

# Programming Assignment #1: Human Compiler

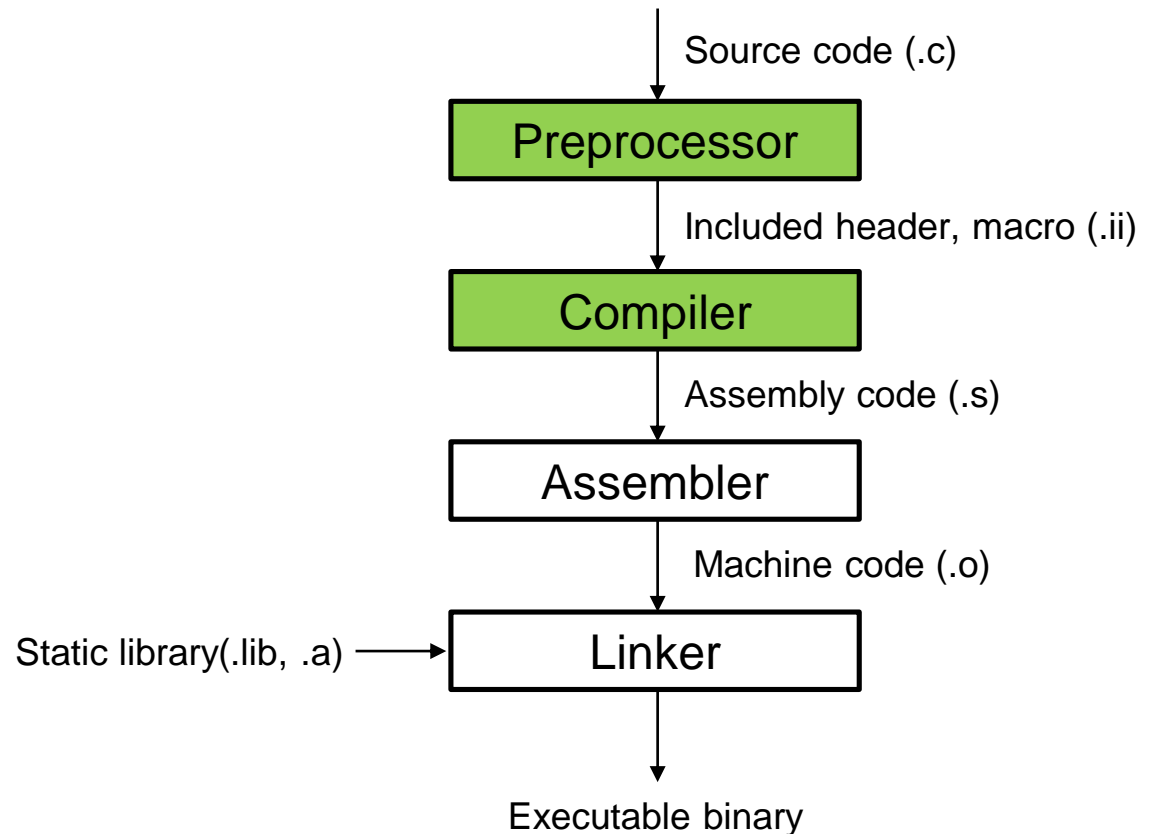
Prof. Jae W. Lee ([jaewlee@snu.ac.kr](mailto:jaewlee@snu.ac.kr))

Department of Computer Science and Engineering  
Seoul National University

TA: Jeonghun Gong, Yunho Jin

# Goal of this project

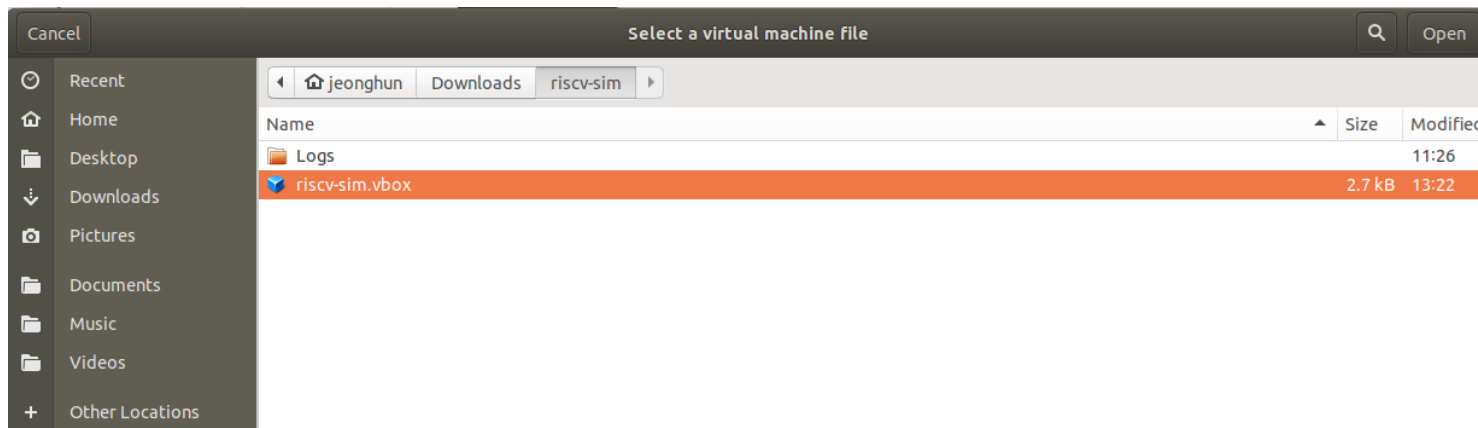
- You will compile C source code into **64-bit RISC-V** assembly.



# Experimental setup

## ■ You will use RISC-V ISA simulator on Linux: Two options

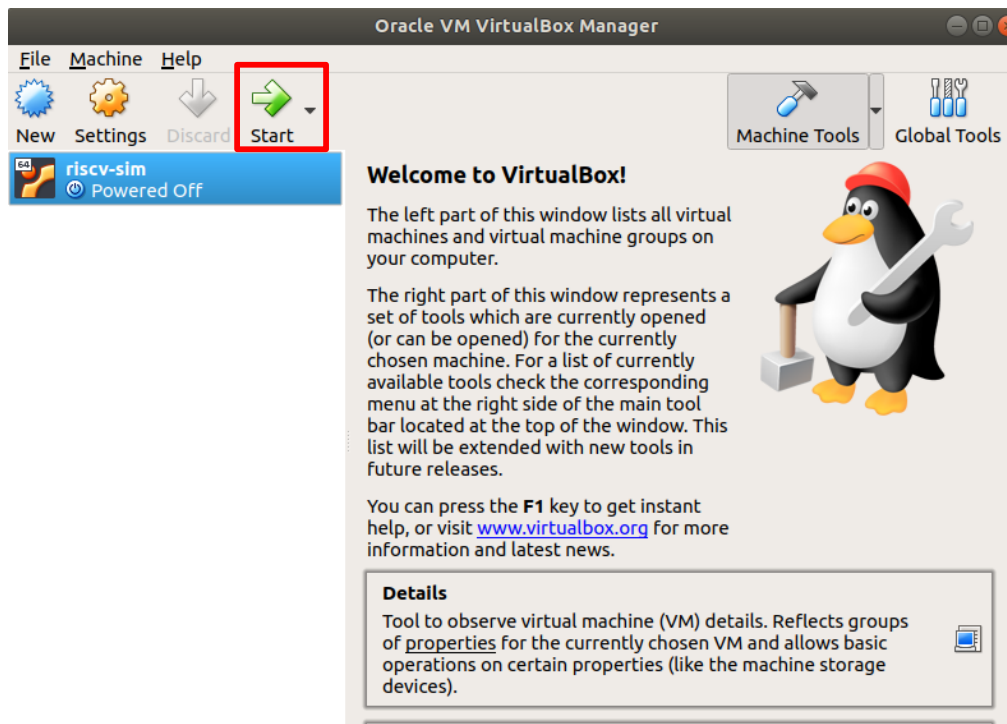
- Option 1: Use Virtual Machine on your Windows/Mac Os machine
  - Download virtualbox executable from website and install it.  
<https://www.virtualbox.org/wiki/Downloads>
  - Download virtualbox image from eTL and extract it.
  - Press (Ctrl + A) in main menu and select virtualbox image.
- Option 2: Use your own Linux box
  - CAVEAT: Grading will be done on our VM



# Experimental setup

## ■ Option 1: Use Virtual Machine on your Windows/Mac

- Boot your image (Linux password: 1234).
- Everything is set including template codes.
- Template code is in ~/PA1/



# Experimental setup

## ■ Option 2: Use your own Linux box: Environment setup

- Ubuntu/Debian distributions are assumed.
- Before get started, add these two lines on your ~/.bashrc
  - You can use a different RISCv installation path if you want.

```
export RISCv='/opt/riscv/'  
PATH="$PATH:$RISCv/bin"
```

- Then, type “source ~/.bashrc” on your command line.
- Make your directory.


```
$> sudo mkdir $RISCv
```

```
$> sudo chown -R [your_username] $RISCv
```

# Experimental setup

## ■ Option 2: Use your own Linux box: Environment setup

- Download build.sh from eTL.
- Before get started, check the number of CPU cores of your PC with `lscpu` command.
- Run build.sh with argument `NUM_THREADS=[core count]`
  - Default number of threads is 8 if not specified.
- This script will automatically download and setup your environment.
  - It will ask for your password during installation.



```
jeonghun@NEETProduction:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                32
On-line CPU(s) list:   0-31
Thread(s) per core:    2
Core(s) per socket:    16
Socket(s):              1
NUMA node(s):          1

jeonghun@NEETProduction:/mnt/randisk$ ls
build.sh  tmp
jeonghun@NEETProduction:/mnt/randisk$ NUM_THREADS=32 ./build.sh
```

A terminal window showing the execution of the `lscpu` command, which displays system information including 32 CPU cores. A red box highlights the value '32' next to 'CPU(s):'. Another red box highlights the value '32' in the command `NUM_THREADS=32 ./build.sh`, with a red arrow pointing from the first box to the second.

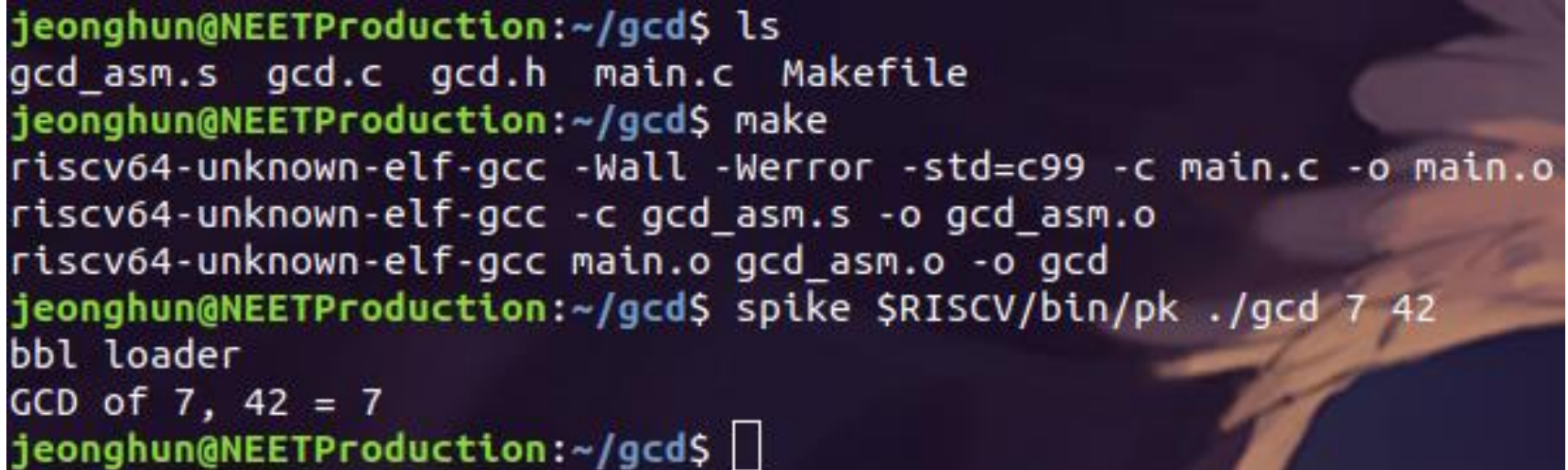
# Experimental setup

## ■ How to execute your code (for both Option 1 and 2):

In terminal

```
$> Make
```

```
$> spike $RISCV/bin/pk ./binary [arg1] [arg2] ...
```

A terminal window with a dark background and light-colored text. The prompt is 'jeonghun@NEETProduction:~/gcd\$'. The user enters 'ls', showing files 'gcd\_asm.s', 'gcd.c', 'gcd.h', 'main.c', and 'Makefile'. Then they enter 'make', which runs two compilation commands using 'riscv64-unknown-elf-gcc'. Finally, they enter 'spike \$RISCV/bin/pk ./gcd 7 42', which outputs 'bbl loader' and 'GCD of 7, 42 = 7'. The prompt returns to 'jeonghun@NEETProduction:~/gcd\$' with a cursor.

```
jeonghun@NEETProduction:~/gcd$ ls
gcd_asm.s gcd.c gcd.h main.c Makefile
jeonghun@NEETProduction:~/gcd$ make
riscv64-unknown-elf-gcc -Wall -Werror -std=c99 -c main.c -o main.o
riscv64-unknown-elf-gcc -c gcd_asm.s -o gcd_asm.o
riscv64-unknown-elf-gcc main.o gcd_asm.o -o gcd
jeonghun@NEETProduction:~/gcd$ spike $RISCV/bin/pk ./gcd 7 42
bbl loader
GCD of 7, 42 = 7
jeonghun@NEETProduction:~/gcd$
```

# Problem 1: Greatest common divisor

## ■ Calculate the greatest common divisor (GCD) of two integers.

- Write your code on `gcd_asm.s`
- Refer to `gcd.c` (reference code) for algorithm.
- Operands are stored at register `a0`, `a1`.
- Store the answer to register `a0` and return.
- Execution:

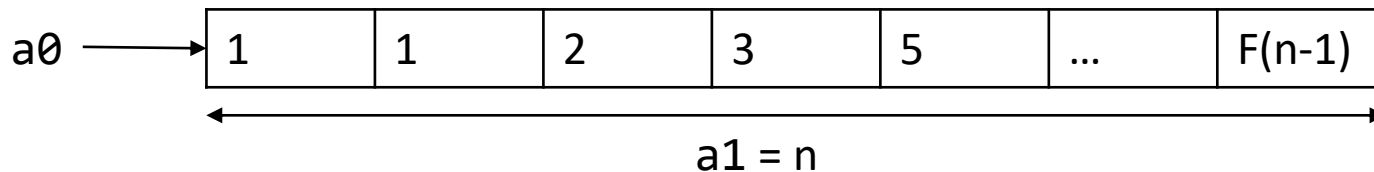
```
$> spike $RISCV/bin/pk ./gcd [lhs] [rhs]
```



# Problem 2: Fibonacci sequence

- **Store the given count of Fibonacci sequence on specified memory location.**
  - Write your code on `fibonacci_asm.s`
  - Starting address of the Fibonacci sequence is stored in register `a0`.
  - The length of the sequence (`n`) is stored in register `a1`.
  - Return value (`a0`) is the memory address having answer.
  - Execution:  

```
$> spike $RISCV/bin/pk ./fibonacci [count]
```

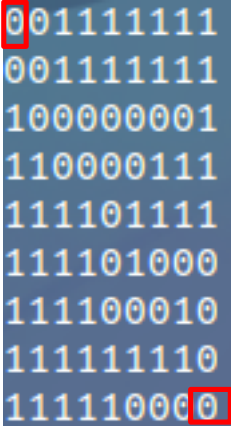


# Problem 3: Maze Solving

## ■ Find the length of the shortest path to solve a given maze.

- Maze is stored in array (reg a0).
- Width (reg a1) and height (reg a2) of array are given.
- Each entry of array represents the state of pixel.
  - (1: Blocked, 0: Open)
- Starting point is (0,0) of array.
- Ending point is (width – 1, height – 1) of array.
- Refer to maze.c for algorithm.
- If this maze can't be solved in 20 steps, return -1; else, return the length of the shortest path.
- Execution:

```
$> spike $RISCV/bin/pk ./maze [filename]
```

Start  End

0	0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	1	1
1	1	0	0	0	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	0
1	1	1	1	0	0	0	1	0	0
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0

# Submission

## ■ Write-up

- Briefly describe your implementation ( $\leq 5$  pages)
- Filename: [student\_id].pdf (example: 2019-12345.pdf)
- **Please** submit it in **PDF** format. Other formats are not accepted.

## ■ Compress your source code and write-up into a single zip file

- Compress gcd\_asm.s, fibonacci\_asm.s, maze\_asm.s and your write-up
- Filename should be [student\_id].zip (example: 2019-12345.zip).
- **Please** submit it in **ZIP** format. Other formats are not accepted.

## ■ Submission deadline: 2019. 9. 30 (Mon) 23:59