CSE 390 B Spring 2021

Hack Assembly & Reflection

Social Computing Discussion, Hack Assembly Details, & Project 4 Tools

Agenda

- Social Reflection Prompt I Discussion
- Reading Review and Q&A
- Implementing Memory Devices
- Exercise: Writing Hack Programs
- Project 4 Overview

Agenda

- **Social Reflection Prompt I Discussion**
- Reading Review and Q&A
- Implementing Memory Devices
- Exercise: Writing Hack Programs
- Project 4 Overview

Social Reflection Prompt I Discussion

- Share the article you found for your first social reflection!

- Things to include:
  - Give a summary of the article
  - Share how it changed/influenced your thinking
  - Share your follow up questions
  - Feel free to have discussions beyond these bullet points too!

Agenda

- Social Reflection Prompt I Discussion
- **Reading Review and Q&A**
- Implementing Memory Devices
- Exercise: Writing Hack Programs
- Project 4 Overview

Hexadecimal

- Base 16 number system
  - Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Commonly used for referring to memory addresses
  - Really easy to convert between binary and hexadecimal
  - Hexadecimal uses less digits to represent a value than binary
- Use the prefix 0x to indicate a number is written in hexadecimal
  - 32 is decimal, 0x32 is hexadecimal

Hexadecimal Digit Equivalencies

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Converting Between Binary and Hexadecimal

- There is a 1 to 1 mapping between 4 binary digits and a single hexadecimal digit
  - 2^4 = 16!
- Converting between binary and hexadecimal is as easy as swapping out 4 binary digits for the corresponding hexadecimal digit (or vice versa)
- Example 0x3A is 0011 1010
  - 0x3 == 0011
  - 0xA == 1010

Hack Assembly: Input/Output

- Two memory maps are created for you by underlying hardware (all you have to do is use them)
  - Screen is a *huge* map where each pixel is one bit
  - Keyboard is a single 16-bit word map w/ code of current key

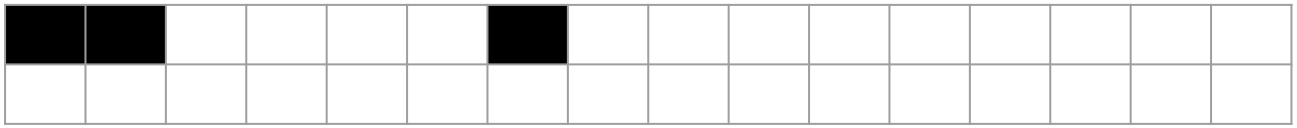
Example:

```
@KBD
D=M

@SCREEN
M=D
```

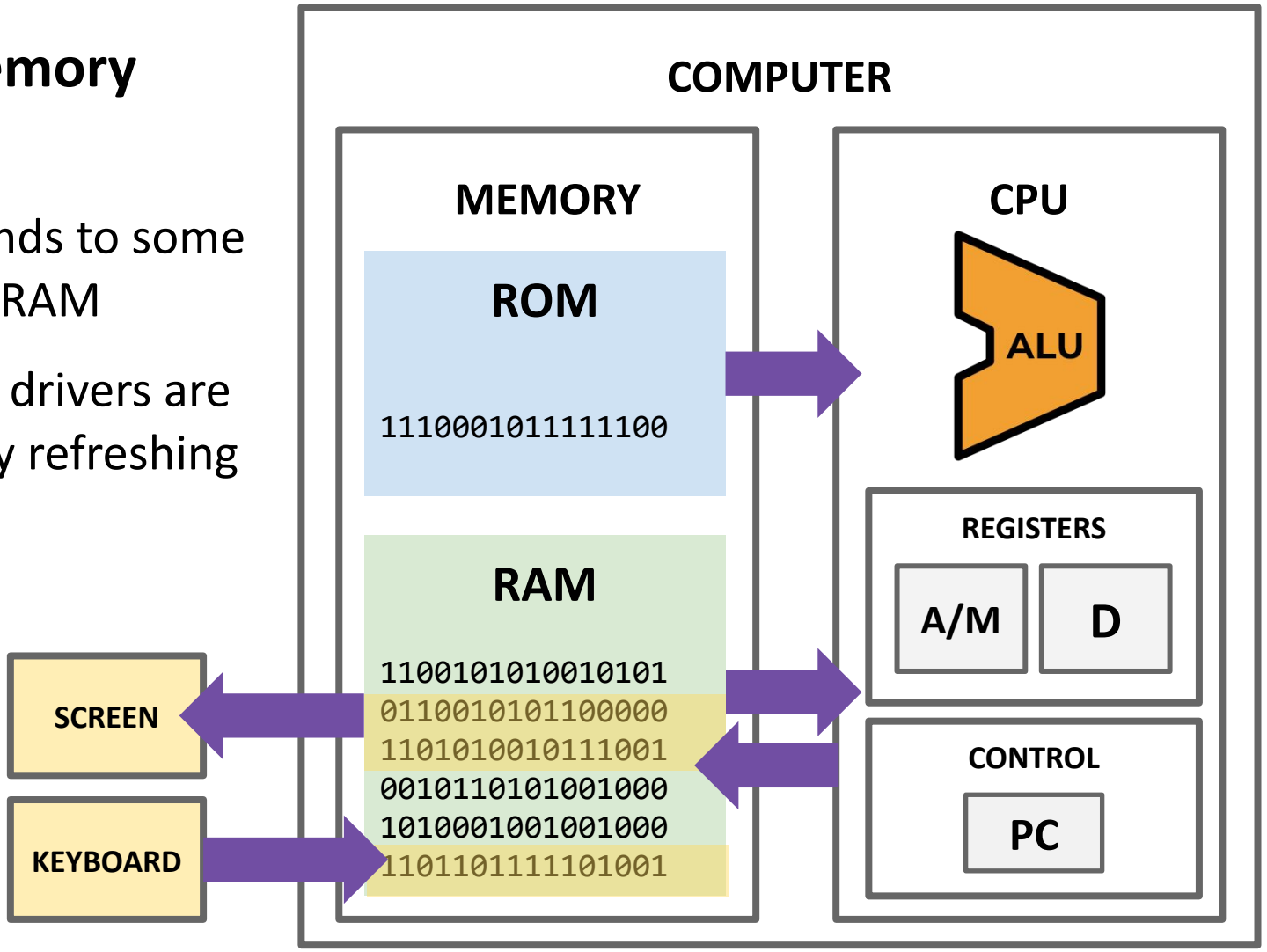
D contains code of current key (e.g. 67 for "C")

First 16 bits of screen (top left) show binary for 67



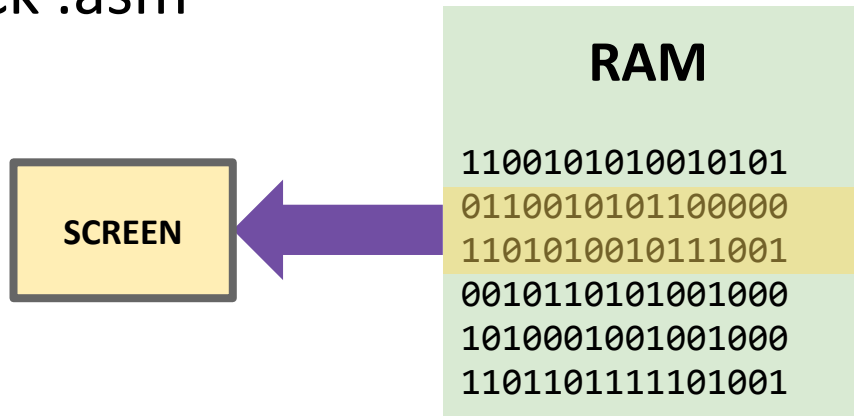
Hack: Input/Output

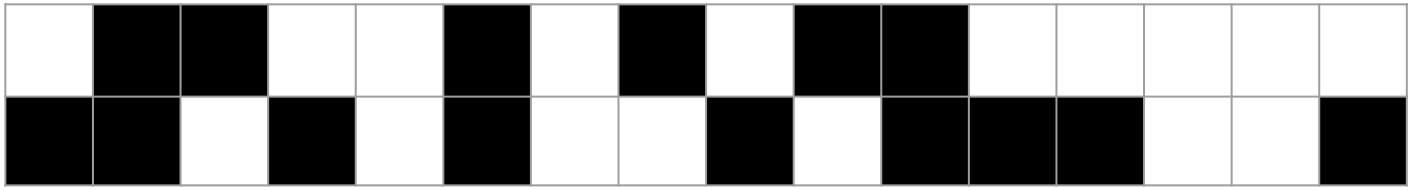
- I/O is **memory mapped**
  - Corresponds to some region of RAM
  - Low-level drivers are constantly refreshing



Hack: Memory Mapped Output

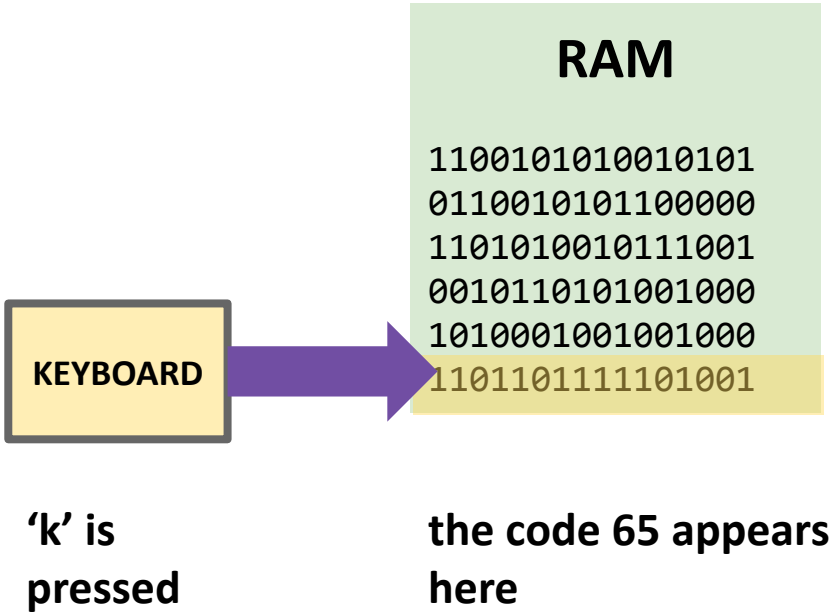
- Each bit of the screen memory map corresponds to one pixel (1: black, 0: white)
- The start of the memory map is accessible via the SCREEN constant in Hack .asm





Hack: Memory Mapped Input

- A single 16-bit word in memory is constantly refreshed with the scan code of the keyboard button being pressed
- This spot in memory accessible via the KBD constant in Hack .asm



Reading Q&A

Agenda

- Social Reflection Prompt I Discussion
- Reading Review and Q&A
- **Implementing Memory Devices**
- Exercise: Writing Hack Programs
- Project 4 Overview

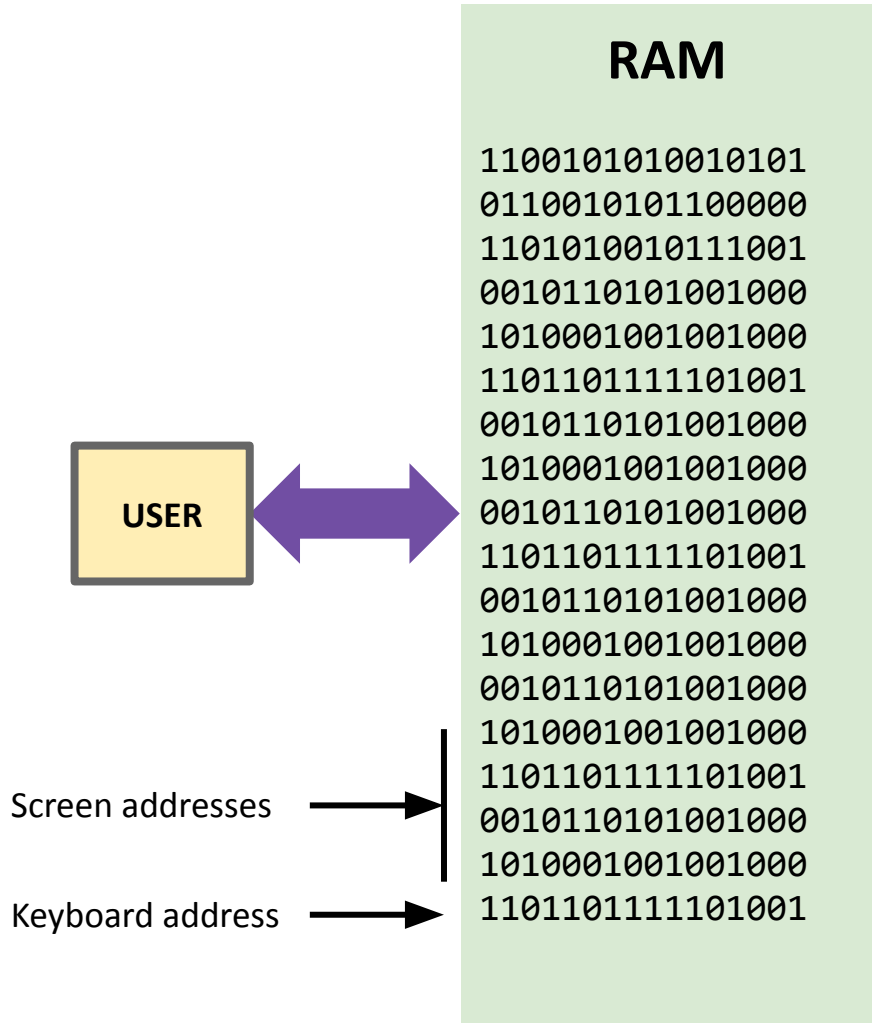
Hack: External Memory Abstraction

- Programmer sees one RAM32K memory region
  - Only 16K + 8K + 1 words/addresses of this are being used
- Split into three parts: Screen, Keyboard, and the rest
  - Screen: 8K words/addresses
  - Keyboard: 1 word/address
  - Rest: 16K words/addresses (used for data and instructions)
- Programmer can use the same interface to interact with the Screen, Keyboard, or normal RAM
  - Just specify address/value/load/etc.
  - Address determines what you are interacting with

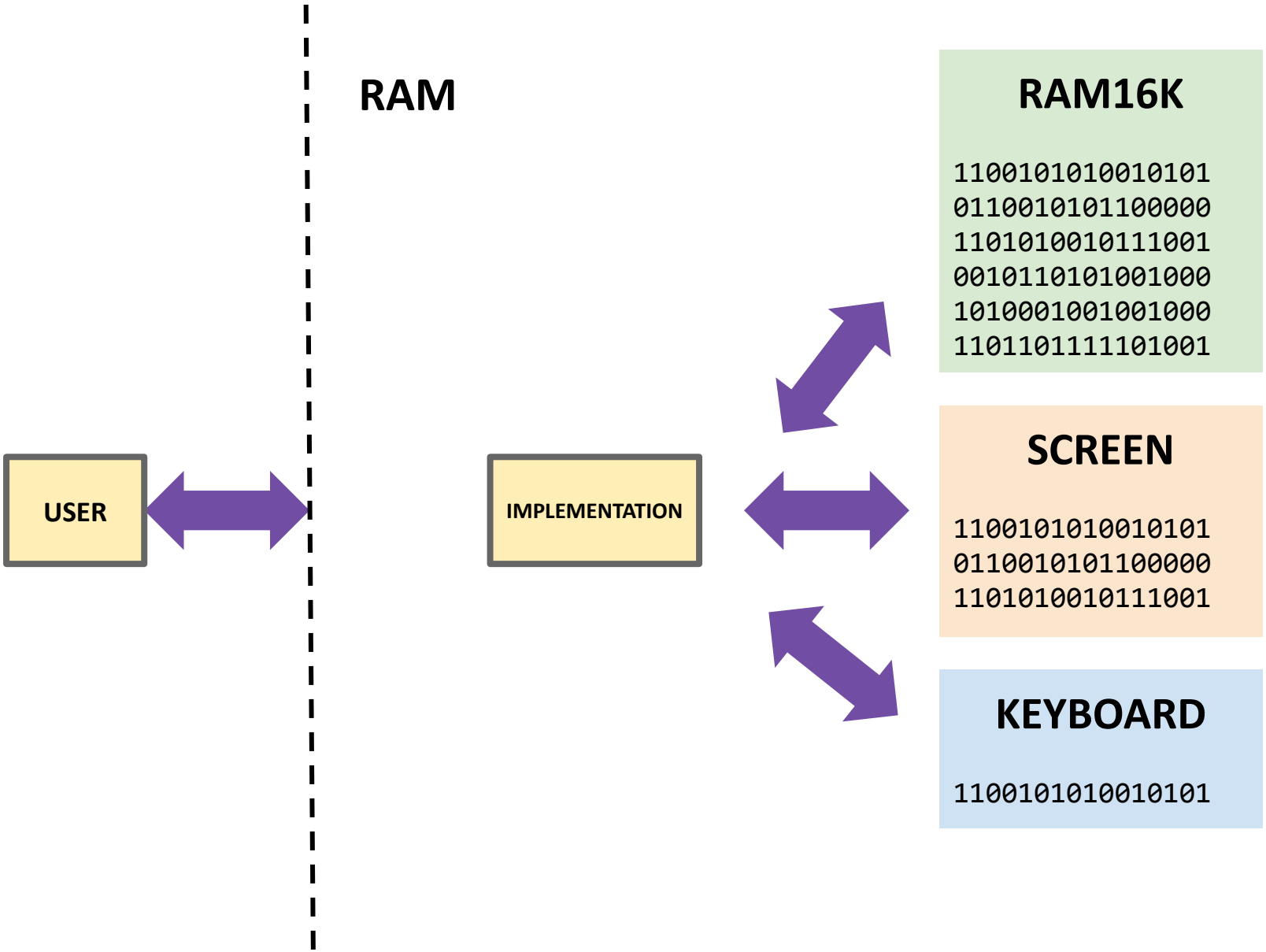
Hack: Internal Memory Implementation

- In reality, no need for a separate memory chip for memory devices
  - "drivers" are code that relay changes in memory values to the device
- In Hack, it's not quite as simple as one RAM32K chip…
  - Use internal Keyboard and Screen chips so our virtual computer can detect/show changes in the keyboard and screen
  - Again, NOT inherent to memory mapped devices, just to Hack
- Our Memory chip has 3 subchips:
  - Screen, Keyboard, and RAM16K (we give you Screen and Keyboard to use, you've already implemented RAM16K)
  - Processes the address given by the programmer and relays the request to the appropriate subchip

Hack: Memory Abstraction User View



Hack: Memory Abstraction Internal View

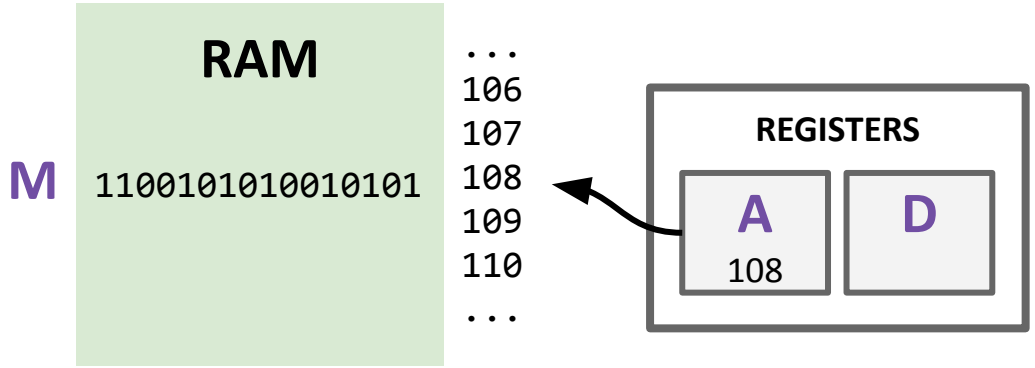


Agenda

- Social Reflection Prompt I Discussion
- Reading Review and Q&A
- Implementing Memory Devices
- **Exercise: Writing Hack Programs**
- Project 4 Overview

Review: Hack: Registers

- **D Register**: For storing data
- **A Register**: For storing data AND addressing memory
- **M "Register"**: The 16-bit word of memory currently being referenced by the address in A



Review: Hack: A-Instructions

Syntax: `@value`

- **`value`** can either be:
  - non-negative decimal constant
  - symbol referring to a constant

Semantics:

- Stores **`value`** in the A register

Review: Hack: C-Instructions

Syntax: `dest = comp ; jump` (**dest** or **jump** is optional)

- **dest** is a combination of destination registers:

```
M, D, MD, A, AM, AD, AMD
```

- **comp** is a computation:

```
0, 1, -1, D, A, !D, !A, -D, -A, D+1, A+1, D-1, A-1, D+A, D-A,
A-D, D&A, D|A M, !M, -M, M+1, M-1, D+M, D-M, M-D, D&M, D|M
```

- **jump** is an unconditional or conditional jump:

```
JGT, JEQ, JGE, JLT, JNE, JLE, JMP
```

Semantics:

- Computes value of **comp**
- Stores results in **dest** (if specified)
- If **jump** is specified and condition is true (by testing **comp** result), jump to instruction ROM[A]

Example: Implementing Multiplication

- Let's write a program that multiplies R0 and R1 and stores the result in R2.
  - Remember we don't have a multiply operation!
  - Will have to use add and loops to get the job done

- Roadmap
  - Start with pseudo code using loops/if/etc
  - Remove loops/if/etc by using jumps in our pseudocode
  - Convert to assembly

Example: Implementing Multiplication

- Goal R0 * R1 = R2
- Pseudocode (adds R0 to the result R1 times):

```
R2 = 0
while (R1 > 0) {
   R2 = R0 + R2
   R1 = R1 - 1
}
```

Example: Implementing Multiplication

- Remove loops from pseudocode
- Uses labels to notate important sections of the code

```
R2 = 0
while (R1 > 0) {
  R2 = R0 + R2
  R1 = R1 - 1
}
```

Attempt 1: What happens when R1 is 0? What should happen?

```
START:
  R2 = 0
LOOP:
   R2 = R0 + R2
   R1 = R1 - 1
   IF R1 > 0 JMP LOOP
END:
   INFINITE LOOP
```

Example: Implementing Multiplication

- Remove loops from pseudocode
- Uses labels to notate important sections of the code

```
R2 = 0
while (R1 > 0) {
   R2 = R0 + R2
   R1 = R1 - 1
}
```

Attempt 2: Problem solved!

```
START:
   R2 = 0
LOOP:
    IF R1 <= 0 JMP to END
    R2 = R0 + R2
    R1 = R1 - 1
    JMP LOOP
END:
    INFINITE LOOP
```

Example: Implementing Multiplication

- Convert to Hack Assembly

```
START:
    R2 = 0
LOOP:
    IF R1 <= 0 JMP to END
    R2 = R0 + R2
    R1 = R1 - 1
    JMP LOOP
END:
    INFINITE LOOP
```

```
(START)
    @R2
    M = 0
(LOOP)


(END)
```

Example: Implementing Multiplication

- Convert to Hack Assembly

```
START:
    R2 = 0
LOOP:
    IF R1 <= 0 JMP to END
    R2 = R0 + R2
    R1 = R1 - 1
    JMP LOOP
END:
    INFINITE LOOP
```

```
(START)
    @R2
    M = 0
(LOOP)
    @R1
    D = A
    @END
    D; JLE

(END)
```

What's not quite right here?

Example: Implementing Multiplication

- Convert to Hack Assembly

```
START:
    R2 = 0
LOOP:
    IF R1 <= 0 JMP to END
    R2 = R0 + R2
    R1 = R1 - 1
    JMP LOOP
END:
    INFINITE LOOP
```

```
(START)
    @R2
    M = 0
(LOOP)
    @R1
    D = M
    @END
    D; JLE

(END)
```

Example: Implementing Multiplication

- Convert to Hack Assembly

```
START:
    R2 = 0
LOOP:
    IF R1 <= 0 JMP to END
    R2 = R0 + R2
    R1 = R1 - 1
    JMP LOOP
END:
    INFINITE LOOP
```

```
(START)
    @R2
    M = 0
(LOOP)
    @R1
    D = M
    @END
    D; JLE
    @R0
    D = M
    @R2
    M = M + D

(END)
```

Example: Implementing Multiplication

- Convert to Hack Assembly

```
START:
    R2 = 0
LOOP:
    IF R1 <= 0 JMP to END
    R2 = R0 + R2
    R1 = R1 - 1
    JMP LOOP
END:
    INFINITE LOOP
```

```
(START)
    @R2
    M = 0
(LOOP)
    @R1
    D = M
    @END
    D; JLE
    @R0
    D = M
    @R2
    M = M + D
    @R1
    M = M - 1
    @LOOP
    0; JMP
(END)
```

Example: Implementing Multiplication

- Convert to Hack Assembly

```
START:
    R2 = 0
LOOP:
    IF R1 <= 0 JMP to END
    R2 = R0 + R2
    R1 = R1 - 1
    JMP LOOP
END:
    INFINITE LOOP
```

```
(START)
    @R2
    M = 0
(LOOP)
    @R1
    D = M
    @END
    D; JLE
    @R0
    D = M
    @R2
    M = M + D
    @R1
    M = M - 1
    @LOOP
    0; JMP
(END)
    @END
    0; JMP
```

Practice Exercise

- Implement an absolute value function, which stores the absolute value of R0 in R1
- We've pushed starter code and test files to your repos under `projects/lecture/`
  - You'll have to run a "git pull" to update your repo first
- You can work on scratch paper if you'd rather, but this might be a good chance to become familiar with the new tools
- Use your fellow classmates to get help as you work on the problems! Encourage you all to try the problem/tools yourselves but make sure ask questions when you have them
  - I'll broadcast reminders to check in with your classmates
- Feel free to pull up the project 4 spec/lecture slides/readings/etc. to reference as you work

Agenda

- Social Reflection Prompt I Discussion
- Reading Review and Q&A
- Implementing Memory Devices
- Exercise: Writing Hack Programs
- **Project 4 Overview**

Project 4: Annotation Specs

- Annotate Project 4 Spec
  - Identify 5 annotation strategies that you want to try
  - Practice these strategies on the P4 Spec

- Fill out the Assignment Timeline
  - Divide up Project 4 into doable chunks for the days you plan to work on the assignment
  - Describe each day's task in as much detail as possible

Project 4: Annotation Specs

- Complete Annotation Reflection
  - Reflect on the strategies you used and why or why not they were effective

- Submit a copy of your annotations along with the Assignment Timeline document and the Annotation Reflection document

Project 4 Overview

- PART I: Annotation
  - Come prepared to your upcoming TA 1:1 to do some Project 4 spec reading and identifying annotation strategies you would want to use
- PART II: Assembly Language
- PART III: Building a Computer Part I (Memory)

Wrapping Up

What's in store for Week 5?

- Building a Computer!
- Exam Preparation
- Project 5 Released

Reminders

- Project 2 Grades Released on Gradescope
- Project 3 Due Tonight 11:59PM PDT