Task 2 — Prove Your Local RISC–V Setup (Run, Disassemble, Decode)

Goal

- Run 3–4 RISC–V C programs on your own machine (compiled with your local toolchain, executed with `spike pk`).
- For each program, generate assembly and include a screenshot of the `main:` section.
- Manually decode RISC–V integer instructions and document them in your GitHub repository.
- Ensure outputs are unique to your PC by embedding username, hostname, machine ID, and timestamps.

Deadline: complete within one week from starting Task 2.

A. Uniqueness mechanism (do this before compiling)

Set identity variables in your **Linux host shell** so each build is user/machine specific.

```
export U=$(id -un)
export H=$(hostname -s)
export M=$(cat /etc/machine-id | head -c 16)
export T=$(date -u +%Y-%m-%dT%H:%M:%SZ)
export E=$(date +%s)
```

B. Common header (save as `unique.h`)

```
#ifndef UNIQUE_H
#define UNIQUE_H
#include <stdio.h>
#include <stdint.h>
#include <time.h>

#ifndef USERNAME
#define USERNAME "unknown_user"
#endif
#ifndef HOSTNAME
#define HOSTNAME "unknown_host"
#endif
#ifndef MACHINE_ID
#define MACHINE_ID "unknown_machine"
#endif
#ifndef BUILD_UTC
#define BUILD_UTC "unknown_time"
```

```
#endif
#ifndef BUILD_EPOCH
#define BUILD_EPOCH 0
#endif

static uint64_t fnv1a64(const char *s) {
    const uint64_t OFF = 14695981039346656037ULL, PRIME = 1099511628211ULL;
    uint64_t h = OFF;
    for (const unsigned char *p=(const unsigned char*)s; *p; ++p) {
        h ^= *p; h *= PRIME;
    }
    return h;
}

static void uniq_print_header(const char *program_name) {
    time_t now = time(NULL);
    char buf[512];
    int n = snprintf(buf, sizeof(buf), "%s|%s|%s|%s|%ld|%s|%s",
                     USERNAME, HOSTNAME, MACHINE_ID, BUILD_UTC,
                     (long)BUILD_EPOCH, __VERSION__, program_name);
    (void)n;
    uint64_t proof = fnv1a64(buf);

    char rbuf[600];
    snprintf(rbuf, sizeof(rbuf), "%s|run_epoch=%ld", buf, (long)now);
    uint64_t runid = fnv1a64(rbuf);

    printf("=== RISC-V Proof Header ===\n");
    printf("User       : %s\n", USERNAME);
    printf("Host       : %s\n", HOSTNAME);
    printf("MachineID  : %s\n", MACHINE_ID);
    printf("BuildUTC   : %s\n", BUILD_UTC);
    printf("BuildEpoch : %ld\n", (long)BUILD_EPOCH);
    printf("GCC        : %s\n", __VERSION__);
    printf("PointerBits: %d\n", (int)(8*(int)sizeof(void*)));
    printf("Program    : %s\n", program_name);
    printf("ProofID    : 0x%016llx\n", (unsigned long long)proof);
    printf("RunID      : 0x%016llx\n", (unsigned long long)runid);
    printf("============================\n");
}
#endif
```

C. Programs to implement

Each program must include `unique.h` and print the header first.

1) factorial.c

```
#include "unique.h"
static unsigned long long fact(unsigned n){ return (n<2)?1ULL:n*fact(n-1); }
int main(void){
    uniq_print_header("factorial");
    unsigned n = 12;
    printf("n=%u, n!=%llu\n", n, fact(n));
    return 0;
}
```

2) max_array.c

```
#include "unique.h"
int main(void){
    uniq_print_header("max_array");
    int a[] = {42,-7,19,88,3,88,5,-100,37};
    int n = sizeof(a)/sizeof(a[0]), max=a[0];
    for(int i=1;i<n;i++) if(a[i]>max) max=a[i];
    printf("Array length=%d, Max=%d\n", n, max);
    return 0;
}
```

3) bitops.c

```
#include "unique.h"
int main(void){
    uniq_print_header("bitops");
    unsigned x=0xA5A5A5A5u, y=0x0F0F1234u;
    printf("x&y=0x%08X\n", x&y);
    printf("x|y=0x%08X\n", x|y);
    printf("x^y=0x%08X\n", x^y);
    printf("x<<3=0x%08X\n", x<<3);
    printf("y>>2=0x%08X\n", y>>2);
    return 0;
}
```

4) bubble_sort.c

```
#include "unique.h"
void bubble(int *a,int n){ for(int i=0;i<n-1;i++) for(int j=0;j<n-1-i;j++) if(a[j]>a[j
    +1]){int t=a[j];a[j]=a[j+1];a[j+1]=t;} }
int main(void){
    uniq_print_header("bubble_sort");
    int a[]={9,4,1,7,3,8,2,6,5}, n=sizeof(a)/sizeof(a[0]);
    bubble(a,n);
    printf("Sorted:"); for(int i=0;i<n;i++) printf(" %d",a[i]); puts("");
    return 0;
}
```

D. Build, run, and capture evidence

Compile example:

```
riscv64-unknown-elf-gcc -O0 -g -march=rv64imac -mabi=lp64 \
  -DUSERNAME="\"$U\"" -DHOSTNAME="\"$H\"" -DMACHINE_ID="\"$M\"" \
  -DBUILD_UTC="\"$T\"" -DBUILD_EPOCH=$E \
  factorial.c -o factorial
```

Run on Spike:

```
spike pk ./factorial
```

E. Produce assembly and disassembly

Assembly (.s):

```
riscv64-unknown-elf-gcc -O0 -S factorial.c -o factorial.s
```

Disassembly of main only:

```
riscv64-unknown-elf-objdump -d ./factorial | sed -n '/<main>:/,/^$/p' | tee
    factorial_main_objdump.txt
```

F. Instruction decoding (integer type)

Decode at least 5 RISC-V integer instructions from your '.s' or '.objdump'. Use the following format in a markdown file:

Instruction	Opcode	rd	rs1	rs2	funct3	funct7	Binary	Description
`add x5,x6,x7`	0110011	x5	x6	x7	000	0000000	`0000000 00111 00110 000 00101 0110011`	x5 = x6 + x7
...	...	...	...	...	...	...	...	...

G. Repository structure

```
riscv-task2-<your-username>/
- unique.h
- factorial.c
- factorial.s
- factorial_main_objdump.txt
- factorial_output.png
- factorial_main_asm.png
- max_array.c
- ...
- instruction_decoding.md
- README.md
```

README should include:

- Output of `spike -version`
- Output of `riscv64-unknown-elf-gcc -v`
- The exact compile commands
- A note confirming ProofID/RunID are visible in each output screenshot

Evaluation Notes

- **ProofID** is unique per user+host+machine+build
- **RunID** adds per-run randomness
- Your '.s' and '.objdump' files must match your compiled code