

Week 1

Task 1 Install & Sanity-Check the Toolchain

- Step-by-step tar -xzf ... command

```
pranay@pranay-VMware-Virtual-Platform:~$ cd ~/Downloads
pranay@pranay-VMware-Virtual-Platform:~/Downloads$ tar -xvzf riscv-toolchain-rv3
2imac-x86_64-ubuntu.tar.gz
opt/riscv/
opt/riscv/riscv32-unknown-linux-gnu/
opt/riscv/riscv32-unknown-linux-gnu/bin/
opt/riscv/riscv32-unknown-linux-gnu/bin/objcopy
opt/riscv/riscv32-unknown-linux-gnu/bin/ld
opt/riscv/riscv32-unknown-linux-gnu/bin/strip
```

- export PATH=\$HOME/riscv/bin:\$PATH lines for ~/.bashrc.

```
pranay@pranay-VMware-Virtual-Platform: ~/Downloads
GNU nano 7.2 /home/pranay/.bashrc
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi
export PATH="/opt/riscv/bin:$PATH"

```

^G Help	^O Write Out	^W Where Is	^K Cut	^T Execute	^C Location
^X Exit	^R Read File	^_\ Replace	^U Paste	^J Justify	^/ Go To Line

- Verification commands (riscv32-unknown-elf-gcc --version, etc.).

```
/home/pranay/Downloads/opt/riscv/bin/riscv32-unknown-elf-gcc
pranay@pranay-VMware-Virtual-Platform:~/Downloads$ nano ~/.bashrc
Downloads$ source ~/.bashrc
pranay@pranay-VMware-Virtual-Platform:~/Downloads$ riscv32-unknown-elf-gcc --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.

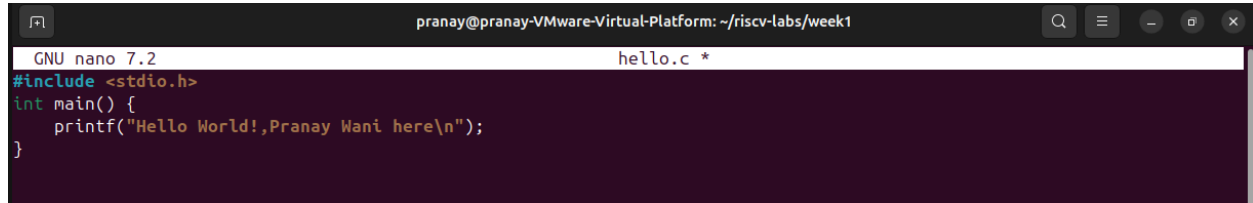
pranay@pranay-VMware-Virtual-Platform:~/Downloads$ riscv32-unknown-elf-objdump --version
GNU objdump (GNU Binutils) 2.43.1
Copyright (C) 2024 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.
pranay@pranay-VMware-Virtual-Platform:~/Downloads$ ^C
pranay@pranay-VMware-Virtual-Platform:~/Downloads$ riscv32-unknown-elf-objdump --version
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.
pranay@pranay-VMware-Virtual-Platform:~/Downloads$
```

Key Concepts

- PATH Modification: Prepending the toolchain's `bin` directory ensures the system prioritizes these binaries over others with the same name [36](#).
- ABI/ISA Flags: `-march=rv32imac` and `-mabi=ilp32` align with the target processor's architecture [15](#).
- Privilege Levels: System-wide installation (`/opt/riscv`) requires `sudo`, while user installations (`~/riscv`) avoid permission issues [1](#).

Task 2 Compile “Hello, RISC-V”

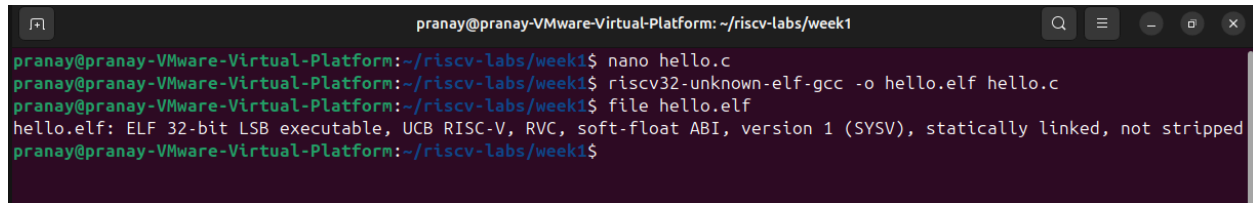
Copy the one-line `printf("Hello...")` program.



```
pranay@pranay-VMware-Virtual-Platform: ~/riscv-labs/week1
GNU nano 7.2 hello.c *
#include <stdio.h>
int main() {
    printf("Hello World!,Pranay Wani here\n");
}
```

Run `riscv32-unknown-elf-gcc -march=rv32imc -mabi=ilp32 -o hello.elf hello.c`.

Use file `hello.elf` to confirm it's 32-bit RISC-V.



```
pranay@pranay-VMware-Virtual-Platform: ~/riscv-labs/week1$ nano hello.c
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$ riscv32-unknown-elf-gcc -o hello.elf hello.c
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$ file hello.elf
hello.elf: ELF 32-bit LSB executable, UCB RISC-V, RVC, soft-float ABI, version 1 (SYSV), statically linked, not stripped
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$
```

Key Concepts Explained

1. Target Architecture Flags

- `-march=rv32imc`: Specifies RV32 base ISA with:
 - I: Base integer instruction set
 - M: Integer multiplication/division extension
 - C: Compressed instructions (16-bit variants) [36](#)
- `-mabi=ilp32`: Defines 32-bit ABI where:
 - `int`, `long`, and pointers are 32-bit
 - `long long` remains 64-bit [26](#)

2. Toolchain Components

- `riscv32-unknown-elf-gcc`: Cross-compiler targeting bare-metal RISC-V systems
- Requires newlib C library for embedded targets (no Linux syscalls) [86](#)

3. ELF Structure

The output `hello.elf` contains:

- RISC-V specific ELF headers
- `.text` section with compressed (C extension) instructions
- `.rodata` section storing the string literal
- Relocation metadata for embedded systems [86](#)

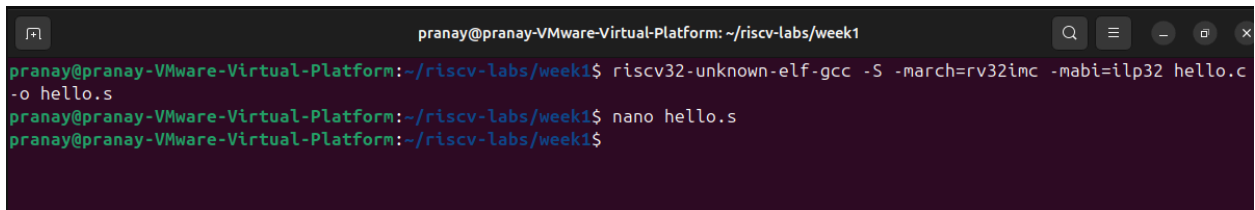
4. Header Dependency

`#include <stdio.h>` provides:

- Declaration for `printf()` function
- Standard I/O macros
- Required for linking against newlib's stdio implementation

Task 3 From C to Assembly

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

A terminal window with a dark background and light green text. The window title is 'pranay@pranay-VMware-Virtual-Platform: ~/riscv-labs/week1'. The terminal shows three lines of commands and their outputs: 1. 'riscv32-unknown-elf-gcc -S -march=rv32imc -mabi=ilp32 hello.c -o hello.s' followed by a new prompt. 2. 'nano hello.s' followed by a new prompt. 3. A new prompt without a command.

```
pranay@pranay-VMware-Virtual-Platform: ~/riscv-labs/week1
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$ riscv32-unknown-elf-gcc -S -march=rv32imc -mabi=ilp32 hello.c -o hello.s
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$ nano hello.s
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$
```

```
pranay@pranay-VMware-Virtual-Platform: ~/riscv-labs/week1
GNU nano 7.2 hello.s
.file "hello.c"
.option nopic
.attribute arch, "rv32i2p1_m2p0_c2p0"
.attribute unaligned_access, 0
.attribute stack_align, 16
.text
.section .rodata
.align 2
.LC0:
.string "Hello world!"
.text
.align 1
.globl main
.type main, @function
main:
    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    printf
    li      a5,0
    mv      a0,a5
    lw      ra,12(sp)
    lw      s0,8(sp)
    addi    sp,sp,16
    jr      ra
```

Prologue

1. **addi sp, sp, -16**

- Adjusts the stack pointer (**sp**) to create space for local variables/registers (16 bytes here).
- RISC-V stacks grow downward, so **sp** decreases to allocate space.

2. **sw ra, 12(sp)**

- Saves the return address (**ra**) to the stack.

-
- Necessary if `main` calls other functions (though not needed in this minimal example).

3. `sw s0, 8(sp)`

- Saves the frame pointer (`s0`), which points to the start of the stack frame.
- Optional but common in unoptimized code.

4. `addi s0, sp, 16`

- Sets the frame pointer (`s0`) to the top of the allocated stack space.
- Simplifies access to function arguments/local variables.

Epilogue

1. `lw ra, 12(sp)`

- Restores the return address from the stack to `ra`.

2. `lw s0, 8(sp)`

- Restores the original frame pointer (`s0`).

3. `addi sp, sp, 16`

- Deallocates the 16 bytes of stack space by moving `sp` back to its original position.

4. `ret`

- Returns to the address stored in `ra` (the caller function).

Key Architectural Concepts

1. Stack Management

- Stack Allocation: `addi sp, sp, -16` creates space for:
 - Return address (4 bytes)

- Saved registers (8 bytes)
- Stack alignment (RISC-V requires 16-byte alignment for function calls)
- Frame Pointer (s0): Acts as stable reference for local variables during stack adjustments

2. Calling Convention

- Return Address (ra): Must be preserved when making nested calls (though not needed here for simple main())
- Argument/Return Registers: a0-a7 for arguments, a0 for return value (hence `li a0, 0`)

3. Instruction Characteristics

- `addi`: Immediate addition (used for stack adjustments)
- `sw/lw`: Store/load word (32-bit) operations
- `ret`: Pseudonym for `j alr zero, ra, 0` (jump to return address)

4. Optimization Impact

The `-O0` flag ensures:

- Explicit stack frame creation
- No instruction reordering
- Preservation of all intermediate steps for debugging

Task 4 Hex Dump & disassembly

```
pranay@pranay-VMware-Virtual-Platform: ~/riscv-labs/week1
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$ riscv32-unknown-elf-objcopy -O ihex hello.elf hello.hex
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$ riscv32-unknown-elf-objdump -d hello.elf > hello.dump
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$ nano hello.dump
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$ nano hello.hex
pranay@pranay-VMware-Virtual-Platform:~/riscv-labs/week1$
```

Convert ELF to Intel Hex

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

```
pranay@pranay-VMware-Virtual-Platform: ~/riscv-labs/week1
GNU nano 7.2 hello.dump
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: 73a000ef     jal     107f8 <__call_exitprocs>
 100c2: f181a783     lw      a5,-232(gp) # 1dcb8 <__stdio_exit_handler>
 100c6: c391          beqz    a5,100ca <exit+0x16>
 100c8: 9782          jalr    a5
 100ca: 8522          mv      a0,s0
 100cc: 48e080ef     jal     1855a <_exit>

000100d0 <register_fini>:
 100d0: 00000793     li      a5,0
 100d4: c791          beqz    a5,100e0 <register_fini+0x10>
 100d6: 6551          lui     a0,0x14
 100d8: d5250513     addi    a0,a0,-686 # 13d52 <__libc_fini_array>
 100dc: 7ea0006f     j       108c6 <atexit>
 100e0: 8082          ret

000100e2 <_start>:
```

Disassemble the ELF

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


```

pranay@pranay-VMware-Virtual-Platform: ~/riscv-labs/week1
GNU nano 7.2 hello.dump
100cc:      48e080ef      jal      1855a <_exit>

000100d0 <register_fini>:
100d0:      00000793      li      a5,0
100d4:      c791      beqz    a5,100e0 <register_fini+0x10>
100d6:      6551      lui    a0,0x14
100d8:      d5250513     addi    a0,a0,-686 # 13d52 <__libc_fini_array>
100dc:      7ea0006f      j      108c6 <atexit>
100e0:      8082      ret

000100e2 <_start>:
100e2:      0000e197      auipc   gp,0xe
100e6:      cbe18193     addi    gp,gp,-834 # 1dda0 <__global_pointer$>
100ea:      f1818513     addi    a0,gp,-232 # 1dcb8 <__stdio_exit_handler>
100ee:      24418613     addi    a2,gp,580 # 1dfe4 <__BSS_END__>
100f2:      8e09      sub     a2,a2,a0
100f4:      4581      li      a1,0
100f6:      2da9      jal     10750 <memset>
100f8:      00000517     auipc   a0,0x0
100fc:      7ce50513     addi    a0,a0,1998 # 108c6 <atexit>
10100:      c519      beqz    a0,1010e <_start+0x2c>
10102:      00004517     auipc   a0,0x4
10106:      c5050513     addi    a0,a0,-944 # 13d52 <__libc_fini_array>
1010a:      7bc000ef     jal     108c6 <atexit>
1010e:      2be1      jal     106e6 <__libc_init_array>
10110:      4502      lw      a0,0(sp)
10112:      004c      addi    a1,sp,4
10114:      4601      li      a2,0

```

```

pranay@pranay-VMware-Virtual-Platform: ~/riscv-labs/week1
GNU nano 7.2 hello.hex
:020000021000EC
:1000B4004111814522C406C62A84EF00A07383A798
:1000C40081F191C382972285EF80E0489307000075
:1000D40091C75165130525D56F00A07E828097E1F5
:1000E4000009381E1CB138581F113864124098EAD
:1000F4008145A92D170500001305E57C19C5174591
:100104000000130505C5EF00C07BE12B02454C0040
:100114000146B12071BF411122C483C781F306C6D1
:1001240091EF9307000081CB71651305C5589700C3
:100134000000E70000008547238CF1F2B24022441E
:10014400410182809307000091CB71659385C1F3CF
:100154001305C55817030000670000008280411191
:1001640006C622C40008F167138587D0212E814773
:100174003E85B240224441018280014582807566F9
:10018400CD6575651306065A9385654D1305055BA4
:10019400A9AC4C41411122C406C6938741F52A8477
:1001A4006384F500EF30E0320C449387C1FB638530
:1001B400F5002285EF30E0314C4493874102638897
:1001C400F50022852244B24041016F308030B240B4
:1001D40022444101828001458280011122CCC16701
:1001E400138441F506CE26CA4AC84EC652C41147E6
:1001F4009387271821468145138501FB23ACF1F031
:1002040058C4232004002320400232404002322AE
:10021400040623280400232A0400232C04003D237D
:10022400C167416AC1694169C164130A4A4B938930
:10023400E94E1309E9539384A457A5072146814540
:10024400138581017CD823204403232234032324EF
:10025400240344D440CC2324040623260406232860

```

Detailed Breakdown of an Instruction

Example: `addi sp,sp,-16`

- **Address:** `0x10074`
Where this instruction resides in memory.
- **Opcode:** `1141` (hex)
 - In RISC-V, compressed instructions (16-bit) start with `0x0000` to `0xFFFF`.
 - `1141` in binary: `0001 0001 0100 0001`.
 - Decoded as:
 - **opcode:** `001` (C.ADDI16SP instruction in compressed format).
 - **imm:** `-16` (encoded as a 6-bit immediate).
 - **rd/rs1:** `sp` (stack pointer, register `x2`).
- **Mnemonic:** `addi`
Adds the immediate value to the source register and stores the result in the destination register.
- **Operands:** `sp,sp,-16`
 - `sp` (destination) = `sp` (source) + `-16` (immediate).
 - Adjusts the stack pointer to allocate 16 bytes of space.

Architecture Considerations

1. Instruction Length:
 - 32-bit: Base RV32I instructions (e.g., `lw`, `add`)
 - 16-bit: Compressed C extension (e.g., `c.addi`)
2. Endianness: RISC-V uses little-endian byte order
3. Address Alignment: Jump targets must be 4-byte aligned for non-compressed instructions

Key Tool Features

objdump Flags

-
- `-M no-aliases`: Shows raw instruction names (e.g., `jal` instead of `call`)
 - `-s`: Mixes source code with assembly (requires compiled with `-g`)
 - `-j .text`: Disassembles only code section

objcopy Formats

5 ABI & Register Cheat- Sheet

Table mapping x0-x31 to zero, ra, sp, gp, tp, t0-t6, s0-s11, a0-a7.

Reg #	ABI Name	Role / Convention
x0	zero	Hardwired to zero. Writes are ignored.
x1	ra	Return Address: Stores return address for jal/jalr (caller-saved).
x2	sp	Stack Pointer: Manages the stack (callee-saved).
x3	gp	Global Pointer: Points to global/static data (not used in standard code).
x4	tp	Thread Pointer: Used for thread-local storage (OS/kernel-specific).
x5	t0	Temporary 0: Caller-saved temporary register.
x6	t1	Temporary 1: Caller-saved temporary register.
x7	t2	Temporary 2: Caller-saved temporary register.
x8	s0/fp	Saved 0 / Frame Pointer: Callee-saved. Optional frame pointer.
x9	s1	Saved 1: Callee-saved. Preserved across function calls.
x10	a0	Argument 0 / Return Value 0: First function argument or return value.
x11	a1	Argument 1 / Return Value 1: Second function argument or return value.
x12	a2	Argument 2: Third function argument.
x13	a3	Argument 3: Fourth function argument.
x14	a4	Argument 4: Fifth function argument.

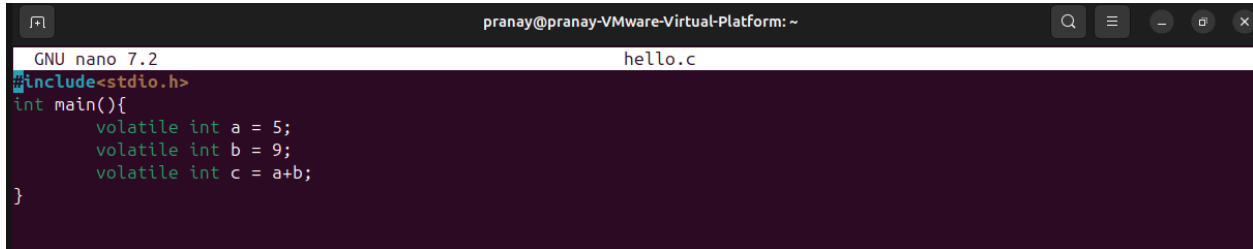
x15	a5	Argument 5: Sixth function argument.
x16	a6	Argument 6: Seventh function argument.
x17	a7	Argument 7: Eighth function argument (also used for syscall numbers).
x18	s2	Saved 2: Callee-saved.
x19	s3	Saved 3: Callee-saved.
x20	s4	Saved 4: Callee-saved.
x21	s5	Saved 5: Callee-saved.
x22	s6	Saved 6: Callee-saved.
x23	s7	Saved 7: Callee-saved.
x24	s8	Saved 8: Callee-saved.
x25	s9	Saved 9: Callee-saved.
x26	s10	Saved 10: Callee-saved.
x27	s11	Saved 11: Callee-saved.
x28	t3	Temporary 3: Caller-saved.
x29	t4	Temporary 4: Caller-saved.
x30	t5	Temporary 5: Caller-saved.
x31	t6	Temporary 6: Caller-saved.

Calling convention summary: a0-a7 = args/returns, s-regs callee-saved, t-regs

Register Type**Registers****Role**

Arguments	a0-a7	Pass function arguments (first 8 args). a0-a1 also hold return values.
Caller-Saved	t0-t6	Caller must save these before calling a function (volatile).
Callee-Saved	s0-s11	Callee must preserve these across function calls (non-volatile).
Special	zero, sp, gp, tp, ra	Hardwired or system roles (e.g., stack management, return address).

6 Stepping with GDB



```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2 hello.c  
#include<stdio.h>  
int main(){  
    volatile int a = 5;  
    volatile int b = 9;  
    volatile int c = a+b;  
}
```

riscv32-unknown-elf-gdb hello.elf

(gdb) target sim

(gdb) load

(gdb) break main

(gdb) run

```
pranay@pranay-VMware-Virtual-Platform: ~  
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv32-unknown-elf-gdb hello.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 hello.elf...  
(No debugging symbols found in hello.elf)  
(gdb) target sim  
Connected to the simulator.  
(gdb) load  
Loading section .text, size 0x48 lma 10094  
Start address 10094  
Transfer rate: 576 bits in <1 sec.  
(gdb) break main  
Breakpoint 1 at 0x100d8  
(gdb) run  
Starting program: /home/pranay/hello.elf  
  
Breakpoint 1, 0x000100d8 in main ()  
(gdb) █
```

7

```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2                                uart.c  
// uart.c  
#define UART0 0x10000000  
void uart_putc(char c) { *(volatile char*)UART0 = c; }  
void uart_puts(const char *s) { while (*s) uart_putc(*s++); }  
void main() {  
    uart_puts("Hello, UART!\n");  
    while (1);  
}
```

```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2                                startup.s  
.section .text  
.globl _start  
_start:  
    la sp, _stack_top  
    call main  
1: j 1b  
  
.section .bss  
.space 4096  
_stack_top:
```



```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2 linker.ld  
ENTRY(_start)  
SECTIONS {  
    . = 0x80000000;  
    .text : { *(.text*) }  
    .bss : { *(.bss*) }  
    . = ALIGN(4);  
    _stack_top = .;  
}
```

```
pranay@pranay-VMware-Virtual-Platform: ~  
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano uart.c  
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv32-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -nostdlib -T linker.  
ld -o uart.elf uart.c startup.s  
qemu-system-riscv32 -machine virt -nographic -bios none -kernel uart.elf  
/home/pranay/Downloads/opt/riscv/bin/../lib/gcc/riscv32-unknown-elf/14.2.0/../../../../riscv32-unknown-elf/bin/ld: warni  
ng: uart.elf has a LOAD segment with RWX permissions
```

8 Exploring GCC Optimisation

```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2 test.c  
// test.c  
  
int square(int x) {  
    int y = x * x;  
    return y;  
}  
  
int main() {  
    int a = 5;  
    int b = square(a);  
    return 0;  
}
```

```

pranay@pranay-VMware-Virtual-Platform: ~
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano test.c
(base) pranay@pranay-VMware-Virtual-Platform:~$ gcc -S -O0 test.c -o test_00.s
gcc -S -O2 test.c -o test_02.s
(base) pranay@pranay-VMware-Virtual-Platform:~$ vimdif test_00.s test_02.s
Command 'vimdiff' not found, but can be installed with:
sudo apt install vim          # version 2:9.1.0016-1ubuntu7.6, or
sudo apt install neovim      # version 0.7.2-8
sudo apt install vim-gtk3    # version 2:9.1.0016-1ubuntu7.6
sudo apt install vim-motif   # version 2:9.1.0016-1ubuntu7.6
sudo apt install vim-nox     # version 2:9.1.0016-1ubuntu7.6
(base) pranay@pranay-VMware-Virtual-Platform:~$

```

Feature	-O0	-O2
Function calls	Preserved (square remains)	Inlined into main
Stack usage	High (every var stored)	Minimal or none
Unused variables	Preserved	Eliminated
Register use	Minimal	Aggressive and efficient
Assembly length	Long and verbose	Short and optimized

9 Inline Assembly Basics

```

GNU nano 7.2                                riscv.h
// riscv.h
#ifndef _RISCV_H
#define _RISCV_H

#include <stdint.h>

static inline uint32_t rdcycle(void) {
    uint32_t c;
    // "volatile" = no compiler optimization/reordering
    // "csrr" = RISC-V CSR read instruction
    // "cycle" = CSR address 0xC00 (cycle counter)
    // "=r"(c) = store result in register → variable c
    asm volatile ("csrr %0, cycle" : "=r"(c));
    return c;
}

#endif

```

```
pranay@pranay-VMware-Virtual-Platform: ~
GNU nano 7.2 main.c
// main.c
#include "riscv.h"

int main() {
    uint32_t start = rdcycle();
    // Example: Measure a delay loop
    for (int i = 0; i < 100; i++) { asm volatile ("nop"); }
    uint32_t end = rdcycle();
    uint32_t cycles = end - start;

    // Print via UART (use your UART code from earlier)
    uart_puts("Cycles: ");
    uart_put_hex(cycles);
    uart_puts("\n");
    while(1);
}
```

```
Unpacking gcc-riscv64-unknown-elf (13.2.0-11ubuntu1+12) ...
Setting up binutils-riscv64-unknown-elf (2.42-1ubuntu1+6) ...
Setting up gcc-riscv64-unknown-elf (13.2.0-11ubuntu1+12) ...
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.4) ...
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -nostdlib -I. -o main.elf main.c startup.s
riscv.h: Assembler messages:
riscv.h:13: Error: unrecognized opcode `csrr a5,cycle', extension `zicsr' required
(base) pranay@pranay-VMware-Virtual-Platform:~$ qemu-system-riscv32 -machine virt -nographic -bios none -kernel main.elf
main.elf: No such file or directory
qemu-system-riscv32: could not load kernel 'main.elf'
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano main.c
(base) pranay@pranay-VMware-Virtual-Platform:~$ qemu-system-riscv32 -machine virt -nographic -bios none -kernel main.elf
main.elf: No such file or directory
qemu-system-riscv32: could not load kernel 'main.elf'
(base) pranay@pranay-VMware-Virtual-Platform:~$ qemu-system-riscv32 -machine virt -nographic -bios none -kernel main.elf
main.elf: No such file or directory
qemu-system-riscv32: could not load kernel 'main.elf'
(base) pranay@pranay-VMware-Virtual-Platform:~$
```

10 memory Mapped I/O Demo

```
pranay@pranay-VMware-Virtual-Platform: ~
GNU nano 7.2 gpio.c
#include <stdint.h>

int main(void) {
    // Declare GPIO register as volatile to prevent optimization
    volatile uint32_t *gpio = (volatile uint32_t*)0x10012000;

    // Toggle the GPIO (bitwise XOR with 0x1)
    *gpio ^= 0x1;

    // Infinite loop (bare-metal programs never exit)
    while (1);
}
```

```
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano gpio.c
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv32-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -nostdlib -O0 -o gpio.o elf gpio.c
/home/pranay/Downloads/opt/riscv/bin/../lib/gcc/riscv32-unknown-elf/14.2.0/../../../../riscv32-unknown-elf/bin/ld: warning: cannot find entry symbol _start; defaulting to 00010094
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv32-unknown-elf-objdump -d gpio.elf

gpio.elf:      file format elf32-littleriscv

Disassembly of section .text:

00010094 <main>:
10094:      1101          addi    sp,sp,-32
10096:      ce06          sw      ra,28(sp)
10098:      cc22          sw      s0,24(sp)
1009a:      1000          addi    s0,sp,32
1009c:      100127b7      lui     a5,0x10012
100a0:      fef42623      sw      a5,-20(s0)
100a4:      fec42783      lw      a5,-20(s0)
100a8:      439c          lw      a5,0(a5)
100aa:      0017c713      xori    a4,a5,1
100ae:      fec42783      lw      a5,-20(s0)
100b2:      c398          sw      a4,0(a5)
100b4:      a001          j       100b4 <main+0x20>
(base) pranay@pranay-VMware-Virtual-Platform:~$
```

Key Components Explained

1. `volatile` Keyword

- **Purpose:**

- Forces the compiler to generate a memory access **every time** `*gpio` is read/written.
- Prevents the compiler from:
 - Removing "redundant" stores (`*gpio ^= 0x1` → `*gpio = 0x1` if optimized).
 - Reordering or merging writes.

- **Why it's needed:**

GPIO registers are **memory-mapped I/O (MMIO)**. The compiler doesn't know these accesses have side effects (e.g., changing hardware state).

2. Pointer Type

- `uint32_t*`:

Ensures 32-bit aligned access (critical for MMIO registers).

- **Cast to volatile:**

Explicitly marks this pointer as referring to hardware (not regular RAM).

3. Toggle Operation

- ***gpio ^= 0x1:**

Bitwise XOR to flip the least significant bit. For example:

- If GPIO was 0x0, becomes 0x1.
- If GPIO was 0x1, becomes 0x0.

11 Linker Script 101

```
pranay@pranay-VMware-Virtual-Platform: ~
GNU nano 7.2 linker.ld
/* linker.ld */
ENTRY(_start)      /* Entry point symbol (defined in startup code) */

SECTIONS {
    . = 0x00000000; /* Flash starts at 0x00000000 */
    .text : {
        *(.text*) /* All code sections (.text, .text.*) */
    }

    . = 0x10000000; /* SRAM starts at 0x10000000 */
    .data : {
        *(.data*) /* All initialized data sections */
    }
}
```

```
pranay@pranay-VMware-Virtual-Platform: ~
GNU nano 7.2 main.c
#define UART_BASE ((volatile char*)0x10000000)

void uart_putc(char c) {
    *UART_BASE = c;
}

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

int main() {
    uart_puts("Hello, RISC-V!\n");
    while (1);
}
```

```

100b4:      a001                j      100b4 <main+0x20>
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano gpio.c
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano linker.ld
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano linker.ld
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv32-unknown-elf-gcc -T linker.ld -nostdlib -o program.elf startup.s main.c
main.c:5:10: fatal error: uart.h: No such file or directory
   5 | #include "uart.h"
     |           ^~~~~~
compilation terminated.
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano main.c
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv32-unknown-elf-gcc -T linker.ld -nostdlib -o program.elf startup.s main.c
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv32-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -nostdlib -T linker.ld -o uart.elf startup.s main.c
(base) pranay@pranay-VMware-Virtual-Platform:~$ qemu-system-riscv32 -machine virt -nographic -bios none -kernel uart.elf

```

Why Flash and SRAM Addresses Differ

Section	Typical Location	Purpose
<code>.text</code>	Flash (0x00000000)	Code: Non-volatile, read-only. Executes directly from Flash.
<code>.data</code>	SRAM (0x10000000)	Initialized Data: Volatile, read-write. Copied from Flash to SRAM at startup.
<ul style="list-style-type: none"> Flash: Retains data when power is off. Ideal for code and read-only data. SRAM: Faster read/write access. Required for variables that change at runtime. 		

Hardware Memory Mapping

- Flash:**
 - Non-volatile memory (retains data without power).
 - Typically mapped to lower addresses (e.g., `0x00000000`).
 - Stores code (`.text`), read-only data (`.rodata`), and initial values for variables (`.data`).

- **SRAM:**

- Volatile memory (loses data on power-off).
- Mapped to higher addresses (e.g., **0x10000000**).
- Stores runtime data (**.data**, **.bss**, stack, and heap).

12 Start-up Code& crt0

```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2 crt0.S  
section .text  
.globl _start  
_start:  
    la sp, _stack_top    # Set stack pointer  
    call main             # Call main() in C  
1: j 1b                   # Infinite loop if main() returns
```

```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2 linker.ld *  
ENTRY(_start)  
SECTIONS {  
    . = 0x80000000;        /* QEMU "virt" machine entry point */  
    .text : { *(.text) } /* Code */  
    . = ALIGN(4);  
    _stack_top = . + 4096; /* 4KB stack */  
}
```

```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2 main.c *  
int main() {  
    while (1); // Bare-metal programs never exit  
    return 0;  
}
```

```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2 crt0.S  
#include "pulpino.h"  
  
#define EXCEPTION_STACK_SIZE 96  
  
/* ===== [ entry ] ===== */  
.section .text  
reset_handler:  
  
/* set 0 in mtvec (base for IVT) */  
csrrw x0, mtvec, x0  
  
/* set all registers to zero */  
mv x1, x0  
mv x2, x1  
mv x3, x1  
mv x4, x1  
mv x5, x1  
mv x6, x1  
mv x7, x1  
mv x8, x1  
mv x9, x1  
mv x10, x1  
mv x11, x1  
mv x12, x1  
mv x13, x1  
  
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark  
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line  M-E Redo      M-6 Copy
```

```
pranay@pranay-VMware-Virtual-Platform: ~  
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano crt0.S  
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -nostdlib -T linker.  
ld -o program.elf crt0.S main.c  
crt0.S: Assembler messages:  
crt0.S:22: Error: unrecognized opcode `csrrw x0,mtvec,x0', extension `zicsr' required  
crt0.S:201: Error: unrecognized opcode `csrr x28,0x7B0', extension `zicsr' required  
crt0.S:202: Error: unrecognized opcode `csrr x29,0x7B1', extension `zicsr' required  
crt0.S:203: Error: unrecognized opcode `csrr x30,0x7B2', extension `zicsr' required  
crt0.S:207: Error: unrecognized opcode `csrr x28,0x7B4', extension `zicsr' required  
crt0.S:208: Error: unrecognized opcode `csrr x29,0x7B5', extension `zicsr' required  
crt0.S:209: Error: unrecognized opcode `csrr x30,0x7B6', extension `zicsr' required  
crt0.S:220: Error: unrecognized opcode `csrrw x0,0x7B4,x28', extension `zicsr' required  
crt0.S:221: Error: unrecognized opcode `csrrw x0,0x7B5,x29', extension `zicsr' required  
crt0.S:222: Error: unrecognized opcode `csrrw x0,0x7B6,x30', extension `zicsr' required  
crt0.S:226: Error: unrecognized opcode `csrrw x0,0x7B0,x28', extension `zicsr' required  
crt0.S:227: Error: unrecognized opcode `csrrw x0,0x7B1,x29', extension `zicsr' required  
crt0.S:228: Error: unrecognized opcode `csrrw x0,0x7B2,x30', extension `zicsr' required  
(base) pranay@pranay-VMware-Virtual-Platform:~$
```



```

pranay@pranay-VMware-Virtual-Platform: ~
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-gcc -c main.c -o main.o
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-ld -T link.ld crt0.o main.o -o program.elf
riscv64-unknown-elf-ld: crt0.o: in function 'reset_handler':
(.text+0x44): undefined reference to '_stack_start'
riscv64-unknown-elf-ld: crt0.o: in function '_start':
(.text+0x4c): undefined reference to '_bss_start'
riscv64-unknown-elf-ld: (.text+0x54): undefined reference to '_bss_end'
riscv64-unknown-elf-ld: crt0.o: in function 'zero_loop_end':
(.text+0x6a): undefined reference to '__libc_init_array'
riscv64-unknown-elf-ld: crt0.o: in function 'main_entry':
(.text+0x76): undefined reference to 'uart_set_cfg'
riscv64-unknown-elf-ld: (.text+0x84): undefined reference to 'uart_wait_tx_done'
riscv64-unknown-elf-ld: (.text+0x8a): undefined reference to 'exit'
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_I2C_ASM':
(.text+0x9e): undefined reference to 'ISR_I2C'
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_UART_ASM':
(.text+0xb2): undefined reference to 'ISR_UART'
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_GPIO_ASM':
(.text+0xc6): undefined reference to 'ISR_GPIO'
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_SPIM0_ASM':
(.text+0xda): undefined reference to 'ISR_SPIM0'
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_SPIM1_ASM':
(.text+0xee): undefined reference to 'ISR_SPIM1'
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_TA_CMP_ASM':
(.text+0x102): undefined reference to 'ISR_TA_CMP'
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_TA_OVF_ASM':
(.text+0x116): undefined reference to 'ISR_TA_OVF'
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_TB_CMP_ASM':
(.text+0x12a): undefined reference to 'ISR_TB_CMP'
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_TB_OVF_ASM':
(.text+0x13e): undefined reference to 'ISR_TB_OVF'
riscv64-unknown-elf-ld: crt0.o: in function 'illegal_insn_handler':

```

```

pranay@pranay-VMware-Virtual-Platform: ~
GNU nano 7.2                                link.ld
SECTIONS {
    . = 0x80000000;

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

    .rodata : { *(.rodata*) }

    .data : {
        __data_start = .;
        *(.data*)
        __data_end = .;
    }

    .bss : {
        __bss_start = .;
        *(.bss*)
        *(COMMON)
        __bss_end = .;
    }

    .stack : {
        . = ALIGN(4);
        _stack_top = . + 0x1000;
    }
}

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo     M-A Set Mark
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/_ Go To Line M-E Redo     M-6 Copy

```

13 Interrupt Primer

```

pranay@pranay-VMware-Virtual-Platform: ~
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano main.c
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano linker.ld
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -nostdlib -T linker.
ld -o program.elf crt0.S main.c
crt0.S: Assembler messages:
crt0.S:22: Error: unrecognized opcode `csrrw x0,mtvec,x0', extension `zicsr' required
crt0.S:201: Error: unrecognized opcode `csrr x28,0x7B0', extension `zicsr' required
crt0.S:202: Error: unrecognized opcode `csrr x29,0x7B1', extension `zicsr' required
crt0.S:203: Error: unrecognized opcode `csrr x30,0x7B2', extension `zicsr' required
crt0.S:207: Error: unrecognized opcode `csrr x28,0x7B4', extension `zicsr' required
crt0.S:208: Error: unrecognized opcode `csrr x29,0x7B5', extension `zicsr' required
crt0.S:209: Error: unrecognized opcode `csrr x30,0x7B6', extension `zicsr' required
crt0.S:220: Error: unrecognized opcode `csrrw x0,0x7B4,x28', extension `zicsr' required
crt0.S:221: Error: unrecognized opcode `csrrw x0,0x7B5,x29', extension `zicsr' required
crt0.S:222: Error: unrecognized opcode `csrrw x0,0x7B6,x30', extension `zicsr' required
crt0.S:226: Error: unrecognized opcode `csrrw x0,0x7B0,x28', extension `zicsr' required
crt0.S:227: Error: unrecognized opcode `csrrw x0,0x7B1,x29', extension `zicsr' required
crt0.S:228: Error: unrecognized opcode `csrrw x0,0x7B2,x30', extension `zicsr' required
main.c: Assembler messages:
main.c:15: Error: unrecognized opcode `csrw mtvec,a5', extension `zicsr' required
main.c:16: Error: unrecognized opcode `csrsi mie,0x80', extension `zicsr' required
main.c:17: Error: unrecognized opcode `csrsi mstatus,0x8', extension `zicsr' required
(base) pranay@pranay-VMware-Virtual-Platform:~$ qemu-system-riscv32 -machine virt -nographic -bios none -kernel program.
elf
program.elf: No such file or directory
qemu-system-riscv32: could not load kernel 'program.elf'
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -nostdlib -T linker.
ld -o program.elf crt0.S main.c
crt0.S: Assembler messages:
crt0.S:22: Error: unrecognized opcode `csrrw x0,mtvec,x0', extension `zicsr' required
crt0.S:201: Error: unrecognized opcode `csrr x28,0x7B0', extension `zicsr' required
crt0.S:202: Error: unrecognized opcode `csrr x29,0x7B1', extension `zicsr' required

```

14 rv32imac vs rv32imc - What's the "A" ?

The '**A**' extension in RISC-V (Atomic Operations) introduces instructions for atomic memory access, critical for synchronization in multi-threaded and multi-core environments. Here's a breakdown:

Key Instructions Added by the 'A' Extension

1. Load-Reserved (**lr.w**) & Store-Conditional (**sc.w**)

- **lr.w rd, (rs1)**: Loads a word from memory into **rd** and marks the address as reserved.
- **sc.w rd, rs2, (rs1)**: Stores **rs2** to memory **only if** the reservation is still valid. Sets **rd** to 0 on success, 1 on failure.
- **Use**: Implement lock-free data structures (e.g., spinlocks).

2. Atomic Memory Operations (AMOs)

Perform read-modify-write operations atomically:

- `amoadd.w rd, rs2, (rs1)`: Atomically adds `rs2` to the value at `rs1`, stores result back, and returns original value to `rd`.
- `amoswap.w`, `amoand.w`, `amoxor.w`, etc.: Swap, AND, XOR, etc.
- **Use**: Efficient counters, mutexes, or shared resource management.

Why the 'A' Extension is Useful

- **Lock-Free Synchronization**: Enables safe concurrent access to shared data without software locks.
- **Operating Systems**: Critical for implementing thread-safe kernels, semaphores, and interrupt handlers.
- **Performance**: Hardware-supported atomics are faster than software-based solutions (e.g., disabling interrupts).
- **Correctness**: Prevents race conditions in multi-core systems.

Example Use Case

c

```
// Atomic increment using AMOADD.W
void atomic_inc(uint32_t *ptr) {
    asm volatile ("amoadd.w zero, %0, %1" : "r"(1), "A"(*ptr));
}
```

This increments `*ptr` atomically, even if multiple cores access it simultaneously.

rv32imac vs rv32imc

- **rv32imac**: Includes Atomic instructions.
- **rv32imc**: Excludes atomics; unsuitable for multi-core/threaded systems.

Applications needing 'A':

- Real-time OS kernels.
- Multi-threaded applications.
- Embedded systems with shared peripherals.

Without 'A', synchronization requires complex software workarounds, risking inefficiency or race conditions.

```
pranay@pranay-VMware-Virtual-Platform: ~  
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-gcc -march=rv32imac -mabi=ilp32 -nostdlib -T linker.  
ld -o atomic.elf crt0.S main.c  
crt0.S: Assembler messages:  
crt0.S:22: Error: unrecognized opcode `csrrw x0,mtvec,x0', extension `zicsr' required  
crt0.S:201: Error: unrecognized opcode `csrr x28,0x7B0', extension `zicsr' required  
crt0.S:202: Error: unrecognized opcode `csrr x29,0x7B1', extension `zicsr' required  
crt0.S:203: Error: unrecognized opcode `csrr x30,0x7B2', extension `zicsr' required  
crt0.S:207: Error: unrecognized opcode `csrr x28,0x7B4', extension `zicsr' required  
crt0.S:208: Error: unrecognized opcode `csrr x29,0x7B5', extension `zicsr' required  
crt0.S:209: Error: unrecognized opcode `csrr x30,0x7B6', extension `zicsr' required  
crt0.S:220: Error: unrecognized opcode `csrrw x0,0x7B4,x28', extension `zicsr' required  
crt0.S:221: Error: unrecognized opcode `csrrw x0,0x7B5,x29', extension `zicsr' required  
crt0.S:222: Error: unrecognized opcode `csrrw x0,0x7B6,x30', extension `zicsr' required  
crt0.S:226: Error: unrecognized opcode `csrrw x0,0x7B0,x28', extension `zicsr' required  
crt0.S:227: Error: unrecognized opcode `csrrw x0,0x7B1,x29', extension `zicsr' required  
crt0.S:228: Error: unrecognized opcode `csrrw x0,0x7B2,x30', extension `zicsr' required  
main.c: Assembler messages:  
main.c:15: Error: unrecognized opcode `csrw mtvec,a5', extension `zicsr' required  
main.c:16: Error: unrecognized opcode `csrsi mie,0x80', extension `zicsr' required  
main.c:17: Error: unrecognized opcode `csrsi mstatus,0x8', extension `zicsr' required  
(base) pranay@pranay-VMware-Virtual-Platform:~$ qemu-system-riscv32 -machine virt -nographic -bios none -kernel atomic.e  
lf  
atomic.elf: No such file or directory  
qemu-system-riscv32: could not load kernel 'atomic.elf'  
(base) pranay@pranay-VMware-Virtual-Platform:~$
```

```

pranay@pranay-VMware-Virtual-Platform: ~
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-gcc -march=rv32imac_zicsr -mabi=ilp32 -nostdlib -T l
inker.ld -o atomic.elf crt0.S main.c
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccYejiKZ.o: in function `reset_handler
':
(.text+0x44): undefined reference to `__stack_start'
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccYejiKZ.o: in function `_start':
(.text+0x4c): undefined reference to `__bss_start'
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: (.text+0x54): undefined reference to `bss_
end'
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccYejiKZ.o: in function `zero_loop_end
':
(.text+0x6a): undefined reference to `__libc_init_array'
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccYejiKZ.o: in function `main_entry':
(.text+0x76): undefined reference to `uart_set_cfg'
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: (.text+0x84): undefined reference to `uart_
wait_tx_done'
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: (.text+0x8a): undefined reference to `exit'
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccYejiKZ.o: in function `ISR_I2C_ASM':
(.text+0x9e): undefined reference to `ISR_I2C'
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccYejiKZ.o: in function `ISR_UART_ASM'
:
(.text+0xb2): undefined reference to `ISR_UART'
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccYejiKZ.o: in function `ISR_GPIO_ASM'
:
(.text+0xc6): undefined reference to `ISR_GPIO'
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccYejiKZ.o: in function `ISR_SPI0_ASM'
:

```

15 Atomic Test Program

```

pranay@pranay-VMware-Virtual-Platform: ~
GNU nano 7.2 main.c
#include <stdint.h>

#define UART_BASE ((volatile char*)0x10000000)
void uart_putc(char c) { *UART_BASE = c; }
void uart_puts(const char *s) { while (*s) uart_putc(*s++); }

typedef struct { volatile uint32_t lock; } spinlock_t;

void spin_lock(spinlock_t *lock) {
    uint32_t tmp;
    do {
        __asm__ volatile (
            "lr.w %0, %1\n"
            "bnez %0, 1f\n"
            "li %0, 1\n"
            "sc.w %0, %0, %1\n"
            "bnez %0, 1b\n"
            "1:"
            : "=&r" (tmp)
            : "A" (lock->lock)
            : "memory"
        );
    } while (tmp != 0);
}

void spin_unlock(spinlock_t *lock) {
    __asm__ volatile (
        "amoswap.w zero, zero, %0"
    );
}

```

[Read 63 lines]

```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2 linker.ls  
ENTRY(_start)  
SECTIONS {  
    . = 0x80000000;  
    .text : { *(.text) }  
    . = ALIGN(4);  
    _stack_top = . + 4096;  
}  
  
pranay@pranay-VMware-Virtual-Platform: ~  
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano linker.ls  
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano main.c  
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-gcc -march=rv32imac_zicsr -mabi=ilp32 -nostdlib -T linker.ld -o lock.elf crt0.S main.c  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: warning: lock.elf has a LOAD segment with R/WX permissions  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccmRzqBI.o: in function `reset_handler':  
.:  
(.text+0x44): undefined reference to `_stack_start'  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccmRzqBI.o: in function `_start':  
(.text+0x4c): undefined reference to `_bss_start'  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: (.text+0x54): undefined reference to `_bss_end'  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccmRzqBI.o: in function `zero_loop_end':  
.:  
(.text+0x6a): undefined reference to `__libc_init_array'  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccmRzqBI.o: in function `main_entry':  
(.text+0x76): undefined reference to `uart_set_cfg'  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: (.text+0x84): undefined reference to `uart_wait_tx_done'  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: (.text+0x8a): undefined reference to `exit'  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccmRzqBI.o: in function `ISR_I2C_ASM':  
(.text+0x9e): undefined reference to `ISR_I2C'  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccmRzqBI.o: in function `ISR_UART_ASM':  
.:  
(.text+0xb2): undefined reference to `ISR_UART'  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccmRzqBI.o: in function `ISR_GPIO_ASM':  
.:  
(.text+0xc6): undefined reference to `ISR_GPIO'  
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccmRzqBI.o: in function `ISR_SPI0_ASM':  
.:  
(.text+0xda): undefined reference to `ISR_SPI0'
```

```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2 syscalls.c *  
int _read(int fd, char *buf, int len) { return 0; }  
int _close(int fd) { return 0; }  
int _fstat(int fd, struct stat *st) { return 0; }  
int _isatty(int fd) { return 1; }  
off_t _lseek(int fd, off_t offset, int whence) { return 0; }  
  
// Memory management  
extern char _heap_start;  
void *_sbrk(int incr) {  
    static char *heap_end = &_heap_start;  
    char *prev_heap_end = heap_end;  
    heap_end += incr;  
    return prev_heap_end;  
}
```

```
pranay@pranay-VMware-Virtual-Platform: ~  
GNU nano 7.2 linker.ld *  
.bss : {  
    _bss_start = .;  
    *(.bss*)  
    *(COMMON)  
    _bss_end = .;  
}  
  
_ = ALIGN(4);  
_heap_start = .;  
_ = _ + 4096; /* 4KB heap */  
_stack_start = .;  
_ = _ + 4096; /* 4KB stack */  
_stack_top = .;  
}
```

```
pranay@pranay-VMware-Virtual-Platform: ~  
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano syscalls.c  
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano linker.ld  
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-gcc -c crt0.S -o crt0.o  
riscv64-unknown-elf-gcc -c main.c -o main.o  
riscv64-unknown-elf-ld -T linker.ld crt0.o main.o -o program.elf  
riscv64-unknown-elf-objdump -d program.elf > program.asm  
riscv64-unknown-elf-ld: warning: program.elf has a LOAD segment with RWX permissions  
riscv64-unknown-elf-ld: crt0.o: in function 'reset_handler':  
(.text+0x44): undefined reference to '_stack_start'  
riscv64-unknown-elf-ld: crt0.o: in function '_start':  
(.text+0x4c): undefined reference to '_bss_start'  
riscv64-unknown-elf-ld: (.text+0x54): undefined reference to '_bss_end'  
riscv64-unknown-elf-ld: crt0.o: in function 'zero_loop_end':  
(.text+0x6a): undefined reference to '__libc_init_array'  
riscv64-unknown-elf-ld: crt0.o: in function 'main_entry':  
(.text+0x76): undefined reference to 'uart_set_cfg'  
riscv64-unknown-elf-ld: (.text+0x84): undefined reference to 'uart_wait_tx_done'  
riscv64-unknown-elf-ld: (.text+0x8a): undefined reference to 'exit'  
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_I2C_ASM':  
(.text+0x9e): undefined reference to 'ISR_I2C'  
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_UART_ASM':  
(.text+0xb2): undefined reference to 'ISR_UART'  
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_GPIO_ASM':  
(.text+0xc6): undefined reference to 'ISR_GPIO'  
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_SPIM0_ASM':  
(.text+0xda): undefined reference to 'ISR_SPIM0'  
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_SPIM1_ASM':  
(.text+0xee): undefined reference to 'ISR_SPIM1'  
riscv64-unknown-elf-ld: crt0.o: in function 'ISR_TA_CMP_ASM':  
(.text+0x102): undefined reference to 'ISR_TA_CMP'
```

17 Endianness & Struct Packing

```
(base) pranay@pranay-VMware-Virtual-Platform:~$ nano endian.c
(base) pranay@pranay-VMware-Virtual-Platform:~$ sudo apt install libriscv64-unknown-elf-newlib
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package libriscv64-unknown-elf-newlib
(base) pranay@pranay-VMware-Virtual-Platform:~$ riscv64-unknown-elf-gcc -march=rv32imac_zicsr -mabi=ilp32 -T linker.ld
o endian.elf crt0.S endian.c
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccE6Syce.o: in function `_start':
(.text+0x4c): multiple definition of `_start'; crt0.o:(.text+0x4c): first defined here
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccE6Syce.o: in function `_init':
(.text+0x202): multiple definition of `_init'; crt0.o:(.text+0x202): first defined here
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: /tmp/ccE6Syce.o: in function `_init':
(.text+0x202): multiple definition of `_fini'; crt0.o:(.text+0x202): first defined here
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: cannot find -lc: No such file or directory
/usr/lib/gcc/riscv64-unknown-elf/13.2.0/../../../../riscv64-unknown-elf/bin/ld: cannot find -lgloss: No such file or direc
ory
collect2: error: ld returned 1 exit status
(base) pranay@pranay-VMware-Virtual-Platform:~$ qemu-system-riscv32 -machine virt -nographic -bios none -kernel endian.
lf
endian.elf: No such file or directory
qemu-system-riscv32: could not load kernel 'endian.elf'
```