University of Nottingham
UK | CHINA | MALAYSIA

Machine Language (Part 2)

Dr. Wooi Ping Cheah

Lab Project

- Project objectives:
  - Practice Hack assembly language programming.
  - Utilize build-in symbols, variables, labels and pointers.

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.

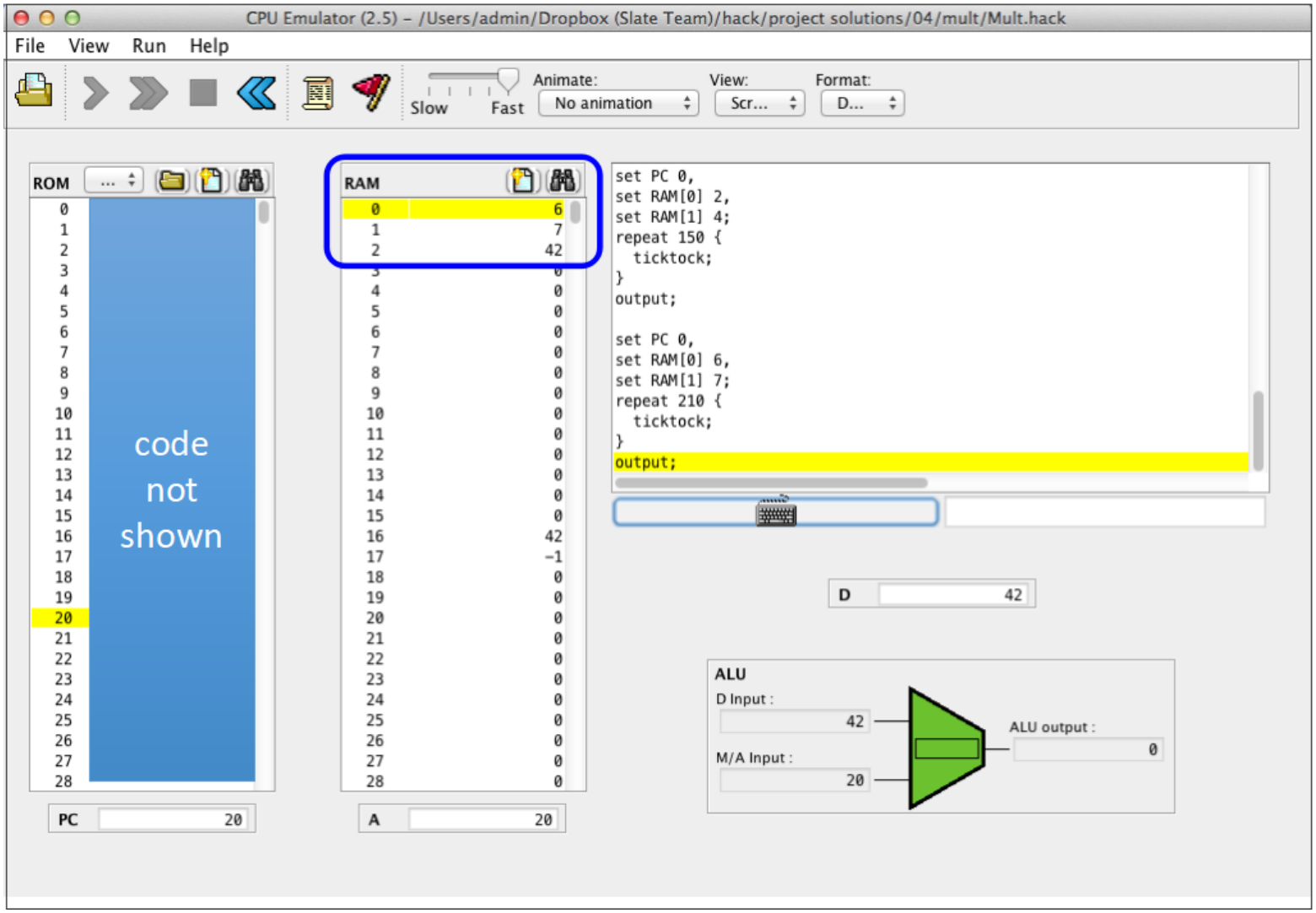
Task 1: sum a sequence of numbers

```
    // Computes RAM[1] = 1+...+RAM[0]
    // Usage: put a number in RAM[0]
0   @16   // RAM[16] represents i
1   M=1   // i = 1
2   @17   // RAM[17] represents sum
3   M=0   // sum = 0

4   @16
5   D=M
6   @0
7   D=D-M
8   @18    // if i>RAM[0] goto 18
9   D;JGT

10  @16
11  D=M
12  @17
13  M=D+M  // sum += i
14  @16
15  M=M+1  // i++
16  @4     // goto 4 (loop)
17  0;JMP

18  @17
19  D=M
20  @1
21  M=D    // RAM[1] = sum
22  @22    // program's end
23  0;JMP  // infinite loop
```

- Modify the code to become more readable.
- Utilize variables such as i, sum.
- Utilize R0 and R1.
- Utilize label LOOP, STOP and END.

Task 2: multiply two numbers

- `Mult`: a program performing `R2 = R0 * R1`



Task 2: hint

- Hack instructions do not contain multiplication, only addition and subtraction. You may implement the multiplication using repetitive addition, e.g. 6*4 = 6+6+6+6 = 24.
- Goal: Implement R2 = R0 × R1
- Pseudo code: Implement multiplication as adding R0 to itself R1 times.

```
      times = R1
      R2 = 0
LOOP:
      if times == 0 goto END
      R2 = R2 + R0
      times = times - 1
      goto LOOP
END:
```

Task 3: generate fibonacci series

- Implement a function to generate fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- Input a number in RAM[0], the function returns the next number in the Fibonacci series in RAM[1].
- Example:

  Input: RAM[0] ==21

  Output: RAM[1] ==34

- Example of pseudo-code:

```
   PreNum = 0
   CurNum = 1
(LOOP)
   if CurNum > R0 goto STOP
   NexNum = CurNum+PreNum
   PreNum = CurNum
   CurNum = NexNum
   goto LOOP
(STOP)
   R1 = NexNum
```

Task 4: integer division

- To implement a division function of integers z=x/y, where x, y, z are non-negative integers, and z is the **round-down** value of x/y (i.e. the largest possible integer z that is not greater than x/y.). You should store x, y, z in RAM[0], RAM[1] and RAM[2], respectively.
- Hint: If x = 10, y = 3, you should get z = 3; if x = 10, y = 2, you should get z = 5; if x = 10, y = 10, you should get z = 1; if x = 10, y = 20, you should get z = 0. You may use the idea of repetitive addition, e.g. y+y+y+..+y≤x, and determine the number of y in this inequality, which is z. You can ignore the case of y=0 for this task.

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
the process of working on this project, you will become familiar w
language to machine-language - and you will appreciate visually h
platform. These lessons will be learned in the context of writing a
below.

m. In
ic
vare

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.