RVS (RISC-V Visual Simulator)

Floating Point (FP) Extension Manual v0.07

1. Floating-point data

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

With respect to the floating-point data, it is bound to the representation used on the host architecture. This version of RVS works only with 64-bit floating point numbers.

Floating point constants can be specified in RVS either as regular floating point numbers, e.g. `1.23, 0.67, 34.111,` etc., or in scientific format, e.g. `1.2e+2, 0.75e-3,` etc.

The floating point constants contain a **mantissa** component followed by an optional **exponent** component constructed as shown below.

- **Mantissa:** `2.0, 2., 0.5, .5`
  Starts with a digit or a decimal point and contains no more than one decimal point (no decimal point is ok) and one or more of the following characters {`0123456789`}. An optional sign can be inserted before the leading character.
- **Exponent:** `e2, e+3, E-2, e+02` (in the exponent `02` is decimal, not octal)
  Starts with `e` or `E` followed by one or more decimal digits {`0123456789`}. An optional sign can be inserted right after the `e` or `E`.

Some examples of valid and invalid floating point specifications are provided below.

- Floating point constants with only a mantissa:
  `2.0, 2., 0.5, .5`
- Floating point constants with a mantissa and an exponent:
  `2.0e2, 2.e+3, 0.5E-2, .5e+02, 1e3`
- Invalid floating point constants:
  `e-1` (missing mantissa)
  `0b11e3` (the mantissa not in decimal format)

Note that both the mantissa and the exponent must be specified in decimal format, so binary, octal, and hexadecimal representations cannot be used.

Some floating point constants can, however, be specified initially as integers in non-decimal formats e.g. `0b010` (binary), `010` (octal), or `0x010` (hexadecimal) in which case there will be no decimal point and no exponent. Examples are provided in the following sections.

2. The Define Float (DF) command

DF (Define Float) Accepts multiple constants that are stored in consecutive 64-bit double-words (one constant per double-word.) Some illustrative examples of the use of the DF command follow.

```
DF    2.0, 2., 0.5, .5
```

Memory address	Content	Entry	FP Value
0x0000000000000000	0x4000000000000000	2.0	+2.0e+0
0x0000000000000008	0x4000000000000000	2.	+2.0e+0
0x0000000000000010	0x3fe0000000000000	0.5	+5.0e-1
0x0000000000000018	0x3fe0000000000000	.5	+5.0e-1

```
DF    2.0e2, 2.e+3, 0.5E-2, .5e+02, 1e3
```

Memory address	Content	Entry	FP Value
0x0000000000000000	0x4069000000000000	2.0e2	+2.0e+2
0x0000000000000008	0x409f400000000000	2.e+3	+2.0e+3
0x0000000000000010	0x3f747ae147ae147b	0.5E-2	+5.0e-3
0x0000000000000018	0x4049000000000000	.5e+02	+5.0e+1
0x0000000000000020	0x408f400000000000	1e3	+1.0e+3

DF 0b010,0b10,010,017,10,17,0x010,0x10,-0b10,-010,-10,-0x10

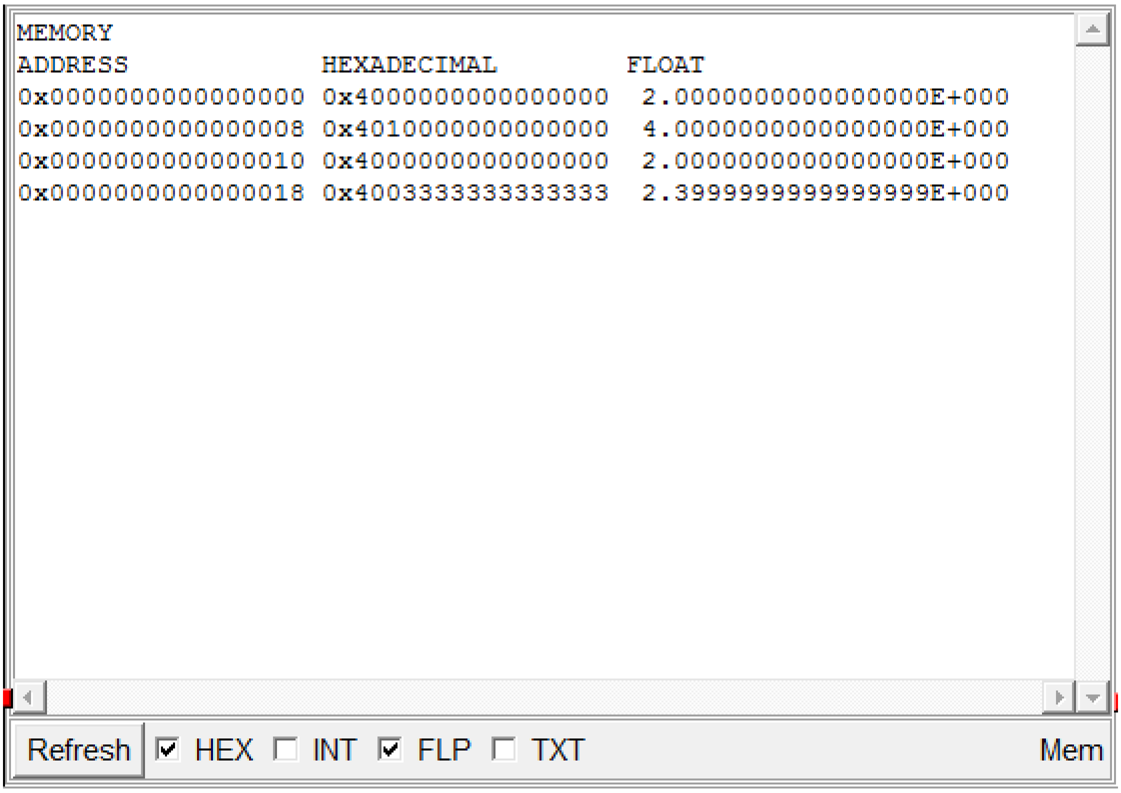
Memory address	**Content**	**Entry**	**FP Value**
0x0000000000000000	0x4000000000000000	0b010	+2.0e+0
0x0000000000000008	0x4000000000000000	0b10	+2.0e+0
0x0000000000000010	0x4020000000000000	010	+8.0e+0
0x0000000000000018	0x402e000000000000	017	+1.5e+1
0x0000000000000020	0x4024000000000000	10	+1.0e+1
0x0000000000000028	0x4031000000000000	17	+1.7e+1
0x0000000000000030	0x4030000000000000	0x010	+1.6e+1
0x0000000000000038	0x4030000000000000	0x10	+1.6e+1
0x0000000000000040	0xc000000000000000	-0b10	-2.0e+0
0x0000000000000048	0xc020000000000000	-010	-8.0e+0
0x0000000000000050	0xc024000000000000	-10	-1.0e+1
0x0000000000000058	0xc030000000000000	-0x10	-1.6e+1

3. Examples

Write the following code in the source window (or in a file and load it)

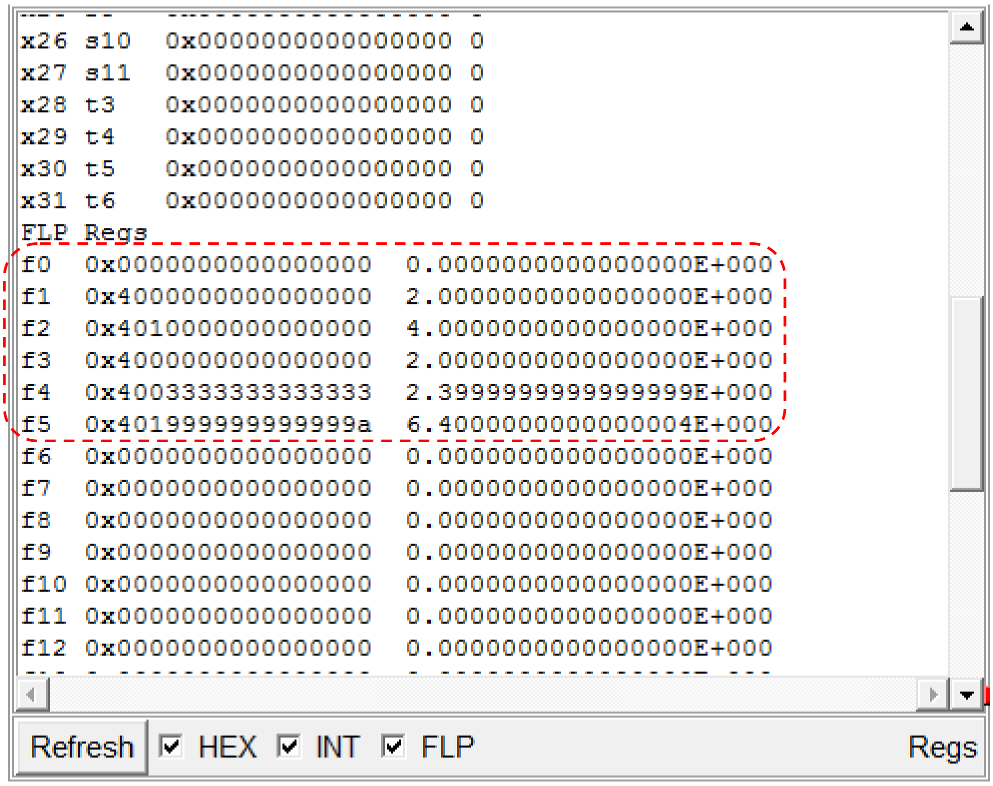
```
DF          2, 4
DF          0b10, 2.4
fld         f1, 0(x0)
fld         f2, 8(x0)
fld         f3, 16(x0)
fld         f4, 24(x0)
fadd.d      f4, f4, f2
```

Compile the above code and check the **Mem** window:



You can see that 4 double words are initialized with the constants `2, 4, 0b10`, and `2.4` respectively. Note that the last value is actually `2.3999999999999` since the exact value of `2.4` cannot be represented as a FP number.

Run the code and check the **Regs** window (you have to scroll down to see the floating point registers):



You can see that registers `f1`, `f2`, `f3`, and `f4` are loaded with the above specified `4` constants. The value in the register `f5` is `6.4` whch is the result of the addition `4+2.4`. Note that while the exact value of `2.4` could not be represented as a FP number, the resulting value of `6.4` was represented as an exact FP number.

List of supported RISC-V FP instructions

31–25	24–20	19–15	14–12	11–7	6–0	
funct7	rs2	rs1	funct3	rd	opcode	**R-type**
imm[11:0]		rs1	funct3	rd	opcode	**I-type**
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	**S-type**

RV32D Standard Extension

imm[11:0]		rs1	011	rd	0000111	**FLD**
imm[11:5]	rs2	rs1	011	imm[4:0]	0100111	**FSD**
0000001	rs2	rs1	rm	rd	1010011	**FADD.D**
0000101	rs2	rs1	rm	rd	1010011	**FSUB.D**
0001001	rs2	rs1	rm	rd	1010011	**FMUL.D**
0001101	rs2	rs1	rm	rd	1010011	**FDIV.D**
0101101	00000	rs1	rm	rd	1010011	**FSQRT.D**
0010001	rs2	rs1	000	rd	1010011	**FSGNJ.D**
0010001	rs2	rs1	001	rd	1010011	**FSGNJN.D**
0010001	rs2	rs1	010	rd	1010011	**FSGNJX.D**
0010101	rs2	rs1	000	rd	1010011	**FMIN.D**
0010101	rs2	rs1	001	rd	1010011	**FMAX.D**
1010001	rs2	rs1	010	rd	1010011	**FEQ.D**
1010001	rs2	rs1	001	rd	1010011	**FLT.D**
1010001	rs2	rs1	000	rd	1010011	**FLE.D**

RV64D Standard Extension (in addition to RV32D)

1100001	00010	rs1	rm	rd	1010011	**FCVT.L.D**
1100001	00011	rs1	rm	rd	1010011	**FCVT.LU.D**
1110001	00000	rs1	000	rd	1010011	**FMV.X.D**
1101001	00010	rs1	rm	rd	1010011	**FCVT.D.L**
1101001	00011	rs1	rm	rd	1010011	**FCVT.D.LU**
1111001	00000	rs1	000	rd	1010011	**FMV.D.X**