Assembler and Input-Output

Dr. Wooi Ping Cheah

Lab Work

- Objectives:
  - Write a Hack assembly program for handling keyboard input and screen output.
  - Practice how to trace the execution of a Hack assembly language program.
  - Practice how to manually translate the Hack assembly program to binary machine language.

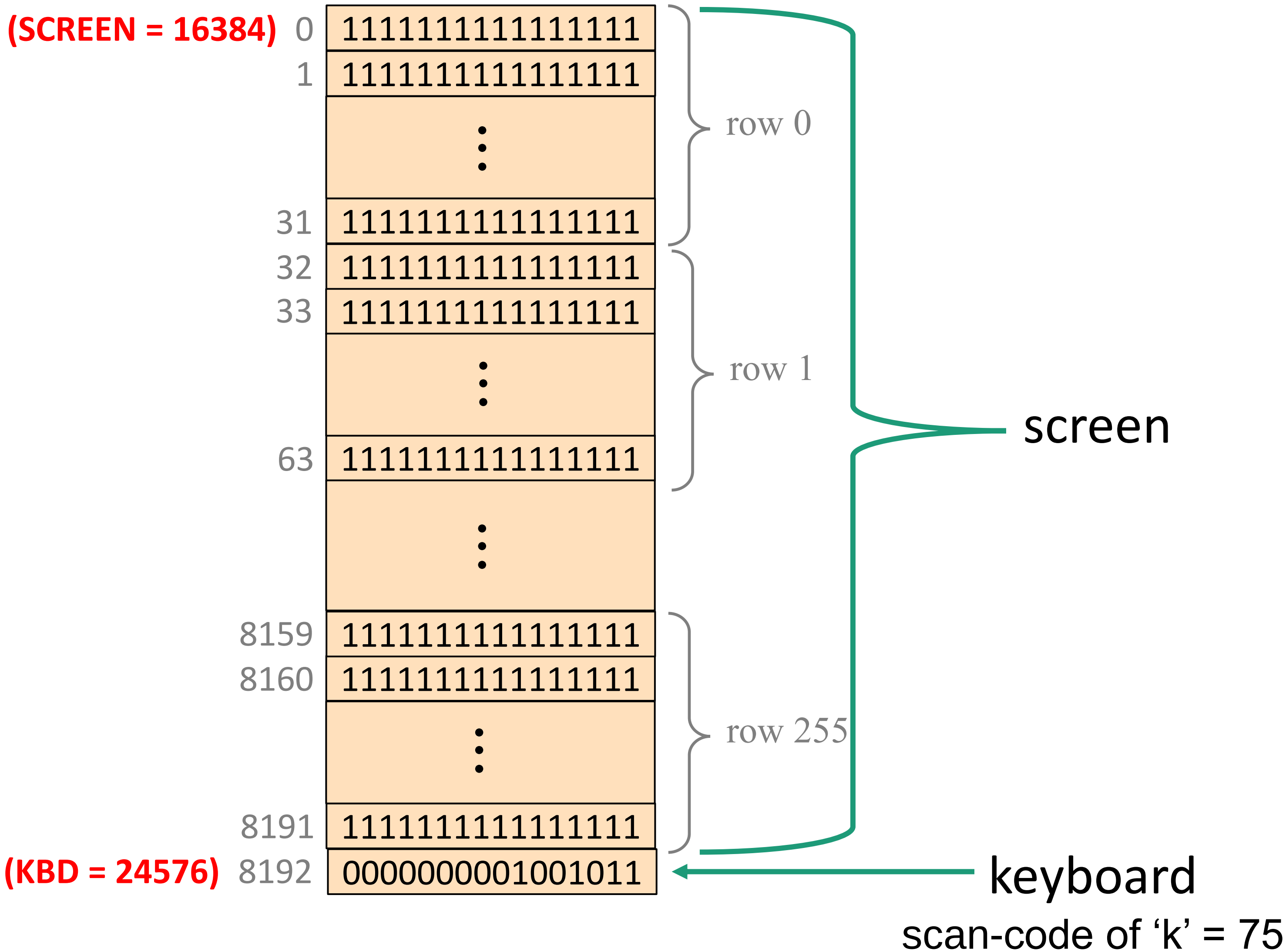
Task 1: fill a rectangle interactively

- `Fill`: a simple interactive program
  - Runs an infinite loop that listens to the keyboard input.
  - When a key is pressed (any key), the program blackens the screen,i.e. writes "black" in every pixel;
  - the screen should remain fully black as long as the key is pressed.
  - When no key is pressed, the program clears the screen, i.e. write "white" in every pixel;
  - the screen should remain fully clear as long as no key is pressed.

Task 1: Fill a Rectangle Interactively

- Implementation strategy
  - ➢Listen to the keyboard
  - ➢To blacken / clear the screen, write code that fills the entire screen memory map with either "white" or "black" pixels
  - ➢Accessing the memory requires working with pointers
  - ➢Hack computer is slow!!!
- Testing
  - ➢Select "no animation"
  - ➢Manual testing (no test scripts).

Task 1: Hints


(SCREEN = 16384)
0
1
31
32
33
63
8159
8160
8191
(KBD = 24576)
8192
1111111111111111
0000000001001011
row 0
row 1
row 255
screen
keyboard
scan-code of ‘k’ = 75

Task 2 – Trace Program Execution

```
@4
D=A
@R0
M=D
@R0
D=M
@n
M=D
@i
M=0
@sum
M=0
```

```
(LOOP)
 @i
 D=M
 @n
 D=D-M
 @STOP
 D;JGT
 @sum
 D=M
 @i
 D=D+M
 @sum
 M=D
 @i
 M=M+1
 @LOOP
 0;JMP
```

```
(STOP)
 @sum
 D=M
 @R1
 M=D
(END)
 @END
 0;JMP
```

Question:

Derive the value of RAM[1] after the execution of this piece of code.

Task 3 - Manually Translate Assembly Code

- Translate the symbolic assembly code to binary code.
  - Add.asm: Add two numbers, // `RAM[0] = 2 + 3`
  - Max.asm: `//RAM[2] = max(RAM[0],RAM[1])`
  - Rectangle.asm //Draw a rectangle

- You may find the binary code by choosing “Bin” option in “CPU Emulator”, but the objective of this lab is to **manually** convert the assembly code to binary code.

Task 3 - Test program: **Add**

Add.asm

```
// Computes RAM[0] = 2 + 3

@2
D=A
@3
D=D+A
@0
M=D
```

Basic things to handle:

- White space
- Instructions

Task 3 - Test program: **Max**

Max.asm

```
// Computes RAM[2] = max(RAM[0],RAM[1])

   @R0
   D=M            // D = RAM[0]
   @R1
   D=D-M          // D = RAM[0] - RAM[1]
   @OUTPUT_RAM0
   D;JGT          // if D>0 goto output RAM[0]

   // Output RAM[1]
   @R1
   D=M
   @R2
   M=D            // RAM[2] = RAM[1]
   @END
   0;JMP

(OUTPUT_RAM0)
   @R0
   D=M
   @R2
   M=D            // RAM[2] = RAM[0]

(END)
   @END
   0;JMP
```

with labels

MaxL.asm

```
// Symbol-less version

   @0
   D=M            // D = RAM[0]
   @1
   D=D-M          // D = RAM[0] - RAM[1]
   @12
   D;JGT          // if D>0 goto output RAM[0]

   // Output RAM[1]
   @1
   D=M
   @2
   M=D            // RAM[2] = RAM[1]
   @16
   0;JMP

   @0
   D=M
   @2
   M=D            // RAM[2] = RAM[0]

   @16
   0;JMP
```

without labels

Task 3 - Test program: **Rectangle**

Rectangle.asm

```
// Rectangle.asm

   @R0
   D=M
   @n
   M=D  // n = RAM[0]

   @i
   M=0  // i = 0

   @SCREEN
   D=A
   @address
   M=D  // base address of Hack
    screen

(LOOP)
   @i
   D=M
   @n
   D=D-M
   @END
   D;JGT  // if i>n goto END
```

with symbols

RectangleL.asm

```
// Symbol-less version

   @0
   D=M
   @16
   M=D   // n = RAM[0]

   @17
   M=0   // i = 0

   @16384
   D=A
   @18
   M=D   // base address of Hack screen

   @17
   D=M
   @16
   D=D-M
   @27
   D;JGT  // if i>n goto END
```

without symbols

Task 3 - Test program: **Rectangle**

Rectangle.asm

```
   @addr
   A=M
   M=-1  //RAM[addr]=1111111111111111

   @i
   M=M+1  // i = i + 1

   @32
   D=A   // D = 32
   @addr
   M=D+M  // addr = addr + 32
   @LOOP
   0;JMP  // goto LOOP

(END)
   @END   // progra
   0;JMP  // infinite loop
```

with symbols

RectangleL.asm

```
   @16
   A=M
   M=-1  //RAM[addr]=1111111111111111

   @18
   M=M+1  // i = i + 1

   @32
   D=A   // D = 32
   @16
   M=D+M  // addr = addr + 32
   @10
   0;JMP  // goto LOOP


   @27   // progran
   0;JMP  // infinite loop
```

without symbols

Project Resources

From NAND to Tetris

Building a Modern Computer From First Principles

www.nand2tetris.org

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

Project 6: The Assembler

Background

Low-level machine programs are rarely written by humans. Typically, they are generated by compilers. Yet humans can inspect the translated code and learn important lessons about how to write their high-level programs better, in a way that avoids low-level pitfalls and exploits the underlying hardware better. One of the key players in this translation process is the *assembler* -- a program designed to translate code written in a symbolic machine language into code written in binary machine language.

This project marks an exciting landmark in our *Nand to Tetris* odyssey: it deals with building the first rung up the software hierarchy, which will eventually end up in the construction of a compiler for a Java-like high-level language. But, first things first.

Objective

Write an Assembler program that translates programs written in the symbolic Hack assembly language into binary code that can execute on the Hack hardware platform built in the previous projects.

Contract

There are three ways to describe the desired behavior of your assembler: (i) Whe
`Prog.asm` file containing a valid Hack assembly language program should be tran

All the necessary project files are available in:

nand2tetris / projects / 06

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.