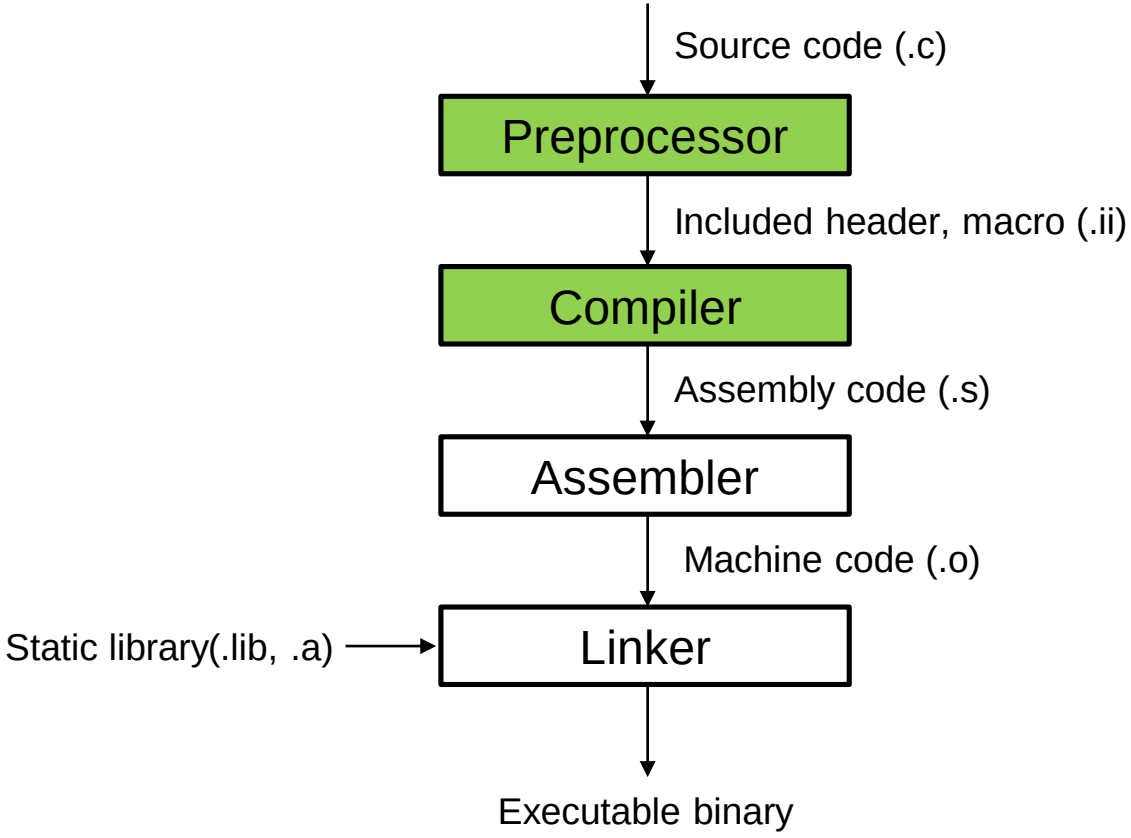
Programming Assignment #1: Human Compiler

Prof. Jae W. Lee (jaewlee@snu.ac.kr)
Department of Computer Science and Engineering
Seoul National University

TA: Jeonghun Gong, Yunho Jin

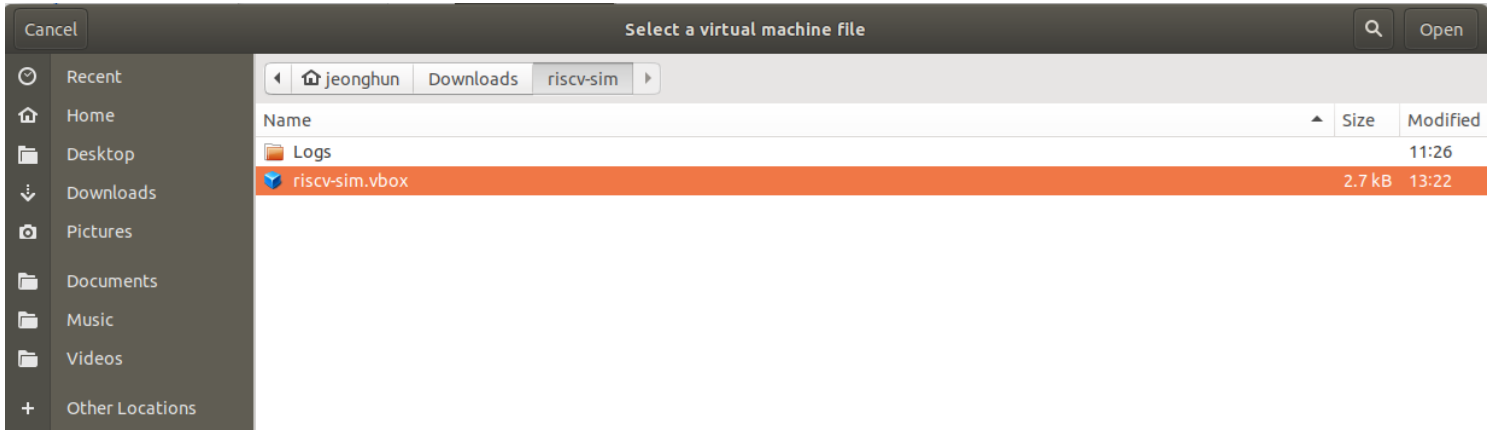
Goal of this project

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



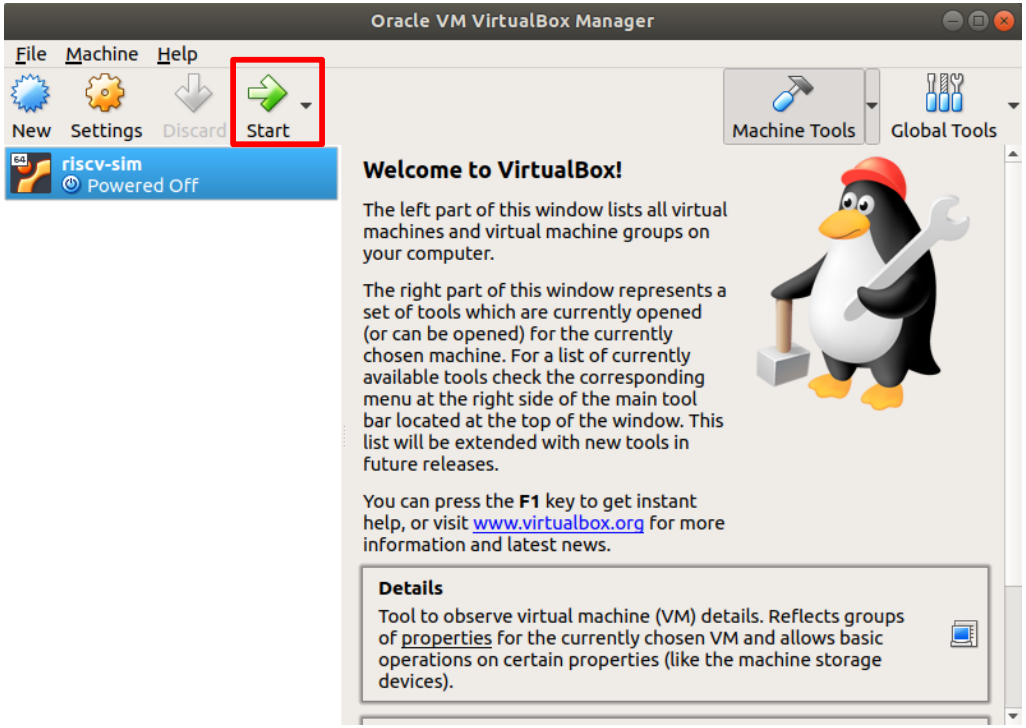
Experimental setup

- **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.

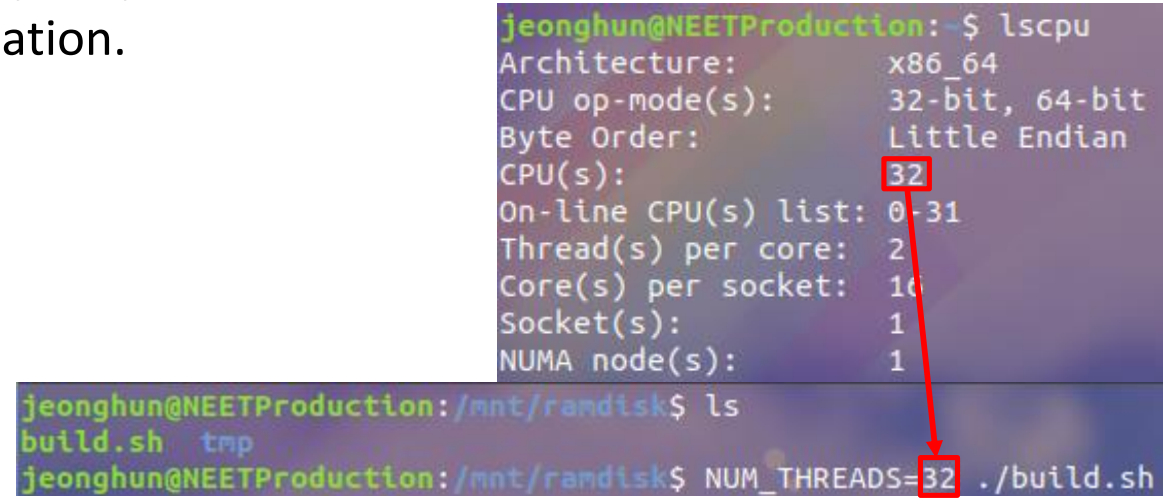
```
export RISCV='/opt/riscv/'
PATH="$PATH:$RISCV/bin"
```

  - 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.

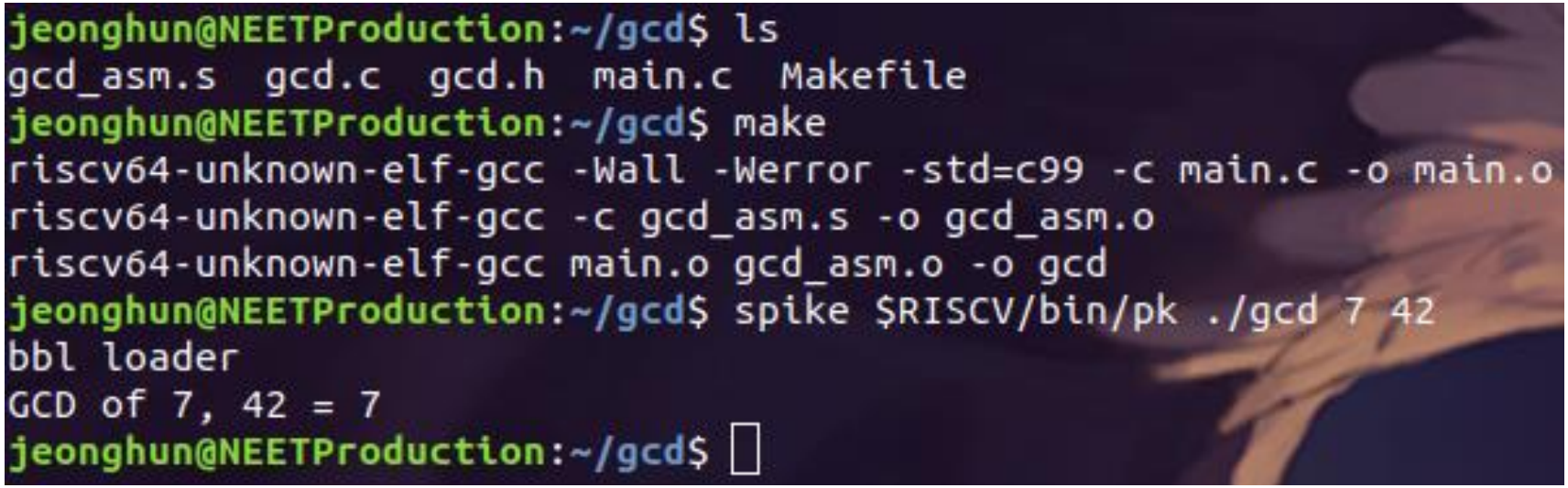


Experimental setup

- **How to execute your code (for both Option 1 and 2):**

In terminal

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



Problem 1: Greatest common divisor

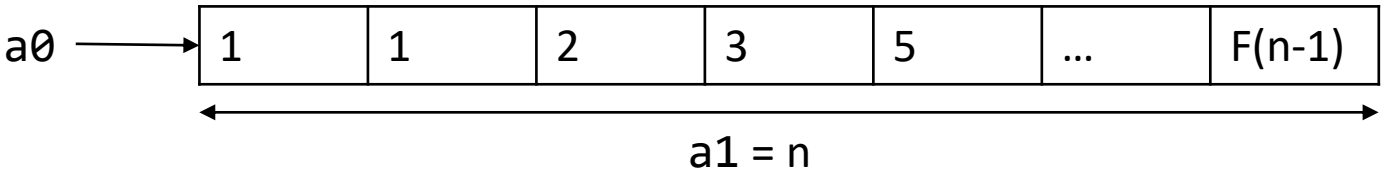
- **Calculate the greatest common divisor (GCD) of two integers.**
  - Write your code on `gcd_asm.s`
  - Refer to `gcd.c` (reference code) for algorithm.
  - Operands are stored at register a0, a1.
  - Store the answer to register a0 and return.
  - Execution:

```
$> 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:

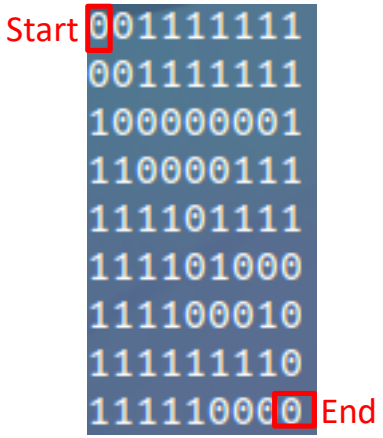
```
$> spike $RISCV/bin/pk ./fibonacci [count]
```



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]
```



Submission

- **Write-up**
  - Briefly describe your implementation (≤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**