Computer Architecture
HW1: RARS Simulator

TA: 何承叡 (Ray)

Due: 2022/10/10 23:59 (UTC+8)

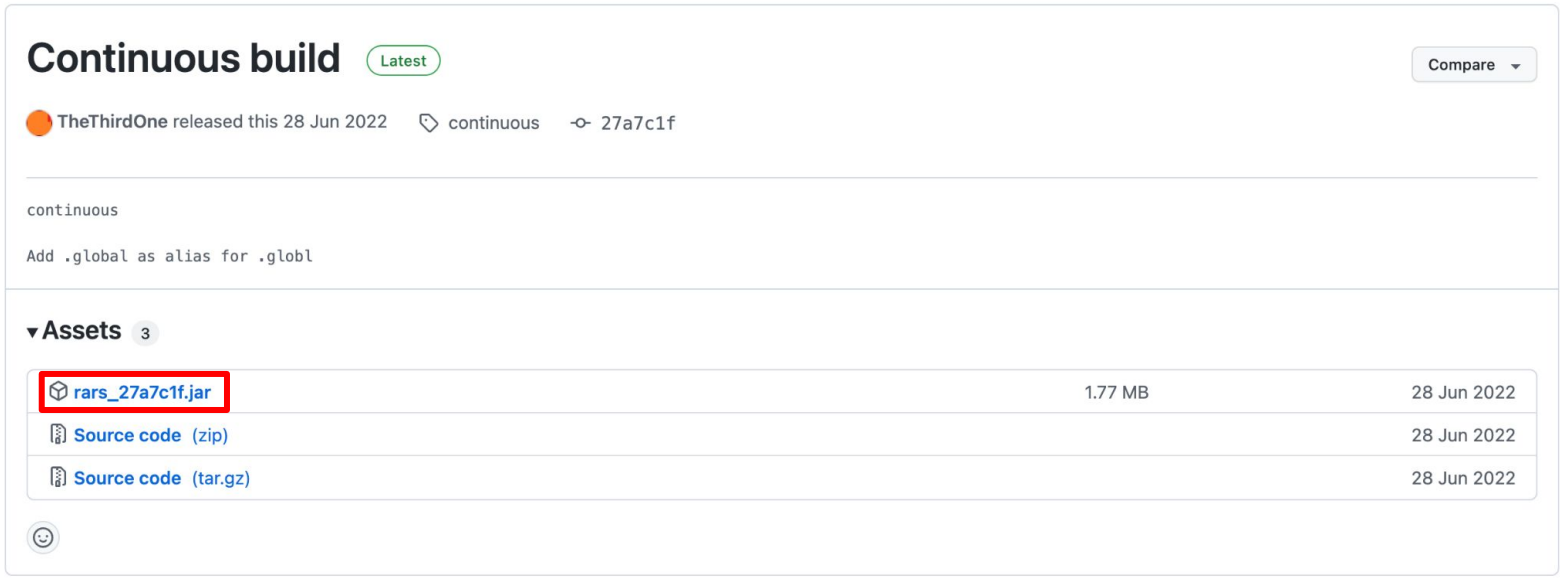
Email: r10943015@ntu.edu.tw

Outline

- RARS: RISC-V Simulator
- GUI of RARS
- HW1-1 Sort Procedure
- HW1-2 Salt-and-Pepper noise denoiser with median filter
- Submission
- Rules

RARS: RISC-V Simulator

- An open source RISC-V assembler and runtime simulator
- Support riscv32 and riscv64 on Windows/Mac/Linux
- Download the .jar file here: https://github.com/TheThirdOne/rars/releases/tag/continuous



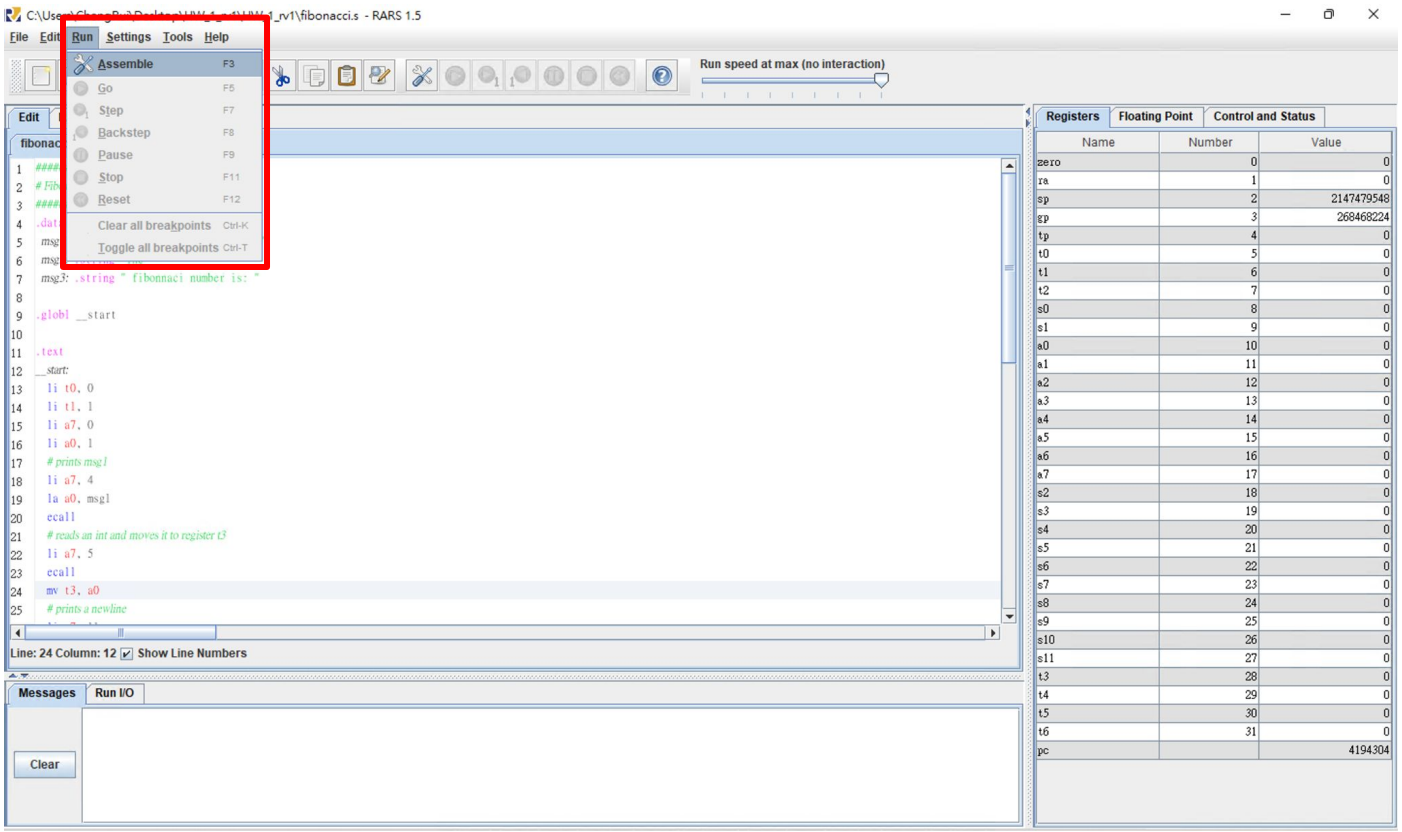
RARS: RISC-V Simulator

- To launch .jar file, make sure you install java
- https://www.java.com/en/download/

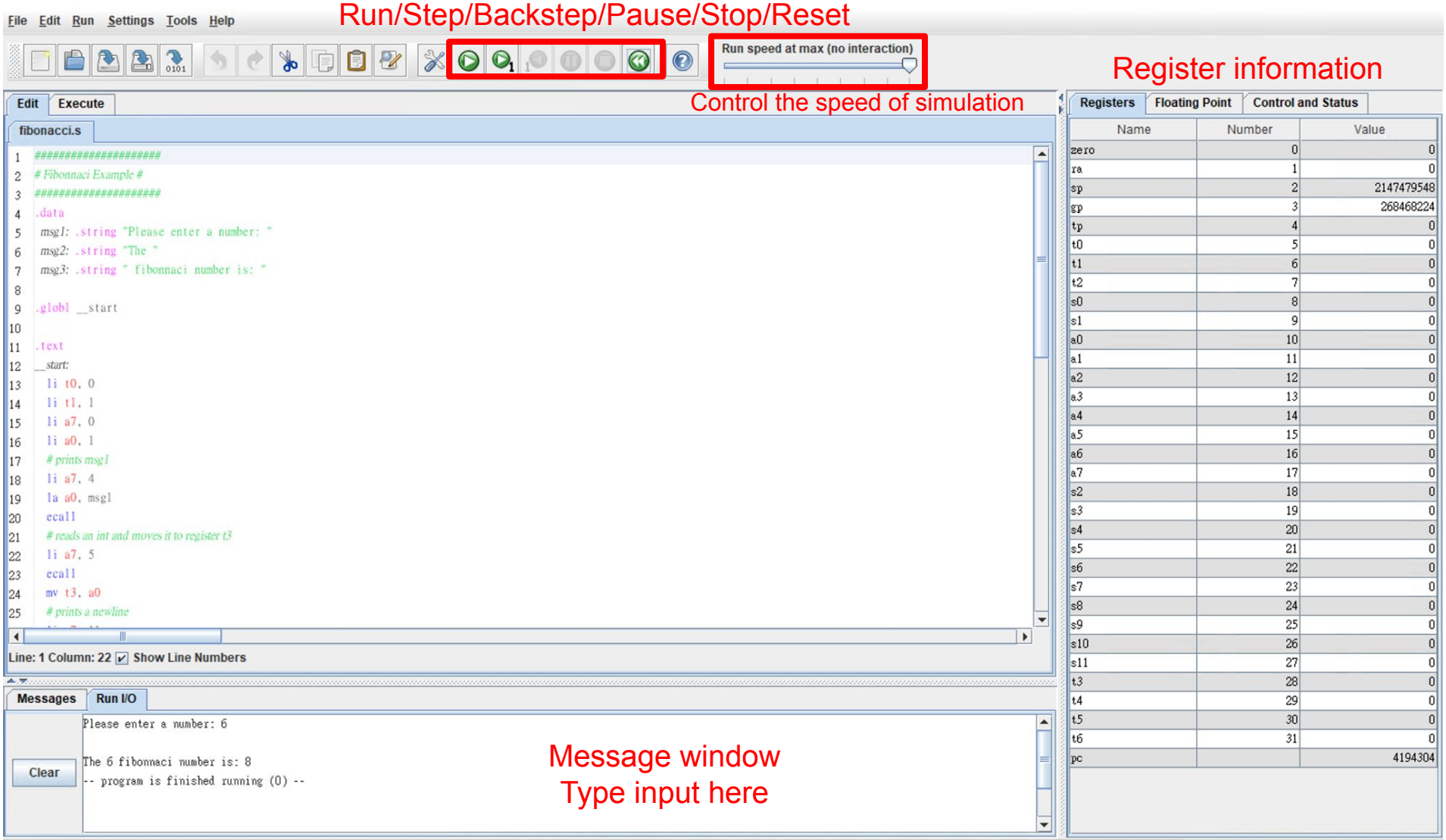
- Click the .jar file to open it

GUI of RARS

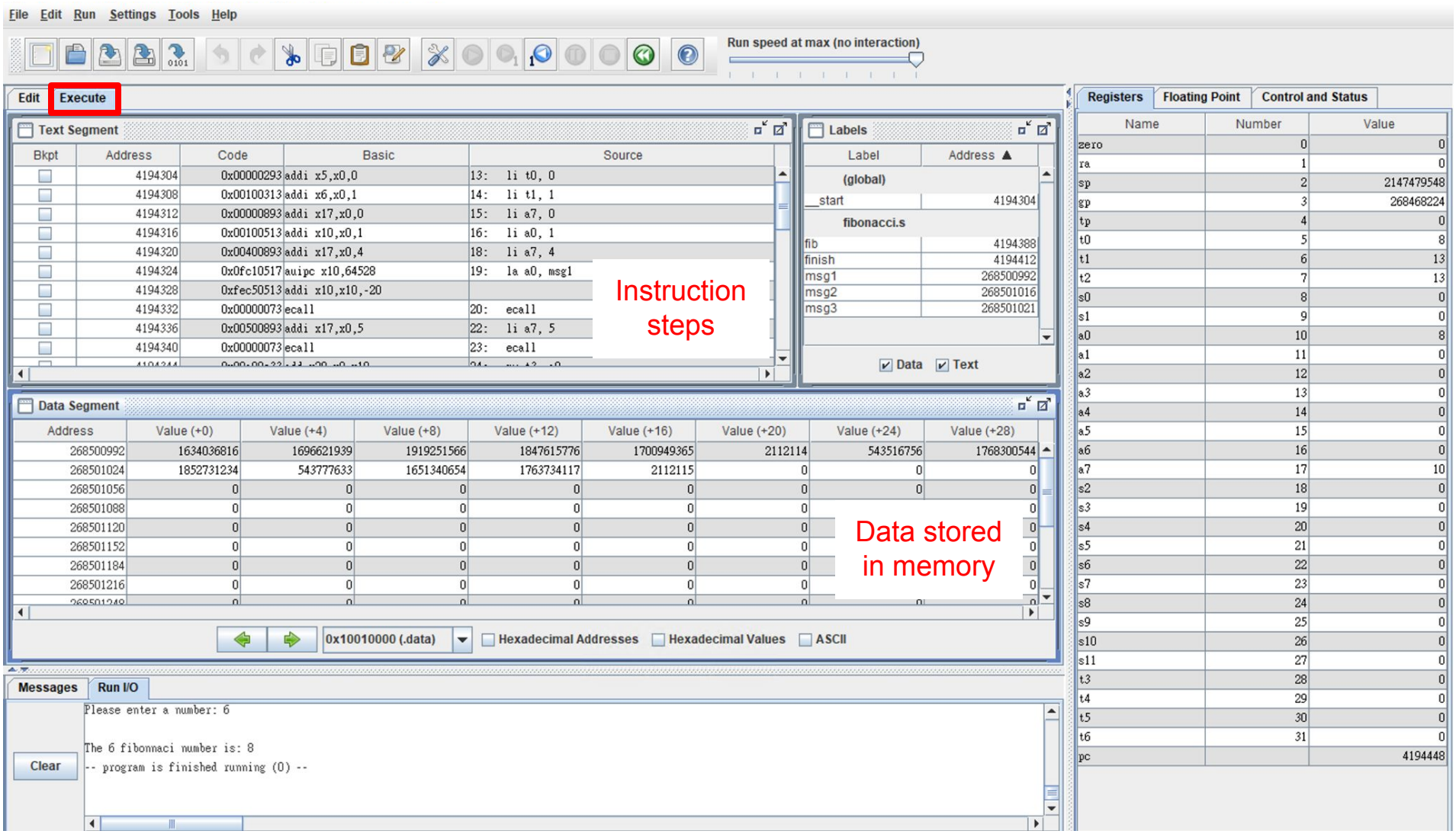
- Open a file
  - File -> Open
- Run the code
  - Run -> Assemble



GUI of RARS

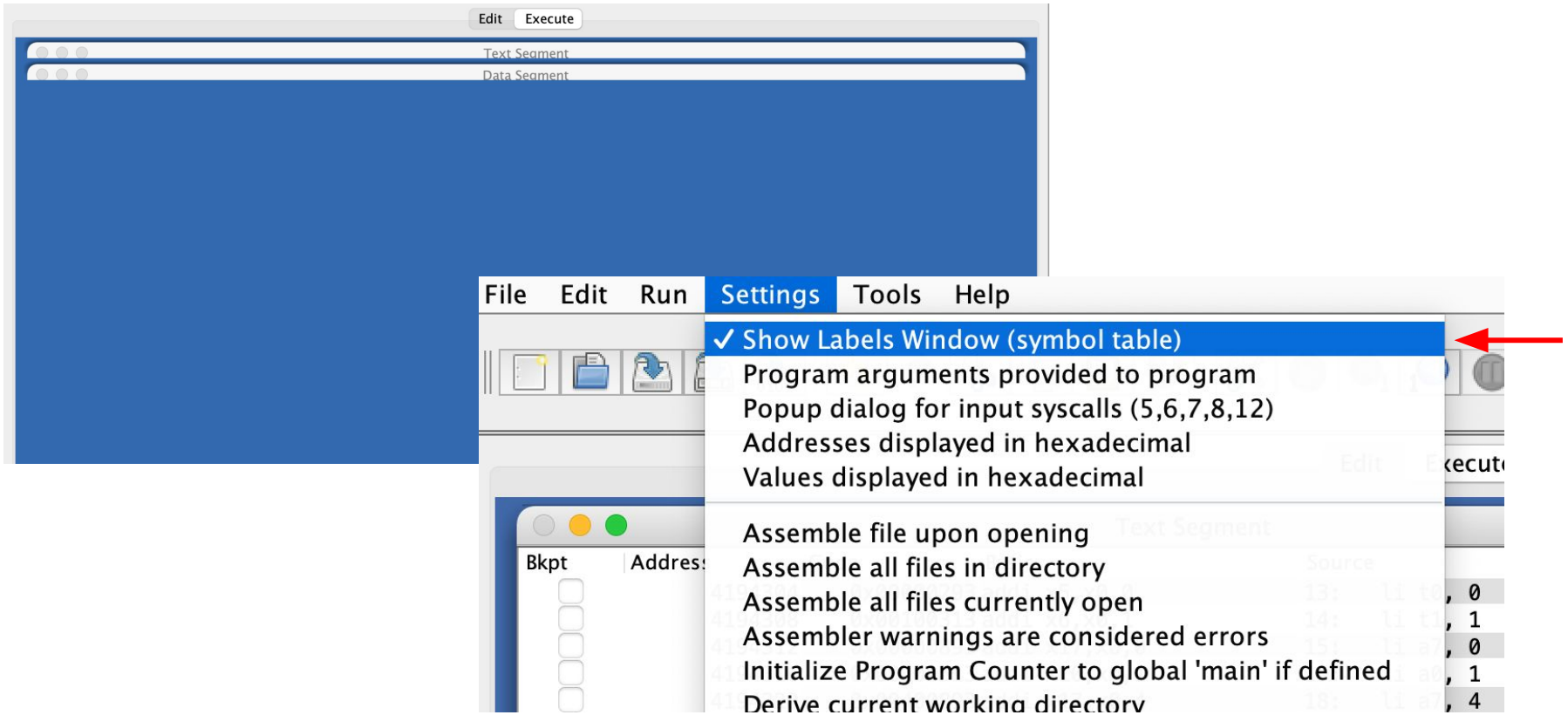

Run/Step/Backstep/Pause/Stop/Reset
Run speed at max (no interaction)
Control the speed of simulation
Register information
File Edit Run Settings Tools Help
Edit Execute
fibonacci.s
Line: 1 Column: 22 Show Line Numbers
Registers Floating Point Control and Status
Name Number Value
zero 0 0
ra 1 0
sp 2 2147479548
gp 3 268468224
tp 4 0
t0 5 0
t1 6 0
t2 7 0
s0 8 0
s1 9 0
a0 10 0
a1 11 0
a2 12 0
a3 13 0
a4 14 0
a5 15 0
a6 16 0
a7 17 0
s2 18 0
s3 19 0
s4 20 0
s5 21 0
s6 22 0
s7 23 0
s8 24 0
s9 25 0
s10 26 0
s11 27 0
t3 28 0
t4 29 0
t5 30 0
t6 31 0
pc 4194304
Messages Run I/O
Clear
Please enter a number: 6
The 6 fibonacci number is: 8
-- program is finished running (0) --
Message window
Type input here

GUI of RARS


File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment
Bkpt Address Code Basic Source
4194304 0x00000293 addi x5,x0,0 13: li t0, 0
4194308 0x00100313 addi x6,x0,1 14: li t1, 1
4194312 0x00000893 addi x17,x0,0 15: li a7, 0
4194316 0x00100513 addi x10,x0,1 16: li a0, 1
4194320 0x00400893 addi x17,x0,4 18: li a7, 4
4194324 0x0fc10517 auipc x10,64528 19: la a0, msg1
4194328 0xfec50513 addi x10,x10,-20
4194332 0x00000073 ecall 20: ecall
4194336 0x00500893 addi x17,x0,5 22: li a7, 5
4194340 0x00000073 ecall 23: ecall
Instruction steps
Labels
Label Address
(global)
__start 4194304
fibonacci.s
fib 4194388
finish 4194412
msg1 268500992
msg2 268501016
msg3 268501021
Data Text
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+12) Value (+16) Value (+20) Value (+24) Value (+28)
268500992 1634036816 1696621939 1919251566 1847615776 1700949365 2112114 543516756 1768300544
268501024 1852731234 543777633 1651340654 1763734117 2112115 0 0 0
268501056 0 0 0 0 0 0 0 0
268501088 0 0 0 0 0 0 0
268501120 0 0 0 0 0 0 0
268501152 0 0 0 0 0 0 0
268501184 0 0 0 0 0 0 0
268501216 0 0 0 0 0 0 0
Data stored in memory
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII
Messages Run I/O
Please enter a number: 6
The 6 fibonacci number is: 8
-- program is finished running (0) --
Clear
Registers Floating Point Control and Status
Name Number Value
zero 0 0
ra 1 0
sp 2 2147479548
gp 3 268468224
tp 4 0
t0 5 8
t1 6 13
t2 7 13
s0 8 0
s1 9 0
a0 10 8
a1 11 0
a2 12 0
a3 13 0
a4 14 0
a5 15 0
a6 16 0
a7 17 10
s2 18 0
s3 19 0
s4 20 0
s5 21 0
s6 22 0
s7 23 0
s8 24 0
s9 25 0
s10 26 0
s11 27 0
t3 28 0
t4 29 0
t5 30 0
t6 31 0
pc 4194448

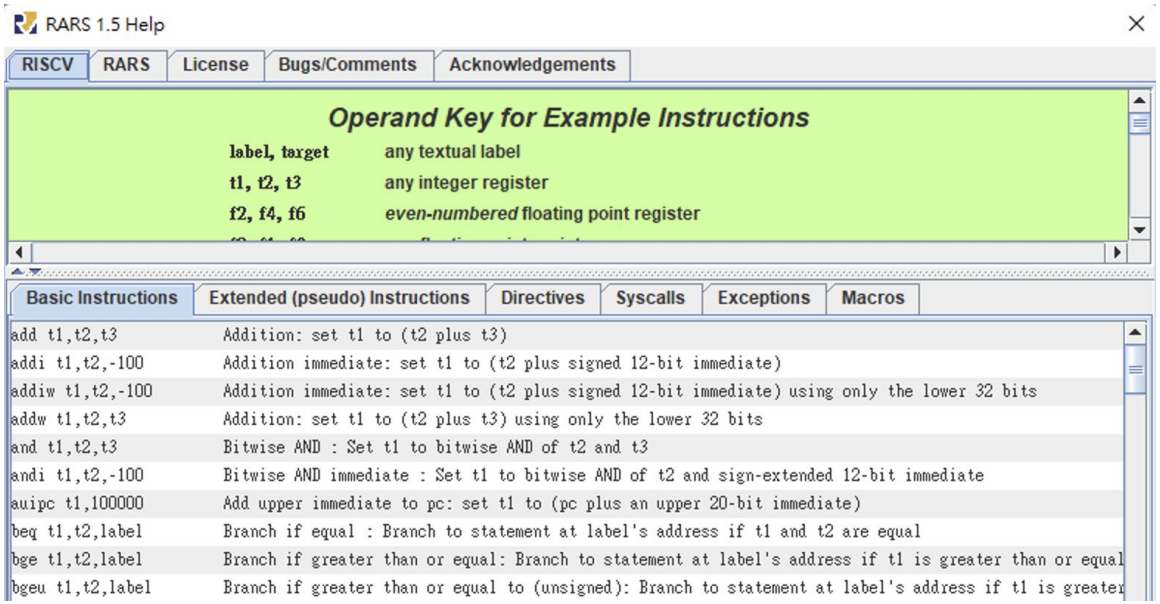
GUI of RARS

- If you cannot open Text and Data section window successfully (especially for Mac user)
  - Settings -> Show Labels Window
  - Issue: https://github.com/TheThirdOne/rars/issues/116



GUI of RARS

- RARS information
  - Help -> Help
- Change to 64-bit RISC-V
  - Settings -> 64 bit
- Dump memory
  - File -> Dump Memory -> Select your preference



HW1-1: Sort Procedure

- Modify HW1_1.s to implement a sorting function
- Input
  - An integer array
  - {10, -2, 4, -7, 6, 9, 3, 1, -5, -8}
- Output
  - A sorted array in **ascending order**
- Useful sorting algorithm
  - Bubble sort…
- Submission
  - Snapshot the sorting result

```
This is HW1_1:
Before sorting:
1  -3  5  2  -4  10  -9  8  -11  23
After sorting:
-11  -9  -4  -3  1  2  5  8  10  23
-- program is finished running (0) --
```

Screenshot example

Template of HW1-1

- The num address is stored in a2
- The length of num is stored in a3
- The **output num address** should be stored in **a2**
- Write your code in the red frame

```
11  main:
12    # Print initiate
13    li a7, 4
14    la a0, str1
15    ecall
16
17    # a2 stores the num address, a3 stores the length of num
18    la a2, num
19    li a3, 10
20    jal prints
21
22    la a2, num
23    li a3, 10
24    jal sort
25
26    # Print result
27    li a7, 4
28    la a0, str2
29    ecall
30
31    la a2, num
32    li a3, 10
33    jal prints
34
35    # End the program
36    li a7, 10
37    ecall
38  #---------------------------------------------Do not modify above text---------------------------------------------
39  sort:
40  ### To Do ###
41
42
43
44    jr ra
45
46
47  #---------------------------------------------Do not modify below text---------------------------------------------
48  # Print function
49  prints:
50    mv t0, zero # for(i=0)
51    # a2 stores the num address, a3 stores the length of num
52    mv t1, a2
```

HW1-2: Denoiser with Median Filter

- Salt-and-pepper noise is an impulse noise caused by sharp and sudden disturbances in an image
  - Since the noise is whether white or black, it can be denoised by a median filter

- For a 3x3 median filter, the filter sorts all 9 numbers in the array and replaces the center element with the median of the sorted array

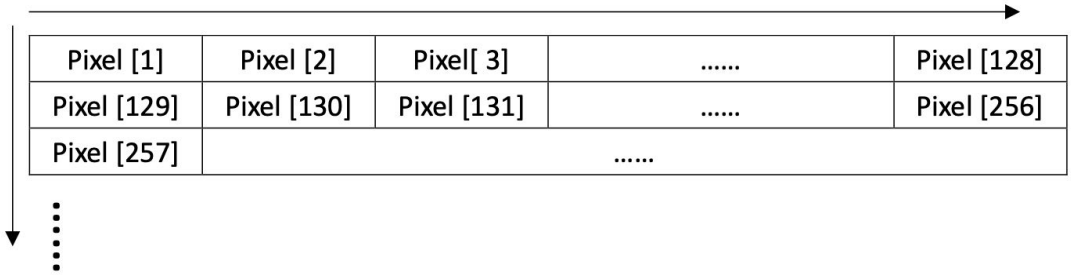
7	5	3
1	0	4
6	2	8

Sorting : 0, 1, 2, 3, 4, 5, 6, 7, 8

7	5	3
1	4	4
6	2	8

HW1-2: Denoiser with Median Filter

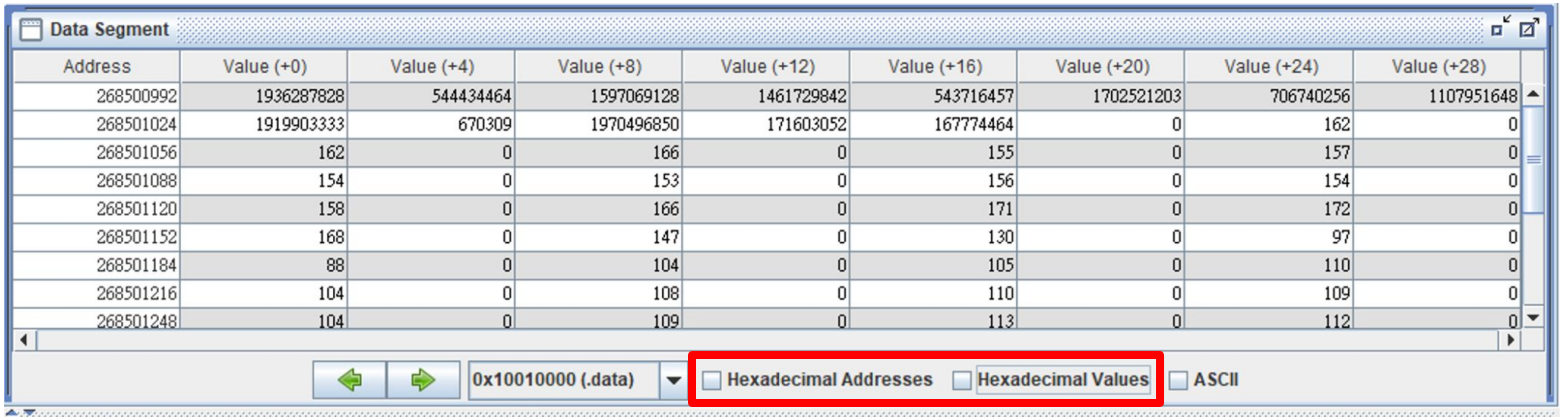
- Apply 3x3 median filter on a 128x128 an image with salt-and-pepper noise



- The filter only work on the input image, which means the previous updated pixel won't affect the latter result
- Do not consider boundary condition, so you don't need to calculate when the center pixel is on or out of the boundaries
  - 128x128 input image -> 126x126 output image

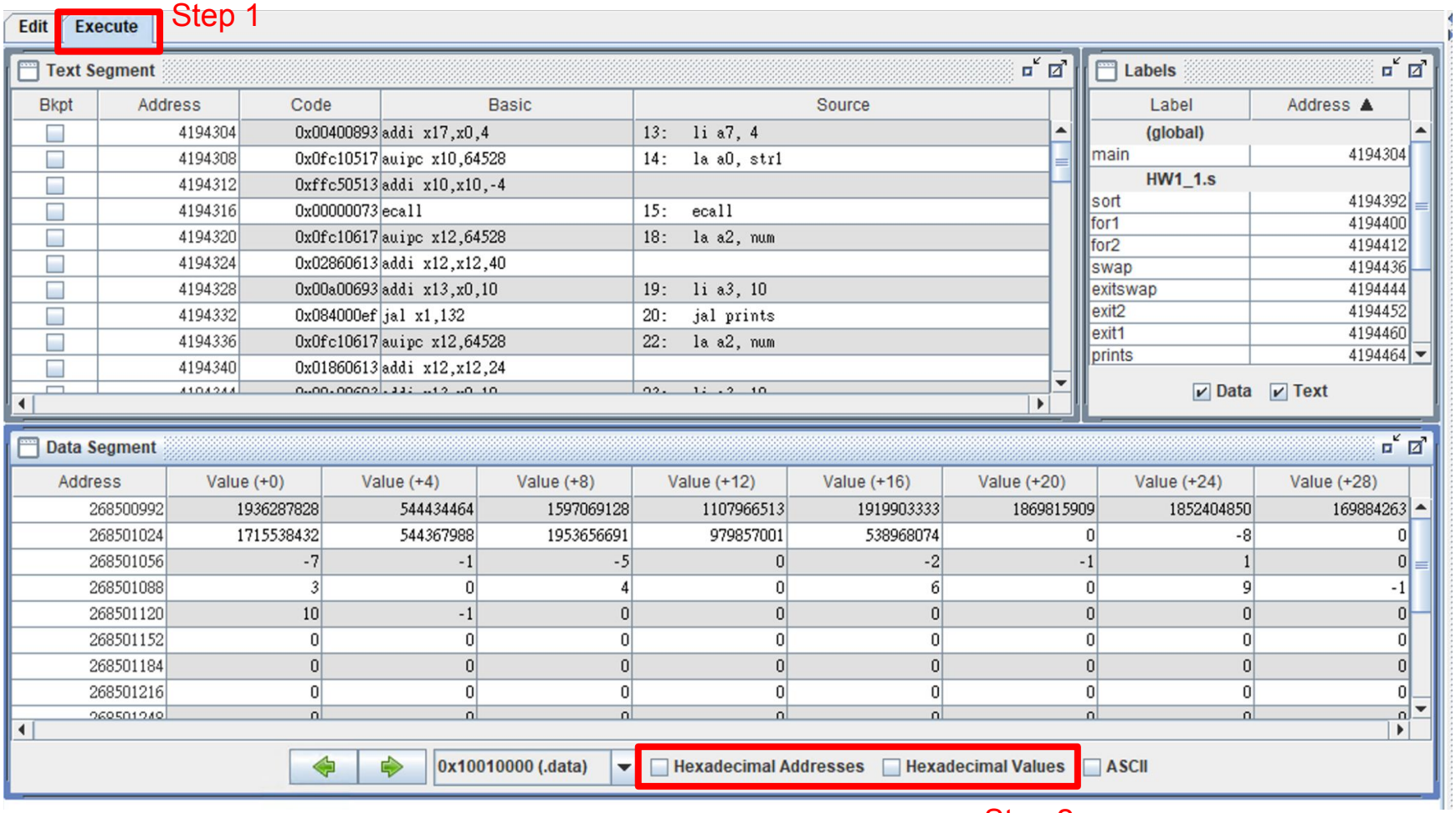
HW1-2: Denoiser with Median Filter

- Modify HW1_2.s to complete your denoising procedure
  - We also provide two small arrays (7x7 and 10x14) in testing_data.txt for you to debug easier
- You should dump memory.txt for our convenience to verify the answer (input image 128x128 version)
  - File -> Dump Memory -> Memory Segment(.data) -> Text/Data -> Dump To File… -> memory.txt
  - Remember to uncheck Hexadecimal Addresses and Hexadecimal Values before dumping memory

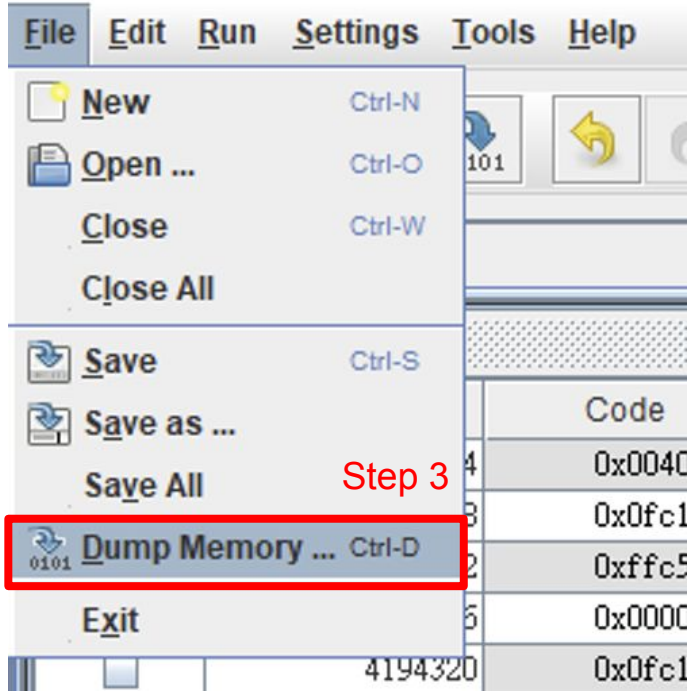


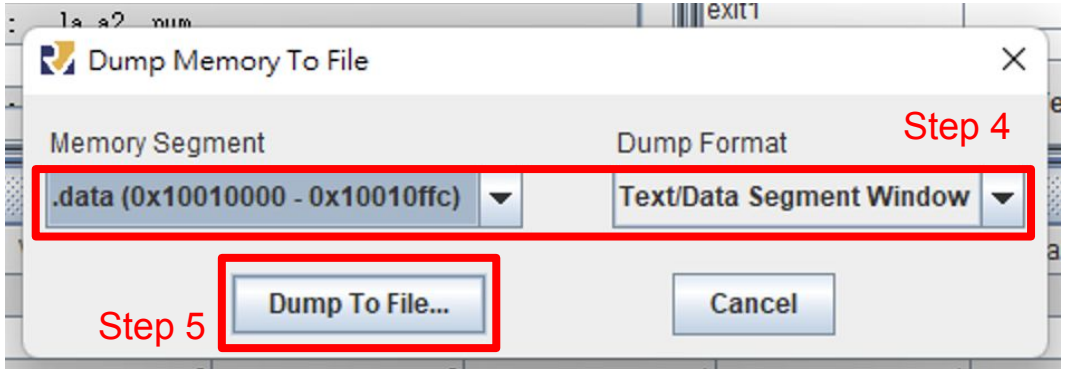
Dump Memory File

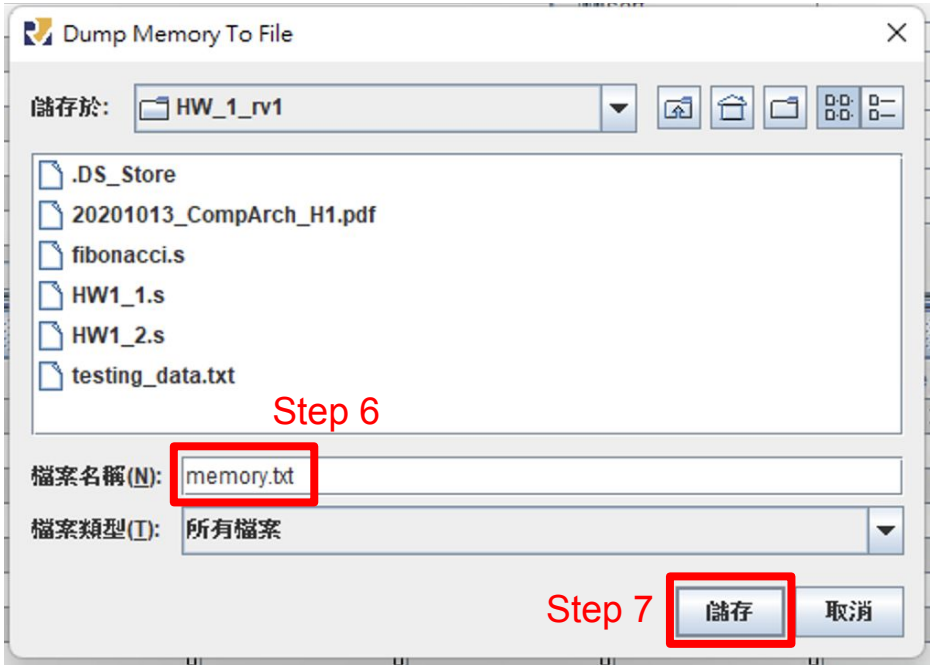
- Uncheck Hexadecimal Addresses and Hexadecimal Values



Dump Memory File







Submission

- HW1-1: snapshot the sorting result on console window and save as HW1_1.jpg
- HW1-2: dump data memory into file memory.txt
- If you have trouble in HW1-2, you can write the implementation details in report.pdf to get partial credit!

Submission

- Deadline: 2022/10/10 23:59 (UTC+8)
  - No late submission allowed
- Hand in results on NTU COOL
- Your homework should be copied into a folder and packed into a zip file with the following naming rules (5% penalty for wrong format)
  - hw1_<student_id>.zip
    - hw1_<student_id>
      - HW1_1.s, HW1_2.s (assembly code)
      - HW1_1.jpg (HW1-1 sorting result screenshot)
      - memory.txt (HW1-2 memory data file)
      - report.pdf (If you can't finish memory.txt)
      - README (If you think you need it)
  - Ex: hw1_r10943006.zip

Rules

- You should finish your homework on your own
- Do NOT share your codes or copy other people's codes
- Do NOT modify the input, output, and any provided instructions