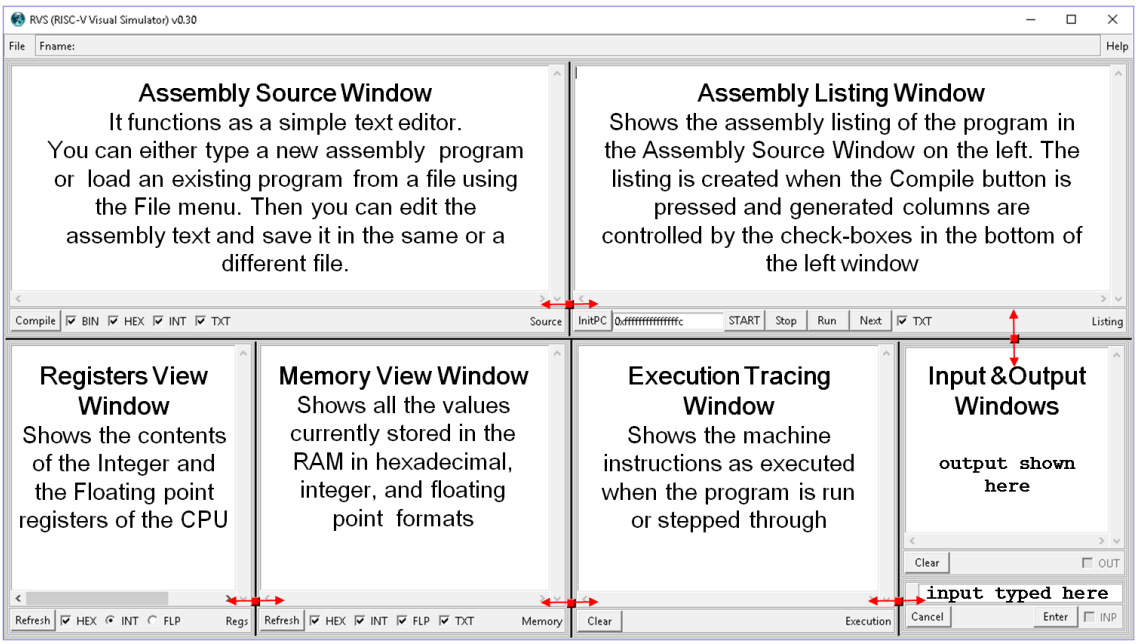
RVS (Risk-V Visual Simulator)

User Manual v0.04

1. Outline

The Risk-V Visual Simulator (RVS) provides a convenient Graphical User Interface (GUI) input, editing, assembly, and execution tracing of RISC-V assembly programs. The GUI consists of six windows as shown below.



The area of each window is freely adjustable by moving the five handles shown in red in the above snapshot as indicated by the arrows superimposed on them. The startup window appears as shown in the following snapshot.

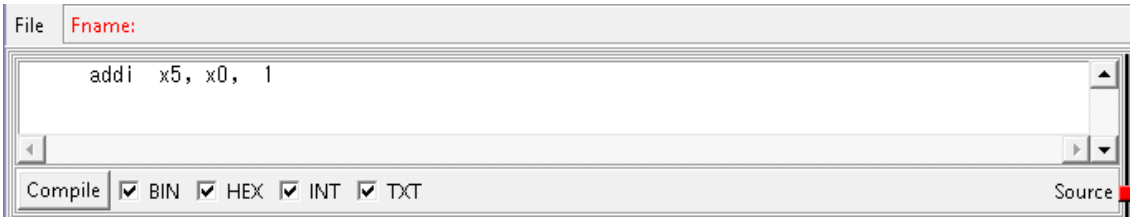

RVS (RISC-V Visual Simulator) v0.30
File
Fname:
Help
ASSEMBLY LISTING
ADDRESS BIN/HEX CODE HEX SOURCE
SYMBOL TABLE
0x00000000ffffffc START
Compile
BIN
HEX
INT
TXT
Source
InitPC
0xfffffffffffffffc
START
Stop
Run
Next
TXT
Listing
x0 zero 0x0000000000000000 0
x1 ra 0x0000000000000000 0
x2 sp 0x0000000000000000 0
x3 gp 0x0000000000000000 0
x4 tp 0x0000000000000000 0
x5 t0 0x0000000000000000 0
x6 t1 0x0000000000000000 0
x7 t2 0x0000000000000000 0
x8 s0 0x0000000000000000 0
x9 s1 0x0000000000000000 0
x10 a0 0x0000000000000000 0
x11 a1 0x0000000000000000 0
x12 a2 0x0000000000000000 0
x13 a3 0x0000000000000000 0
x14 a4 0x0000000000000000 0
x15 a5 0x0000000000000000 0
x16 a6 0x0000000000000000 0
x17 a7 0x0000000000000000 0
x18 s2 0x0000000000000000 0
x19 s3 0x0000000000000000 0
MEMORY
ADDRESS HEXADECIMAL INTEGER
Clear
OUT
Cancel
Enter
INP
Refresh
HEX
INT
FLP
Regs
Refresh
HEX
INT
FLP
TXT
Memory
Clear
Execution

2. The Source

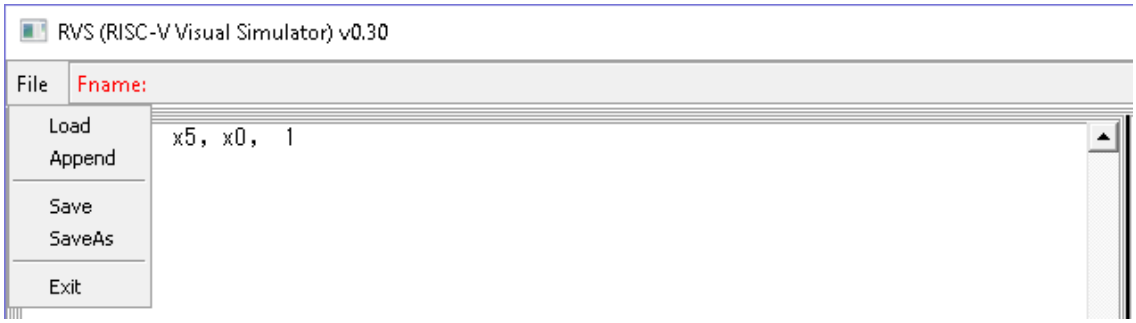
To start with, type in the **Source** window the following simple RISC-V assembly instruction:

```
addi  x5, x0,  1
```

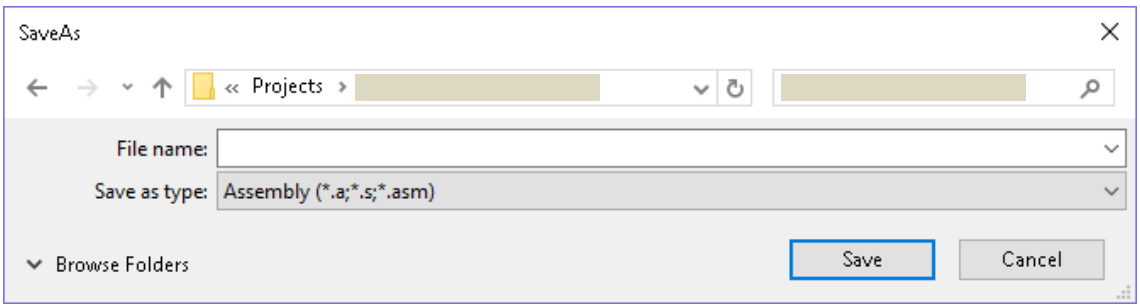
The resulting **Source** window is shown on the following snapshot.



Note that the **Fname:** string next to the **File** menu has turned red. This change in color indicates that you have unsaved changes of the text in the **Source** window. You can save the text from the **Source** window into a file by clicking on the **File** menu as shown below.

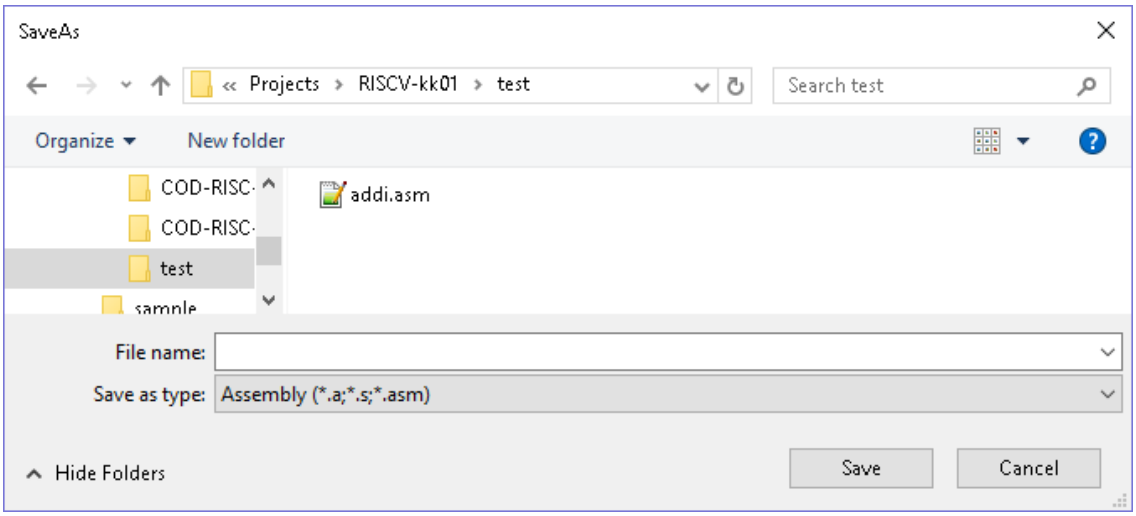


Then select **SaveAs** to obtain the following dialog.

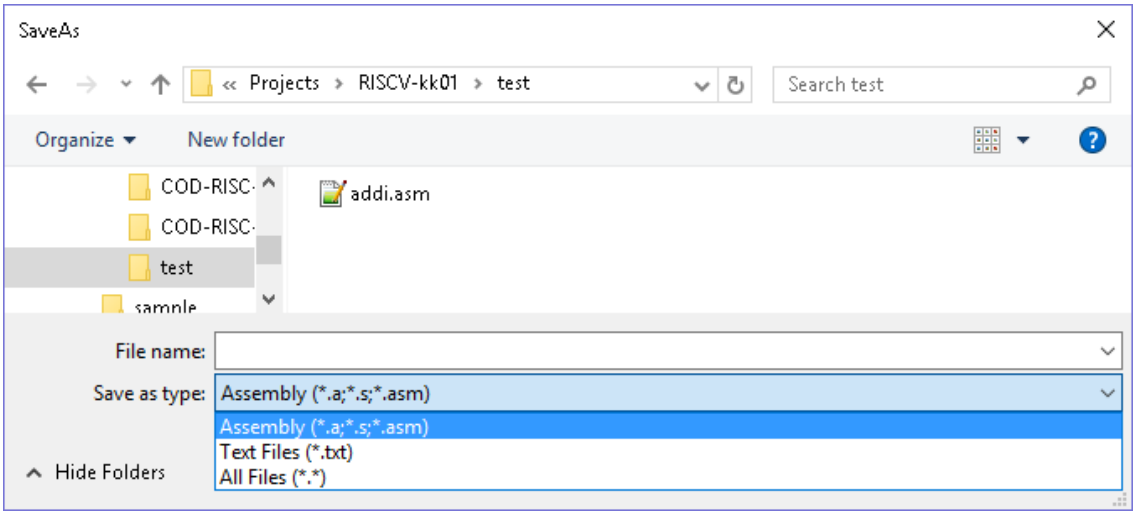


To save the **Source** text in a file, type the file name with its extension (e.g. test.asm) and then click on the **Save** button.

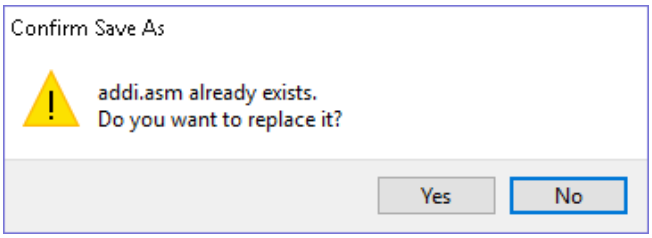
If you want to select a different folder before saving the file, you can click on **Browse Folders** and the above dialog will change as shown in the following snapshot.



The above dialog allows you to change the current directory and see the files in it. Note that only files with extensions shown in the **Save as type:** field, namely .a, .s, and .asm will be visible. To see other files change the **Save as Type:** selection as shown below.

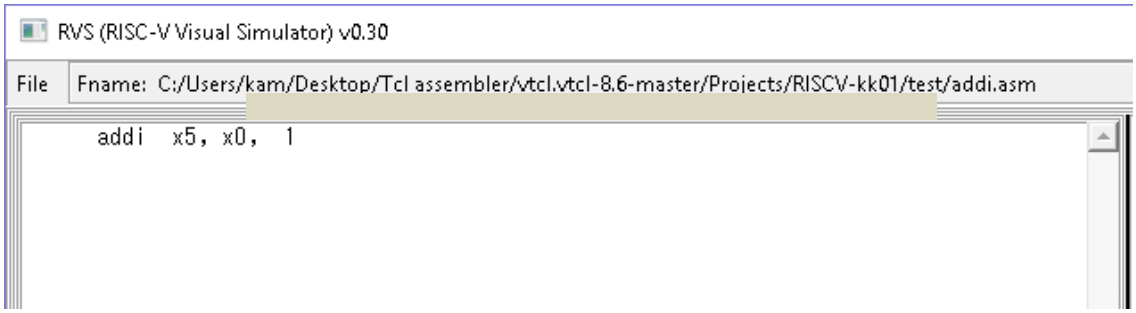


You can see above that a file named **addi.asm** is already present in the current directory. If you try to save your **Source** text under the same name the following warning message will appear.



Clicking on the **Yes** button will overwrite the content of the existing file with your **Source** text.

Once you save your **Source** text in a file, the name of the current file will show after the **Fname:** string.



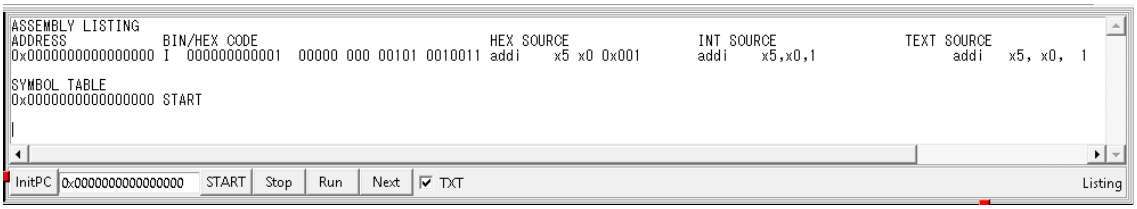
You can save the **Source** text in the current file at any time by selecting **Save** from the **File** menu. (Note that the **Fname:** string is currently shown in black indicating no unsaved changes.)

The **File** menu also includes two options for loading in the **Source** window existing assembly programs from files. The **Load** option will overwrite the current **Source** text with the new one from the file (a warning will be issued in case of unsaved changes.) The **Append** option on the other hand will add the new program text from the file after the last character of the current **Source** text. Note that after using **Append** the file name will be cleared and the unsaved changes flag will be raised.

The last option of the File menu is Exit which just terminates the RVS program (a warning will be issued in case of unsaved changes.)

3. Compilation

Now let us proceed with compiling the text in the **Source** window. Press the **Compile** button and the **Listing** window will change as shown below.



The resulting assembly listing shown above contains five columns as described below:

- ADDRESS

  `0x0000000000000000`

  This is the memory address where the compiled instruction is placed.
- BIN/HEX CODE

  `I  000000000001  00000 000 00101 0010011`

  This is the binary representation of the compiled instruction.
- HEX SOURCE

  `addi   x5 x0 0x001`

  This is the canonical form of the instruction (spaces instead of commas and no parentheses) with all operands in hexadecimal.
- INT SOURCE

  `addi   x5,x0,1`

  This is the standard form of the instruction (with commas and parentheses) with all operands in decimal.
- TEXT SOURCE

  `addi   x5, x0,  1`

  This is the instruction text as entered in the **Source** window.

The inclusion of the above columns in the **Listing** is controlled by the check-boxes in the bottom of the **Source** window. Only the columns corresponding to the checked boxes will appear in the listing when the **Compile** button is pressed. Note that in order to change the columns you will have to re-compile the source.

In our case the instruction was placed at the default address of `0x0000000000000000`. The beginning of the program code was marked by the label START as shown in the

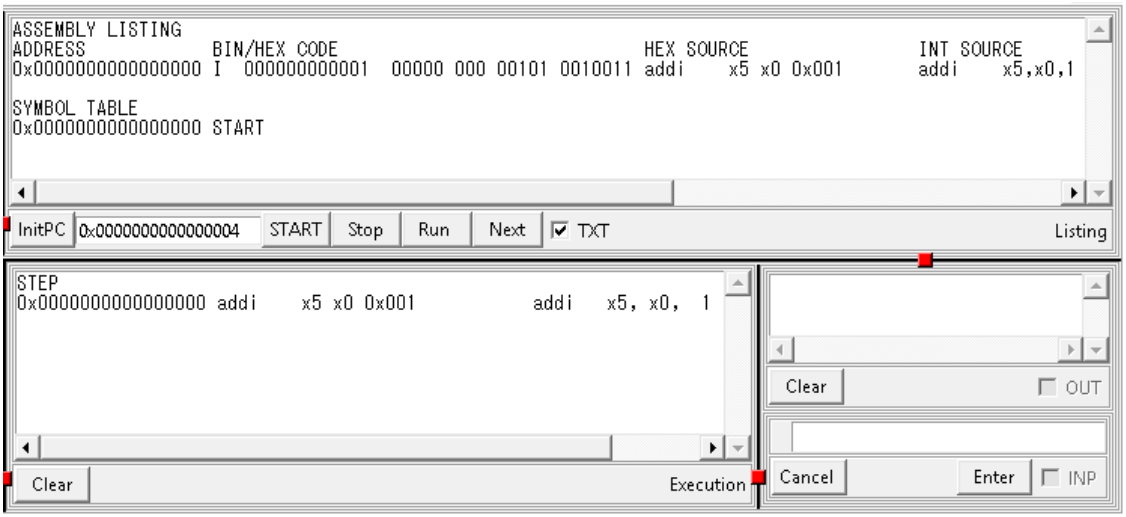
SYMBOL TABLE (`0x0000000000000000 START`).

The following controls are placed in the bottom of the Listing window:

- **InitPC** button with an entry field
  The entry field contains the current value of the Program Counter (PC). The PC can be changed by directly typing into this field. Ressing the InitPC button will reset the PC value to the value of the START label (`0x0000000000000000` in our case).
- **START** button
  Pressing this button will reset the PC to the START value and will run the program.
- **Stop** button
  Pressing this button will stop the program at the current instruction (the address of the current instruction will be shown in the PC entry field)
- **Run** button
  Pressing this button will run the program starting from the current instruction (the one pointed by the current value of the PC)
- **Next** button
  Pressing this button will execute the current instruction (the one pointed by the current value of the PC) and will stop. The PC is automatically adjusted to point to the next instruction. Use this button for stepwise execution of your code.
- **TXT** check-box
  This checkbox controls the inclusion of the source code in the **Execution** tracing

4. Execution

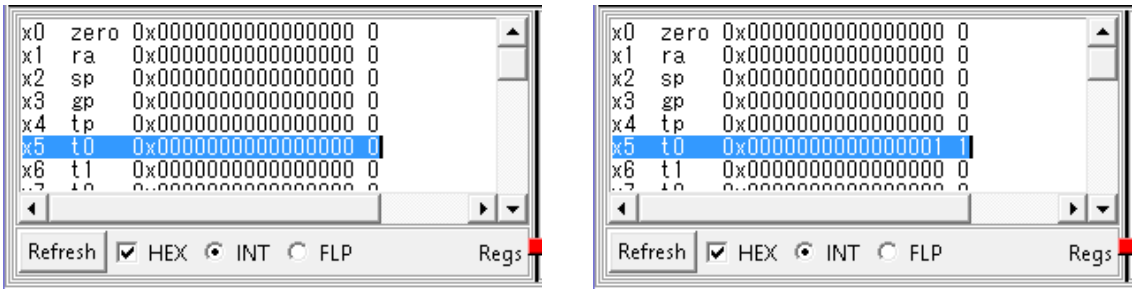
Now press the **Next** button to execute the single compiled instruction in the **Listing**. The **Execution** window is shown below.



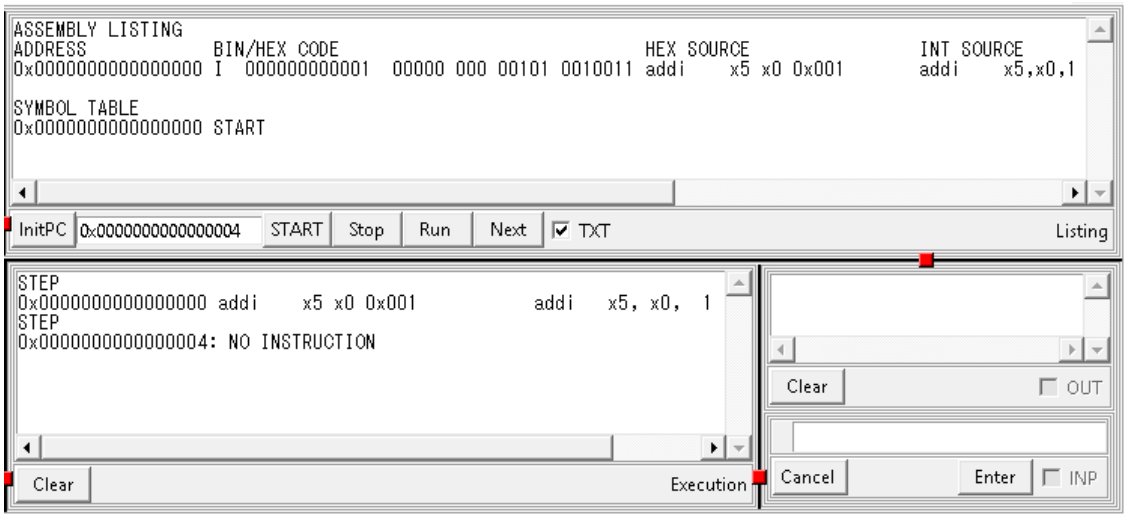
The three columns in the Execution window from left to right are as follows:

- Address
  `0x0000000000000000`
  This is the address of the current instruction.
- Instruction
  `x5 x0 0x001`
  This is the current instruction in canonical form (spaces instead of commas, no parentheses) with all operands in hexadecimal.
- Source
  `addi  x5, x0, 1`
  This is the text of the instruction line as entered in the **Source** window.

The **`addi x5, x0, 1`** instruction adds 1 to the value of register **`x0`** (which is always 0) and stores the result (1 in our case) into register **`x5`**. The values of the registers before (on the left) and after (on the right) the execution of this instruction are shown in the corresponding snapshots of the **Regs** window below.



After the execution of this instruction the PC was incremented by 4 to point at address `0x0000000000000004`. If we press the Next button again the following error message will be shown.



The message `0x0000000000000004: NO INSTRUCTION` indicates that no executable instruction was found at address `0x0000000000000004` so the program was stopped.

5. Internals

The RVS execution of RISC-V programs is based on a Virtual memory (VM) model implemented through hash-based associative memory. In this way the executed code can freely refer and use the entire 64-bit addressing space while the allocated physical memory will be confined only to the necessary minimum. With this model there is no need for an operating system to manage the stack and heap memory allocation.

All virtual memory is considered as initialized to 0s so reading from an address without its prior initialization will return 0s without an error while no physical memory will actually be allocated. Writing to an address, however, will lead to actual physical memory allocation (if the memory has not been allocated already.) This is also true for writing 0s, e.g. actual physical memory will be allocated and maintained for storing 0s. Although less efficient, this is a design decision that greatly facilitates the tracking of the memory use by the executed code as shown in the **Memory** window.

The RVS provides bit-accurate representation of the supported RISC-V machine instructions although it uses different data structures for simulating the execution process.