Stop the chip from running. This is the only command that does not take a memory location.
- ★ Read {location}
  - ○ Reads a number from User Input and saves it at {location}
- ★ Write {location}
  - ○ Takes the number at {location} and sends it to Program Output
- ★ Lda[4] {location}
  - ○ Takes the number at {location} and puts it into the accumulator
- ★ Sta {location}
  - ○ Takes the number in the accumulator and saves it to {location}

[2] There are a some extra commands the Simpletron understands that are not listed here. The Simpletron simulator understands them, and you can use them if you can figure them out. But, they are not supported in all Simpletron Computers. The commands listed above will always be supported by a Simpletron computer. They might not have the same "words" but they will have the same number codes.
[3] Many Simpletrons use the words BRANCH, BRANCHNEG, and BRANCHZERO for the jump, jneg, and jzero words (respectively). This simulator uses the word jump and its derivations because it is shorter and a bit easier to understand.
[4] LDA is short for "Load Accumulator" and STA is short for "Store Accumulator". The names come from a chip called the 6502. It was used for a very popular game console and many other devices. It was such a popular chip, the abbreviations became common.

After you enter one of the commands above, the simulator will turn it into the correct value and save it to where the instruction pointer is currently looking in memory. The instruction pointer will then be increased by one.

How to control the simulator

The simulator is a program itself. Because you don't have a real Simpletron, the simulator has some extra commands that allow you to control the fake computer. Without a power button to push, you need a way to start and stop the machine from inside the program. Without a disk drive, you need a way to get the programs you write into memory. These commands are all specific to this simulator. Other Simpletron computers might not be able to do what this simulator does.

Simulator Commands

- Go
    - The instruction pointer is set to 00, and the CPU starts to run. You can also type "g" all by itself or press <control>-G to start the CPU.
- Stop
    - The CPU is stopped. You can also press <control>-C
- Step
    - Execute one command, then stop. You don't need to type the whole word, you can just type "s" by itself.
- Set {number}
    - Put a value into the accumulator
- Break {location}
    - If the CPU gets to {location} when running, stop running.
- Continue
    - Start running from the current location, useful after your program stops for a break or if you stopped the program while it was running. You can also type "cont" instead of the whole word.
- Clear {location}
    - Undo the break command, so the CPU no longer stops at {location}
- @{location}
    - Move the instruction pointer so that it is pointing to {location}
- #
    - The # and everything after it on the line is ignored. This is useful if you are writing programs in a file and want to make notes or comments.

- {number}
    - A number all by itself is written to the location the instruction pointer is at. The instruction pointer is then increased by one.[5]
- Dumpmem {filename}
    - This will take the current contents of memory and write them to a file named {filename}. If you have a working program and want to make sure you don't lose your work, this is a good way to save it. You can also save the memory to a file and then edit it using a text editor. The file created with dumpmem can be read directly back in with restoremem.
- Restoremem {filename}
    - This will read the contents of {filename} into memory. The file can contain the numbers from a previous dumpmem. It can also contain any simulator or assembly command (except commands that operate on files). Each line of the file will be processed like you had typed it directly into the simulator. Anything that works as a normal simulator command should work from the file.
- Dumpstate {filename}
    - This will take the current state of the Simpletron and save it to a file. It will have the values of the Accumulator, instruction pointer, and a descriptive memory table. This file can't be read back into the simulator using the restoremem command. It is useful if you have a problem and want to look at the entire state of the Simpletron at another time. This does not save the output or the input. There is no way to save that to a file.
- Debug
    - This is similar to dumpstate, but it doesn't save it to a file. Instead, it will show it on the screen when you quit the simulator.
    - If you change your mind, use "nodebug" to turn this option off.
- Dumpprofile {filename}
    - This will dump the information collected when program profiling is turned on. This is the only way to see the profiling information collected when profiling is turned on.
- Profile
    - This will turn on profiling, and reset all associated values. This will keep a log of how often each command is run, where the instruction pointer spends spends the most time executing, and how often memory addresses are accessed.

[5] If you know the correct number codes, you can type the number instead of the assembly commands. For example, you could type 4300 instead of the word "halt". The numbers are the real way the CPU is programmed and controlled.

- Noprofile
  - This will turn profiling off. Any information collected is not changed.
- Resetprofile
  - This will reset all profiling information back to the initial state.
- Reset
  - Restart the simulator. This will change everything back to the way it was when you first started the simulator. Everything will be set back to 0. This does not clear the output.
- Wipe
  - Clear the output window. If you combine with with the reset command, the simulator will be back to the original state it started in.
- Quit
  - Close the simulator. You can just type “q” by itself to quit.

Watch out! The commands “dumpmem” and “dumpstate” will copy over any file that is already there. If the file already exists, you will be prompted to type ‘y’ to confirm that you want to write over the old file. If you type any other key, it will cancel the command. You should be careful when using this command.

Good idea! You can always make sure you use a descriptive name every time you use the commands “dumpmem” or “dumpstate”. If you name the file “addfractions” you will know what it does a lot quicker than if you name it “program1”.

Challenge Programs

Below, you will find some challenge programs. Can you write a program on the Simpletron that solves these programs?

- Add it up
  - Write a program that reads in three numbers from the user and outputs the sum.
  - Write a program that reads 10 numbers and sums them up.
  - Write a program that keeps reading numbers until the user enters a negative number, and outputs the sum of all the positive numbers.
  - Can you change the last program so that it prints the average of the numbers instead of the sum? The average would be the sum divided by how many numbers there were.
- I’m a Product of my environment
  - Write a program that reads in two numbers and outputs the product.
  - Write a program that reads a positive number greater than 1 and adds up all the multiples of that number below 200.
- Decisions, Decisions
  - Read two numbers. If both numbers are negative, or both numbers are positive, subtract the first number from the second. Output the difference. If one number is positive and the other is negative, add the numbers and output the sum.
- Give me the biggest you’ve got!
  - Write a program that reads in two numbers from the user and outputs the largest number.
  - Write a program that reads in five numbers from the user and outputs the largest number.
  - Change both the above programs so they output the smallest number instead.
- We can sort it out
  - Read three numbers, output them in ascending order.
  - Read five numbers, output them in ascending order.
  - Read 10 numbers, output them in ascending order.
  - Change the above programs to output the numbers in descending order.

➢ All that remains

- Write a program that reads two numbers. Divide the first number by the second and output the quotient.
- Write a program that reads two numbers. Divide the first number by the second and output the remainder.
- Write a program that reads two numbers greater than 1 and divides them. Round the quotient to the nearest whole number. 32÷5=6, 33÷5=7. Output the rounded quotient.
- Write a program that reads two numbers greater than 1 and divides them. Output the whole number quotient and then output the first 4 decimal digits. 32÷6= 5 3333 and 32÷5= 6 4000.
- Write a program that reads a number. If the number can be divided by 7, output the quotient. If the number can't be divided by 7, output zero.
- Write a program that reads two numbers. If the first number can be divided by the second, output the quotient. If it can't, output zero.
- Write a program that reads a number greater than 1 and outputs the original number, if it was prime. Otherwise output zero.
- Write a program that reads two numbers greater than 1 and counts how many prime numbers are between them.
- Write a program that reads two numbers greater than 1 and outputs the sum of all the primes numbers between them.
- Write a program that reads one number greater than 1. Output the prime factorization of that number. The prime factorization is the list of prime numbers that you can multiply together to get the number. For example, 24 is 2, 2, 2, 3 because 2 x 2 x 2 x 3 = 24. Note, if the number is smaller than 2 then you need to output zero. There are no primes under 2.

➢ Something Useful

- Write a program that reads two nonzero numbers and outputs the greatest common divisor.
- Write a program that first reads in a numerator and then reads in a denominator. Output the numerator and then the denominator of the fraction in simplest form.
- Write a program that reads in a numerator then a denominator of a fraction, then reads in a numerator and denominator of a second fraction. Multiply them together, and output the numerator and then denominator of the product.
- Write a program that reads in a numerator then a denominator of a fraction, then reads in a numerator and denominator of a second fraction.

Add them together, and output the numerator and then denominator of the sum.

- Write a program that reads in a numerator then a denominator of a fraction, then reads in a numerator and denominator of a second fraction. Subtract the second from the first, and output the numerator and then denominator of the difference.
- Write a program that reads in a numerator then a denominator of a fraction, then reads in a numerator and denominator of a second fraction. Divide the first by the second, and output the numerator and then denominator of the quotient.
- Change the above programs to output the answer in simplest form.
    - NOTE: None of the fraction problem inputs should be zero. If you have space, it would be good habit to check the denominator.

Simpletron Machine Language

- ★ 10xy -- Read {location}
  - ○ Reads a number from User Input and saves it at {location}
- ★ 11xy -- Write {location}
  - ○ Takes the number at {location} and sends it to Program Output
- ★ 20xy -- Lda {location}
  - ○ Takes the number at {location} and puts it into the accumulator
- ★ 21xy -- Sta {location}
  - ○ Takes the number in the accumulator and saves it to {location}
- ★ 30xy -- Add {location}
  - ○ Adds the value saved at {location} to the accumulator
- ★ 31xy -- Sub {location}
  - ○ Subtracts the value saved at {location} from the accumulator
- ★ 32xy -- Div {location}
  - ○ Divides the accumulator by the value saved at {location}
- ★ 33xy Mult {location}
  - ○ Multiplies the accumulator by the value saved at {location}
- ★ 40xy -- Jump {location}
  - ○ Go to {location} before getting the next command
- ★ 41xy -- Jneg {location}
  - ○ If the accumulator is less than 0, go to {location} before getting the next command. If the accumulator is 0 or positive, don't do anything
- ★ 42xy -- Jzero {location}
  - ○ If the accumulator is 0, go to {location} before getting the next command. If the accumulator is anything else, don't do anything
- ★ 4300 -- Halt
  - ○ Stop the chip from running. This is the only command that does not take a memory location.