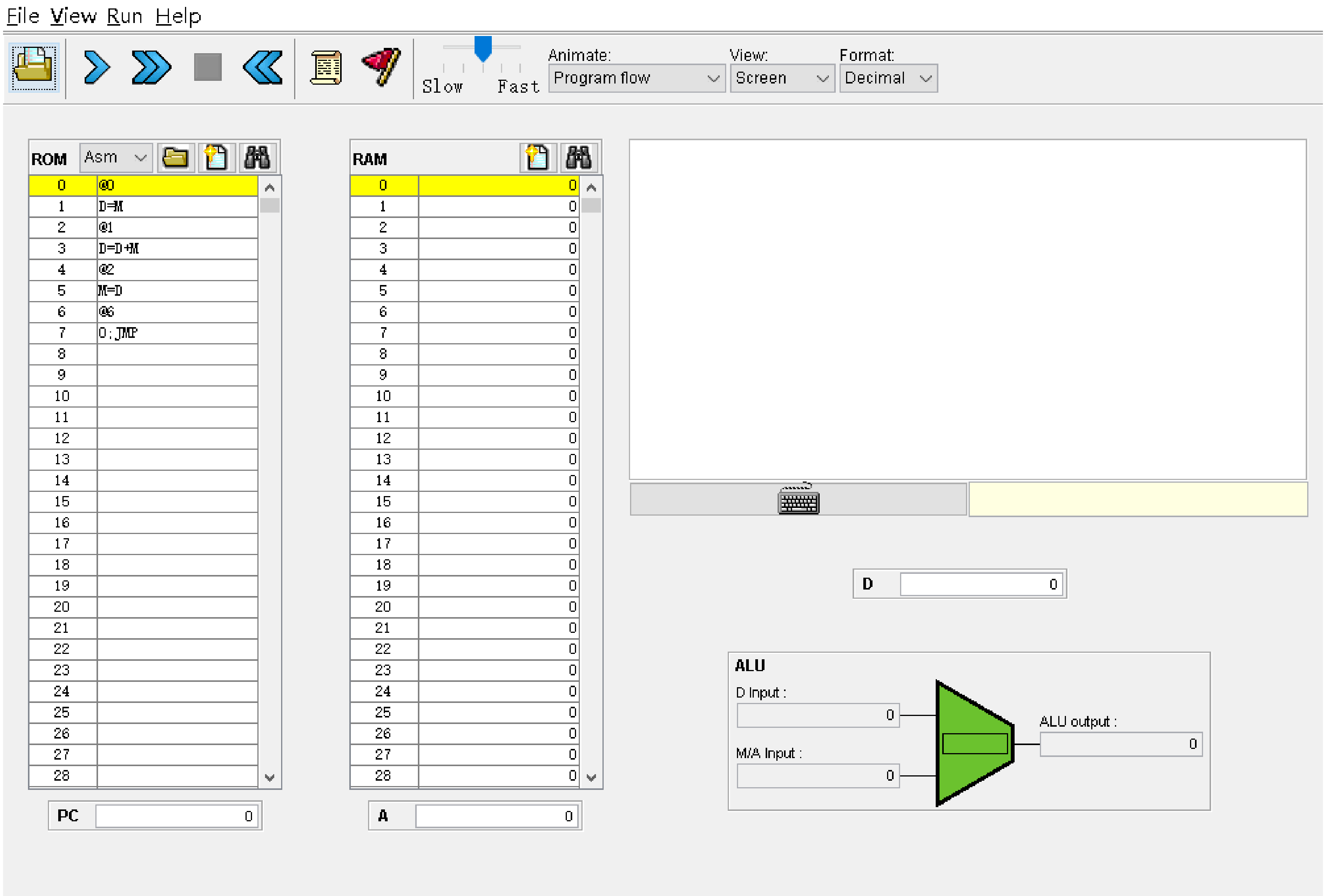
Machine Language

Dr. Wooi Ping Cheah

Lab Project

- Project objectives: to have a taste of
  - Low-level programming
  - Hack assembly language
  - Hack hardware

CPU Emulator



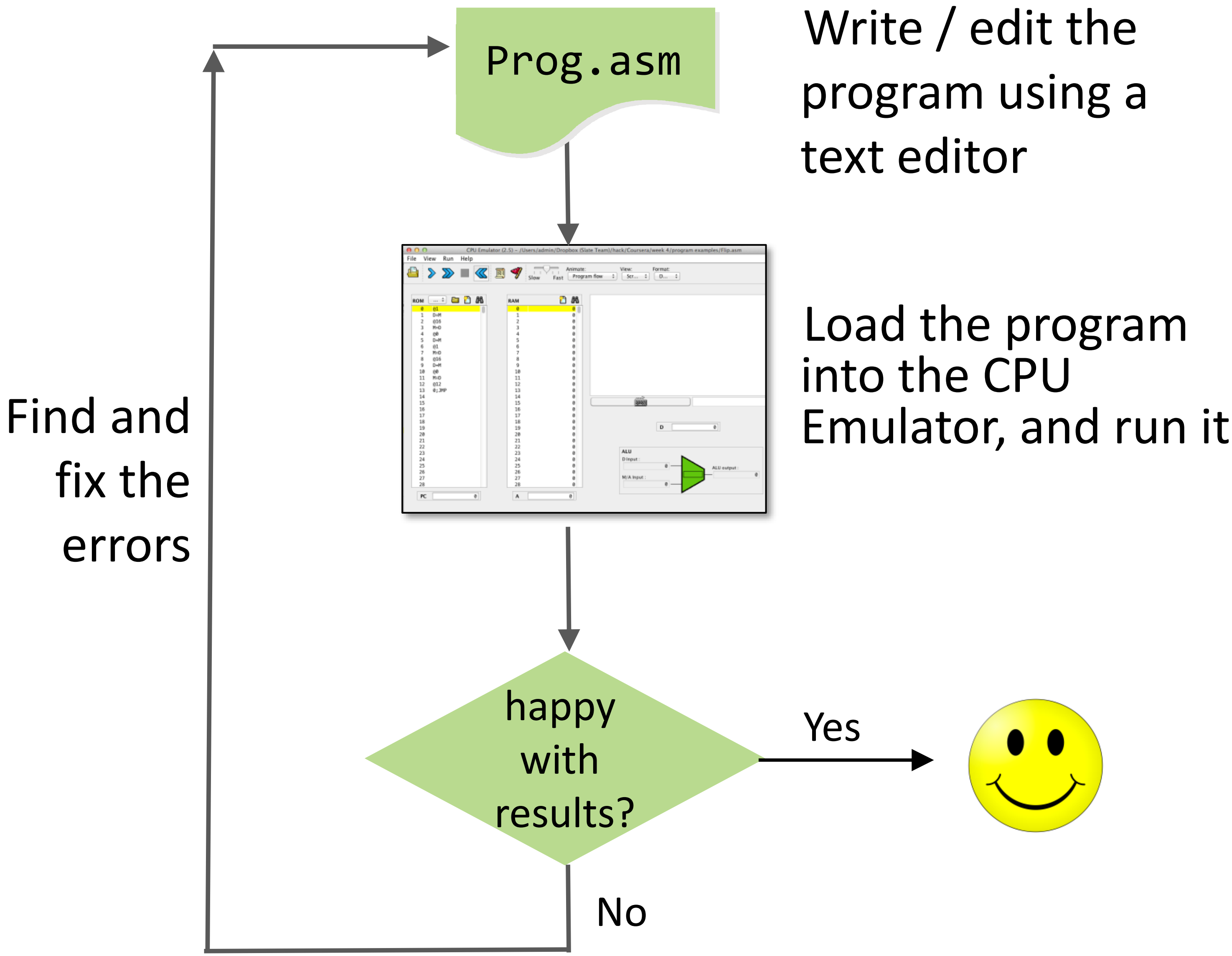
CPU Emulator

- **Slow ... Fast** (change the speed of emulation).
- **Animation**
  - ➢**Program flow**: show how the program proceeds.
  - ➢**Program & data flow**: show how the program proceeds and the data vary.
  - ➢**No animation**: no animation will be shown.
- **View**
  - ➢Script: show script.
  - ➢Output: show running results.
  - ➢Compare: not in used for this lab.
  - ➢Screen: show Hack Computer screen. (256×512, B/W)
- **Format**: show numbers in decimal, hexamal or binary.

Misc

- File suffix
  - Binary code (.hack) files.
  - Assembly language (.asm) files.
- Test procedures
  - Open CPU Emulator.
  - Load xxx.asm into ROM, symbolic code without symbols (e.g. predefined symbols, labels, variables).
  - Load xxx.tst test script, run the test script.

Program development process



Best practice

Well-written low-level code is

- Short
- Efficient
- Elegant
- Self-describing

Technical tips

- Use symbolic variables and labels
- Use sensible variable and label names
- Variables: lower-case
- Labels: upper-case
- Use indentation
- Start with pseudo code.

Terminate a program

Hack assembly code

```
   // Program: Add2.asm
   // Computes: RAM[2] = RAM[0] + RAM[1]
   // Usage: put values in RAM[0], RAM[1]
0  @0
1  D=M   // D = RAM[0]

2  @1
3  D=D+M // D = D + RAM[1]

4  @2
5  M=D   // RAM[2] = D

6  @6
7  0;JMP
```

- Jump to instruction number A (which happens to be 6)
- 0: syntax convention for jmp instructions

translate and load

Memory (ROM)

0	@0
1	D=M
2	@1
3	D=D+M
4	@2
5	M=D
6	@6
7	0;JMP
8	
9	
10	
11	
12	
13	
14	
15	
⋮	
32767	

<u>Best practice:</u>

To terminate a program safely, end it with an infinite loop.

Task 1: add two numbers

- Input: RAM[0] and RAM[1].
- Output: RAM[2] = RAM[0]+RAM[1].
- add2.asm
- A sample code is shown on the previous slide.

Task 2: swap two numbers

- Set RAM[0] = 50, RAM[1] = 100, then swap the value of RAM[0] and RAM[1]. You may use RAM[16] as the temporary variable.
- You may start by modifying the codes on slide 52.

Task 3: signum

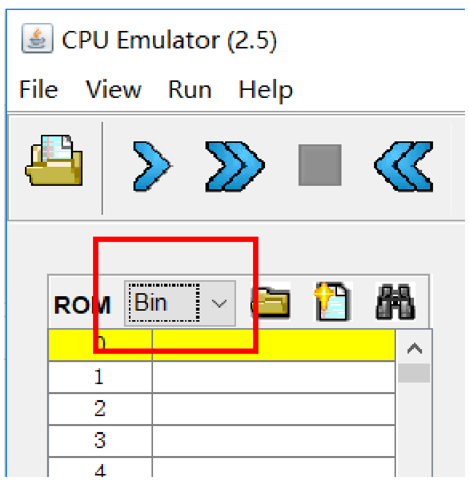
- Implement signum.asm to achieve the following function.

```
// Program: signum.asm
// Computes:
//   if RAM[0]>0
//          RAM[1]=1
//   else
//          RAM[1]=0
// Usage: put a value in RAM[0],
//        run and inspect RAM[1].
```

- Though branching will be covered in the next lecture you may like to do it first.

Task 4: assembly to binary

- For task 1-3, choose one of them, translate the assembly code to binary code, according to the syntax provided on slide 47. Compare your translation with the translation by CPU Emulator.



Project resources

From NAND to Tetris
Building a Modern Computer From First Principles

- Home
- Prerequisites
- Syllabus
- Course
- Book
- Software
- Terms
- Papers
- Talks
- Cool Stuff
- About
- Team
- Q&A

Project 4: Machine Language Programming

Background

Each hardware platform is designed to execute a certain machine language, expressed using agreed-upon binary codes. Writing programs directly in binary code is a possible, yet an unnecessary, tedium. Instead, we can write such programs in a low-level symbolic language, called *assembly*, and have them translated into binary code by a program called *assembler*. In this project you will write some low-level assembly programs, and will be forever thankful for high-level languages like C and Java. (Actually, assembly programming can be a lot of fun, if you are in the right mood; it's an excellent brain teaser, and it allows you to control the underlying machine directly and completely.

Objective

To get a taste of low-level programming in machine language, and ... m. In the process of working on this project, you will become familiar w... ic language to machine-language - and you will appreciate visually h... vare platform. These lessons will be learned in the context of writing a... below.

All the necessary project files are available in:
nand2tetris / projects / 04

Programs

Program	Description	Comments / Tests
Mult.asm	**Multiplication:** In the Hack framework, the top 16 RAM words (`RAM[0] ... RAM[15]`) are also referred to as the so-called *virtual registers* `R0 ... R15`. With this terminology in mind, this program computes the value `R0*R1` and stores the result in `R2`.	For the purpose of this program, we assume that `R0>=0`, `R1>=0`, and `R0*R1<32768` (you are welcome to ponder where this value comes from). Your program need not test these conditions, but rather assume that they hold. To test your program, put some values in RAM[0] and RAM[1], run the code, and inspect RAM[2]. The supplied `Mult.tst` script and `Mult.cmp` compare file are deigned to test your program "officially", running it on several representative values supplied by us.

Acknowlegement

- This set of lecture notes are based on the lecture notes provided by Noam Nisam / Shimon Schocken.
- You may find more information on: www.nand2tetris.org.