**RISCV\_VSD\_SquadronMini**

**Lab exercises of RISCV workshop by Kunal Ghosh**

**Instructor : Kunal Ghosh**

**PRE-REQUISITE: Installing the required applications for the workshop - Virtual Box, Ubuntu on VBBOX and VDI files**

**TASK-1:**

**A: Write a simple C code and compile it with gcc compiler.**

**B: Compile the same code with RISCV compiler to generate the assembly code for the same. Further Evaluate RISCV assembly code for the sample C code with two different options of compilation.**

**NECESSARY INSTALLATIONS**

**Step 1: Setting up the virtual environment to work on**

* Install Oracle Virtual Box, VMBox
* Launch Virtual Machine on VMBox
* Attach the VDI file to the Virtual Machine instance in VMBox

**Step 2: Install Leafpad - the word editor**

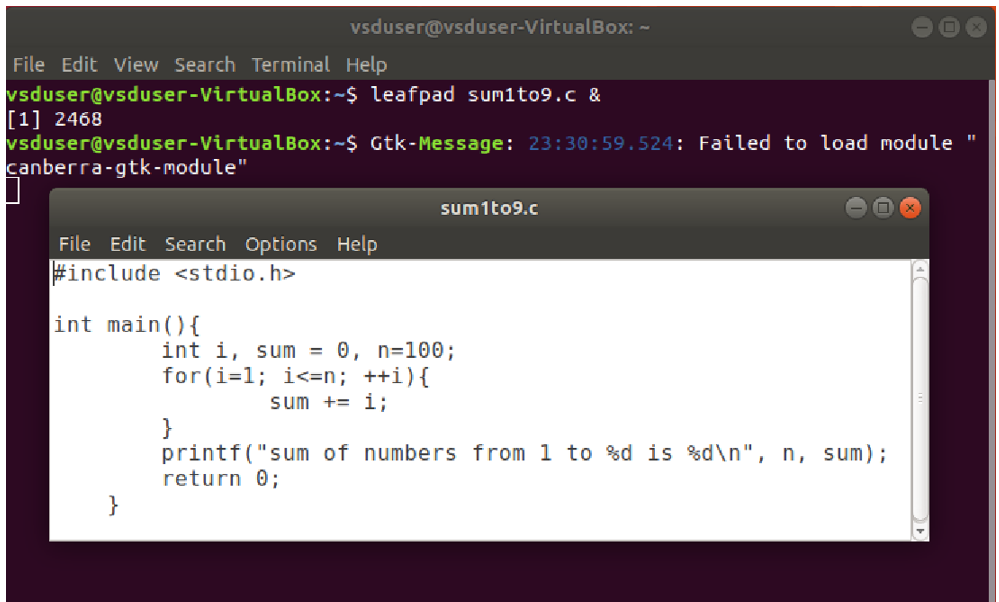
**$ sudo apt install leafpad password - vsdiat**

**TASK 1A - COMPILE AND EXECUTE A SIMPLE C CODE USING GCC COMPILER**

**$ cd --Navigate to home directory:**

**$ leafpad filename.c & This opens a blank file with filename.c, type the c code**

& will open the text editor in the background..

**[](https://private-user-images.githubusercontent.com/51965739/378216312-3a04abb4-3369-4564-bcf8-d73b6b845152.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..khs2J2JTXDvVKppUJr0C0gj7pNEtXYHGr7H1U_3VhMo)**

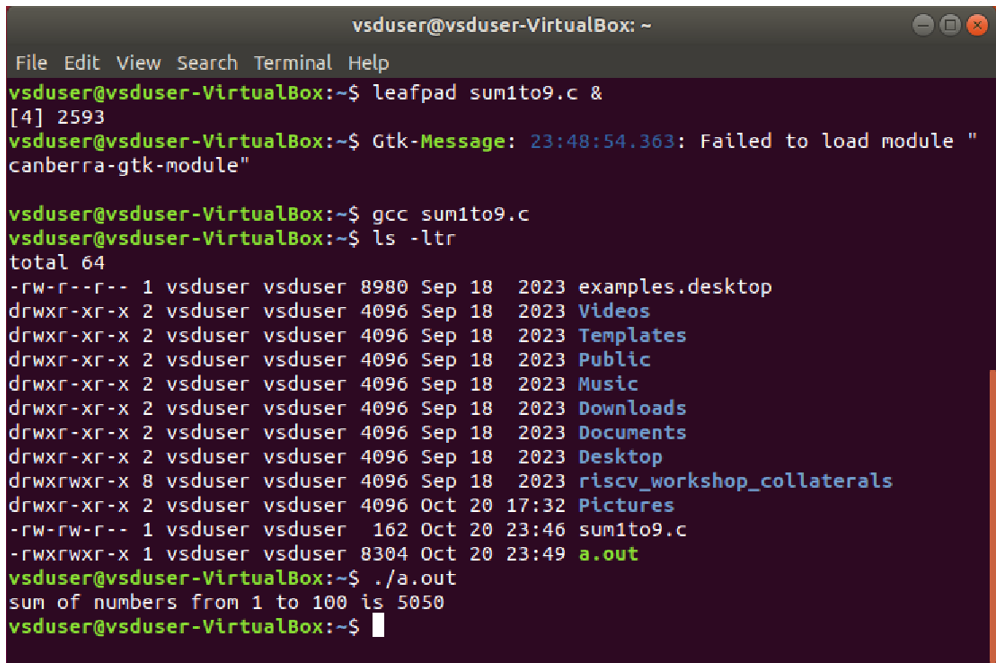
**Save the file Come back to terminal Press entre to come to the home prompt**

**To see the results Run the following commands**

**$ gcc filename.c** compiles the C program, producing an executable named a.out.

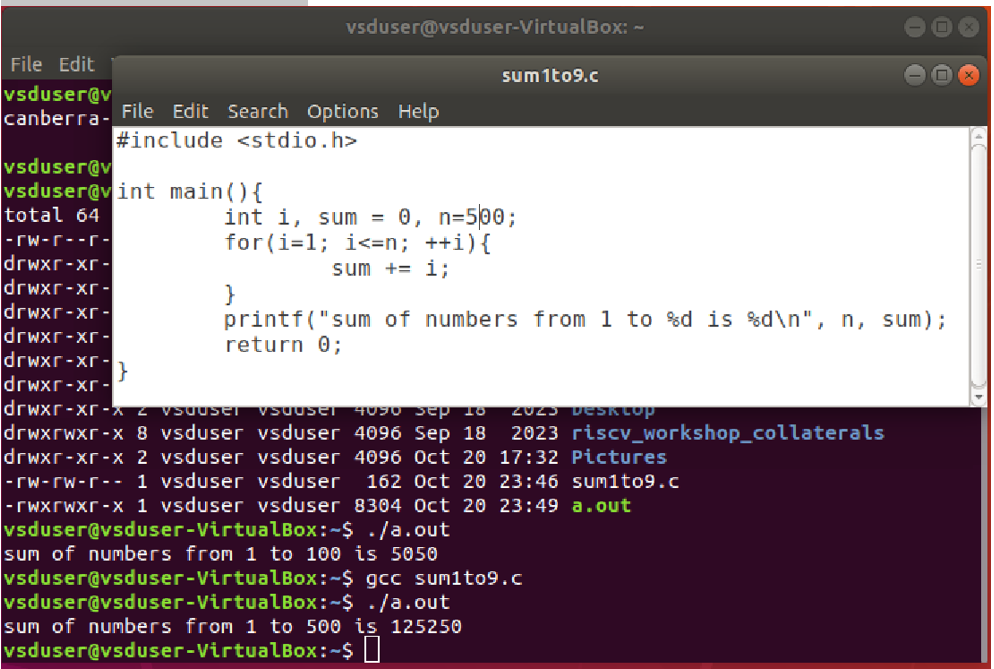
**$ ./a.out** runs that executable, executing the program

* gcc \*.c: This command compiles the C source code file using the GNU Compiler Collection (GCC). The compiled output will be stored in an executable file named a.out in the current directory by default unless a different name is specified using the -o option.
* ./a.out: This command executes the compiled executable file a.out.
* ./ tells the terminal to look for a.out in the current directory and execute it.

**[](https://private-user-images.githubusercontent.com/51965739/378216146-8310c8c6-46cf-4652-96c1-e6174aa564ae.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..WCUVV9vjvYKFuFICllbx3WCBqaNUF63kbeR__qrPDQ4)**

* ls -ltr command is a combination of options for the ls command in Unix/Linux, which lists directory contents.
* *Here's a breakdown of the flags:*
* **-l: Long format**. This option provides a detailed listing that includes file permissions, number of links, owner, group, size, and modification time of the files.
* **-t: Sort by modification time**. Files are sorted by the time they were last modified, with the most recently modified files appearing first.
* **-r: Reverse order**. This option reverses the sorting order. When used with -t, it shows the oldest modified files first (instead of the most recent).
* *ls -ltr: Lists all files and directories in the current directory (or a specified directory) in long format, sorted by the oldest modification time first.*

**Change the value of n in filename.c , Recompile (gcc \*.c) and see the results (./a.out)**

**[](https://private-user-images.githubusercontent.com/51965739/378216209-0f5ec3fa-ba7e-4e16-9324-6fc827f94570.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..uonO3-A4ZVqvhIvrMXA7IuhCFz9x78N3gn4L9_mnnf8)**

**TASK-1B: ASSEMBLY CODE ON RISCV COMPILER AND GCC COMPILER**

**1B.1 Compile C Code with RISC-V Compiler**

**$ cat sum1to9.c**

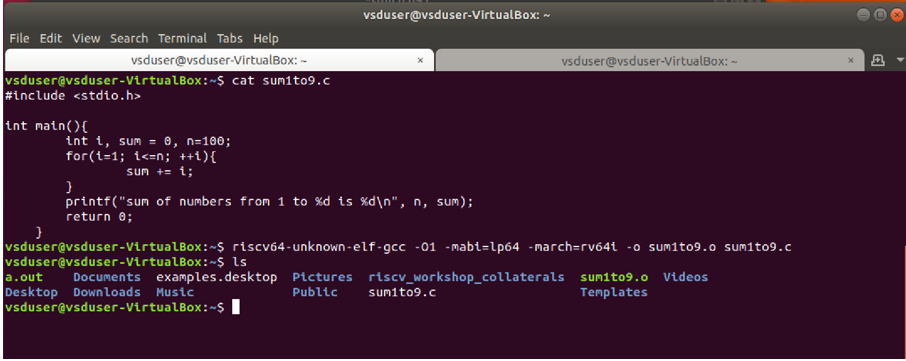
**$ riscv64-unknown-elf-gcc -O1 -mabi=lp64 -march=rv64i -o sum1to9.o sum1to9.c**

**$ ls** --lists all the files in the current directory .. to check \*.o object file is created after compilation

**cat command “concatenation” -shows the content of the \*.c file on the terminal tab**

**riscv64….** execute code using RISC-V Simulator, **creates an object file (\*.o) for source file (\*.c), before linking them together, into the final executable (by default a.out).**

*(NOTE O is in caps for -O1 option)*

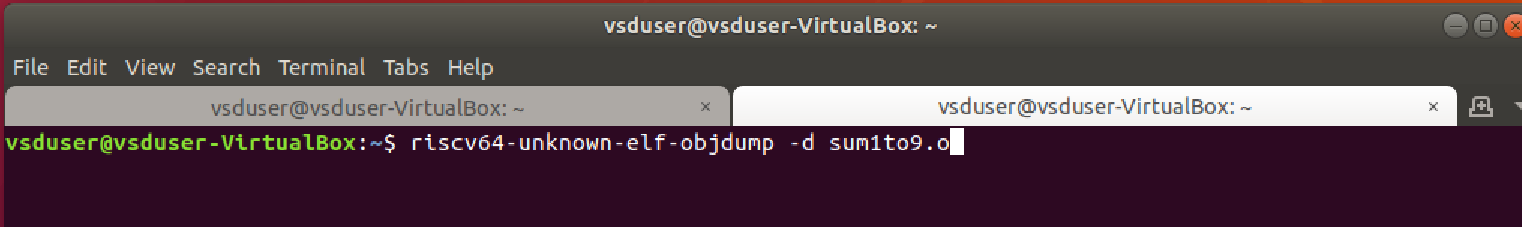
**[](https://private-user-images.githubusercontent.com/51965739/379323755-72db8c59-817a-4a0f-8c8a-3d033af60e96.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..AjY0sHq2Gwc1f3Qpi0YAhpfapZ5ohQmS1aGA-Dcso3U)**

* cat is a command-line utility in Unix-like operating systems (including Linux and macOS) that is used to concatenate and print files to the standard output.
* This command is invoking the RISC-V GCC cross-compiler (riscv64-unknown-elf-gcc) to compile a C program (\*.c) into a RISC-V ELF object file (\*.o) for the 64-bit RISC-V architecture.
* Here’s a breakdown of the individual options:
* **riscv64-unknown-elf-gcc:** This is the cross-compiler for the RISC-V 64-bit architecture. It is used to compile programs that will run on RISC-V systems but are being compiled on a different host (like x86).
* riscv64: Refers to the target architecture, which is RISC-V 64-bit. indicating that the compilation is targeting 64-bit RISC-V processors.
* unknown-elf: Specifies that the target system is an embedded system using the ELF (Executable and Linkable Format).
* unknown: Denotes an unspecified target operating system, indicating that the compilation is not tied to a particular OS.
* elf: Specifies the object file format, which is the standard format for executable files on many systems, including RISC-V.
* gcc: The GNU Compiler Collection (GCC), a widely used compiler for various programming languages, including C.
* -O : optimization level,
  + -Ofast: This flag enables the compiler to use aggressive optimizations that may not always be strictly compliant with the C standard but can potentially improve performance.
  + -O1: This flag enables optimization level 1, which applies basic optimizations to improve performance without significantly increasing compilation time. It's a trade-off between execution speed and compilation speed. It enables basic optimizations that are generally safe and compliant with the C standard but takes somewhat more time, and a lot more memory for a large function.
* -mabi: Specifies the Application Binary Interface (ABI) for the target architecture.
* lp64: Indicates l: Long data type is 64 bits, p: Large Pointer 64-bit, which means that pointers are 64 bits wide and (GET MORE CLARIFICATION ON THIS - 64: Integer and floating-point values are 64 bits. Other data types (like int and long) can be either 32 or 64 bits depending on the compiler flags.)
* -march: Specifies the target processor architecture.
* rv64i: stands for RISC-V 64-bit with the base integer instruction set (I). This is the minimal and core instruction set for a RISC-V 64-bit CPU supported by all RISC-V processors.
* -o \*.o: This specifies the compiled output file name.
* \*.c: This is the input C source file that will be compiled (\*.c).
* In summary, this command compiles the C file sum1to9.c for a 64-bit RISC-V system, generating an optimized object file (sum1to9.o) with 64-bit ABI and RISC-V integer instruction set.
* NEED TO UNDERSTAND MORE ABOUT THIS
* *The compiler performs optimization based on the knowledge it has of the program. Compiling multiple files at once to a single output file mode allows the compiler to use information gained from all of the files when compiling each of them.*
* *Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.*
* *Depending on the target and how GCC was configured, a slightly different set of optimizations may be enabled at each -O level.*
* ***To optimize the program using -Ofast function with RISCV GCC Compiler***
* *-Ofast disregards strict standards compliance in favour of improving speed. -Ofast enables all -O3 optimizations and disables strict floating-point math operations to increase performance. It also enables optimizations that are not valid for all standard-compliant programs. It turns on -ffast-math, -fallow-store-data-races and the Fortran-specific -fstack-arrays, unless -fmax-stack-var-size is specified, and -fno-protect-parens. It turns off -fsemantic-interposition.*
* *This optimization flag enables the most aggressive optimizations. Be cautious with -Ofast, as it might change the behavior of programs that depend on strict adherence to standards.*

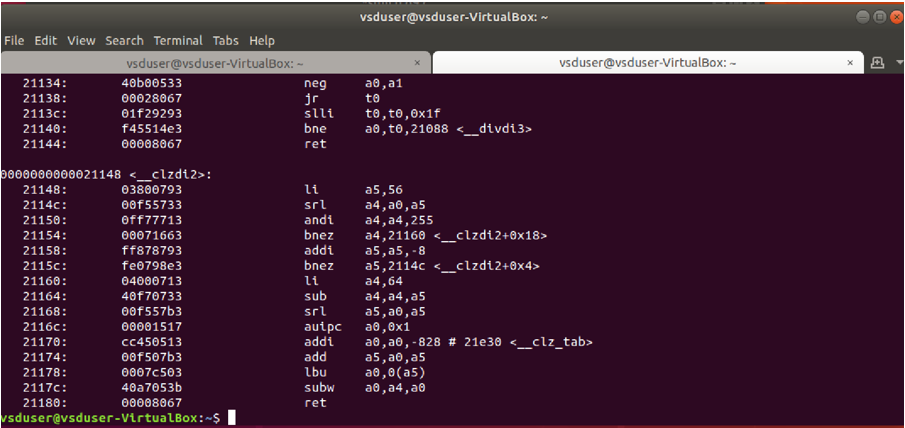
**1B.2 Display the Assembly Code of the written C code after compilation with RISCV compiler**

**To see the contents of the \*.o object file, open a new tab and execute the command**

**$ riscv64-unknown-elf-objdump -d sum1to9.o**

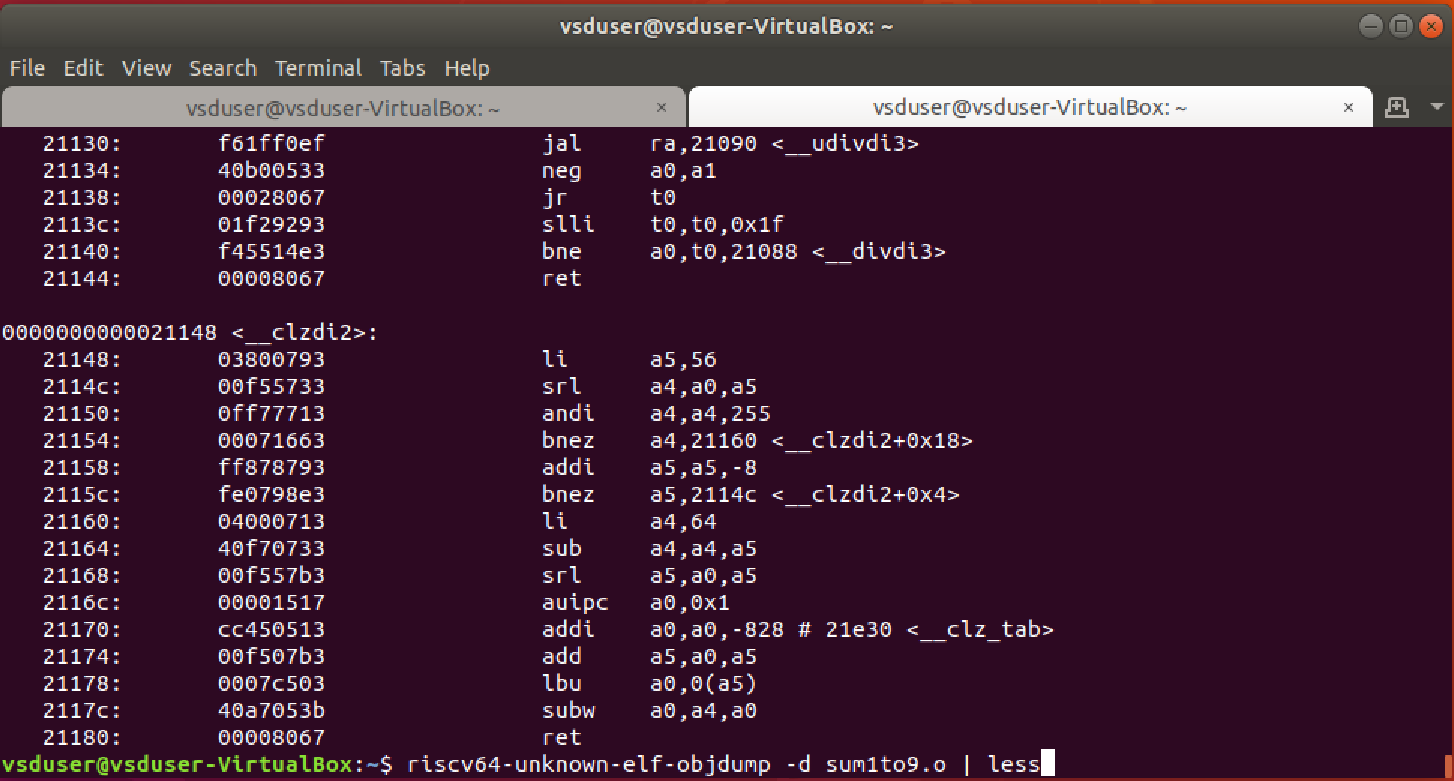
****

**Output will appear as below.** This command is used to view the 'objdump' file contents.

**[](https://private-user-images.githubusercontent.com/51965739/379325002-7e180792-4f71-4c83-a8e0-ce819db24077.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..8QqWswvCds2DE7KY6-1PeWrvUlUxnku9mXyKvgsN9FQ)**

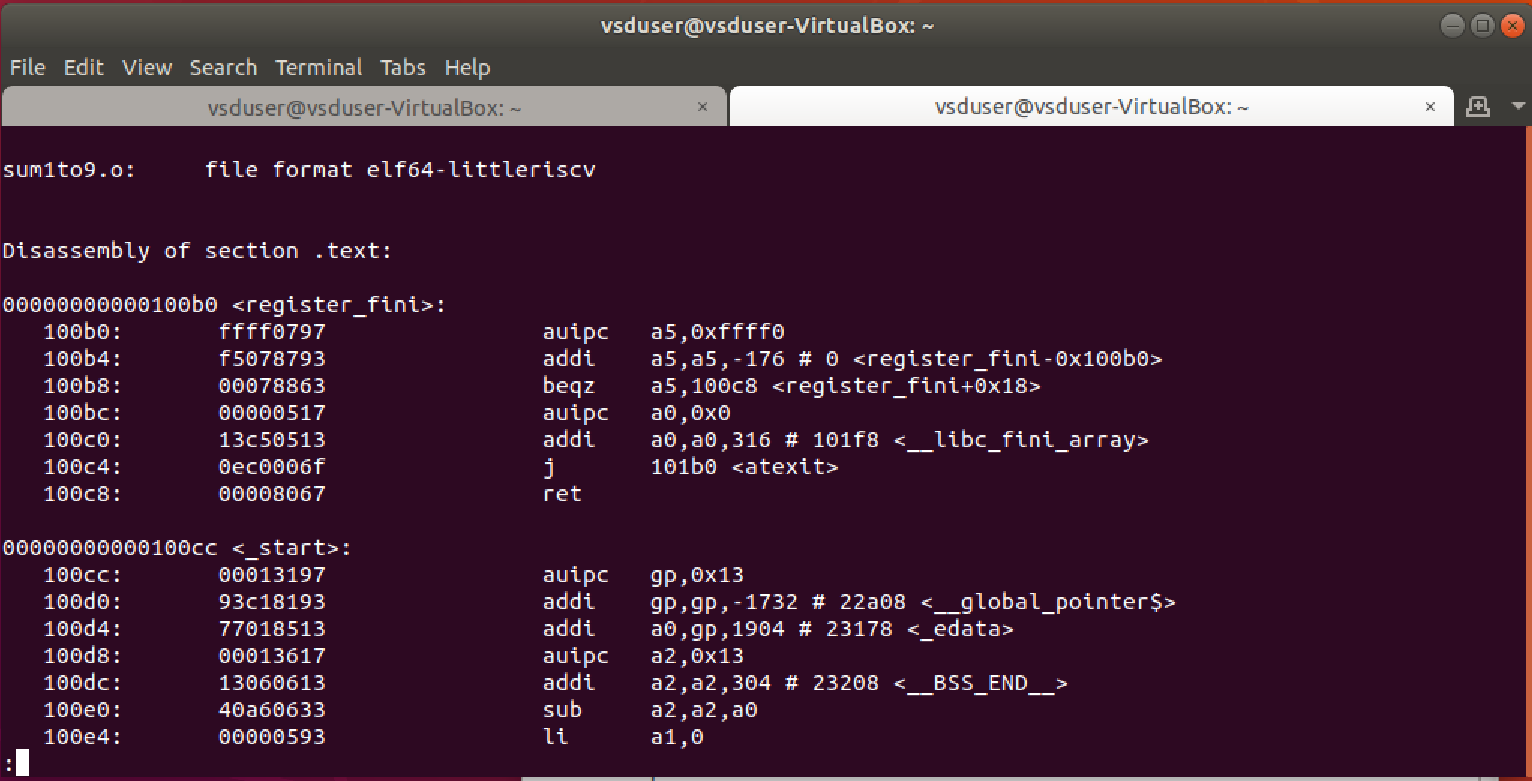
**1B.3 Display the Assembly Code after compilation with RISCV compiler by piping the output to a pager (like less) for easier navigation:** To look into the assembly code generated by RISCV compiler execute the command

**$ riscv64-unknown-elf-objdump -d sum1to9.o | less**



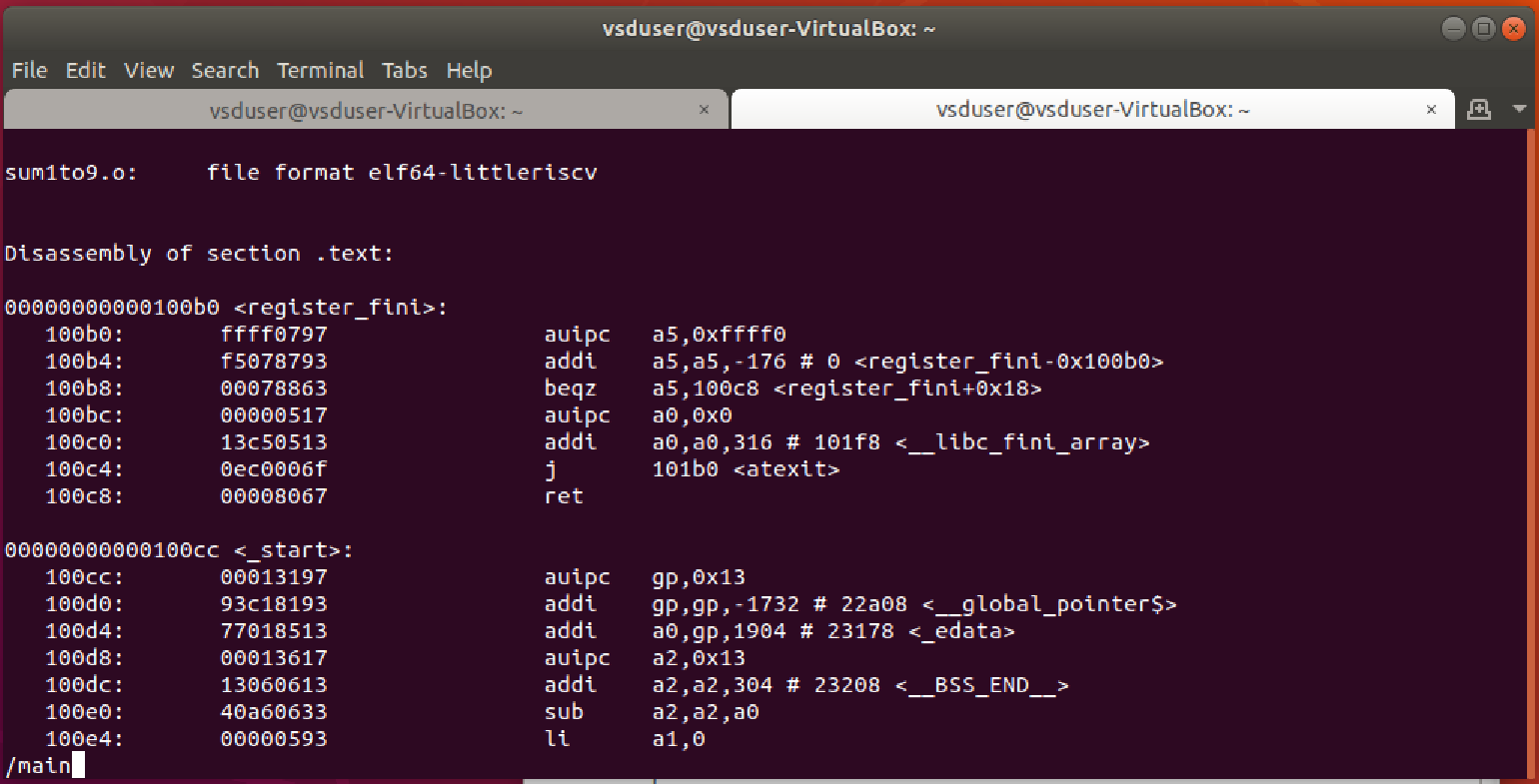
* objdump is a command-line utility used to display information about binary files, such as object files, executables, shared libraries, or core dumps. It is commonly used for debugging, reverse engineering, or inspecting the internal structure of binaries.
* objdump: The GNU Object File Disassembler, a tool used to extract information from object files.
* The riscv64-unknown-elf-objdump is a tool used for disassembling or displaying information about object files or binaries compiled for the RISC-V 64-bit architecture.
* -d: This option is for disassembly. It tells objdump to disassemble the machine code instructions in the object file or executable into human-readable assembly code.
* *To disassemble an object file, you'd first compile the C source file (\*.c) into an object file (\*.o) using the RISC-V compiler, and then use objdump on the compiled object file.*
* | less: The pipe | character redirects the output of objdump as input to the less command. less is a text pager that allows you to view the output in a more manageable way, providing features like scrolling, searching, and viewing multiple pages at once.

**OUTPUT after executing the pipe command**

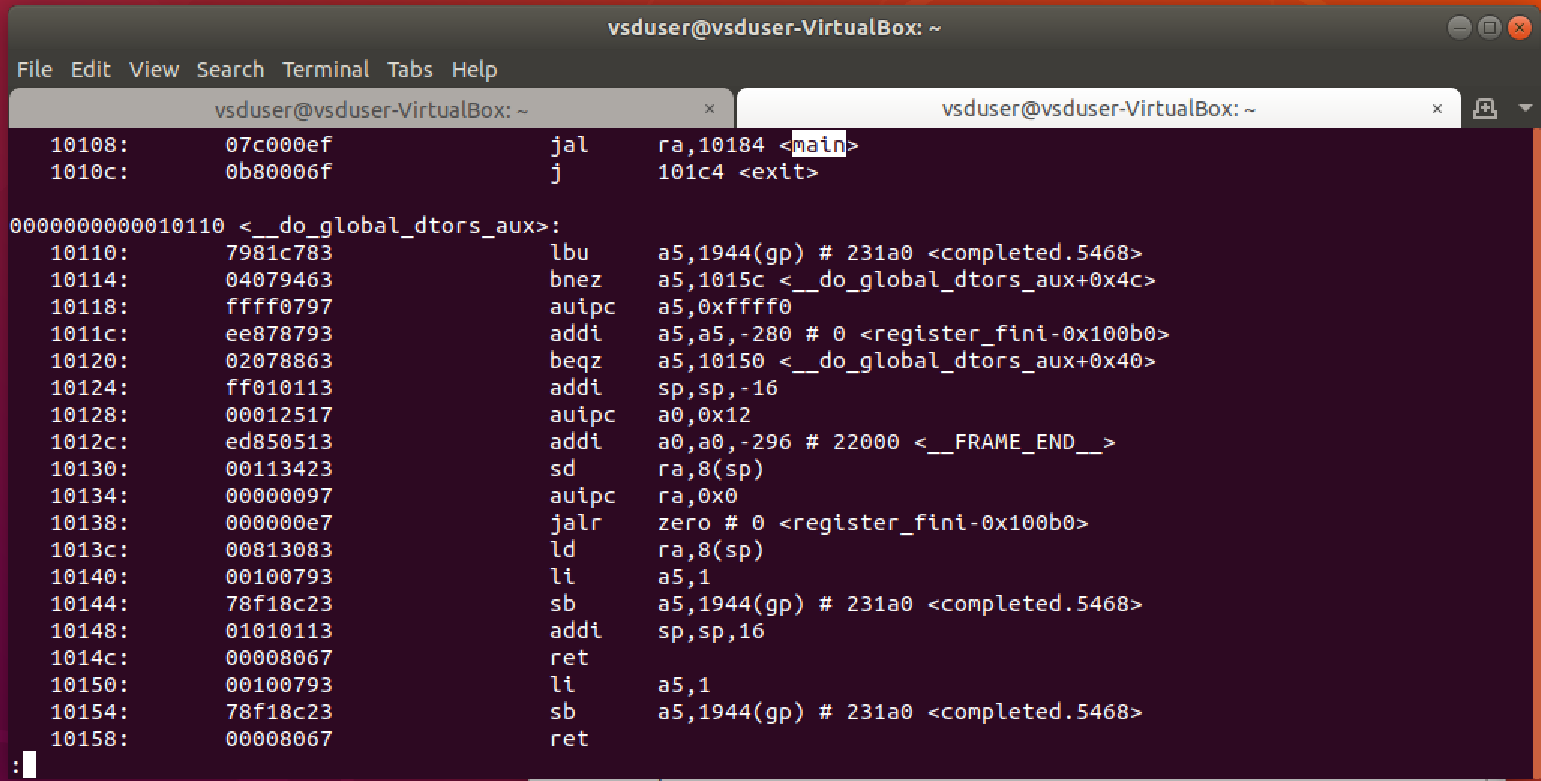
****

**1B.4 Search for the main() function in Assembly Code**

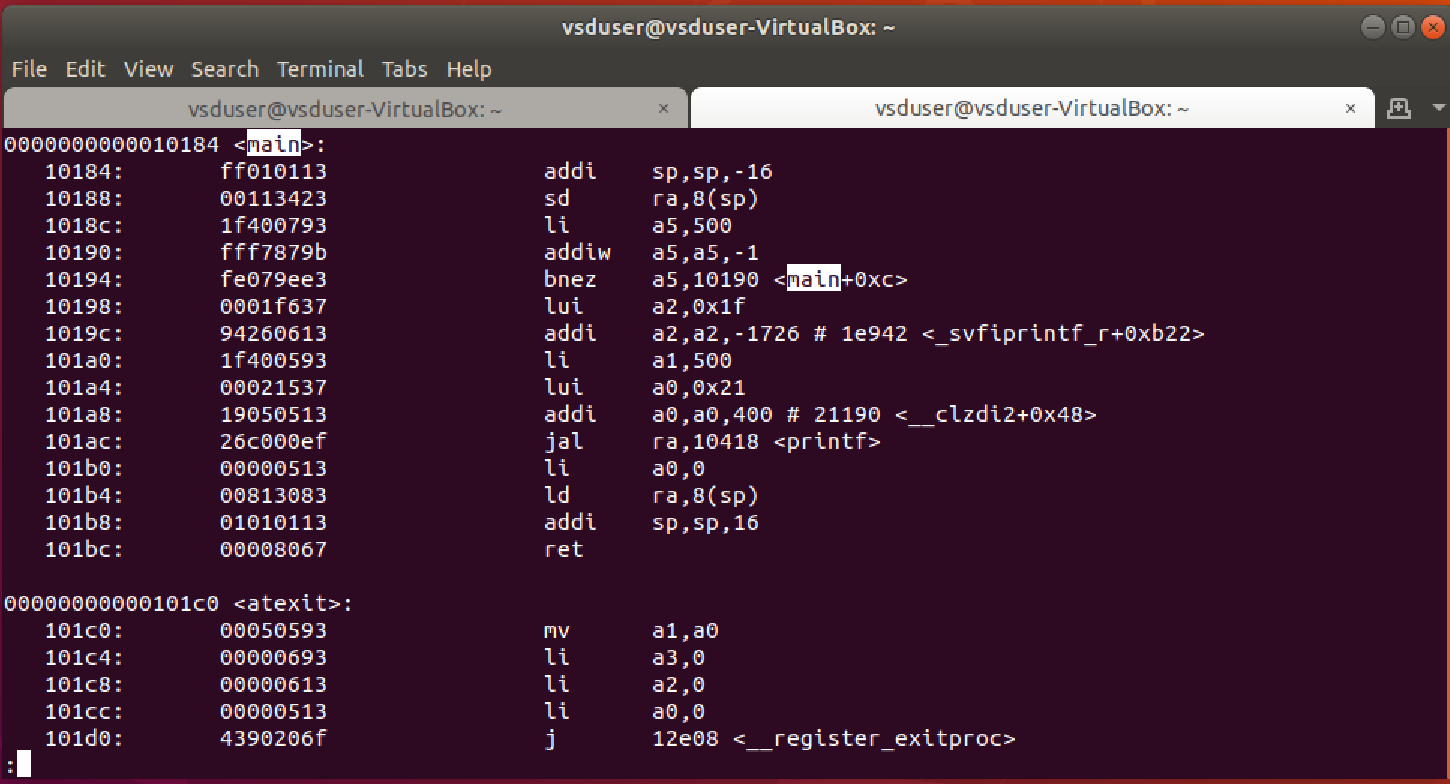
**Type /main**

****

**Output window after typing /main and then pressing enter that takes us to the code location for main**

****

**Highlight main and press n to look out for all the matches of ‘main’**

****

**Starting address of main is 10184 and the next block is at 101c0  
To find the total memory locations (space) taken by main 101c0 – 10184 = 3C (These values are in hex)  
3C (hex) = 60 (decimal)  
No of instructions = (101C0 – 10184)/4 = 15 (divide by 4 because each instruction is taking 4 bytes)**

**The number of assembly instructions generated is 15 with option -O1**

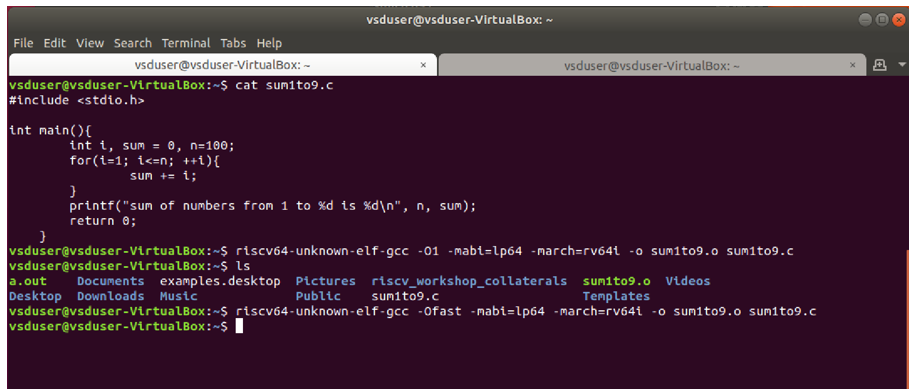
**press q to come out of the file description and come back to the prompt**

**1B.5 Do step 1B1 with -Ofast option to examine and compare the memory space used with -O1 and -Ofast options**

Execute the command with -Ofast option in **Tab 1**

**$ Riscv64-unknown-elf-gcc -Ofast -mabl=lp64 -march=rv64i -o sum1to9.o sum1to9.c**

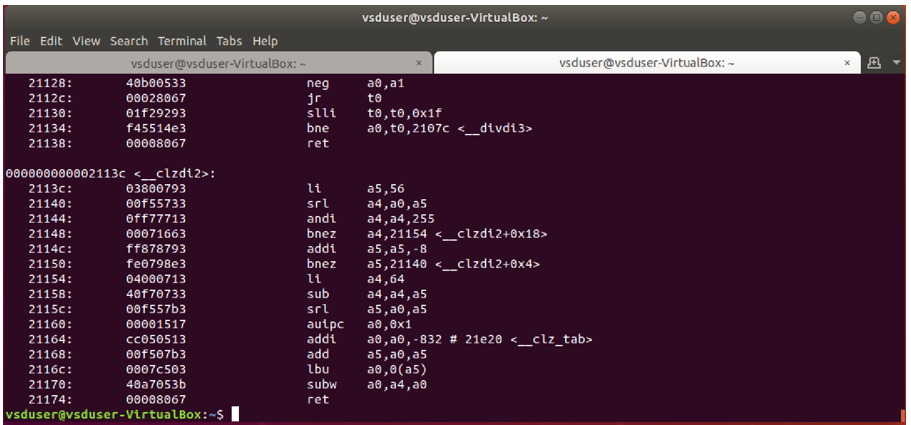
(NOTE O is in caps for -Ofast as well as -O1 options only)

**[](https://private-user-images.githubusercontent.com/51965739/379336144-d176ded5-63ac-4488-8aa7-26720e23ed63.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..Ot0bM_DEAt60WHjrZVZxPz2_crD-d_-kQWN9wJitBpk)**

**1B.5 Run the commands from 1B2 – 1B4 again In tab 2 to examine the assembly code generated**

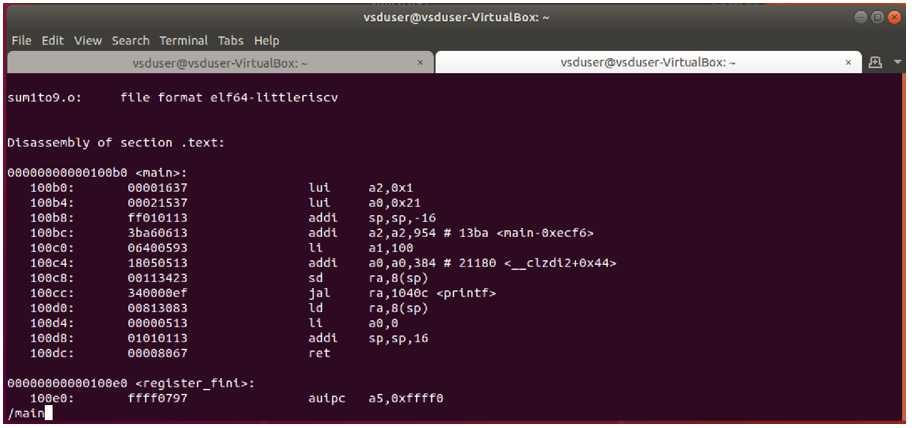
**$ riscv64-unknown-elf-objdump -d sum1to9.o**

Output after executing the command

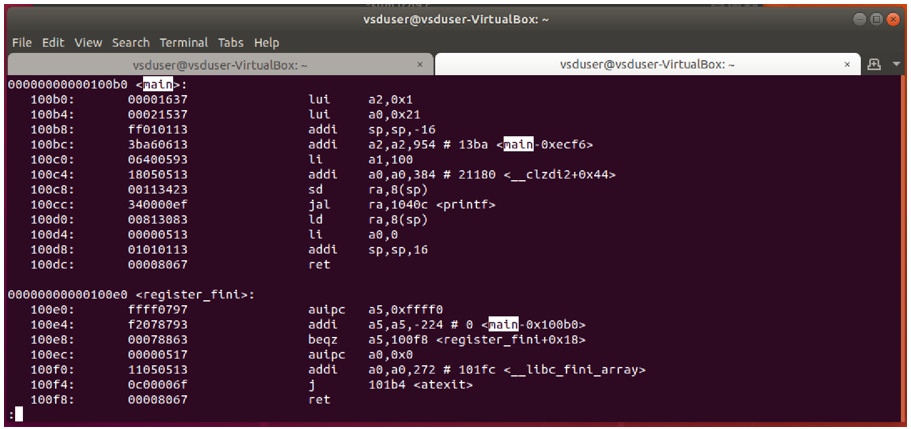
**[](https://private-user-images.githubusercontent.com/51965739/379336346-3e8a2065-f5ee-48d8-b4f8-99d66314a04d.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..K9kE73B9bMPIP3feGz7shwmGD5Rxfd-Ugk-BZ98kwkU)**

**$ riscv64-unknown-elf-objdump -d sum1to9.o | less**

Output after executing the command

**[](https://private-user-images.githubusercontent.com/51965739/379336573-678d4fae-351e-450f-a4da-2059b7b7d393.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..VYpxxKrSxIN4M-LVH2HPbrSwvaxjzN_hZL1Hoek1Oe4)type /main and press enter**

**Highlight main and press n to look out for all the matches of ‘main’**

**[](https://private-user-images.githubusercontent.com/51965739/379337350-ebee73fc-e401-470d-b174-c9e12e0e202e.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..dMOu0i0QY_EPdBPYmsj9ETh-7WFoh5MIydjtTrCKRVM)**

**Starting address of main is 100b0 and the next block is at 100e0  
To find the total memory locations (space) taken by main 100e0 – 100bo = 30 (These values are in hex)  
3C (hex) = 48 (decimal)  
No of instructions = (101C0 – 10184)/4 = 12 (divide by 4 because each instruction is taking 4 bytes)**

**The number of assembly instructions generated is 12 with option -Ofast**

press q to come out of the file description to come back to the prompt

**The number of assembly instructions generated is 12 with option -Ofast**

**The number of assembly instructions generated is 15 with option -O1**