

# Computer Organization 2022

## Lab 2 – Assembly Lab I

### ( **Exercise & Report Format** )

**Video link** : <https://youtu.be/t9y12xV-M1I>

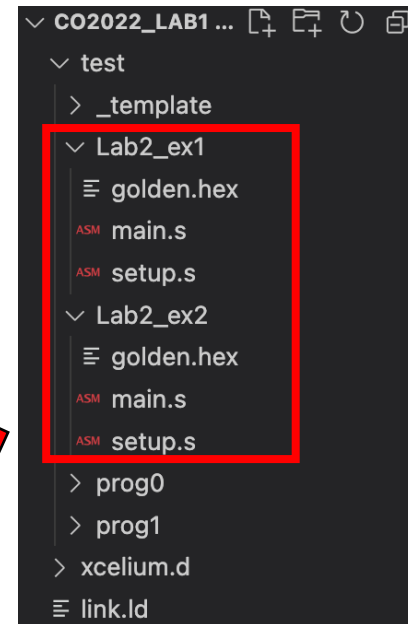
# Exercise Overview

## There are 2 exercises for Lab2

- In each exercise, TA would provide C code
- First, You need to read C code & write golden answer & write assembly code yourself to implement the program described in the C code
- Then, using the environment introduced in Lab1 to do cross-compilation, simulation with RISC-V CPU provided by TA, result checking & waveform explaining

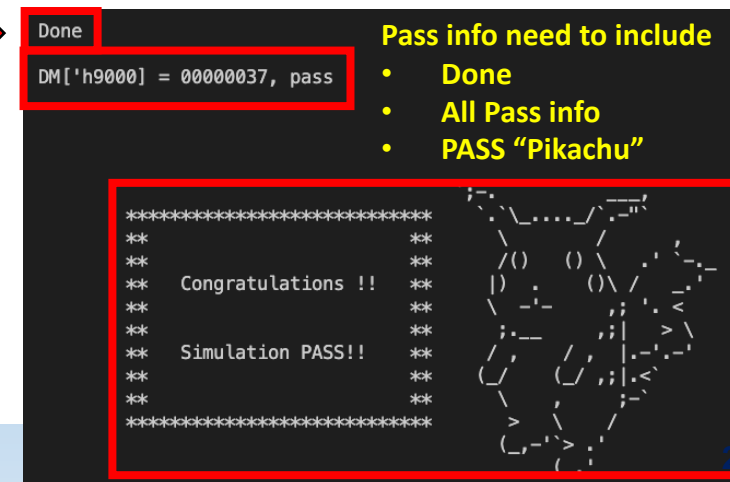
## [ Exercise Steps ]

1. Create two copies from \_template under /CO2022\_Lab1/test and rename Lab2\_ex1 and Lab2\_ex2
2. **Read** the C Code & **Write** golden answer yourself in **"golden.hex"**
3. **Write** assembly code yourself in **"main.s"** to implement the program described in C code
4. Cross-compile assembly code with RISC-V Toolchain => Generate .hex & .dump
5. Simulate running the program(.hex) you wrote on the CPU provided by TA with NC-Verilog => Generate result & .fsdb
6. Check the print information in the terminal & **Screenshot the pass information**
7. Check the waveform(.fsdb) with nWave & **Explain the waveform with "main.dump"**  
(You can reference the explanation TA demonstrated in Lab1 part3)

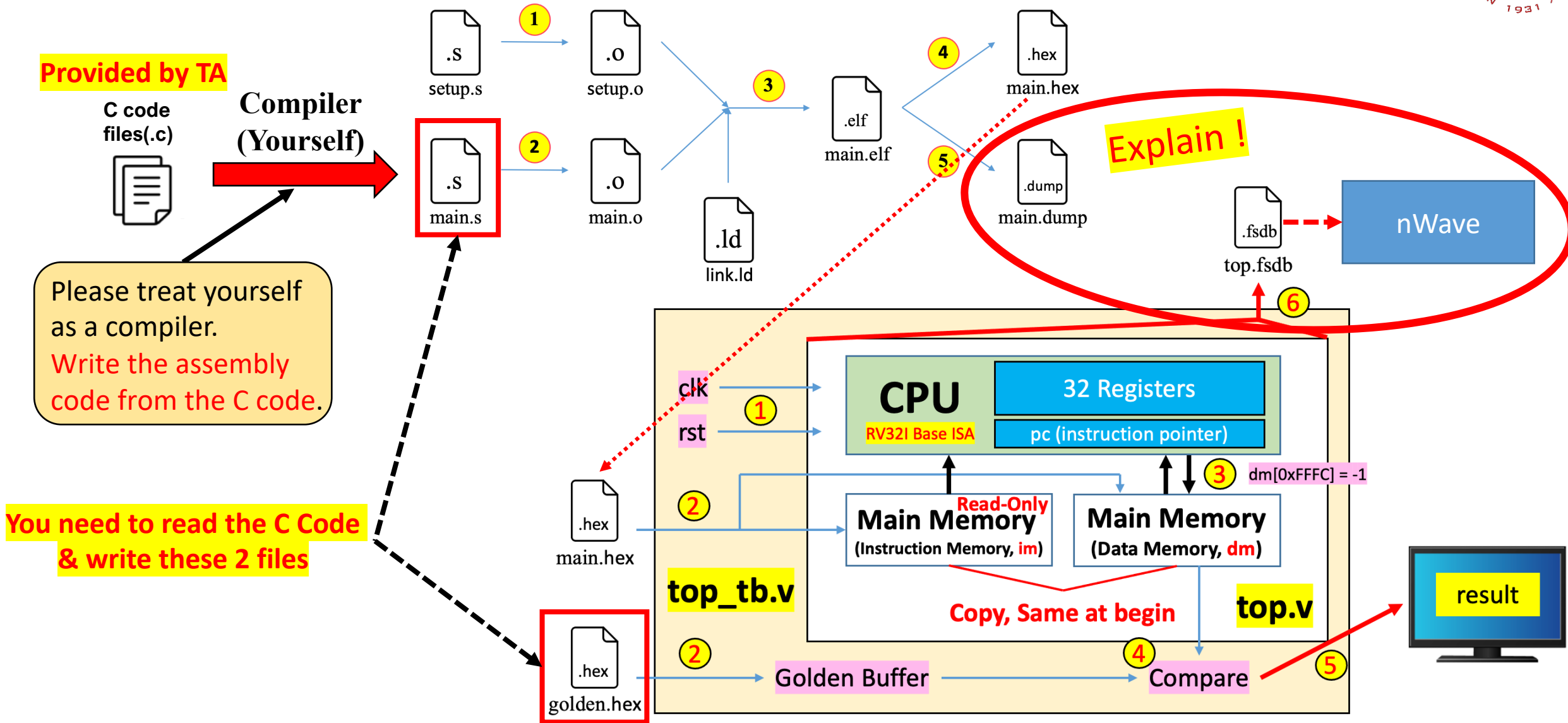


```
20 # #####
21 # ### Main Program
22 # #####
23
24 # ...
25
26 # #####
```

main.s (Write the assembly code here)



# Exercise Overview



# Exercise 1


1. Please read the C code, calculate the golden answer yourself and write the golden answer in /test/Lab2\_ex1/golden.hex yourself  
(If you miscalculate the answers, there will be errors later)
2. Please write the RISC-V assembly code in /test/Lab2\_ex1/main.s yourself to implement the program described in the C code  
(If the program you write is wrong, there will be errors later)
3. Please use RISC-V Toolchain to cross-compile the assembly code (setup.s & main.s => main.hex & main.dump)
  - Please compare the differences between main.s & main.dump (e.g. How Pseudo Instructions correspond to actual instructions)
4. Please use NC-Verilog to simulate running the program(main.hex) you wrote on the CPU provided by TA => get result & top.fsdb  
(If either step 1 or 2 is wrong, it will get wrong here)  
(If both step 1 & 2 are wrong, it could be right or wrong here)  
(TA will use the correct golden.hex written by TA to test your program)
5. Check the result in the terminal & Screenshot the pass information
6. Check the waveform(top.fsdb) with nWave & Explain the waveform with “main.dump”

```
1  int main() {
2      int *answer = (int *) 0x9000;
3
4      /*
5       * Attention : You cannot treat variable as a known value
6       *              You only know that variable is an integer
7       */
8      int a = 9;
9      int b = -13;
10     int ans0 = (a*5 + 30) * 7 + b*3;
11     int ans1 = a*(-3) + b*(-5);
12     int ans2 = abs(a) + abs(b); // ans2 = |a| + |b|
13     int ans3 = a%4 + b%4;
14     int ans4 = (int)(a / 8) + (int)(b / 8);
15
16     int c = 7;
17     int d = -15;
18     int ans5 = 100 * 5.625;
19     int ans6 = -5 * 3.5;
20     int ans7 = 3 * 0.75;
21     int ans8 = (int)(c * 0.75) + (int)(d * 0.75);
22
23     *answer = ans0;
24     *(answer+1) = ans1;
25     *(answer+2) = ans2;
26     *(answer+3) = ans3;
27     *(answer+4) = ans4;
28     *(answer+5) = ans5;
29     *(answer+6) = ans6;
30     *(answer+7) = ans7;
31     *(answer+8) = ans8;
32
33     return 0;
34 }
```

## Attention :

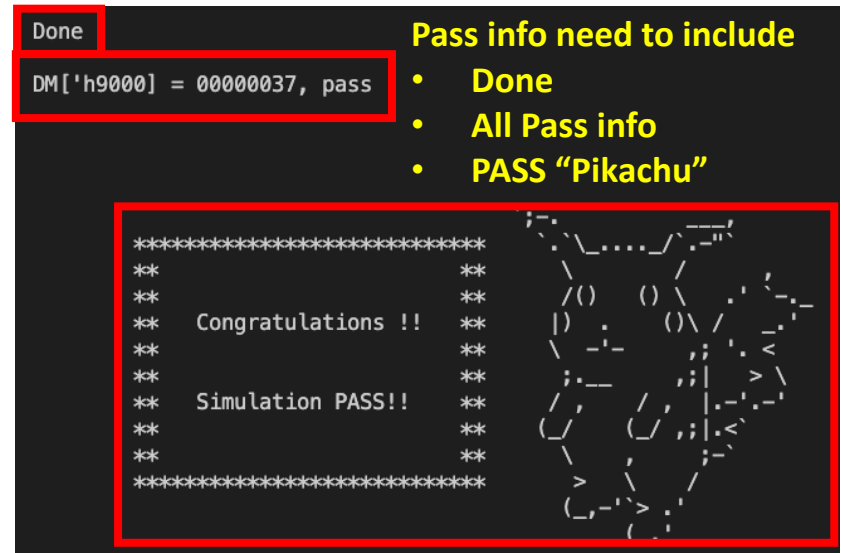
- You cannot treat variable as a known value
- You only know that variable is an integer

# Exercise 1 (Report Requirements)

1. Please screenshot your golden.hex (**Need 9 answers**)
2. Please explain how you implement the following C code with RISC-V assembly code we learned this time
  1. `Variable * -3`
  2. `abs(Variable)`
  3. `Variable % 4`
  4. `(int)(Variable / 8)`
  5. `(int)(100 * 5.625)`
  6. `(int)(-5 * 3.5)`
  7. `(int)(3 * 0.75)`
  8. `(int)(Variable * 0.75)`
3. Please explain if just using the assembly code we learned this time is enough to do `Variable * Variable`, how or why not?
4. Please compare the differences between `main.s` & `main.dump` (e.g. How Pseudo Instructions correspond to actual instructions & other differences you found)
5. Please screenshot the pass information 
6. Please explain the waveform at the 8 locations listed above with “`main.dump`”

**Attention again :**


- You cannot treat variable as a known value
- You only know that variable is an integer



# Exercise 2

## `asm volatile ();`

- This is a way to write assembly code in C code
  - The compiler will just copy the contents in () to the location in the .s that corresponds to C code without doing anything with it
1. Please read the C code, calculate the golden answer yourself and write the golden answer in /test/Lab2\_ex2/golden.hex yourself
  2. Please write the RISC-V assembly code in /test/Lab2\_ex2/main.s yourself to implement the program described in the C code and copy the contents of `asm volatile` under your code
  3. Then do same things as exercise 1
    - Cross-Compile
    - Simulate
    - Check result
    - Check waveform



```
# #####  
# ### Main Program  
# #####  
  
# write the assembly code here  
# ...  
  
    la    s0, _answer  
    lb    t0, 0(s0)  
    sh    t0, 8(s0)  
    lhu   t1, 0(s0)  
    sw    t1, 12(s0)  
    lw    t2, 4(s0)  
    sb    t2, 16(s0)  
  
# #####
```

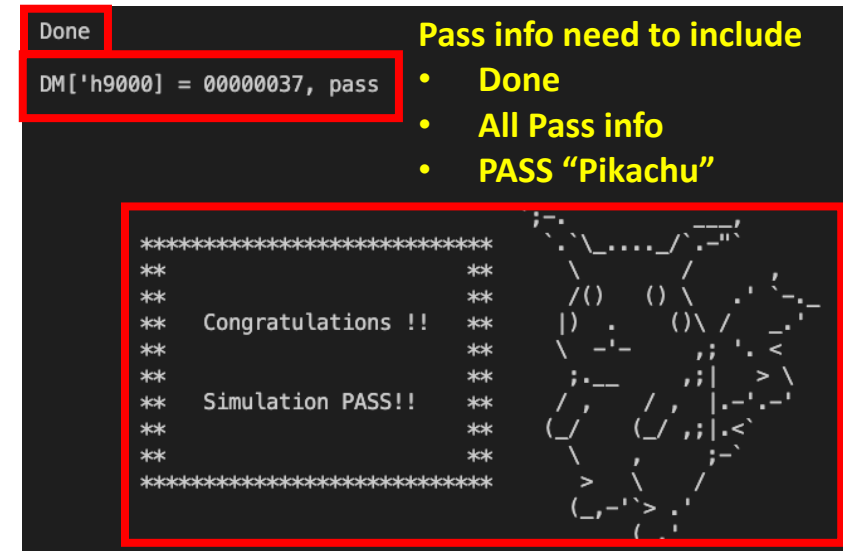
```
1  int main() {  
2      int *answer = (int *) 0x9000;  
3  
4      int ans0 = -775687803; // 0xD1C3F185  
5      int ans1 = 3365119; // 0x003358FF  
6  
7      *answer = ans0;  
8      *(answer+1) = ans1;  
9  
10     asm volatile (  
11         "la    s0, _answer\n\t"  
12         "lb    t0, 0(s0)\n\t"  
13         "sh    t0, 8(s0)\n\t"  
14         "lhu   t1, 0(s0)\n\t"  
15         "sw    t1, 12(s0)\n\t"  
16         "lw    t2, 4(s0)\n\t"  
17         "sb    t2, 16(s0)\n\t"  
18     );  
19  
20     return 0;  
21 }
```

**A way to write assembly code in C code**



# Exercise 2 (Report Requirements)

1. Please screenshot your golden.hex (**Need 5 answers**)
- 2. Please explain how you put long integers (0xD1C3F185, 0x003358FF) into registers
- 3. Please explain how you figured out the answers at 0x9008, 0x900c, 0x9010
4. Please compare the differences between main.s & main.dump  
(e.g. How Pseudo Instructions correspond to actual instructions & other differences you found)
5. Please screenshot the pass information
6. Please use the waveform & “main.dump”  
to explain and verify your calculations are correct



The screenshot shows a debugger window with a dark background. At the top left, a small box says "Done". Below it, a line of code is highlighted: `DM['h9000'] = 00000037, pass`. To the right of this, the text "Pass info need to include" is followed by a bulleted list: 

- Done
- All Pass info
- PASS "Pikachu"

 Below the code, there is a large block of text enclosed in a red border. It contains a congratulatory message: 

```
*****
**                                     **
**      Congratulations !!            **
**                                     **
**      Simulation PASS!!            **
**                                     **
*****
```

 To the right of this text is a large, stylized ASCII art of a Pikachu's head.

# Question List

Q1 :  
 Why IALIGN of RV32I is 32 bits ?  
 Why IALIGN need to support 16 bits ?  
 Why IALIGN have no greater than 32 bits (e.g. 64 / 128 bits) ?

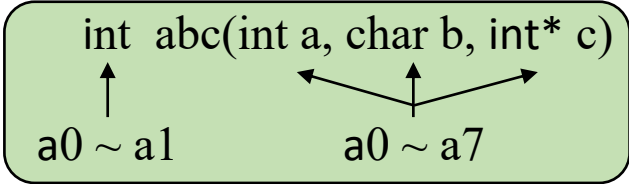
Q4 : Why there is no lwu in RV32I ?

Q5 : What is the addressing mode of Load & Store ?

Q2 : Why temporary registers and saved registers are not numbered sequentially ?



# Registers	Base	Version	Status	
	RVWMO	2.0	Ratified	Weak Memory Ordering
32	RV32I	2.1	Ratified	Base Integer Instruction Set, 32-bit
32	RV64I	2.1	Ratified	Base Integer Instruction Set, 64-bit
16	RV32E	1.9	Draft	Base Integer Instruction Set ( <b>embedded</b> ), 32-bit
32	RV128I	1.7	Draft	Base Integer Instruction Set, 128-bit



Q3 : Why return value needs 2 registers (a0, a1) ?

	int	long	pointer
ilp32/ilp32f/ilp32d	32-bit	32-bit	32-bit
lp64/lp64f/lp64d	32-bit	64-bit	64-bit

	ILP32	LP64	LLP64	ILP64
char	8	8	8	8
short	16	16	16	16
int	32	32	32	64
long	32	64	32	64
long long	64	64	64	64
void *	32	64	64	64

LLP = long long & pointer

hint → RISC-V 只支援這六種



# Report Format (Chinese or English are both ok)

- Cover (There is a default format of the report on the moodle)
- Contents of the report
  1. Answers of “Question List”
  2. Report Requirement of “Exercise 1”

助教在評分的時候  
會依據你的解釋程度給分  
解釋的越“完整清楚簡潔”  
分數就會越高
  3. Report Requirement of “Exercise 2”

Please convert the report to pdf before submitting to moodle

# File structure for submission

