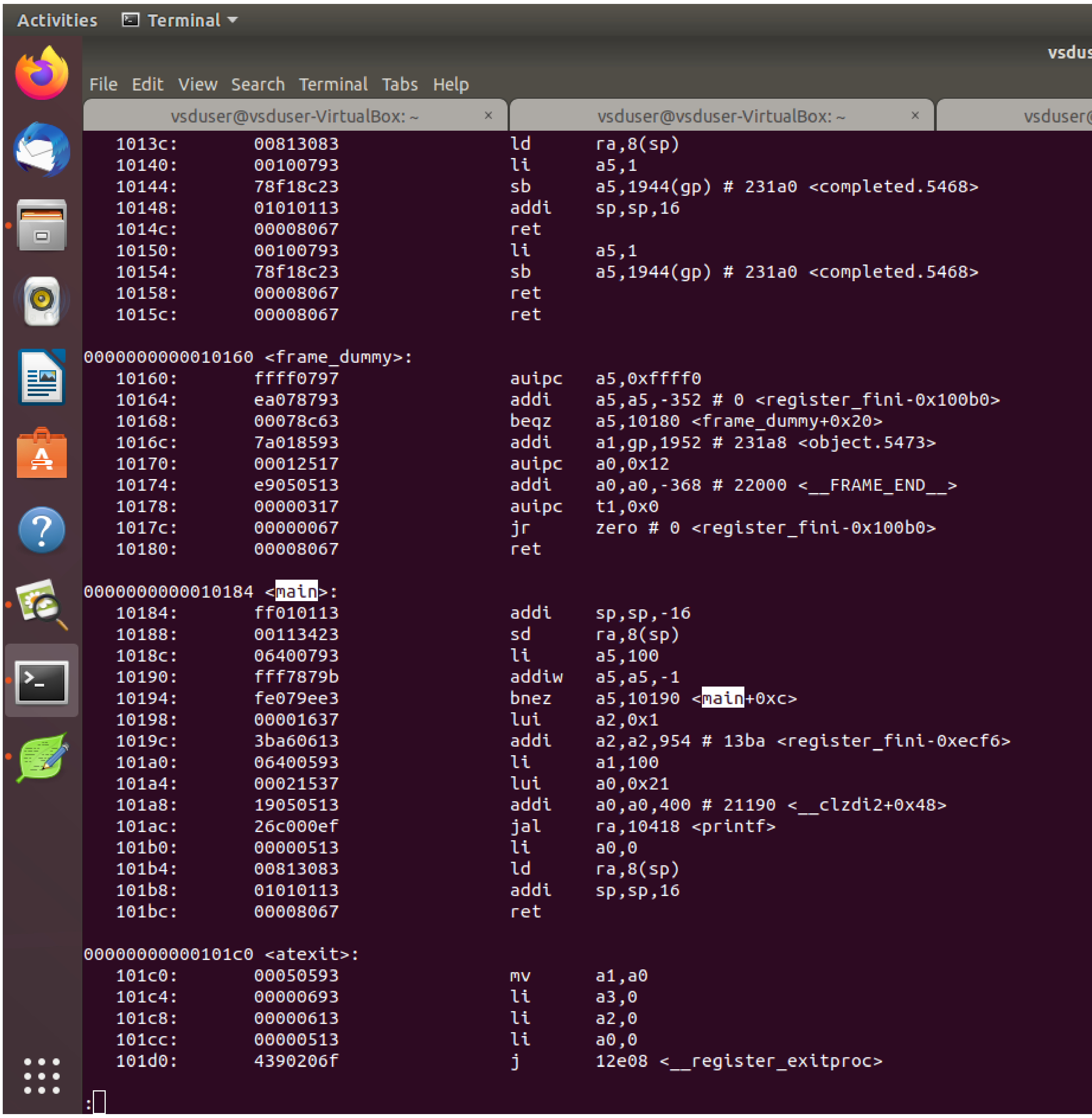
RISC-V (Reduced Instruction Set Computer - Version 5) is an open-source and modular **instruction set architecture (ISA)** based on the **RISC (Reduced Instruction Set Computing)** principles. It was developed at **UC Berkeley** and is gaining popularity in embedded systems, processors, and AI applications.

Key Features:

- **Open-source:** Free to use, modify, and implement without licensing fees.
- **Simplicity & Modularity:** Has a **base integer ISA** (RV32I, RV64I, RV128I) with optional extensions (e.g., floating-point, vector, atomic operations).
- **Scalability:** Used in embedded devices, microcontrollers, and high-performance computing.
- **Efficiency:** Optimized for performance, power consumption, and area (PPA).

RISC-V Instruction Types:

1. **R-Type:** Register operations (e.g., add, sub, or, and).
2. **I-Type:** Immediate-based operations (e.g., addi, lw).
3. **S-Type:** Store instructions (e.g., sw, sh).
4. **B-Type:** Branching instructions (e.g., beq, bne).
5. **U-Type:** Upper immediate instructions (lui, auipc).
6. **J-Type:** Jump instructions (jal).


Activities
Terminal
vsdus
File Edit View Search Terminal Tabs Help
vsduser@vsduser-VirtualBox: ~
vsduser@vsduser-VirtualBox: ~
vsduser@

```
   1013c:       00813083                ld      ra,8(sp)
   10140:       00100793                li      a5,1
   10144:       78f18c23                sb      a5,1944(gp) # 231a0 <completed.5468>
   10148:       01010113                addi    sp,sp,16
   1014c:       00008067                ret
   10150:       00100793                li      a5,1
   10154:       78f18c23                sb      a5,1944(gp) # 231a0 <completed.5468>
   10158:       00008067                ret
   1015c:       00008067                ret

0000000000010160 <frame_dummy>:
   10160:       ffff0797                auipc   a5,0xffff0
   10164:       ea078793                addi    a5,a5,-352 # 0 <register_fini-0x100b0>
   10168:       00078c63                beqz    a5,10180 <frame_dummy+0x20>
   1016c:       7a018593                addi    a1,gp,1952 # 231a8 <object.5473>
   10170:       00012517                auipc   a0,0x12
   10174:       e9050513                addi    a0,a0,-368 # 22000 <__FRAME_END__>
   10178:       00000317                auipc   t1,0x0
   1017c:       00000067                jr      zero # 0 <register_fini-0x100b0>
   10180:       00008067                ret

0000000000010184 <main>:
   10184:       ff010113                addi    sp,sp,-16
   10188:       00113423                sd      ra,8(sp)
   1018c:       06400793                li      a5,100
   10190:       fff7879b                addiw   a5,a5,-1
   10194:       fe079ee3                bnez    a5,10190 <main+0xc>
   10198:       00001637                lui     a2,0x1
   1019c:       3ba60613                addi    a2,a2,954 # 13ba <register_fini-0xecf6>
   101a0:       06400593                li      a1,100
   101a4:       00021537                lui     a0,0x21
   101a8:       19050513                addi    a0,a0,400 # 21190 <__clzdi2+0x48>
   101ac:       26c000ef                jal     ra,10418 <printf>
   101b0:       00000513                li      a0,0
   101b4:       00813083                ld      ra,8(sp)
   101b8:       01010113                addi    sp,sp,16
   101bc:       00008067                ret

00000000000101c0 <atexit>:
   101c0:       00050593                mv      a1,a0
   101c4:       00000693                li      a3,0
   101c8:       00000613                li      a2,0
   101cc:       00000513                li      a0,0
   101d0:       4390206f                j       12e08 <__register_exitproc>
:
```

INSTRUCTIONS

1) lui a2,0x1

- This belongs to the U-Type instruction format.

U- Type format:

Immediate (20-bit)	Destination Register (rd, 5-bit)	Opcode (7-bit)
imm[31:12]	rd	opcode

According to given instruction: lui a2,0x1

- a2 corresponds to x12.
- 0x1 is the immediate value.

Breaking it Down:

- Immediate: 00000000000000000001 (20-bit, represents 0x1)
- rd (Destination Register): 01100 (x12)
- Opcode: 0110111 (for lui)

Final 32-bit Binary Representation:

00000000000000000001 01100 0110111

2) addi sp,sp,-16

- This belongs to the I-Type instruction format.

I-Type Format:

Immediate (12-bit)	Source Register (rs1, 5-bit)	Function (funct3, 3-bit)	Destination Register (rd, 5-bit)	Opcode (7-bit)
imm[11:0]	rs1	funct3	rd	opcode

According to given instruction: addi sp, sp, -16

- sp corresponds to x2.
- Immediate value: -16 (which is 0xFFF0 in 12-bit signed representation).

Breaking it Down:

- Immediate: 111111111000 (-16 in signed 12-bit format)
- rs1 (Base Register): 00010 (x2 → sp)
- funct3: 000 (for addi)
- rd (Destination Register): 00010 (x2 → sp)
- Opcode: 0010011 (for addi)

Final 32-bit Binary Representation:

111111111000 00010 000 00010 0010011

3) li a1, 100

- li (Load Immediate) is a pseudo-instruction in RISC-V.
- If the immediate fits in 12 bits: addi a1, zero, 100
- If the immediate is larger than 12 bits, lui and addi are used.
- Since 100 (0x64) fits in a 12-bit signed immediate, it is translated to: addi a1, zero, 100
- addi (Add Immediate) belongs to the I-Type instruction format.

I-Type Format: addi a1, zero, 100

Immediate (12-bit)	rs1 (Base Register)	funct3 (3-bit)	rd (Destination Register)	Opcode (7-bit)
imm[11:0]	rs1	funct3	rd	opcode

According to given instruction:

- Immediate: 000000011001 (100 in signed 12-bit format)
- rs1 (Base Register): 00000 (x0 → zero)
- funct3: 000 (for addi)
- rd (Destination Register): 01011 (x11 → a1)
- Opcode: 0010011 (for addi)

Final 32-bit Binary Representation:
000000011001 00000 000 01011 0010011

4) sub a2, a2, a0

- sub subtracts a0 from a2 and stores the result in a2.

R-Type Format:

Function (funct7, 7-bit)	Source Register 2 (rs2, 5-bit)	Source Register 1 (rs1, 5-bit)	Function (funct3, 3-bit)	Destination Register (rd, 5-bit)	Opcode (7-bit)
funct7	rs2	rs1	funct3	rd	opcode

Breaking it Down:

- funct7: 0100000 (for sub)
- rs2 (Second Source Register): 01010 (x10 → a0)
- rs1 (First Source Register): 01100 (x12 → a2)
- funct3: 000 (for sub)
- rd (Destination Register): 01100 (x12 → a2)
- Opcode: 0110011 (for R-type instructions)

Final 32-bit Binary Representation:
0100000 01010 01100 000 01100 0110011

5) Instruction: lw a0, 0(sp)

- lw loads a 32-bit value from memory into a register.

I-Type Format:

Immediate (12-bit)	Source Register (rs1, 5-bit)	Function (funct3, 3-bit)	Destination Register (rd, 5-bit)	Opcode (7-bit)
imm[11:0]	rs1	funct3	rd	opcode

Breaking it Down:

- Immediate: 000000000000 (0 in signed 12-bit format)
- rs1 (Base Register): 00010 (x2 → sp)
- funct3: 010 (for lw)
- rd (Destination Register): 01010 (x10 → a0)
- Opcode: 0000011 (for lw)

6) Instruction: jal ra, 102EC <memset>

- jal performs a jump-and-link operation, storing the return address in ra.

J-Type Format:

Immediate (20-bit)	Destination Register (rd, 5-bit)	Opcode (7-bit)
imm[20]	10:1	11

Breaking it Down:

- Immediate: 00000010001011101100 (102EC in signed 20-bit format, adjusted for J-type encoding)
  - imm[20] = 0
  - imm[10:1] = 1000101110
  - imm[11] = 1
  - imm[19:12] = 00000010

- rd (Destination Register): 00001 (x1 → ra)
- Opcode: 1101111 (for jal)

Final 32-bit Binary Representation:
000000100010 1000101110 1 00000010 00001 1101111

7) Instruction: ld ra, 8(sp)

- ld loads a 64-bit value from memory into a register.

I-Type Format:

Immediate (12-bit)	Source Register (rs1, 5-bit)	Function (funct3, 3-bit)	Destination Register (rd, 5-bit)	Opcode (7-bit)
imm[11:0]	rs1	funct3	rd	opcode

Breaking it Down:

- Immediate: 000000001000 (8 in signed 12-bit format)
- rs1 (Base Register): 00010 (x2 → sp)
- funct3: 011 (for ld)
- rd (Destination Register): 00001 (x1 → ra)
- Opcode: 0000011 (for ld)

Final 32-bit Binary Representation:
000000001000 00010 011 00001 0000011

8) mv a1, a0

- Pseudo-instruction for addi a1, a0, 0

I-Type Format:

Immediate (12-bit)	Source Register (rs1, 5-bit)	Function (funct3, 3-bit)	Destination Register (rd, 5-bit)	Opcode (7-bit)
imm[11:0]	rs1	funct3	rd	opcode

Breaking it Down:

- Immediate: 000000000000 (0 in signed 12-bit format)
- rs1 (Source Register): 01010 (a0 = x10)
- funct3: 000 (for addi)
- rd (Destination Register): 01011 (a1 = x11)
- Opcode: 0010011 (for addi)

Final 32-bit Binary Representation

000000000000 01010 000 01011 0010011

9) sd ra, 8(sp)

- Store Doubleword

- Instruction Type: S-Type
- Immediate (split format): 000000000100 (8 in 12-bit signed format)
- rs2 (Data to store): 00001 (ra = x1)
- rs1 (Base Register): 00010 (sp = x2)
- funct3: 011 (for sd)
- Opcode: 0100011 (for store operations)

10) jalr a5

- Jump and Link Register

I-Type Format:

Immediate (12-bit)	Source Register (rs1, 5-bit)	Function (funct3, 3-bit)	Destination Register (rd, 5-bit)	Opcode (7-bit)
imm[11:0]	rs1	funct3	rd	opcode

- Immediate: 000000000000 (0 in signed 12-bit format)
- rs1 (Base Register): 01111 (a5 = x15)
- funct3: 000 (for jalr)
- rd (Destination Register): 01111 (a5 = x15)
- Opcode: 1100111 (for jalr)

11) sh a5, 16(a4)

- Store Halfword

- Instruction Type: S-Type
- Immediate (split format): 000000001000 (16 in 12-bit signed format)
- rs2 (Data to store): 01111 (a5 = x15)
- rs1 (Base Register): 01110 (a4 = x14)
- funct3: 001 (for sh)
- Opcode: 0100011 (for store instructions)

12) sw a5, -116(s0)

(Store Word)

- Instruction Type: S-Type
- Immediate (split format): 111110010100 (-116 in two's complement)
- rs2 (Data to store): 01111 (a5 = x15)
- rs1 (Base Register): 01000 (s0 = x8)
- funct3: 010 (for sw)
- Opcode: 0100011 (for store instructions)

Final 32-bit Binary Representation

13) and a3, a2, a3

(Bitwise AND)

- Instruction Type: R-Type
- funct7: 0000000
- rs2 (Second Operand): 01101 (a3 = x13)
- rs1 (First Operand): 01100 (a2 = x12)
- funct3: 111 (for and)

- rd (Destination Register): 01101 (a3 = x13)
- Opcode: 0110011 (for R-type ALU operations)

14) andi a5, a5, 26
(Bitwise AND with Immediate)

- Instruction Type: I-Type
- Immediate: 000000011010 (26 in binary)
- rs1 (Source Register): 01111 (a5 = x15)
- funct3: 111 (for andi)
- rd (Destination Register): 01111 (a5 = x15)
- Opcode: 0010011 (for immediate ALU operations)

15) or a5, a3, a5
(Bitwise OR)

- Instruction Type: R-Type
- funct7: 0000000
- rs2 (Second Operand): 01111 (a5 = x15)
- rs1 (First Operand): 01101 (a3 = x13)
- funct3: 110 (for or)
- rd (Destination Register): 01111 (a5 = x15)
- Opcode: 0110011 (for R-type ALU operations)