HW 2

2024 Computer Organization : Performance Modeling for the μRISC-V Processor

Chia-Heng Tu

Dept. of Computer Science and Information Engineering

National Cheng Kung University

Outline

- Overview
- Structure
- Exercise
- Judge
- Submission
- Deadline
- How to Mail TAs

Overview

1. The performance model used in this assignment uses **performance metrics** to summarize the delivered performance of a RISC-V instruction executed on the target RISC-V processor, including the effect of the CPU pipeline and the memory subsystem.

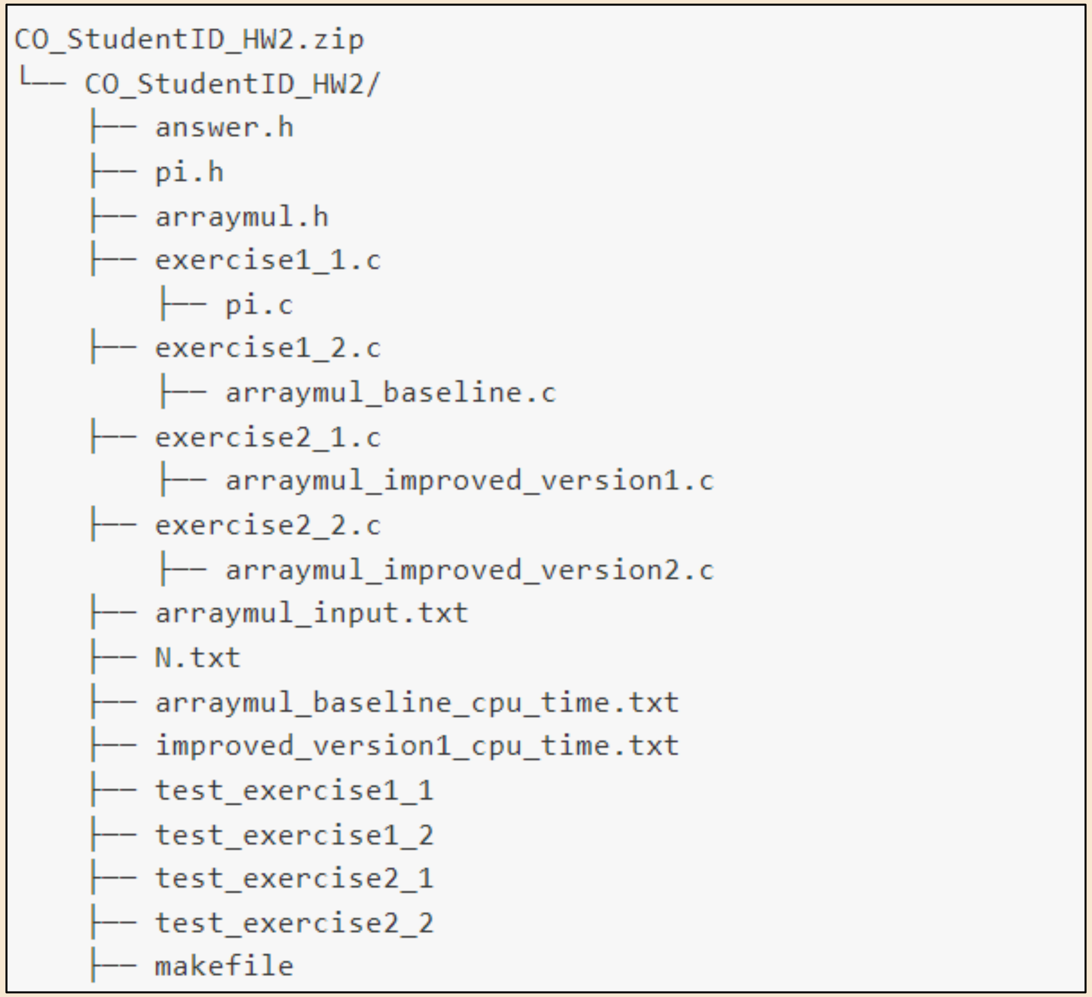
2. Given the above modeling concept, you will need to collect the performance data of your program to derive the CPU execution time. Specifically, you need to record the instruction counts of different types of RISC-V instructions.

   - The instructions are categorized into seven types and their instruction counts should be recorded (accumulated) in the seven counters: add_cnt, sub_cnt, mul_cnt, div_cnt, lw_cnt, sw_cnt, and others_cnt, respectively.

   - The CPIs for the seven types of instructions, add_CPI, sub_CPI, mul_CPI, div_CPI, lw_CPI, sw_CPI, and others_CPI as constants.

Structure

You will receive the code with the following structure:



```
CO_StudentID_HW2.zip
└── CO_StudentID_HW2/
    ├── answer.h
    ├── pi.h
    ├── arraymul.h
    ├── exercise1_1.c
        ├── pi.c
    ├── exercise1_2.c
        ├── arraymul_baseline.c
    ├── exercise2_1.c
        ├── arraymul_improved_version1.c
    ├── exercise2_2.c
        ├── arraymul_improved_version2.c
    ├── arraymul_input.txt
    ├── N.txt
    ├── arraymul_baseline_cpu_time.txt
    ├── improved_version1_cpu_time.txt
    ├── test_exercise1_1
    ├── test_exercise1_2
    ├── test_exercise2_1
    ├── test_exercise2_2
    ├── makefile
```

Structure – Exercise

File description for Exercise1-1 and 1-2:

```
CO_StudentID_HW2.zip
└── CO_StudentID_HW2/
    ├── answer.h
    ├── pi.h
    ├── arraymul.h
    ├── exercise1_1.c
        ├── pi.c
    ├── exercise1_2.c
        ├── arraymul_baseline.c
    ├── exercise2_1.c
        ├── arraymul_improved_version1.c
    ├── exercise2_2.c
        ├── arraymul_improved_version2.c
    ├── arraymul_input.txt
    ├── N.txt
    ├── arraymul_baseline_cpu_time.txt
    ├── improved_version1_cpu_time.txt
    ├── test_exercise1_1
    ├── test_exercise1_2
    ├── test_exercise2_1
    ├── test_exercise2_2
    ├── makefile
```

Lab2 exercise1-1

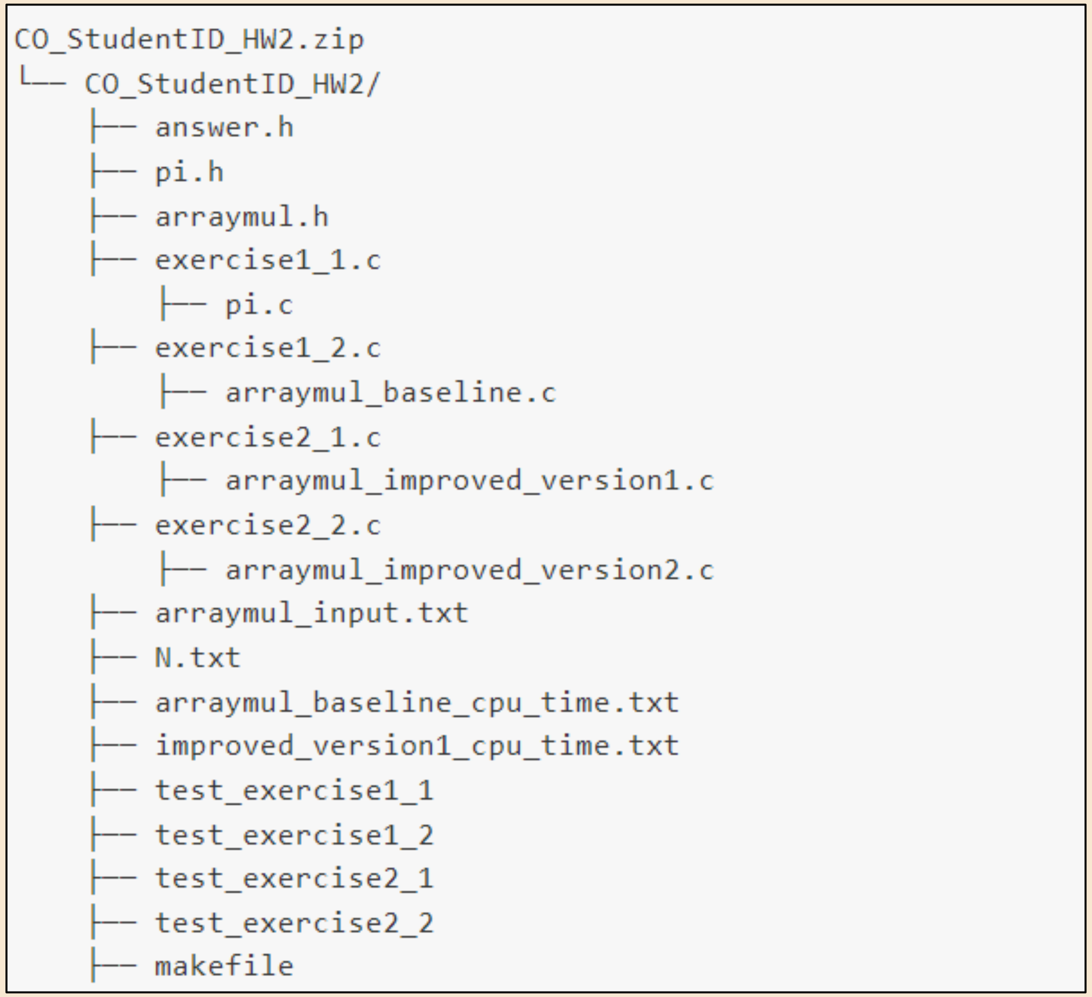
- exercise1_1.c: main file, starting from this file.
- pi.h: definition of exercise1_1.c.
- pi.c: implementation answer here.
- answer.h: compute performance here.
- N.txt: write down your answer in this file.

Lab2 exercise1-2

- exercise1_2.c: main file, starting from this file.
- arraymul.h: definition of exercise1_2.c, exercise2_1.c, exercise2_2.c.
- arraymul_baseline.c: implementation answer here.
- answer.h: compute performance here.
- arraymul_baseline_cpu_time.txt: cpu time is stored in this file.

Structure – Exercise

File description for Exercise2-1 and 2-2:



```
CO_StudentID_HW2.zip
└── CO_StudentID_HW2/
    ├── answer.h
    ├── pi.h
    ├── arraymul.h
    ├── exercise1_1.c
        ├── pi.c
    ├── exercise1_2.c
        ├── arraymul_baseline.c
    ├── exercise2_1.c
        ├── arraymul_improved_version1.c
    ├── exercise2_2.c
        ├── arraymul_improved_version2.c
    ├── arraymul_input.txt
    ├── N.txt
    ├── arraymul_baseline_cpu_time.txt
    ├── improved_version1_cpu_time.txt
    ├── test_exercise1_1
    ├── test_exercise1_2
    ├── test_exercise2_1
    ├── test_exercise2_2
    ├── makefile
```

Lab2 exercise2-1

- exercise2_1.c: main file, starting from this file.
- arraymul.h: definition of exercise1_2.c, exercise2_1.c, exercise2_2.c.
- arraymul_improved_version1.c: implementation answer here.
- answer.h: compute performance here.
- improved_version1_cpu_time.txt: cpu time is stored in this file.

Lab2 exercise2-2

- exercise2_2.c: main file, starting from this file.
- arraymul.h: definition of exercise1_2.c, exercise2_1.c, exercise2_2.c.
- arraymul_improved_version2.c : implementation answer here.
- answer.h: compute performance here.

Structure – Judge files

You will receive the four judge programs to test your exercises.

```
CO_StudentID_HW2.zip
└── CO_StudentID_HW2/
    ├── answer.h
    ├── pi.h
    ├── arraymul.h
    ├── exercise1_1.c
        ├── pi.c
    ├── exercise1_2.c
        ├── arraymul_baseline.c
    ├── exercise2_1.c
        ├── arraymul_improved_version1.c
    ├── exercise2_2.c
        ├── arraymul_improved_version2.c
    ├── arraymul_input.txt
    ├── N.txt
    ├── arraymul_baseline_cpu_time.txt
    ├── improved_version1_cpu_time.txt
    ├── test_exercise1_1
    ├── test_exercise1_2
    ├── test_exercise2_1
    ├── test_exercise2_2
    ├── makefile
```

Common

- makefile: help you compile and run code.
- test_execise1_1: testing your execise1_1.
- test_execise1_2: testing your execise1_2.
- test_execise2_1: testing your execise2_1.
- test_execise2_2: testing your execise2_2.

Structure – Modify files

You have to modify five files in this assignment.

```
CO_StudentID_HW2.zip
└── CO_StudentID_HW2/
    ├── answer.h
    ├── pi.h
    ├── arraymul.h
    ├── exercise1_1.c
    │   ├── pi.c
    ├── exercise1_2.c
    │   ├── arraymul_baseline.c
    ├── exercise2_1.c
    │   ├── arraymul_improved_version1.c
    ├── exercise2_2.c
    │   ├── arraymul_improved_version2.c
    ├── arraymul_input.txt
    ├── N.txt
    ├── arraymul_baseline_cpu_time.txt
    ├── improved_version1_cpu_time.txt
    ├── test_exercise1_1
    ├── test_exercise1_2
    ├── test_exercise2_1
    ├── test_exercise2_2
    ├── makefile
```

You should write your answer in:

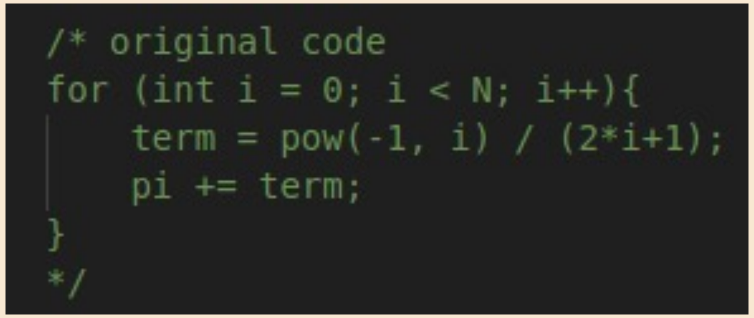
- answer.h
- pi.c
- N.txt
- arraymul_baseline.c
- arraymul_improved_version1.c
- arraymul_improved_version2.c

Don't modify other files. But you can read other files to help you finish your homework.

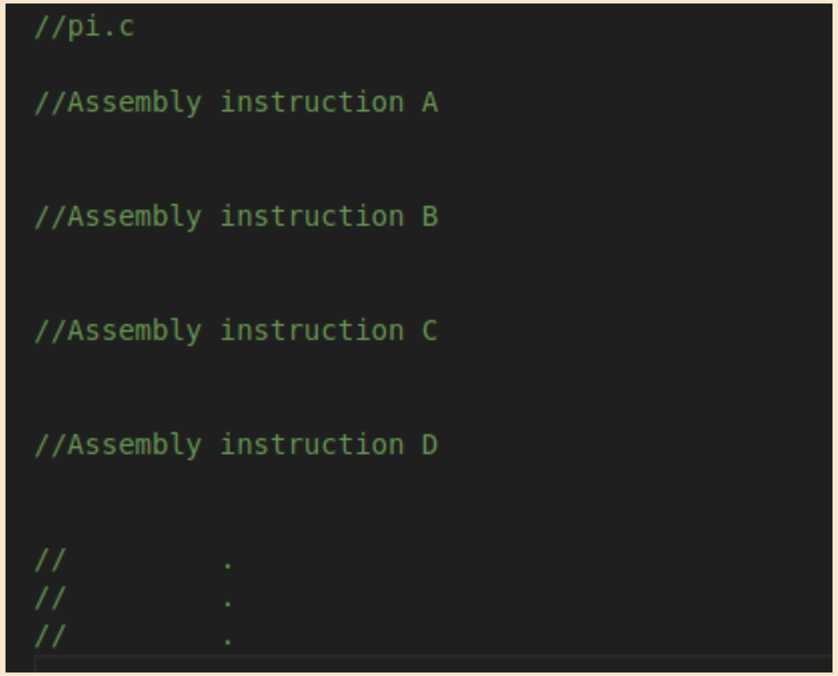
Exercise1-1(40%): Calculating π

Step1: Translate C code to Assembly in pi.c

① Variables N is defined in the header files(pi.h) and is used to determine the number of iterations, you can only change N(the number of iterations) in pi.h.



```
/* original code
for (int i = 0; i < N; i++){
    term = pow(-1, i) / (2*i+1);
    pi += term;
}
*/
```

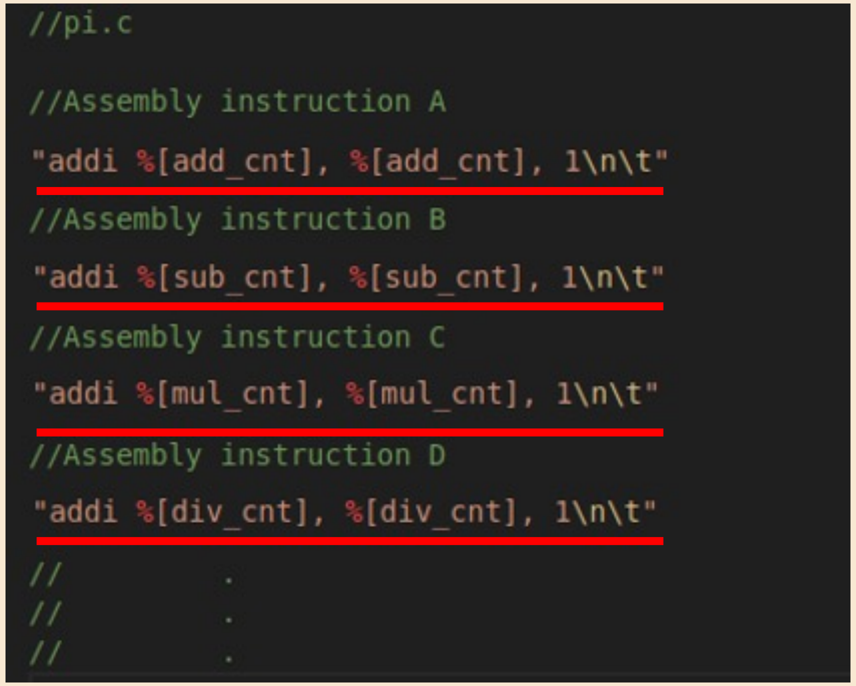


```
//pi.c

//Assembly instruction A


//Assembly instruction B


//Assembly instruction C


//Assembly instruction D


//          .
//          .
//          .
```

Exercise1-1(40%): Calculating π

Step2: Insert the assembly code into pi.c to count the number of instructions, according to the types of the instructions, and to store the accumulated counts in the respective counters.

- The seven counters values which are define in pi.h:
  - add_cnt (4%)
  - sub_cnt (4%)
  - mul_cnt (4%)
  - div_cnt (4%)
  - lw_cnt (4%)
  - sw_cnt (4%)
  - others_cnt(4%)

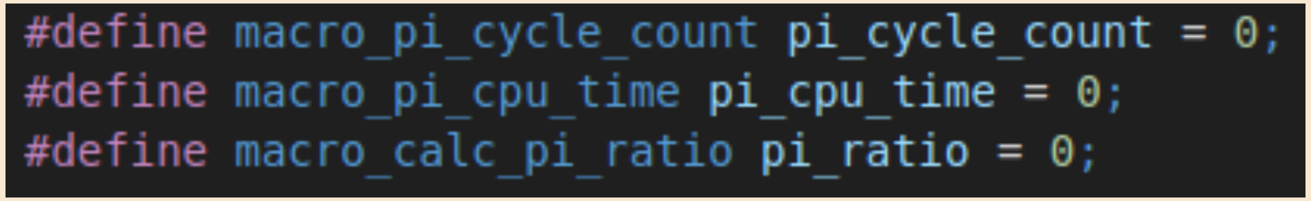


```
//pi.c

//Assembly instruction A
"addi %[add_cnt], %[add_cnt], 1\n\t"
//Assembly instruction B
"addi %[sub_cnt], %[sub_cnt], 1\n\t"
//Assembly instruction C
"addi %[mul_cnt], %[mul_cnt], 1\n\t"
//Assembly instruction D
"addi %[div_cnt], %[div_cnt], 1\n\t"
//          .
//          .
//          .
```

Exercise1-1(40%): Calculating π

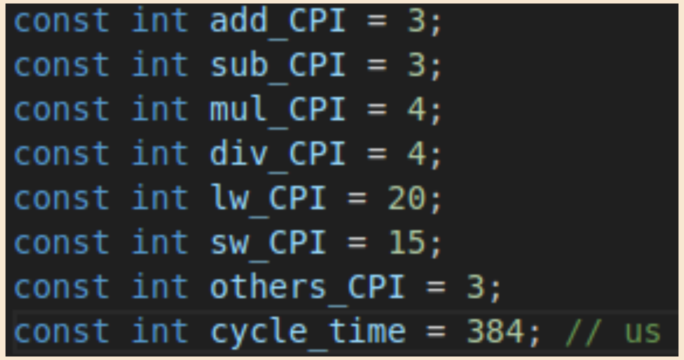
Step3: Add the related code(formula) in answer.h to set up the value for performance data.

- answer.h



```
#define macro_pi_cycle_count pi_cycle_count = 0;
#define macro_pi_cpu_time pi_cpu_time = 0;
#define macro_calc_pi_ratio pi_ratio = 0;
```

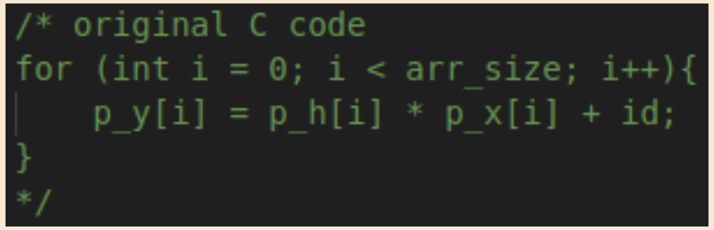
- pi.h



```
const int add_CPI = 3;
const int sub_CPI = 3;
const int mul_CPI = 4;
const int div_CPI = 4;
const int lw_CPI = 20;
const int sw_CPI = 15;
const int others_CPI = 3;
const int cycle_time = 384; // us
```

- The total cycle count (pi_cycle_count)(4%)
  - ➢ formula: Clock Cycles = Instruction Count * Cycles per Instruction
- The CPU time (pi_cpu_time)(4%)
  - ➢ formula: Instruction Count * CPI * Clock Cycle Time
- What is the maximum value of N and the program is still memory bound? Please manually record the answer into N.txt. (4%)
  - ➢ formula: (clock cycles for the instructions other than load/store instructions) / (clock cycles for all the instructions)

Exercise1-2(40%): Array multiplication



```
/* original C code
for (int i = 0; i < arr_size; i++){
    p_y[i] = p_h[i] * p_x[i] + id;
}
*/
```

Step1: Translate C code to Assembly in arraymul_baseline.c

Step2: Insert the assembly code into arraymul_baseline.c to count the number of instructions, according to the types of the instructions, and store the accumulated counts in the respective counters

① Variables arr_size is defined in the header files(arraymul.h) and is used to determine the number of array size, you can only change arr_size(array size) in arraymul.h, $\log_2$(arr_size) must be a integer.

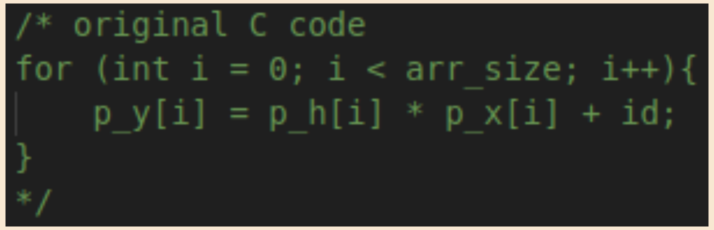
Step3: Add the related code(formula) in answer.h to set up the value for performance data.

- The seven counters values(28%)
- The total cycle count (arraymul_baseline_cycle_count)(4%)
- The CPU time (arraymul_baseline_cpu_time)(4%)
- Choose any arr_size but **$\log_2$(arr_size)** must be a integer and answer the question: Is this program a CPU bound or Memory bound program?(4%)

Exercise2-1(50%): Vectorized Code

- Some useful vector extension instruction: vsetvli, vle16.v vle32.v, vse16.v, vse32.v, vadd.vv, vmseq.vv…
- Read V extension spec(how to use instruction, read example).
- riscv-v-spec-1.0.pdf
- Introduction to the RISC-V Vector Extension

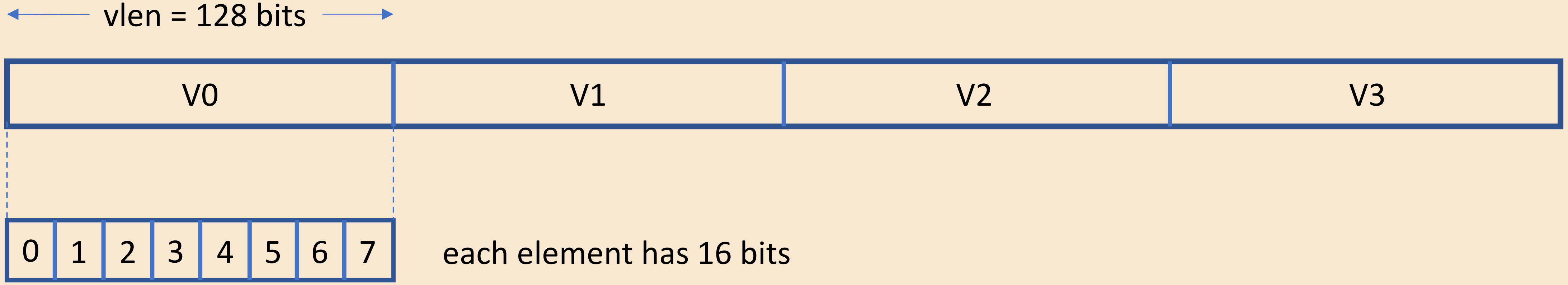
Exercise2-1(50%): Vectorized Code



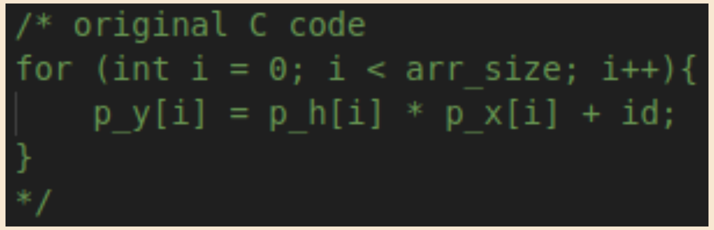
```
/* original C code
for (int i = 0; i < arr_size; i++){
    p_y[i] = p_h[i] * p_x[i] + id;
}
*/
```

Step1: Using the RISC-V V extension to Rewrite Assembly Code in exercise 1-2 and Report the Performance Statistics, run Spike simulator with custom configurations(vlen=128, elen =16).

① The Vector Register Width (vlen)
② The Element Width (elen)



Exercise2-1(50%): Vectorized Code



```
/* original C code
for (int i = 0; i < arr_size; i++){
    p_y[i] = p_h[i] * p_x[i] + id;
}
*/
```

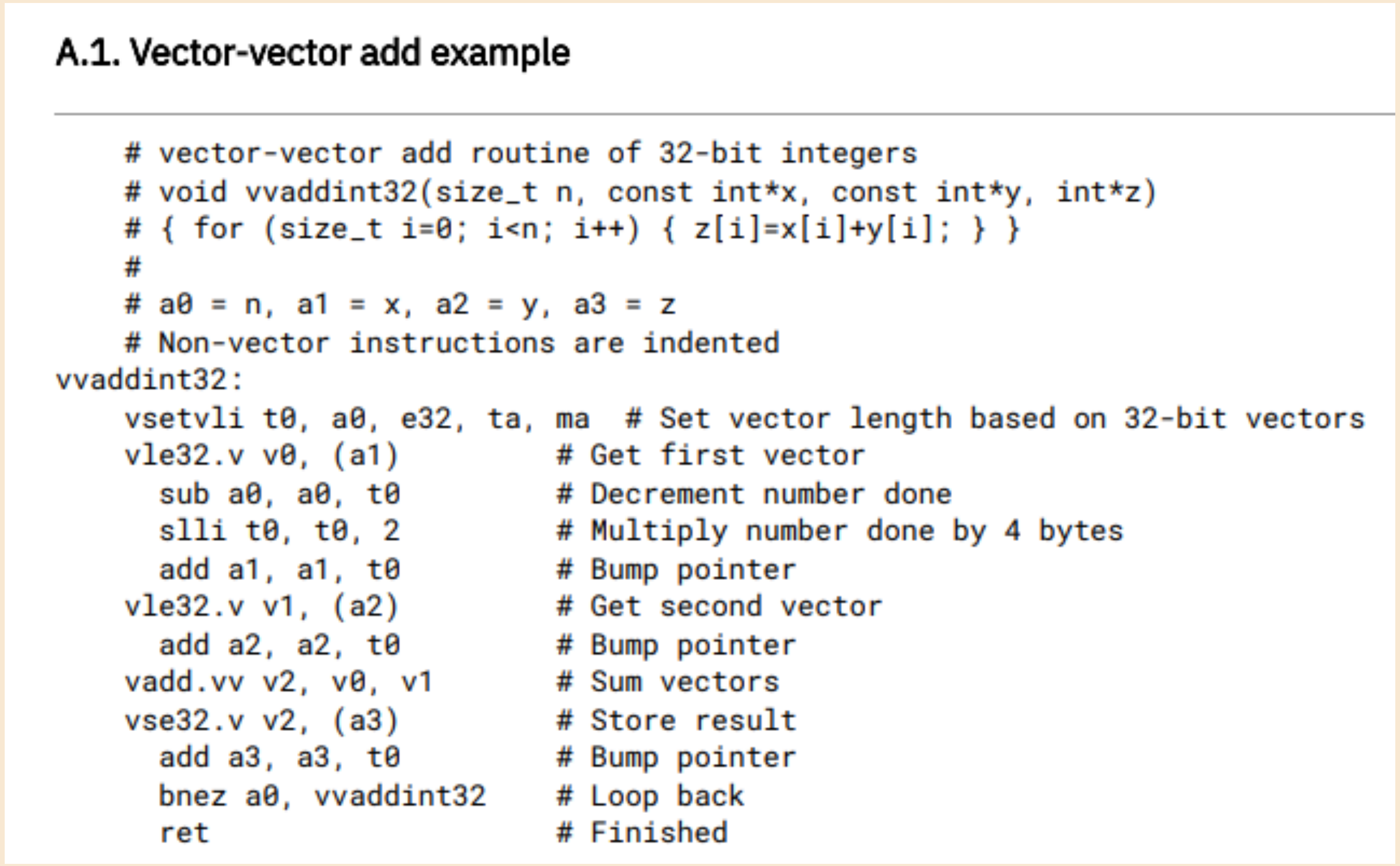
Step2: Insert the assembly code into arraymul_improved_version1.c to count the number of instructions, according to the types of the instructions, and to store the accumulated counts in the respective counters.

Step3: Add the related code(formula) in answer.h to set up the value for performance data.

- The seven counters values(28%)
- The total cycle count (improved_version1_cycle_count)(4%)
- The CPU time (improved_version1_cpu_time)(4%)
- Speedup(14%)
  - If 6 < speedup, you get (14 pt).
  - If 4 < speedup < 6, you get (9 pt).
  - If 2 < speedup < 4, you get (5 pt).
  - If speedup < 2, you get (0 pt).

You have to complete exercise1-2 first!

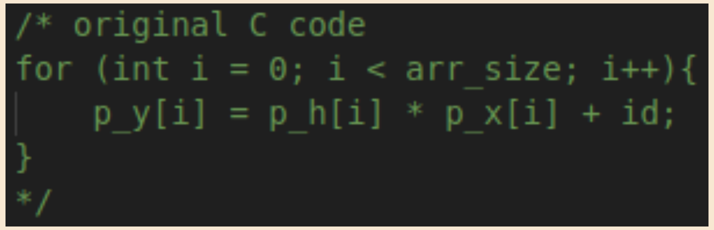
Exercise2-1(50%): Vectorized Code Example



A.1. Vector-vector add example

```
    # vector-vector add routine of 32-bit integers
    # void vvaddint32(size_t n, const int*x, const int*y, int*z)
    # { for (size_t i=0; i<n; i++) { z[i]=x[i]+y[i]; } }
    #
    # a0 = n, a1 = x, a2 = y, a3 = z
    # Non-vector instructions are indented
vvaddint32:
    vsetvli t0, a0, e32, ta, ma  # Set vector length based on 32-bit vectors
    vle32.v v0, (a1)         # Get first vector
      sub a0, a0, t0         # Decrement number done
      slli t0, t0, 2         # Multiply number done by 4 bytes
      add a1, a1, t0         # Bump pointer
    vle32.v v1, (a2)         # Get second vector
      add a2, a2, t0         # Bump pointer
    vadd.vv v2, v0, v1       # Sum vectors
    vse32.v v2, (a3)         # Store result
      add a3, a3, t0         # Bump pointer
      bnez a0, vvaddint32    # Loop back
      ret                    # Finished
```

Exercise2-2(10%): Vectorized Code

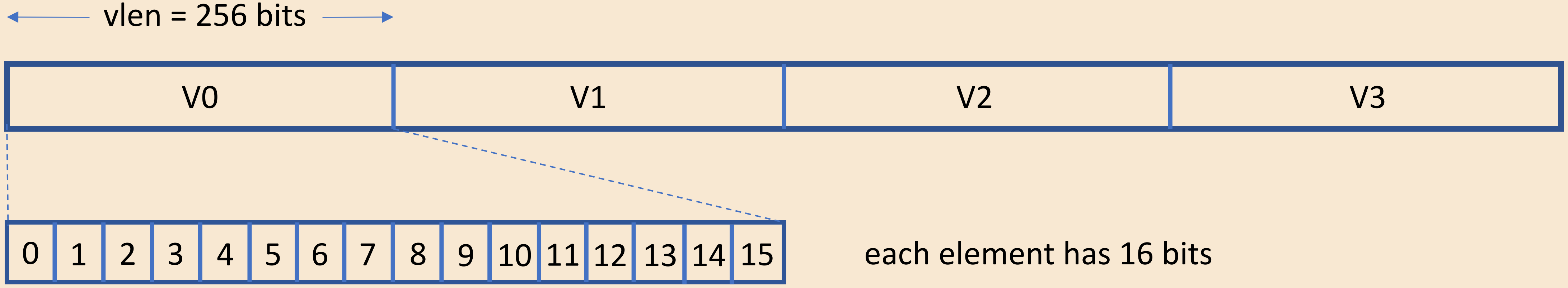


```
/* original C code
for (int i = 0; i < arr_size; i++){
    p_y[i] = p_h[i] * p_x[i] + id;
}
*/
```

Step1: Using the RISC-V V extension to Rewrite Assembly Code in exercise 2-1 and Report the Performance Statistics, run Spike simulator with custom configurations(vlen=256, elen=16) with the same speed as 2-1.

Step2: Insert the assembly code into arraymul_improved_version2.c to count the number of instructions, according to the types of the instructions, and to store the accumulated counts in the respective counters.

Step3: Add the related code(formula) in answer.h to set up the value for performance data.



each element has 16 bits

You have to complete exercise2-1 first!

- Speedup(10%)

Judge

We use makefile to judge your program. You can use the judge program to get the testing score by typing judge in your terminal.

- Test your Exercise1-1 in lab2

```
$ make judge_exercise1_1
```

- Test your Exercise1-2 in lab2

```
$ make judge_exercise1_2
```

- Test your Exercise2-1 in lab2

```
$ make judge_exercise2_1
```

- Test your Exercise2-2 in lab2

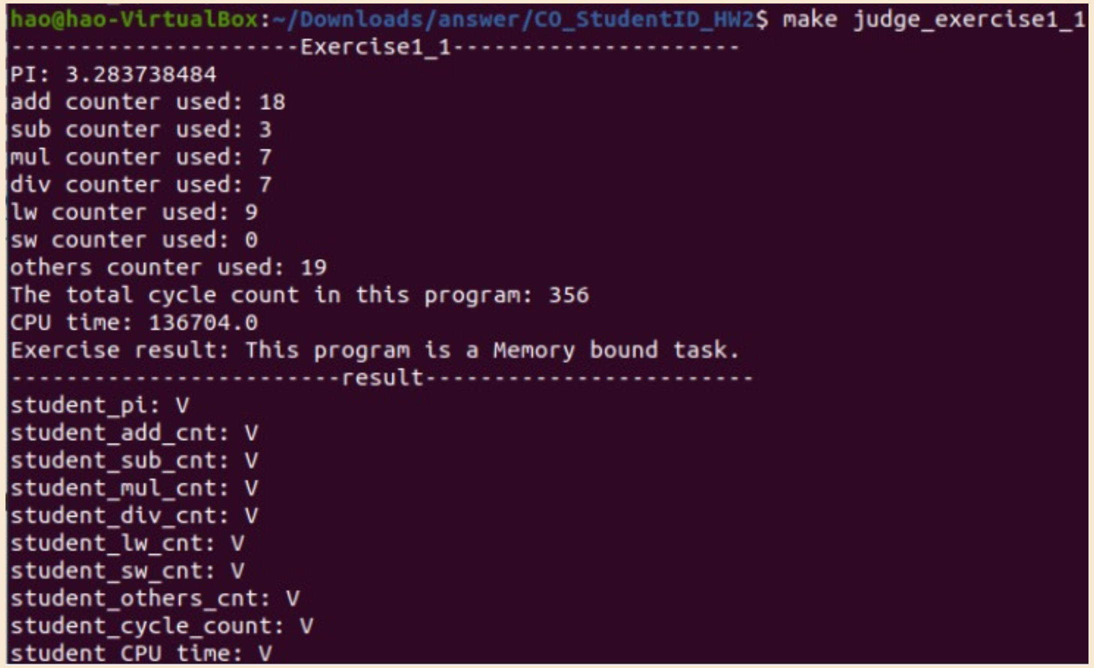
```
$ make judge_exercise2_2
```

Judge

We use makefile to judge your program. You can use the judge program to get the testing score by typing judge in your terminal.

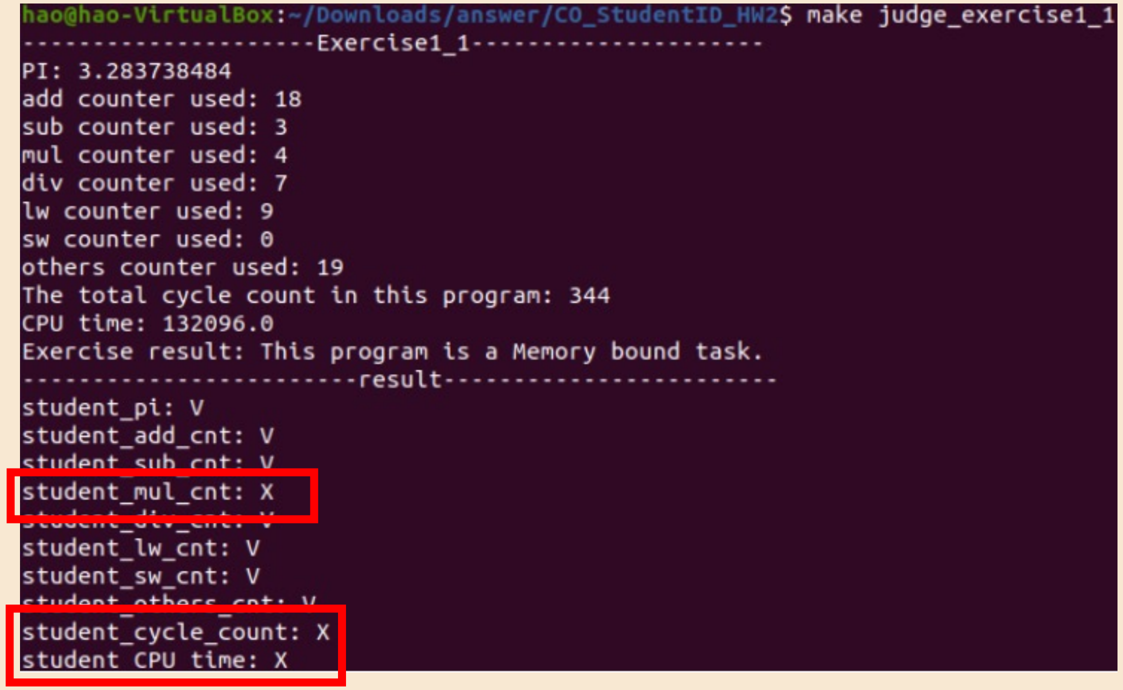
```
$ make judge_exercise1_1
```

- **Pass**



```
hao@hao-VirtualBox:~/Downloads/answer/CO_StudentID_HW2$ make judge_exercise1_1
--------------------Exercise1_1--------------------
PI: 3.283738484
add counter used: 18
sub counter used: 3
mul counter used: 7
div counter used: 7
lw counter used: 9
sw counter used: 0
others counter used: 19
The total cycle count in this program: 356
CPU time: 136704.0
Exercise result: This program is a Memory bound task.
------------------------result----------------------
student_pi: V
student_add_cnt: V
student_sub_cnt: V
student_mul_cnt: V
student_div_cnt: V
student_lw_cnt: V
student_sw_cnt: V
student_others_cnt: V
student_cycle_count: V
student_CPU_time: V
```

- **Error**



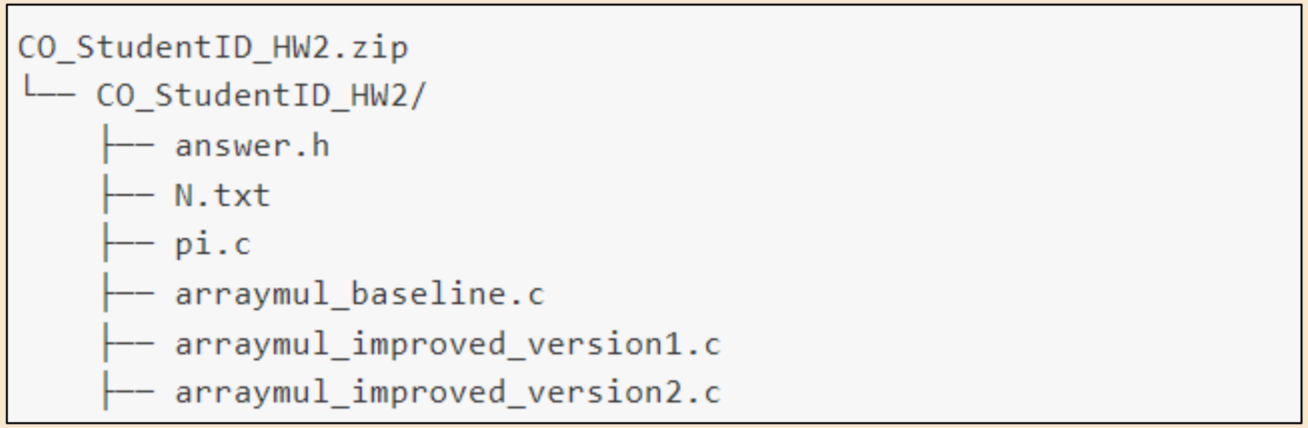
```
hao@hao-VirtualBox:~/Downloads/answer/CO_StudentID_HW2$ make judge_exercise1_1
--------------------Exercise1_1--------------------
PI: 3.283738484
add counter used: 18
sub counter used: 3
mul counter used: 4
div counter used: 7
lw counter used: 9
sw counter used: 0
others counter used: 19
The total cycle count in this program: 344
CPU time: 132096.0
Exercise result: This program is a Memory bound task.
------------------------result----------------------
student_pi: V
student_add_cnt: V
student_sub_cnt: V
student_mul_cnt: X
student_div_cnt: V
student_lw_cnt: V
student_sw_cnt: V
student_others_cnt: V
student_cycle_count: X
student_CPU_time: X
```

Submission

We assume your developed code is inside the folder: CO_StudentID_HW2. Please follow the instructions below to submit your programming assignment.

1. Compress your source code into a zip file.
2. Submit your homework with NCKU Moodle.
3. The zipped file and its internal directory organization of your developed code should be similar to the example below.

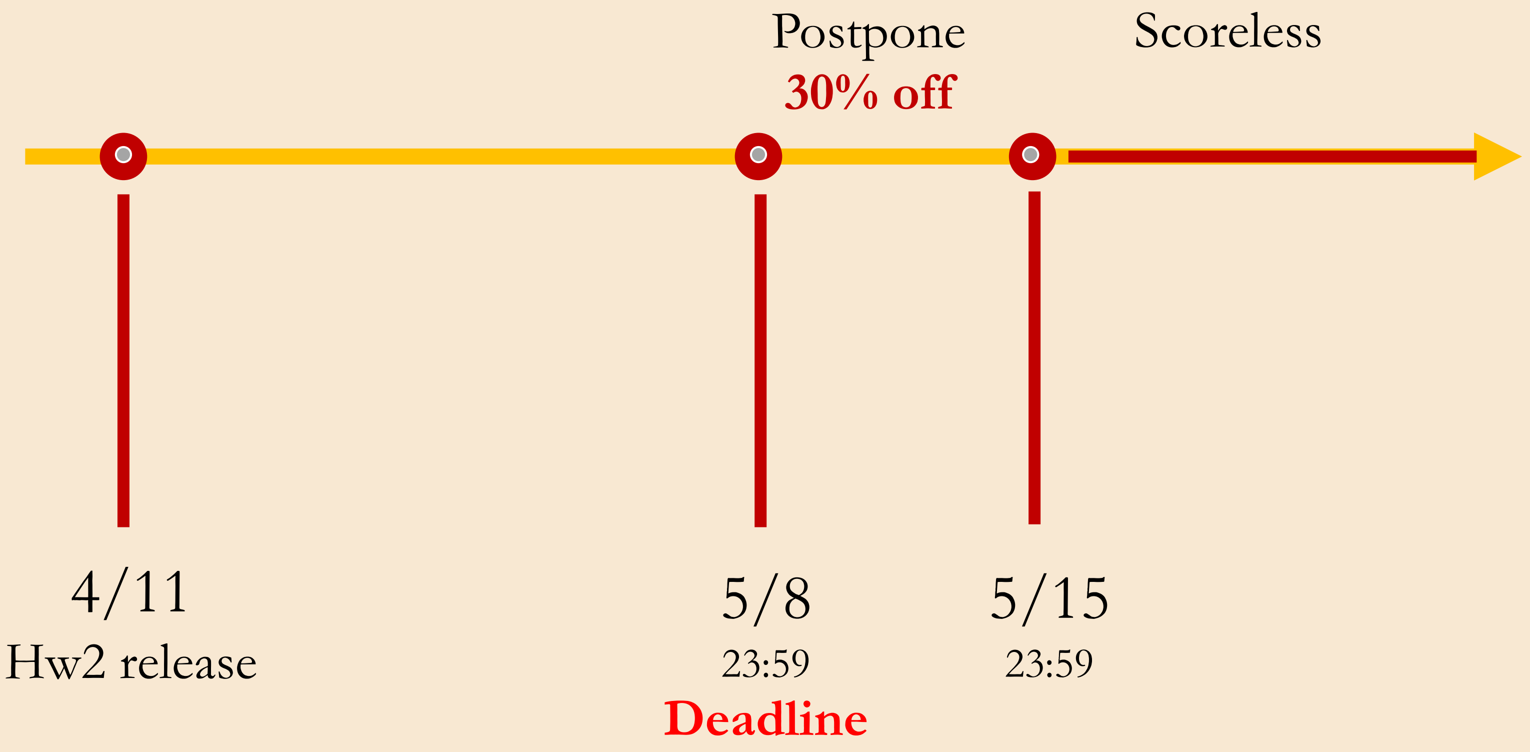
NOTE: Replace all StudentID with your student ID number!!



```
CO_StudentID_HW2.zip
└── CO_StudentID_HW2/
    ├── answer.h
    ├── N.txt
    ├── pi.c
    ├── arraymul_baseline.c
    ├── arraymul_improved_version1.c
    ├── arraymul_improved_version2.c
```

!!! Incorrect format (either the file structure or file name) will lose 10 points. !!!

Deadline


Postpone
30% off
Scoreless
4/11
Hw2 release
5/8
23:59
Deadline
5/15
23:59

How to Mail TAs

- Send mail to asrlab@csie.ncku.edu.tw, not any TA's mail!!
- Email subject starts with "[Comp2024]"
- Thoroughly read document before asking questions.