RVS (Risk-V Visual Simulator)

Assembler Manual v0.05

1. Syntax

Every instruction or assembly command must be on a separate line and cannot be continued on a new line.

Every line starts with an optional label, followed by an instruction/command mnemonic code, followed by a comma-separated list of parameters, followed by an optional comment.

Labels are denoted by a colon (:) as their last character. Everything from the first non-white character after the beginning of the line until the first colon character will be identified as a label (including the space/special characters, if any.)

The field separators in the list of parameters are commas, space characters, and open and close parentheses.

Comments are preceded by a semi-colon character (;), a number sign (#), or by 2 slash characters (//). The comment always continues to the end of the line. If it starts at the beginning of the line then the entire line is a comment line.

2. Arrangement of code and data in memory

The assembly begins at the default memory address of 0. This can be changed by the ORG assembly command.

Each assembled machine instruction takes one word (4 bytes) so the memory pointer is increased by 4. Machine instructions are automatically aligned at word boundaries (the address is divisible by 4.)

Each assembled Define command (**`DD, DW, DH, DB, DC, DM`**) takes one or more double-words so the memory pointer is increased by a multiple of 8 bytes. Define commands are automatically aligned at double-word boundaries (the address is divisible by 8.) Unused bits are filled with 0s.

There are no limitations with respect to the way code and data should be arranged in the memory as long as no overlapping occurs. In case of overlapping the RVS will issue an error message and stop the source text processing right at the overlapping line.

The default starting address of the assembled program which will be the initial value of the Program Counter (PC) is defined as the address of the first compiled machine instruction. Note that it may be different from the lowest address of the compiled machine instructions if **`ORG`** is used. If necessary, use the **`START`** label (e.g. put it just in front of the instruction you want to start with) to specify a different start address.

3. **Constants** (see the examples in the Define commands section below)

- **Binary:** `0b010, 0b10`
  Starts with `0b` or `0B` and contains only the following characters {`01`}. An optional sign can be inserted before the leading 0b or `0B`.
- **Octal:** `010, 017` (`10` and `17` are decimal, not octal)
  Starts with `0` and contains only the following characters {`01234567`}. An optional sign can be inserted before the leading 0.
- **Decimal:** `10, 17` (`010` and `017` are octal, not decimal)
  Starts with a non-zero digit and contains only the following characters {`0123456789`}. An optional sign can be inserted before the leading digit.
- **Hexadecimal:** `0x010, 0x10, 0Xab, 0xCD`
  Starts with `0x` or `0X` and contains only the following characters {`0123456789abcdefABCDEF`}. An optional sign can be inserted before the leading `0x` or `0X`.
- **Characters:** `"101", "0101", "123", "abcd"`
  A sequence of characters enclosed in double quotes. Each character is converted to a 8-bit value corresponding to its ASCII code.

4. Define commands

DD (Define Double-words) Specified binary, octal, integer, and hexadecimal constants are converted to 64-bit unsigned (or signed, if preceded by a minus sign) integers. Accepts multiple constants that are stored in consecutive 64-bit double-words (1 constant per double-word.)

```
DD     0b010,0b10,010,017,10,17,0x010,0x10,-0b10,-010,-10,-0x10
```

```
0x0000000000000000 DD 0x0000000000000002    DD 0b010
0x0000000000000008 DD 0x0000000000000002    DD 0b10
0x0000000000000010 DD 0x0000000000000008    DD 010
0x0000000000000018 DD 0x000000000000000f    DD 017
0x0000000000000020 DD 0x000000000000000a    DD 10
0x0000000000000028 DD 0x0000000000000011    DD 17
0x0000000000000030 DD 0x0000000000000010    DD 0x010
0x0000000000000038 DD 0x0000000000000010    DD 0x10
0x0000000000000040 DD 0xfffffffffffffffe    DD -0b10
0x0000000000000048 DD 0xfffffffffffffff8    DD -010
0x0000000000000050 DD 0xfffffffffffffff6    DD -10
0x0000000000000058 DD 0xfffffffffffffff0    DD -0x10
```

DW (Define Words) Specified binary, octal, integer, and hexadecimal constants are converted to 32-bit unsigned (or signed, if preceded by a minus sign) integers. Accepts multiple constants that are stored in consecutive 32-bit words (2 constant per double-word in little endian order.) If necessary, the last double-word is padded with 0s.

```
DW     0b010,0b10,010,017,10,17,0x010,0x10,-0b10,-010,-10,-0x10
```

```
0x0000000000000000 DD 0x0000000200000002    DD 0b010,0b10
0x0000000000000008 DD 0x0000000f00000008    DD 010,017
0x0000000000000010 DD 0x000000110000000a    DD 10,17
0x0000000000000018 DD 0x0000001000000010    DD 0x010,0x10
0x0000000000000020 DD 0xfffffff8fffffffe    DD -0b10,-010
0x0000000000000028 DD 0xfffffff0fffffff6    DD -10,-0x10
```

DH (Define Half-words) Specified binary, octal, integer, and hexadecimal constants are converted to 16-bit unsigned (or signed, if preceded by a minus sign) integers. Accepts multiple constants that are stored in consecutive 16-bit half-words (4 constant per double-word in little endian order.) If necessary, the last double-word is padded with 0s.

```
DH      0b010,0b10,010,017,10,17,0x010,0x10,-0b10,-010,-10,-0x10
```

```
0x0000000000000000 DD 0x000f000800020002  DD 0b010,0b10,010,017
0x0000000000000008 DD 0x001000100011000a  DD 10,17,0x010,0x10
0x0000000000000010 DD 0xfff0fff6fff8fffe  DD -0b10,-010,-10,-0x10
```

DC (Define Characters) Accepts a single constant which must be a sequence of characters enclosed in double quotes. The ASCII codes of the characters are stored in one or more consecutive 64-bit double-words (8 characters per double-word in little endian order.) If necessary, the last double-word is padded with 0s. Add a trailing \0 to the sequence of characters to make sure that it will be null-terminated irrespectively of its length.

```
DC      "0123456789\0"
```

```
0x0000000000000000 DD 0x3736353433323130
0x0000000000000008 DD 0x0000000000003938
```

DM (Define Memory in double-words) Initializes the specified number of 64-bit double-words with all 0s.

```
DM      3
```

```
0x0000000000000000 DD 0x0000000000000000
0x0000000000000008 DD 0x0000000000000000
0x0000000000000010 DD 0x0000000000000000
```

5. The ORG (Origin) Command

ORG (Origin) Defines the address at which the next assembled data or instruction will be placed.

```
ORG   0x1000
DD    1
ORG   0x800
addi  x5,x0,1
```

```
ASSEMBLY LISTING
ADDRESS            BIN/HEX CODE                                      TEXT SOURCE
0x0000000000000800 I  000000000001  00000 000 00101 0010011          addi   x5,x0,1
0x0000000000001000 DD 0x0000000000000001                             DD     1

SYMBOL TABLE
0x0000000000000800 START
```

6. List of supported RISC-V instructions

31 27 26 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
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20j10:1|11j19:12]				rd	opcode	J-type

RV32I Base Instruction Set

imm[31:12]				rd	0110111	**LUI**
imm[31:12]				rd	0010111	**AUIPC**
imm[20j10:1|11j19:12]				rd	1101111	**JAL**
imm[11:0]		rs1	000	rd	1100111	**JALR**
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	**BEQ**
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	**BNE**
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	**BLT**
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	**BGE**
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	**BLTU**
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	**BGEU**
imm[11:0]		rs1	000	rd	0000011	**LB**
imm[11:0]		rs1	001	rd	0000011	**LH**
imm[11:0]		rs1	010	rd	0000011	**LW**
imm[11:0]		rs1	100	rd	0000011	**LBU**
imm[11:0]		rs1	101	rd	0000011	**LHU**
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	**SB**
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	**SH**
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	**SW**
imm[11:0]		rs1	000	rd	0010011	**ADDI**
imm[11:0]		rs1	010	rd	0010011	**SLTI**
imm[11:0]		rs1	011	rd	0010011	**SLTIU**
imm[11:0]		rs1	100	rd	0010011	**XORI**
imm[11:0]		rs1	110	rd	0010011	**ORI**
imm[11:0]		rs1	111	rd	0010011	**ANDI**
0000000	rs2	rs1	000	rd	0110011	**ADD**
0100000	rs2	rs1	000	rd	0110011	**SUB**
0000000	rs2	rs1	001	rd	0110011	**SLL**
0000000	rs2	rs1	010	rd	0110011	**SLT**
0000000	rs2	rs1	011	rd	0110011	**SLTU**
0000000	rs2	rs1	100	rd	0110011	**XOR**
0000000	rs2	rs1	101	rd	0110011	**SRL**
0100000	rs2	rs1	101	rd	0110011	**SRA**
0000000	rs2	rs1	110	rd	0110011	**OR**
0000000	rs2	rs1	111	rd	0110011	**AND**
000000000000		00000	000	00000	1110011	**ECALL**
000000000001		00000	000	00000	1110011	**EBREAK**

31 27 26 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

RV64I Base Instruction Set (in addition to RV32I)

imm[11:0]		rs1	110	rd	0000011	**LWU**
imm[11:0]		rs1	011	rd	0000011	**LD**
imm[11:5]	rs2	rs1	011	imm[4:0]	0100011	**SD**
000000	shamt	rs1	001	rd	0010011	**SLLI**
000000	shamt	rs1	101	rd	0010011	**SRLI**
010000	shamt	rs1	101	rd	0010011	**SRAI**

RV32M Standard Extension

0000001	rs2	rs1	000	rd	0110011	**MUL**
0000001	rs2	rs1	001	rd	0110011	**MULH**
0000001	rs2	rs1	010	rd	0110011	**MULHSU**
0000001	rs2	rs1	011	rd	0110011	**MULHU**
0000001	rs2	rs1	100	rd	0110011	**DIV**
0000001	rs2	rs1	101	rd	0110011	**DIVU**
0000001	rs2	rs1	110	rd	0110011	**REM**
0000001	rs2	rs1	111	rd	0110011	**REMU**