RVS (Risk-V Visual Simulator)

Input/Output System Calls Manual v0.04

1. Outline

The Risk-V Visual Simulator (RVS) supports user communications (data input and output) with executed assembly code through the I/O window in the bottom left of the main RVS window as shown below enclosed in a red rectangle.

All user communications are carried out through the `ecall` instructions in the format **`ecall rd, rs, imm`**. The first two parameters **`rs`** and **`rd`** can be either integer or float point registers irrespectively of the immediate value **`imm`**. The interpretation of the values in **`rs`** and **`rd`**, however depends on the syscall value placed in **`imm`** as explained below.

The I/O syscalls are blocking which means that the code execution will be suspended while there is input or output in progress. The user interactions with other components of the RVS main window are also blocked so the user can only interact with the I/O window.

2. Output (print services)

Output system calls are carried out through `ecall` instructions in the following format:

`ecall rd, rs, imm`.

The three parameters **`rd`**, **`rs`**, and **`imm`** must be supplied but only two of them, namely the source register **`rs`**, and the immediate value **`imm`** will be used. The value of the destination register **`rd`** is not used and will not be changed.

The following print services are available:

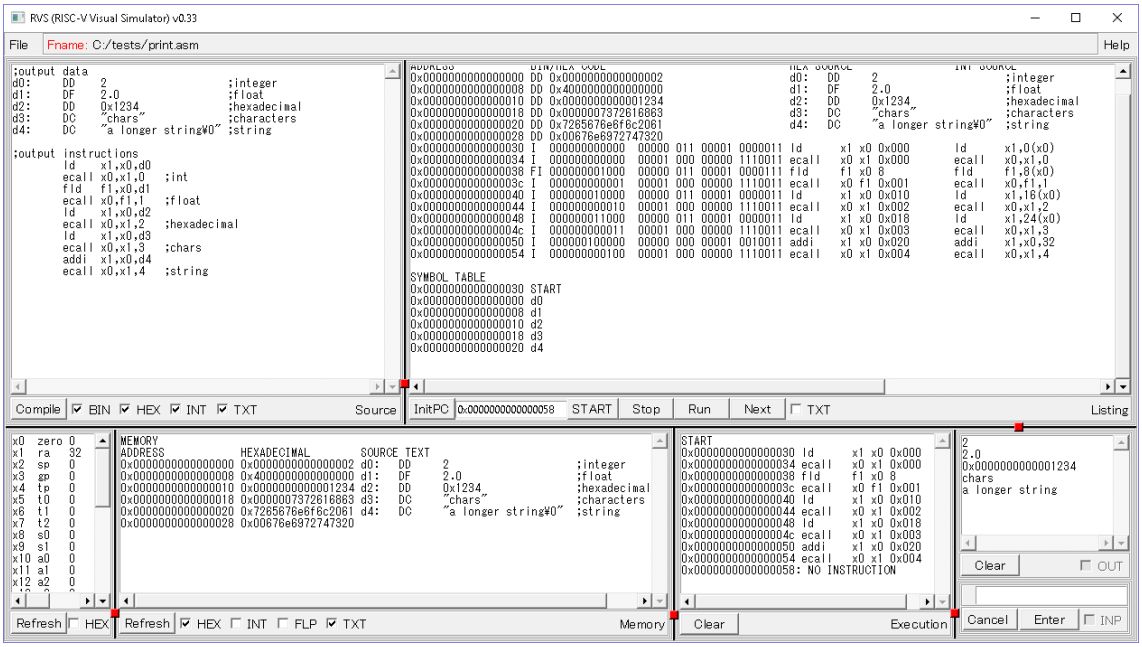
- **print_integer (`ecall rd, rs, 0`)** The value in the source register **`rs`** is printed as a 64-bit signed integer. The value of the destination register **`rd`** is not changed. The syscall code is 0.
- **print_float (`ecall rd, rs, 1`)** The value in the source register **`rs`** is printed as a 64-bit floatin point number. The value of the destination register **`rd`** is not changed. The syscall code is 1.
- **print_hexadecimal (`ecall rd, rs, 2`)** The value in the source register **`rs`** is printed as a 64-bit hexadecimal. This is a convenient way to explore the internal representation of any data including integer and floating point numbers. The value of the destination register **`rd`** is not changed. The syscall code is 2.
- **print_characters (`ecall rd, rs, 3`)** The value in the source register **`rs`** is printed as a sequence of characters, each character corresponding to one byte, in little endian order. The printing stops if a null (0) character is encountered. Note that the sequence of characters does not have to be null-terminated as the maximum number of characters is limited to 8. The value of the destination register **`rd`** is not changed. The syscall code is 3.
- **print_string (`ecall rd, rs, 4`)** The value in the source register **`rs`** is an address of the beginning of a null terminated string. The sequence of characters in the string are printed in little endian order until the null character is encountered. Note that the string must be null-terminated as otherwise the printing may continue past the string end. The value of the destination register **`rd`** is not changed. The syscall code is 4.

An output example. The following assembly language code has been compiled and executed in RVS:

```
;output data
d0:     DD    2                  ;integer
d1:     DF    2.0                ;float
d2:     DD    0x1234             ;hexadecimal
d3:     DC    "chars"            ;characters
d4:     DC    "a longer string¥0" ;string

;output instructions
        ld    x1,x0,d0
        ecall x0,x1,0   ;int
        fld   f1,x0,d1
        ecall x0,f1,1   ;float
        ld    x1,x0,d2
        ecall x0,x1,2   ;hexadecimal
        ld    x1,x0,d3
        ecall x0,x1,3   ;chars
        addi  x1,x0,d4
        ecall x0,x1,4   ;string
```

The resulting RVS window is shown in the following snapshot.



3. Input (read services)

Input system calls are carried out through `ecall` instructions in the following format:

```
ecall rd, rs, imm.
```

The three parameters **rd, rs,** and **imm** must be supplied although only two of them, namely the destination register **rd,** and the immediate value **imm** are sufficient for input. The value of the source register **rs** is ignored if 0. Otherwise it is interpreted as a sequence of characters that is displayed as an input prompt.

The following read services are available:

- **read_integer (ecall rd, rs, 5)** The value entered by the user is converted to a 64-bit signed integer and is stored in the destination register **rd**. The value of the source register **rs** is used as an input prompt if not 0. The syscall code is 5.
- **read _float (ecall rd, rs, 6)** The value entered by the user is converted to a 64-bit floating point number and is stored in the destination register **rd**. The value of the source register **rs** is used as an input prompt if not 0. The syscall code is 6.
- **read _hexadecimal (ecall rd, rs, 7)** The value entered by the user is converted to a 64-bit unsigned integer and is stored in the destination register **rd**. The value of the source register **rs** is used as an input prompt if not 0. The syscall code is 7.
- **read _characters (ecall rd, rs, 8)** The value entered by the user is converted to a sequence of characters, each character corresponding to one byte, in little endian order. If the obtained sequence of bytes is shorter than 8 it is padded with zero bytes on the most significant side to complement it to 8 bytes. Note that the sequence of entered characters does not have to be null-terminated as the maximum number of characters is limited to 8 (the entered sequence of characters is trimmed on its right side if longer than 8 bytes.) The final sequence of 8 bytes is stored in the destination register **rd**. The value of the source register **rs** is used as an input prompt if not 0. The syscall code is 8.
- **read _string (ecall rd, rs, 9)** The value entered by the user is converted to a sequence of characters, each character corresponding to one byte, in little endian order. A null character (a 0-byte) is automatically appended to the entered string. The final sequence of characters is then copied to the memory starting at the address provided in the destination register **rd**. Note that it is

the responsibility of the user to ensure that there is enough memory space for storing the entire entered string including the trailing null character. RVS will silently allow overwriting of data areas but will issue an error message and stop if overwriting of code is about to happen. The value of the source register **rs** is used as an input prompt if not 0. The syscall code is 9.

An input example. The following assembly language code has been compiled and executed in RVS:

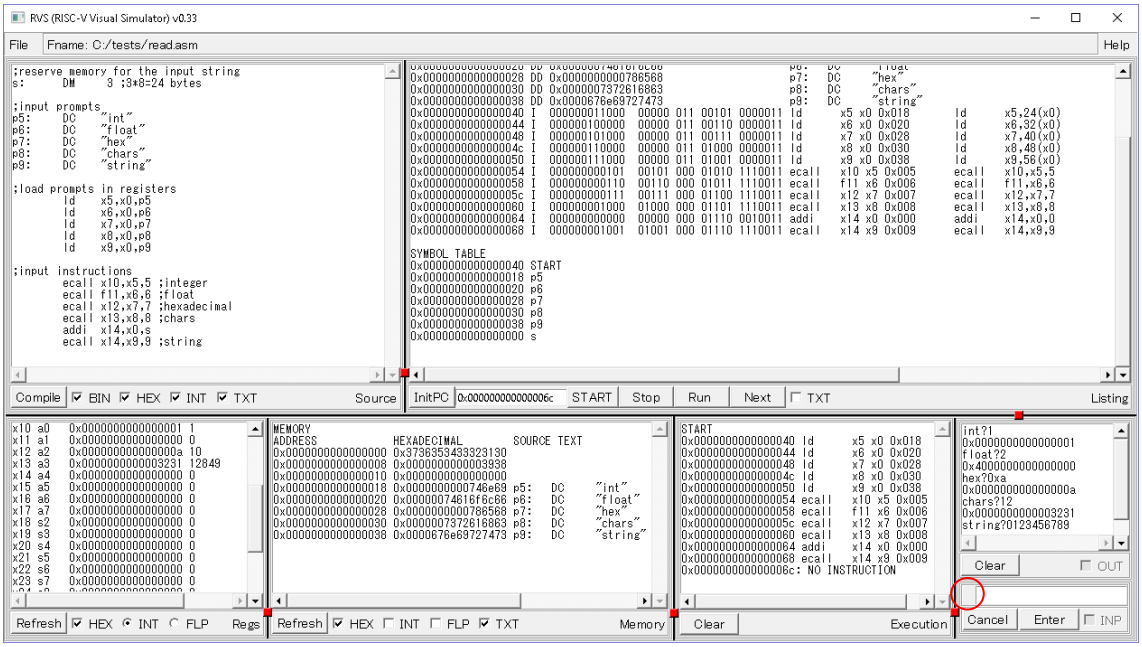
```
;reserve memory for the input string
s:      DM    3 ;3*8=24 bytes

;input prompts
p5:     DC    "int"
p6:     DC    "float"
p7:     DC    "hex"
p8:     DC    "chars"
p9:     DC    "string"

;load prompts in registers
        ld    x5,x0,p5
        ld    x6,x0,p6
        ld    x7,x0,p7
        ld    x8,x0,p8
        ld    x9,x0,p9

;input instructions
        ecall x10,x5,5 ;integer
        ecall f11,x6,6 ;float
        ecall x12,x7,7 ;hexadecimal
        ecall x13,x8,8 ;chars
        addi  x14,x0,s
        ecall x14,x9,9 ;string
```

The resulting RVS window is shown in the following snapshot.



An active input is indicated by a blinking rectangle (its color will be altering from gray to red and vise versa.) The rectangle is denoted by a red circle in the screen snapshot above.

To complete the input process, press the Enter key or click on the **Enter** button in the bottom of the I/O window. If your input cannot be converted to the desired data type an error message will be displayed and the execution of your code will be stopped. Click ether on the **Run** or on the **Next** button in the bottom of the **Listing** window to restart the last input syscall and enter your data again.

If you click on the **Cancel** button the input process will be cancelled, the execution of your program will be suspended, and all components of the GUI will become accessible. You can restart the cancelled input syscall by clicking ether on the **Run** or on the **Next** button in the bottom of the **Listing** window.

RVS Input/Output System Calls

ecall	rd	rs1	imm	I/O System Call
Print Services	Destination	Source	Code	Function
print_integer	N/A	int64	0	rs1 holds the integer to print
print_ float	N/A	float64	1	rs1 holds the float to print
print_hexadecimal	N/A	hex64	2	rs1 holds the value to print as hex
print_characters	N/A	char64 (max 8)	3	rs1 holds up to 8 (padded with nulls) characters to print
print_string	N/A	addr64	4	rs1 holds the address of the null-terminated string to print
Read Services	Destination	Source	Code	Function
read_integer	int64	prompt	5	an integer is read and stored in rd
read _float	float64	prompt	6	a float is read and stored in rd
read _hexadecimal	hex64	prompt	7	a hexadecimal value is read and stored in rd
read _characters	char64 (max 8)	prompt	8	up to 8 characters are read and stored in rd padded with nulls
read _string	addr64	prompt	9	rd holds the address of the buffer for the string to read

Note: Read Services use as a prompt up to 8 (padded with nulls) characters from rs1