Week 1 Task:
1 Install & Sanity-Check the Toolchain

📦 1. Toolchain Extraction

The compressed toolchain archive was extracted using the following command:

```
cd ~/Downloads
tar -xzf riscv-toolchain-rv32imac-x86_64-ubuntu.tar.gz
```

This created a directory named `riscv` in `~/Downloads`, which contains the necessary binaries (`gcc`, `objdump`, `gdb`, etc.).

🔧 2. Environment Setup

To make the RISC-V binaries available system-wide, the path was added to the `~/.bashrc` file:

```
echo 'export PATH=$HOME/Downloads/riscv/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

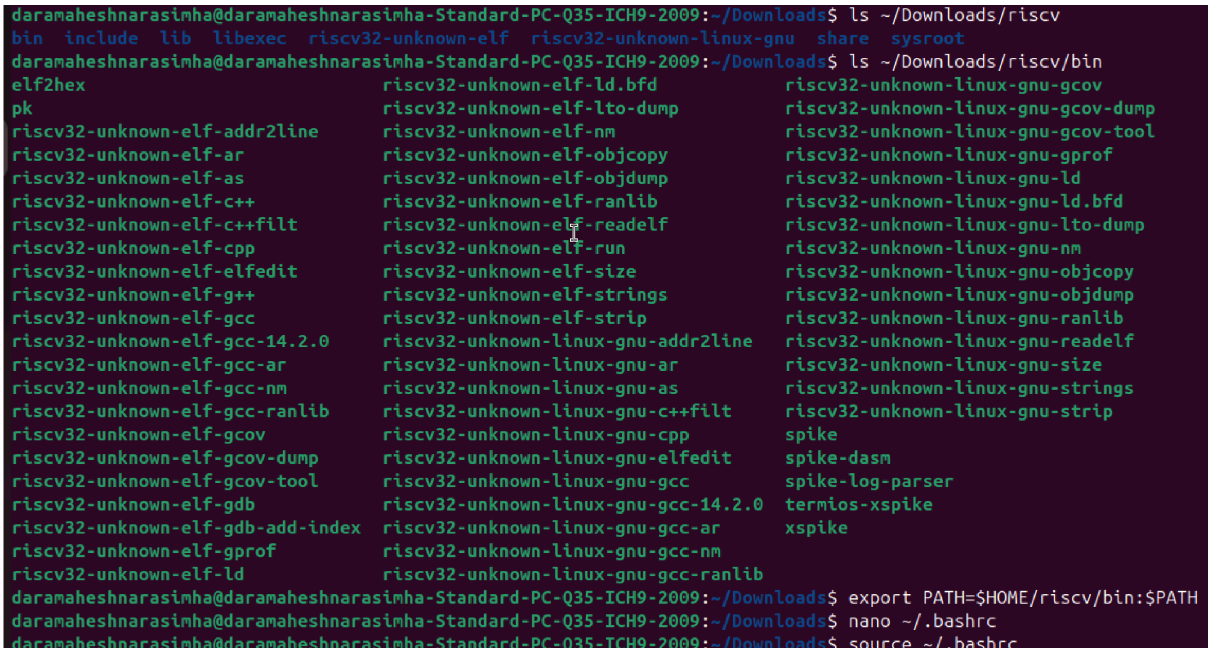
This ensures the tools are available in every terminal session.

✅ 3. Toolchain Verification

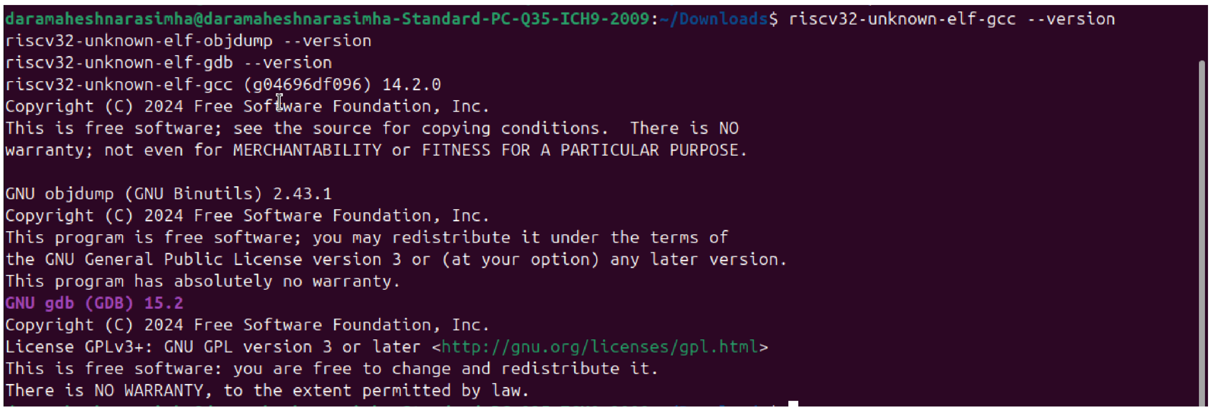
The following commands were used to verify that the toolchain was installed correctly:

```
riscv32-unknown-elf-gcc --version
riscv32-unknown-elf-objdump --version
riscv32-unknown-elf-gdb –version
```

`output:`



```
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ ls ~/Downloads/riscv
bin  include  lib  libexec  riscv32-unknown-elf  riscv32-unknown-linux-gnu  share  sysroot
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ ls ~/Downloads/riscv/bin
elf2hex                            riscv32-unknown-elf-ld.bfd           riscv32-unknown-linux-gnu-gcov
pk                                 riscv32-unknown-elf-lto-dump         riscv32-unknown-linux-gnu-gcov-dump
riscv32-unknown-elf-addr2line      riscv32-unknown-elf-nm               riscv32-unknown-linux-gnu-gcov-tool
riscv32-unknown-elf-ar             riscv32-unknown-elf-objcopy          riscv32-unknown-linux-gnu-gprof
riscv32-unknown-elf-as             riscv32-unknown-elf-objdump          riscv32-unknown-linux-gnu-ld
riscv32-unknown-elf-c++            riscv32-unknown-elf-ranlib           riscv32-unknown-linux-gnu-ld.bfd
riscv32-unknown-elf-c++filt        riscv32-unknown-elf-readelf          riscv32-unknown-linux-gnu-lto-dump
riscv32-unknown-elf-cpp            riscv32-unknown-elf-run              riscv32-unknown-linux-gnu-nm
riscv32-unknown-elf-elfedit        riscv32-unknown-elf-size             riscv32-unknown-linux-gnu-objcopy
riscv32-unknown-elf-g++            riscv32-unknown-elf-strings          riscv32-unknown-linux-gnu-objdump
riscv32-unknown-elf-gcc            riscv32-unknown-elf-strip            riscv32-unknown-linux-gnu-ranlib
riscv32-unknown-elf-gcc-14.2.0     riscv32-unknown-linux-gnu-addr2line  riscv32-unknown-linux-gnu-readelf
riscv32-unknown-elf-gcc-ar         riscv32-unknown-linux-gnu-ar         riscv32-unknown-linux-gnu-size
riscv32-unknown-elf-gcc-nm         riscv32-unknown-linux-gnu-as         riscv32-unknown-linux-gnu-strings
riscv32-unknown-elf-gcc-ranlib     riscv32-unknown-linux-gnu-c++filt    riscv32-unknown-linux-gnu-strip
riscv32-unknown-elf-gcov           riscv32-unknown-linux-gnu-cpp        spike
riscv32-unknown-elf-gcov-dump      riscv32-unknown-linux-gnu-elfedit    spike-dasm
riscv32-unknown-elf-gcov-tool      riscv32-unknown-linux-gnu-gcc        spike-log-parser
riscv32-unknown-elf-gdb            riscv32-unknown-linux-gnu-gcc-14.2.0 termios-xspike
riscv32-unknown-elf-gdb-add-index  riscv32-unknown-linux-gnu-gcc-ar     xspike
riscv32-unknown-elf-gprof          riscv32-unknown-linux-gnu-gcc-nm
riscv32-unknown-elf-ld             riscv32-unknown-linux-gnu-gcc-ranlib
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ export PATH=$HOME/riscv/bin:$PATH
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ nano ~/.bashrc
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ source ~/.bashrc
```



```
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ riscv32-unknown-elf-gcc --version
riscv32-unknown-elf-objdump --version
riscv32-unknown-elf-gdb --version
riscv32-unknown-elf-gcc (g04696df096) 14.2.0
Copyright (C) 2024 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

GNU objdump (GNU Binutils) 2.43.1
Copyright (C) 2024 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.
GNU gdb (GDB) 15.2
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

2 Compile “Hello, RISC-V”:

🎯 Objective

Compile a minimal C program (`hello.c`) that prints "Hello, RISC-V!" targeting the RISC-V `rv32imac` ISA, producing an ELF binary. Confirm the binary is correctly built for the RISC-V 32-bit platform.

🧾 Source Code: `hello.c`

```
#include <stdio.h>

int main() {

    printf("Hello, RISC-V!\n");

    return 0;
```

}

🛠 Compilation Command

The following command was executed from the terminal inside the `~/Downloads` directory:

```
riscv32-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -o hello.elf hello.c
```

Explanation of Flags:

Flag	Description
`-march=rv32imac`	Targets the 32-bit RISC-V ISA with `I`, `M`, `A`, and `C` extensions.
`-mabi=ilp32`	Sets the ABI: integers, longs, and pointers are 32-bit.
`-o hello.elf`	Output binary file in ELF format.
`hello.c`	Input source file.

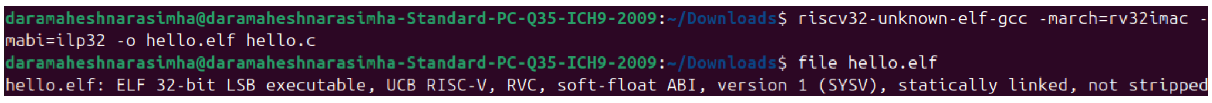
Binary Verification

After successful compilation, the ELF binary was verified using:

```
file hello.elf
```

The RISC-V "Hello, World" program was successfully compiled into a valid ELF binary using the correct architecture (`rv32imac`) and ABI (`ilp32`). This confirms the RISC-V GCC toolchain is working correctly and capable of building programs for 32-bit RISC-V embedded targets.

OUTPUT:



```
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ riscv32-unknown-elf-gcc -march=rv32imac -
mabi=ilp32 -o hello.elf hello.c
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ file hello.elf
hello.elf: ELF 32-bit LSB executable, UCB RISC-V, RVC, soft-float ABI, version 1 (SYSV), statically linked, not stripped
```

3.From C to Assembly

🎯 Objective

Generate the assembly code (`.s`) for a simple "Hello, RISC-V!" C program using the RISC-V GCC cross-compiler. Analyze the prologue and epilogue of the `main` function to understand how RISC-V manages the stack and function calls.

🛠 Step 1: Generate the. s Assembly File

```
riscv32-unknown-elf-gcc -S -O0 -march=rv32imac -mabi=ilp32 hello.c
```

Explanation of Flags:

Flag	Description
`-S`	Generate assembly code instead of object or ELF.

Flag	Description
`-O0`	No compiler optimizations — output is clearer.
`-march=rv32imac`	Target RISC-V architecture with I, M, A, C extensions.
`-mabi=ilp32`	Use 32-bit integer ABI.

Result: This command generated a file called `hello.s` in the working directory.

📄 Step 2: View the `main` Function in `hello.s`

Excerpt from the generated file (`hello.s`):

```
asm
CopyEdit
main:
    addi    sp,sp,-16
    sw      ra,12(sp)
    sw      s0,8(sp)
    addi    s0,sp,16
    ...
    lw      ra,12(sp)
    lw      s0,8(sp)
    addi    sp,sp,16
    ret
```

📘 Step 3: Explain the Prologue and Epilogue

🔹 *Function Prologue (at the beginning of `main`)*

Instruction	Purpose
`addi sp, sp, -16`	Allocate 16 bytes on the stack.
`sw ra, 12(sp)`	Save return address.
`sw s0, 8(sp)`	Save previous frame pointer.
`addi s0, sp, 16`	Establish new frame pointer.

These steps prepare the function's **stack frame** and preserve important registers, so the function can return safely and handle local variables.

🔹 *Function Epilogue (at the end of `main`)*

Instruction	Purpose
`lw ra, 12(sp)`	Restore return address.
`lw s0, 8(sp)`	Restore old frame pointer.
`addi sp, sp, 16`	Deallocate stack space.
`ret`	Return to caller using restored `ra`.

These steps **clean up** the function's stack frame before returning, restoring the program state.

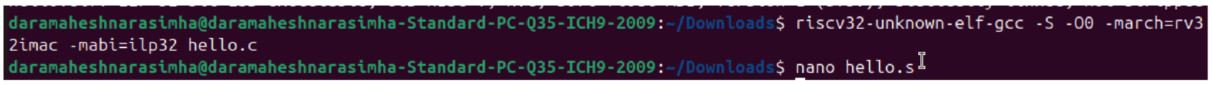
🧠 Why This Matters

The RISC-V calling convention requires each function to:

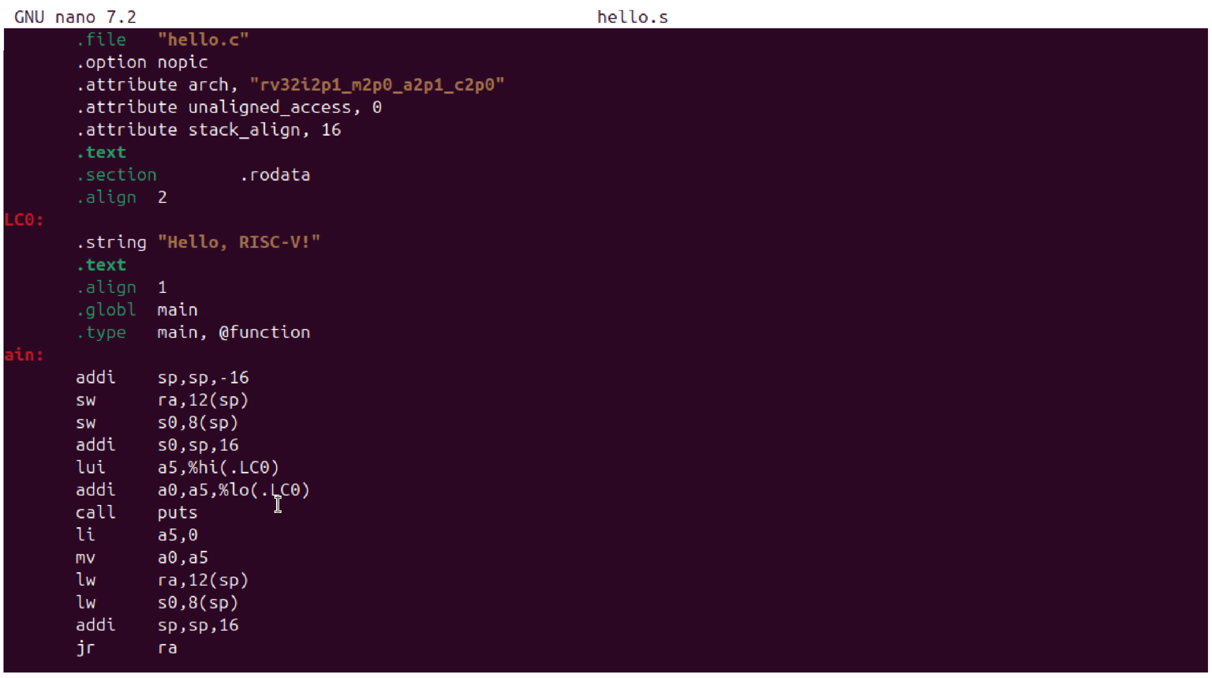
- Preserve the return address (`ra`) and frame pointer (`s0`) if they are used.
- Use the stack to manage local variables and maintain state.
- Follow a consistent stack layout to support debugging and nested function calls.

This understanding is **essential** for low-level programming, debugging, and writing RISC-V assembly directly.

OUTPUT:



```
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ riscv32-unknown-elf-gcc -S -O0 -march=rv3
2imac -mabi=ilp32 hello.c
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ nano hello.s
```



```
 GNU nano 7.2                                         hello.s
        .file   "hello.c"
        .option nopic
        .attribute arch, "rv32i2p1_m2p0_a2p1_c2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .section        .rodata
        .align  2
LC0:
        .string "Hello, RISC-V!"
        .text
        .align  1
        .globl  main
        .type   main, @function
ain:
        addi    sp,sp,-16
        sw      ra,12(sp)
        sw      s0,8(sp)
        addi    s0,sp,16
        lui     a5,%hi(.LC0)
        addi    a0,a5,%lo(.LC0)
        call    puts
        li      a5,0
        mv      a0,a5
        lw      ra,12(sp)
        lw      s0,8(sp)
        addi    sp,sp,16
        jr      ra
```

4. Hex Dump & Disassembly:

Tools used:

- `riscv32-unknown-elf-objdump` — for disassembly
- `riscv32-unknown-elf-objcopy` — for format conversion

📁 Starting File

You have: hello.elf ← compiled ELF binary from earlier steps

🔷 1. Disassemble the ELF (Human-Readable RISC-V Instructions)

🔧 Command:

riscv32-unknown-elf-objdump -d hello.elf > hello.dump

This disassembles all executable sections and saves the output to `hello.dump`.

📄 Sample Output from `hello.dump`:

```
00010074 <main>:

   10074: 1141              addi    sp,sp,-16

   10076: c606              sw      ra,12(sp)

   10078: c422              sw      s0,8(sp)

   1007a: 842e              mv      s0,sp
```

🔷 2. Convert ELF to Intel HEX (Raw Format for ROM/Flash)

🔧 Command:

```
riscv32-unknown-elf-objcopy -O ihex hello.elf hello.hex
```

This creates `hello.hex`, a machine-readable file commonly used for flashing to embedded devices or viewing the binary contents.

📄 Sample Lines from `hello.hex`:

```
:10000000004111141C606C422842E4505C226C226E5

:00000001FF
```

Each line starts with `:` and follows Intel HEX format:

- **Byte count**, **address**, **record type**, **data**, and **checksum**

Tools like ROM programmers or QEMU can use `.hex` to simulate or burn the binary.

🔷 1. View Disassembly in the Terminal

```
riscv32-unknown-elf-objdump -d hello.elf
```

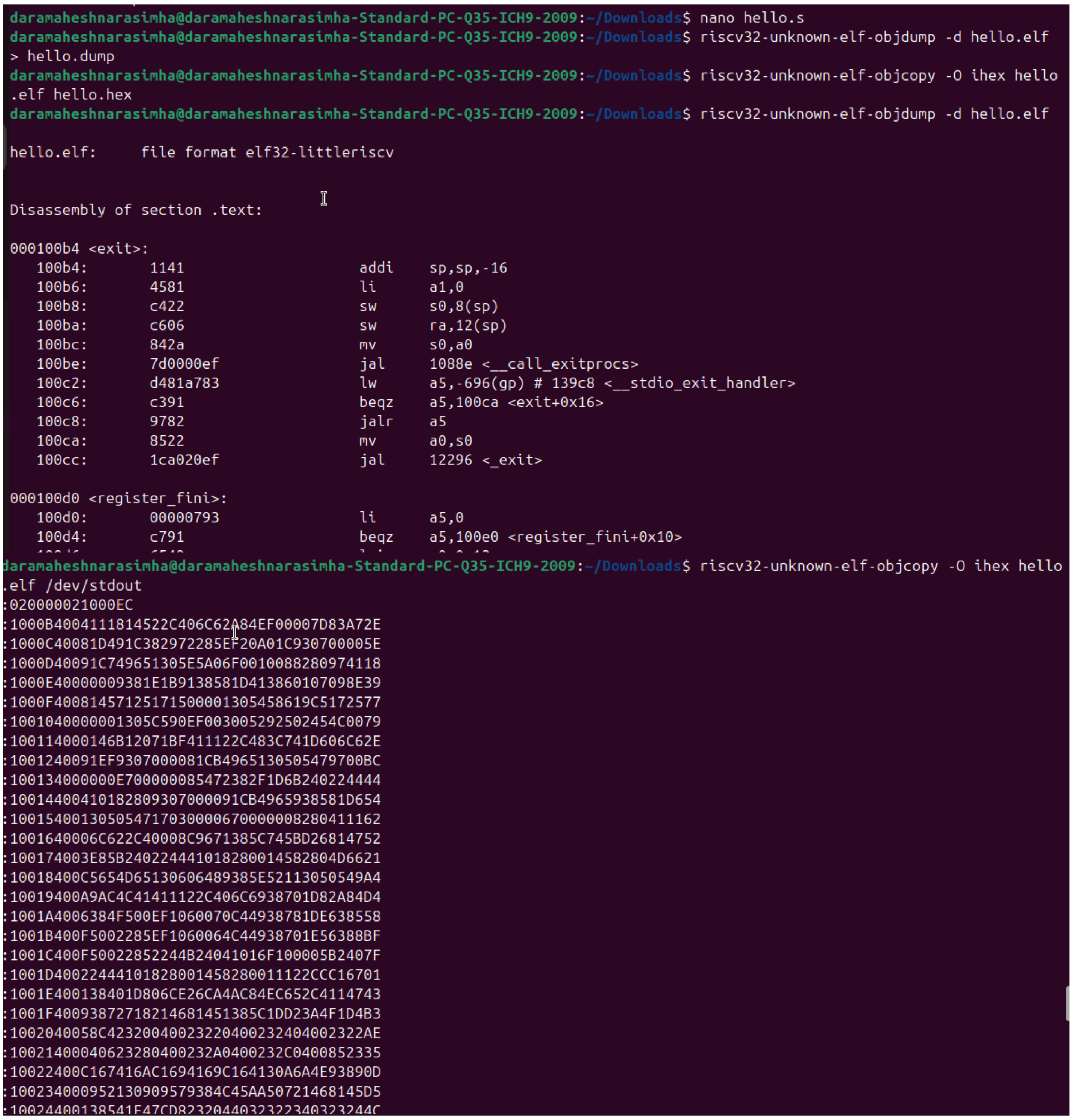
#This prints the disassembly of your ELF file directly to the terminal.

🔷 2. View HEX Output in the Terminal

```
riscv32-unknown-elf-objcopy -O ihex hello.elf /dev/stdout
```

#This prints the HEX version of your ELF file (Intel HEX format) directly to the terminal.

OUTPUT:



```
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ nano hello.s
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ riscv32-unknown-elf-objdump -d hello.elf
> hello.dump
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ riscv32-unknown-elf-objcopy -O ihex hello
.elf hello.hex
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ riscv32-unknown-elf-objdump -d hello.elf

hello.elf:     file format elf32-littleriscv


Disassembly of section .text:

000100b4 <exit>:
   100b4:       1141                    addi    sp,sp,-16
   100b6:       4581                    li      a1,0
   100b8:       c422                    sw      s0,8(sp)
   100ba:       c606                    sw      ra,12(sp)
   100bc:       842a                    mv      s0,a0
   100be:       7d0000ef                jal     1088e <__call_exitprocs>
   100c2:       d481a783                lw      a5,-696(gp) # 139c8 <__stdio_exit_handler>
   100c6:       c391                    beqz    a5,100ca <exit+0x16>
   100c8:       9782                    jalr    a5
   100ca:       8522                    mv      a0,s0
   100cc:       1ca020ef                jal     12296 <_exit>

000100d0 <register_fini>:
   100d0:       00000793                li      a5,0
   100d4:       c791                    beqz    a5,100e0 <register_fini+0x10>
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ riscv32-unknown-elf-objcopy -O ihex hello
.elf /dev/stdout
:020000021000EC
:1000B4004111814522C406C62A84EF00007D83A72E
:1000C40081D491C382972285EF20A01C930700005E
:1000D40091C749651305E5A06F0010088280974118
:1000E40000009381E1B9138581D413860107098E39
:1000F40081457125171500001305458619C5172577
:10010400000001305C590EF003005292502454C0079
:100114000146B12071BF411122C483C741D606C62E
:1001240091EF9307000081CB4965130505479700BC
:100134000000E700000085472382F1D6B240224444
:1001440041018280930700091CB4965938581D654
:10015400130505471703000067000000828041162
:1001640006C622C40008C9671385C745BD26814752
:100174003E85B2402244410182800145828 04D6621
:10018400C5654D6513060648938 5E52113050549A4
:10019400A9AC4C41411122C406C6938701D82A84D4
:1001A4006384F500EF1060070C44938781DE638558
:1001B400F5002285EF1060064C44938701E56388BF
:1001C400F5002285 2244B24041016F100005B2407F
:1001D4002244410182800145828 0011122CCC16701
:1001E400138401D806CE26CA4AC84EC652C4114743
:1001F40093872718214681451385C1DD23A4F1D4B3
:1002040058C4232004002322040023240400 2322AE
:100214000406232804002 32A0400232C0400852335
:10022400C167416AC1694169C164130A6A4E93890D
:10023400095213090957938 4C45AA50721468145D5
:10024400138541F47CD8232044032322340323244C
```

5 ABI & Register Cheat-Sheet:

Here’s a complete **RV32I Register ABI Cheat-Sheet** including all 32 registers, their names, and calling convention roles — essential for understanding RISC-V assembly and function behavior.

🧠 RISC-V RV32 Register Cheat-Sheet

Register	ABI Name	Role / Description	Saved by
x0	zero	Constant zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary	Caller
x6	t1	Temporary	Caller

Register	ABI Name	Role / Description	Saved by
`x7`	`t2`	Temporary	Caller
`x8`	`s0/fp`	Saved register / Frame pointer	Callee
`x9`	`s1`	Saved register	Callee
`x10`	`a0`	Function argument 0 / return value	Caller
`x11`	`a1`	Function argument 1 / return value	Caller
`x12`	`a2`	Function argument 2	Caller
`x13`	`a3`	Function argument 3	Caller
`x14`	`a4`	Function argument 4	Caller
`x15`	`a5`	Function argument 5	Caller
`x16`	`a6`	Function argument 6	Caller
`x17`	`a7`	Function argument 7	Caller
`x18`	`s2`	Saved register	Callee
`x19`	`s3`	Saved register	Callee
`x20`	`s4`	Saved register	Callee
`x21`	`s5`	Saved register	Callee
`x22`	`s6`	Saved register	Callee
`x23`	`s7`	Saved register	Callee
`x24`	`s8`	Saved register	Callee
`x25`	`s9`	Saved register	Callee
`x26`	`s10`	Saved register	Callee
`x27`	`s11`	Saved register	Callee
`x28`	`t3`	Temporary	Caller
`x29`	`t4`	Temporary	Caller
`x30`	`t5`	Temporary	Caller
`x31`	`t6`	Temporary	Caller

🔁 Summary: Calling Convention Rules

🟢 Argument and Return Registers

- **`a0–a7` (x10–x17):**
  - Used to **pass arguments** to functions.
  - `a0/a1` also used for **return values**.
  - **Caller-saved** — caller must save them if it needs to preserve them.

🔵 Saved Registers (callee-saved)

- **`s0–s11` (x8–x9, x18–x27):**
  - Must be **preserved by the callee**.
  - Often used for long-term variables or frame pointers.

🔴 Temporary Registers (caller-saved)

- **`t0–t6` (x5–x7, x28–x31):**
    - **Not preserved** — functions can overwrite these freely.
    - Used for intermediate computations.

⚙️ Special Registers

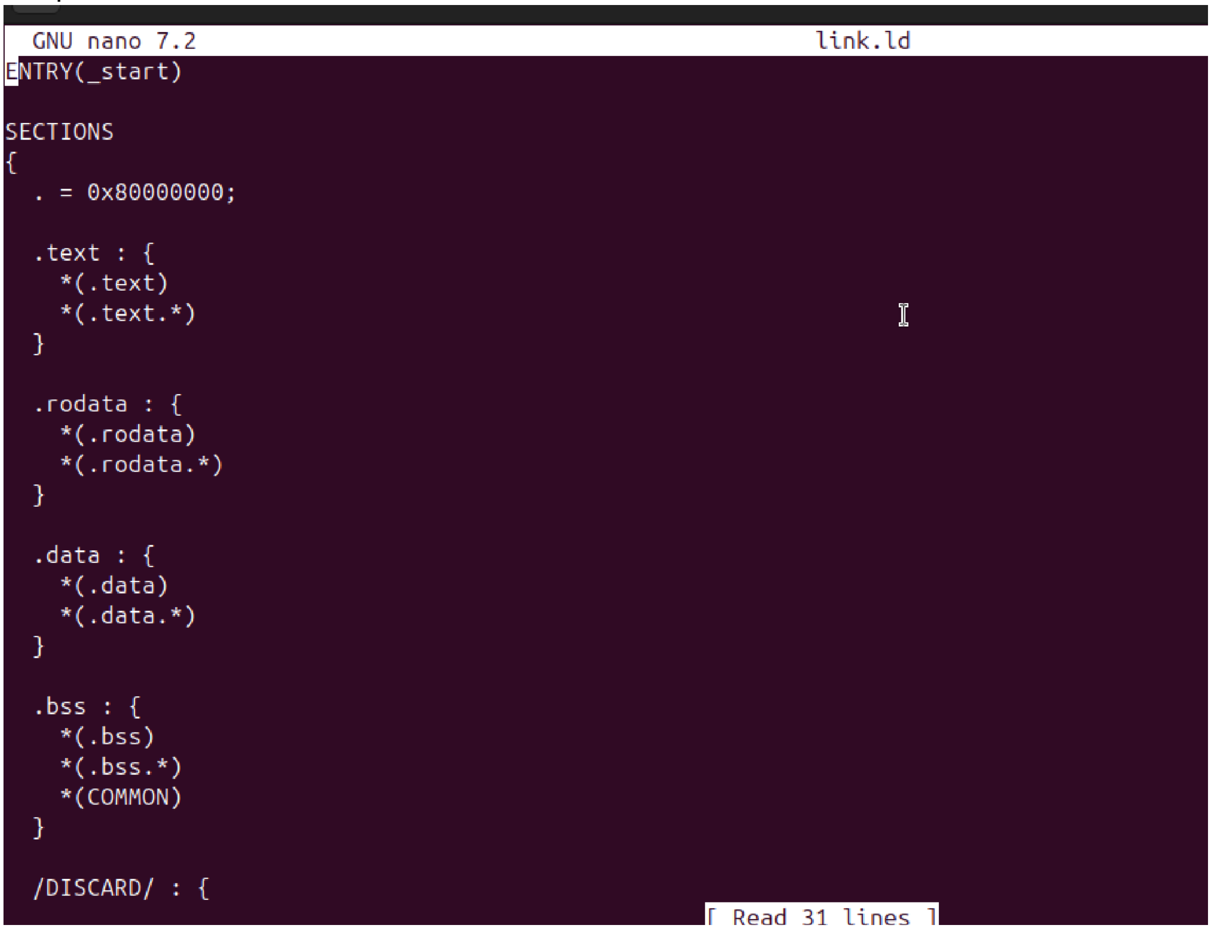
- `zero (x0)`: Always 0 — reads as 0, writes are ignored.
- `ra (x1)`: Return address — set by `jal`/`jalr`, used by `ret`.
- `sp (x2)`: Stack pointer — must always point to the top of the stack.
- `gp (x3)`, `tp (x4)`: Used for global and thread-local data.

6. Stepping with GDB:

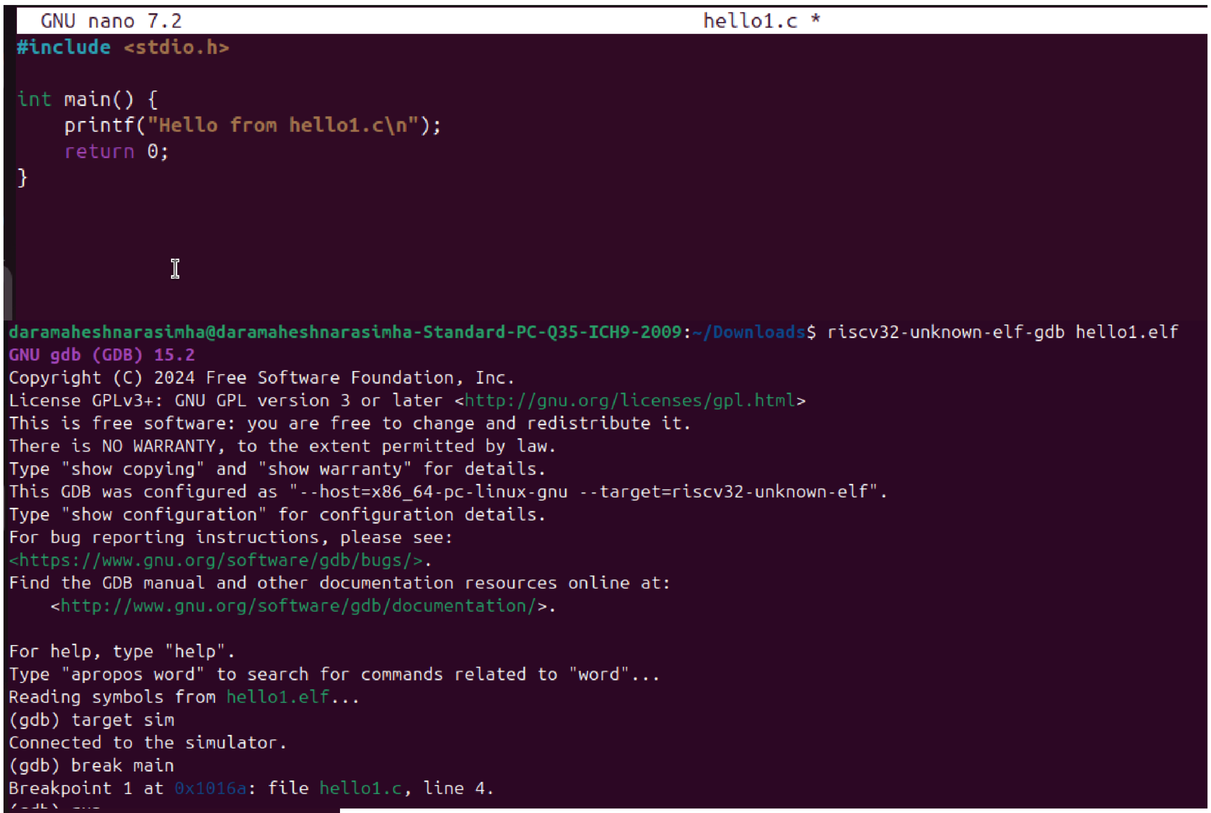
Debugging with GDB:

riscv32-unknown-elf-gdb hello1.elf
(gdb) target sim
(gdb) break main
(gdb) info reg a0

✅ Successfully compiled a RISC-V program with standard GCC.
Outputs:

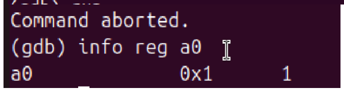


```
  GNU nano 7.2                                      link.ld
ENTRY(_start)

SECTIONS
{
  . = 0x80000000;

  .text : {
    *(.text)
    *(.text.*)
  }

  .rodata : {
    *(.rodata)
    *(.rodata.*)
  }

  .data : {
    *(.data)
    *(.data.*)
  }

  .bss : {
    *(.bss)
    *(.bss.*)
    *(COMMON)
  }

  /DISCARD/ : {
                                                   [ Read 31 lines ]
```



```
  GNU nano 7.2                                              hello1.c *
#include <stdio.h>

int main() {
    printf("Hello from hello1.c\n");
    return 0;
}
```

```
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ riscv32-unknown-elf-gdb hello1.elf
GNU gdb (GDB) 15.2
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv32-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello1.elf...
(gdb) target sim
Connected to the simulator.
(gdb) break main
Breakpoint 1 at 0x1016a: file hello1.c, line 4.
```



```
Command aborted.
(gdb) info reg a0
a0             0x1      1
```

7. Running Under an Emulator

Objective:

To execute a bare-metal RISC-V ELF binary using an emulator like **QEMU** or **Spike** and view UART output in the terminal.

Tools Used:

- `riscv32-unknown-elf-gcc` (Cross-compiler)
- `QEMU` (Emulator for RISC-V)
- `OpenSBI` (Open Source Supervisor Binary Interface)
- `hello2.c` (Bare-metal source file)
- Custom linker script `link1.ld`

Steps Performed:

Wrote Bare-Metal Program (`hello2.c`)
Defined low-level UART functions and a `_start` entry point:

```
#define UART0 0x10000000

void uart_putchar(char c) {
   *(volatile char *)UART0 = c;
}

void uart_puts(const char *s) {
    while (*s) {
        uart_putchar(*s++);
    }
}

void _start() {
    uart_puts("Hello RISC-V from UART!\n");
    while (1) {}
}
```

Created Custom Linker Script (`link1.ld`)

Ensured `.text` section loads **after OpenSBI firmware**, e.g.:

```
ENTRY(_start)

SECTIONS {
  . = 0x80010000;
  .text : { *(.text*) }
  .data : { *(.data*) }
  .bss  : { *(.bss*)  }
}
```

Compiled the Program

Using bare-metal options:

```
riscv32-unknown-elf-gcc -g -march=rv32imac -mabi=ilp32 \
  -T link1.ld -nostartfiles -nostdlib -o hello.elf hello2.c
```

Built OpenSBI Firmware

Installed 64-bit cross-compiler:

```
sudo apt install gcc-riscv64-linux-gnu
```

Compiled OpenSBI:

```
make CROSS_COMPILE=riscv64-linux-gnu- PLATFORM=generic
```

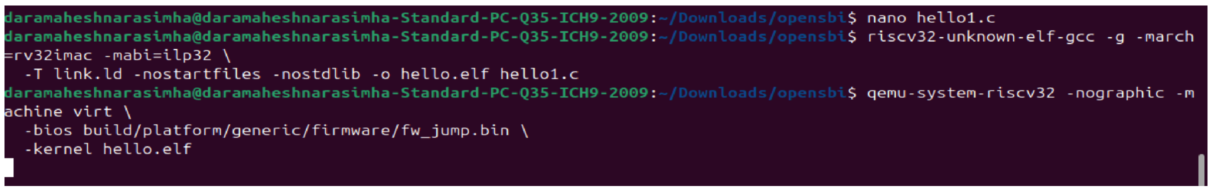
Ran Program in QEMU with OpenSBI

Used the following command:

```
qemu-system-riscv32 -nographic -machine virt \

  -bios build/platform/generic/firmware/fw_jump.bin \

  -kernel hello.elf
```

Output:

```
Hello RISC-V from UART!
```



```
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads/opensbi$ nano hello1.c
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads/opensbi$ riscv32-unknown-elf-gcc -g -march
=rv32imac -mabi=ilp32 \
  -T link.ld -nostartfiles -nostdlib -o hello.elf hello1.c
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads/opensbi$ qemu-system-riscv32 -nographic -m
achine virt \
  -bios build/platform/generic/firmware/fw_jump.bin \
  -kernel hello.elf
```

8.Exploring GCC Optimisation:

Step-by-Step Instructions

🔧 *1. Create a simple C file*

```
nano test.c
the code:
int square(int x) {
    int y = x * x;
    return y;
}
```

⚙️ 2. **Compile with -O0 (no optimization)**

```
riscv32-unknown-elf-gcc -S -O0 -o test_O0.s test.c
```

This creates the file test_O0.s, which contains assembly code with no optimization.

⚙️ 3. **Compile with -O2 (optimized)**

```
riscv32-unknown-elf-gcc -S -O2 -o test_O2.s test.c
```

This creates `test_O2.s`, the optimized assembly version.

👀 *4. View both assembly files*

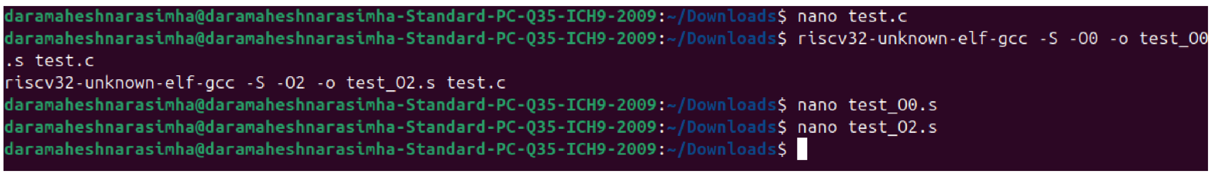
Use a terminal text viewer like `nano`, `less`, or `vim`:

nano test_O0.s

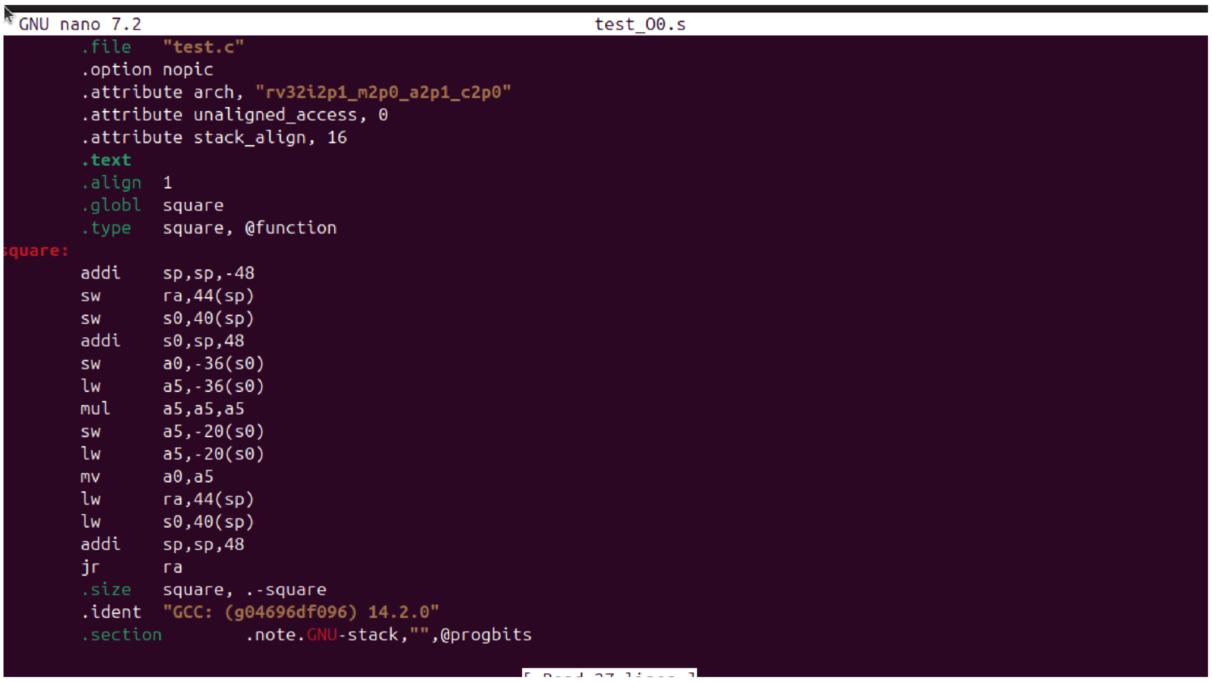
nano test_O2.s

🧠 Key Differences and Why They Happen

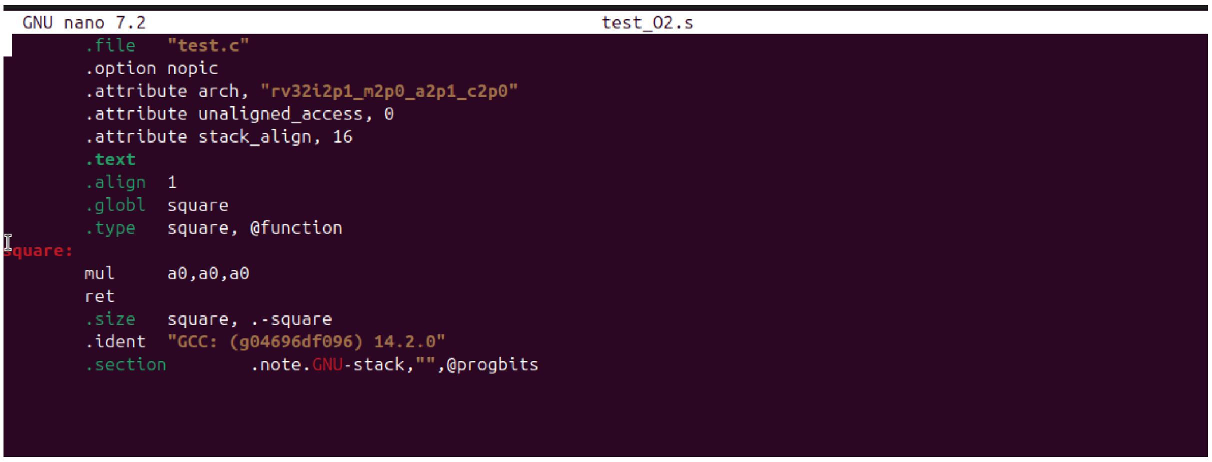
Feature	`-O0`	`-O2`	Explanation
Stack Frame	Yes	No	`-O0` saves registers and uses memory even if not needed. `-O2` avoids stack if possible.
Temporary Variables	All used	Removed	`-O2` eliminates variables like `y` and reuses registers.
Dead Code	Kept	Removed	GCC removes unused or unreachable code at `-O2`.
Register Allocation	Minimal	Aggressive	`-O2` keeps values in registers, not memory.
Inlining	No	Maybe (if appropriate)	Small functions may be inlined at `-O2`. Not shown here but common in other cases.



```
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ nano test.c
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ riscv32-unknown-elf-gcc -S -O0 -o test_O0
.s test.c
riscv32-unknown-elf-gcc -S -O2 -o test_O2.s test.c
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ nano test_O0.s
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ nano test_O2.s
daramaheshnarasimha@daramaheshnarasimha-Standard-PC-Q35-ICH9-2009:~/Downloads$ 
```



```
GNU nano 7.2                                                  test_O0.s
        .file   "test.c"
        .option nopic
        .attribute arch, "rv32i2p1_m2p0_a2p1_c2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .align  1
        .globl  square
        .type   square, @function
square:
        addi    sp,sp,-48
        sw      ra,44(sp)
        sw      s0,40(sp)
        addi    s0,sp,48
        sw      a0,-36(s0)
        lw      a5,-36(s0)
        mul     a5,a5,a5
        sw      a5,-20(s0)
        lw      a5,-20(s0)
        mv      a0,a5
        lw      ra,44(sp)
        lw      s0,40(sp)
        addi    sp,sp,48
        jr      ra
        .size   square, .-square
        .ident  "GCC: (g04696df096) 14.2.0"
        .section        .note.GNU-stack,"",@progbits
```



```
GNU nano 7.2                                                  test_O2.s
        .file   "test.c"
        .option nopic
        .attribute arch, "rv32i2p1_m2p0_a2p1_c2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .align  1
        .globl  square
        .type   square, @function
square:
        mul     a0,a0,a0
        ret
        .size   square, .-square
        .ident  "GCC: (g04696df096) 14.2.0"
        .section        .note.GNU-stack,"",@progbits
```