CSE 390B, 2024 Spring

Building Academic Success Through Bottom-Up Computing

Student Wellness & The Assembler

Stress and Student Wellness, Inside the Assembler, Compilers and The Software Stack, Hack CPU Logic Example

Lecture Outline

- **Stress and Student Wellness**
  - **Investing Time for Self-care and Well-being**
- Inside the Assembler
  - Assembler Motivations and Challenges
  - Parsing, Symbols, Encoding
- Compilers and The Software Stack
  - Steps for Compiling Software
- Hack CPU Logic Example: writeM
  - Project 6 CPU Logic Exercise

Stress and Student Wellness

- Students are generally reporting increasing depression, anxiety, and suicidal thoughts
  - The stress we feel from school can amplify these feelings
- A lack of self-care can hinder our academic performance as a student
- In a survey, over 80% of students felt that emotional or mental difficulties have hurt their academic performance (UW Healthy Minds Survey, 2017)

Stigma and Seeking Help

- Asking for help can be challenging and take humility
- However, seeking help is an important step in taking care of ourselves and helping us perform better academically
- 94% of UW students **disagreed** with the statement, "I would think less of someone who has received mental health treatment." (UW Healthy Minds Survey, 2017)

Student Wellness Discussion

In groups, discuss the following questions:

- What is your typical response for when life circumstances becomes stressful?
- What are some strategies you can utilize for managing stress?
- What actionable steps can you take to foster your own well-being? What are some SMART goals you can set?

Strategies for Managing Stress

- Set aside time for leisure time to do something you enjoy (e.g., reading, knitting, gaming, completing a puzzle, etc.)
- Take care of your body by eating healthily, exercising regularly, getting enough sleep, etc.
- Write down all the things in your mind that is causing the stress you are experiencing
- Take the time to connect with people 1:1 or with a community and share about your life

Looking Out for One Another

- Promote a climate of care among your peers in the UW and Allen School
- Look for warning signs (unusual moods, relationship dynamics, academic patterns, suicidal thoughts)
- Express concern for your peers and let them know that you are there for them
- Point them to resources for seeking additional help

Resources for Seeking Additional Help

- If depression, anxiety, or thoughts about suicide become a pattern, be proactive about reaching out for help

- Several resources available from both UW and within the Allen School
  - UW resources include SAFECAMPUS, UW Counseling Center, etc.
  - See the CSE 390B Resources page for more

Resources for Student Wellness

- **Allen School Advising Team** can help you with your broader academic situation and provide additional resources and suggestions for support. Contact options can be found here.
- **UW COVID-19 Resources** include two lists of resources related to COVID-19, one from general UW and one from the Race and Equity Initiative.
- **UW Mental Health Resources** located here, including urgent help, workshops, counseling, and more
- **UW Resilience Lab** is a group focused on promoting well-being at UW.
- **UW Academic Success Coaches** providing support and resources for helping you achieve your academic goals.
- Coping with COVID-19 Related Stress

Lecture Outline

- Stress and Student Wellness
  - Investing Time for Self-care and Well-being
- **Inside the Assembler**
  - **Assembler Motivations and Challenges**
  - **Parsing, Symbols, Encoding**
- Compilers and The Software Stack
  - Steps for Compiling Software
- Hack CPU Logic Example: writeM
  - Project 6 CPU Logic Exercise

Producing Machine Code

```
while (i < 100)
{
  sum += arr[i];
  i++;
}
```

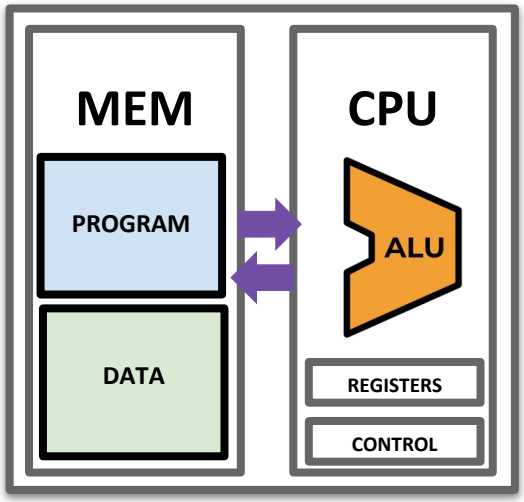
Java

Compile

```
0101110011100110
1011000101010100
1110001011111100
...
```

Machine Code Instructions

Load & Execute

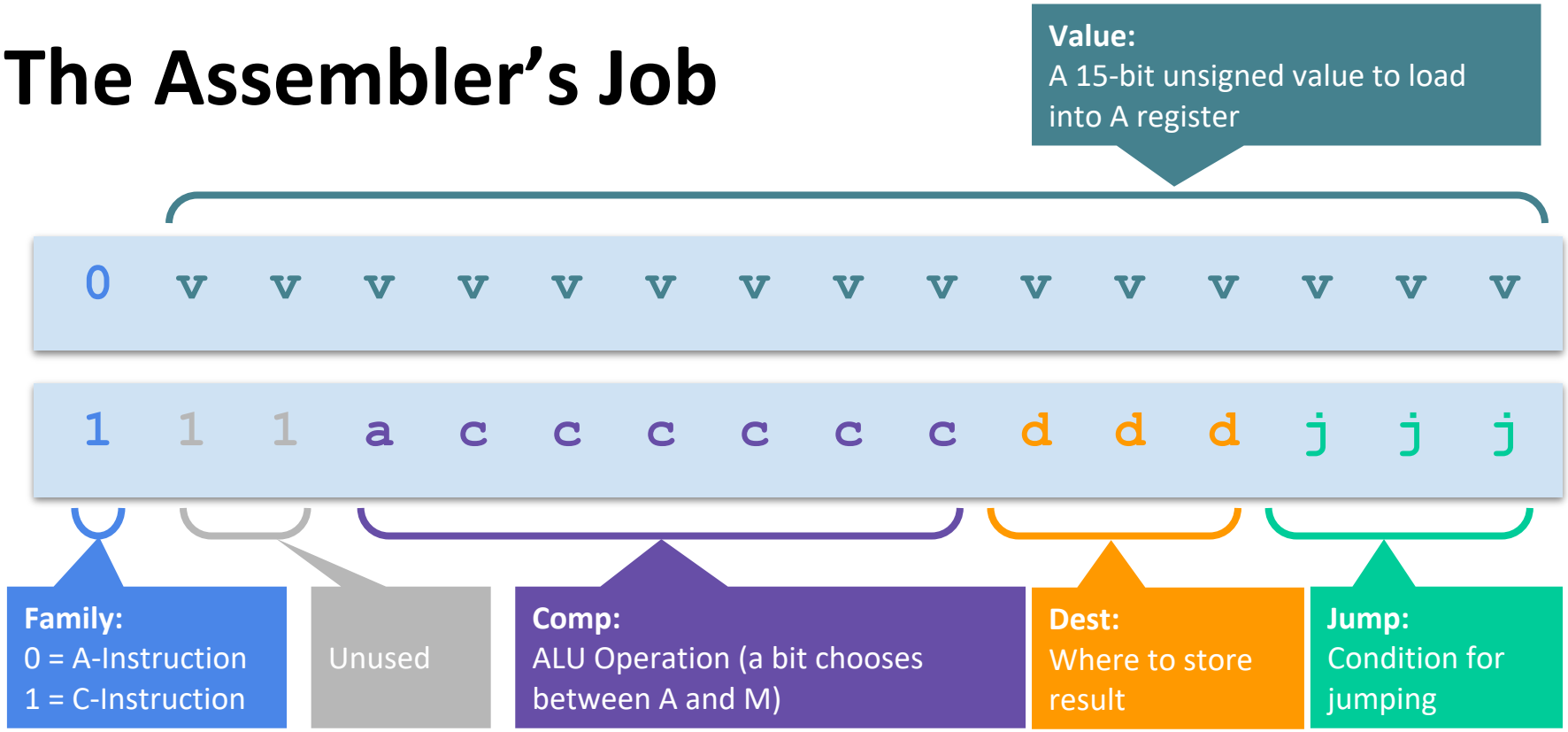


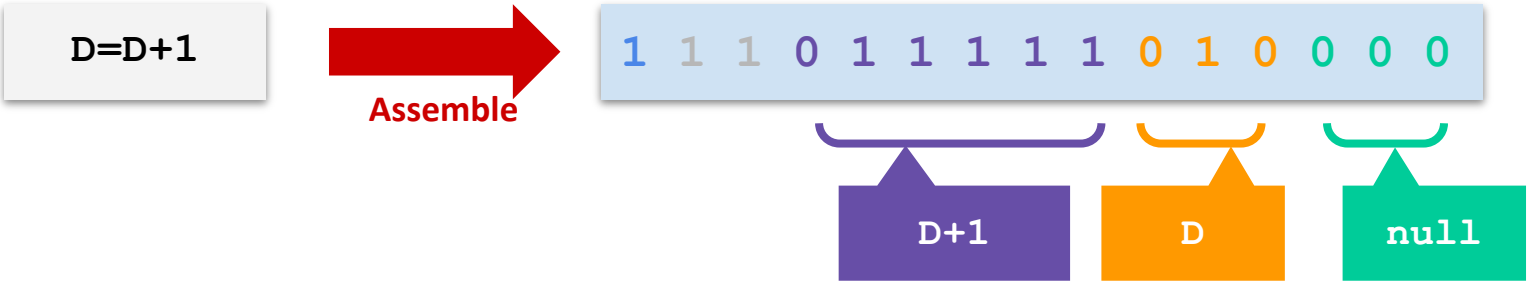
Assemble

```
movq $5, %rdx
addq %rsx, %rdx
movq %rdx, %rax
ret
```

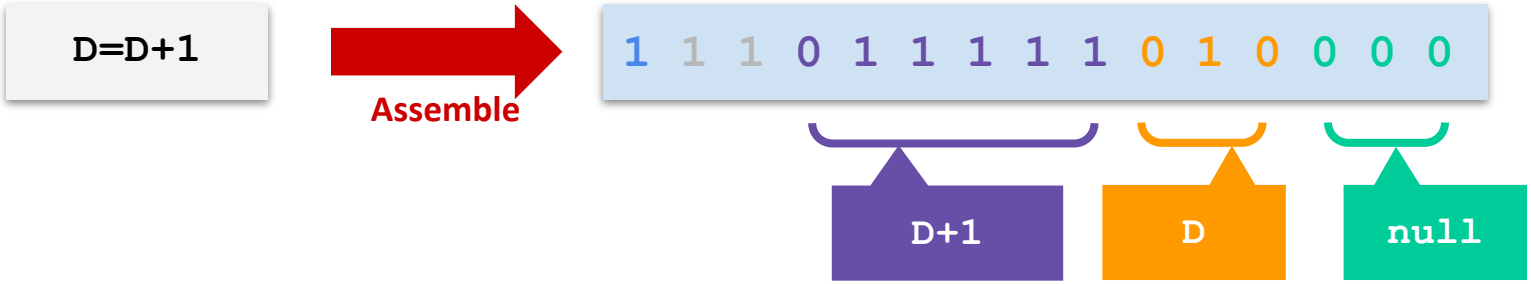
Assembly Language

The Assembler's Job





The Assembler's Job



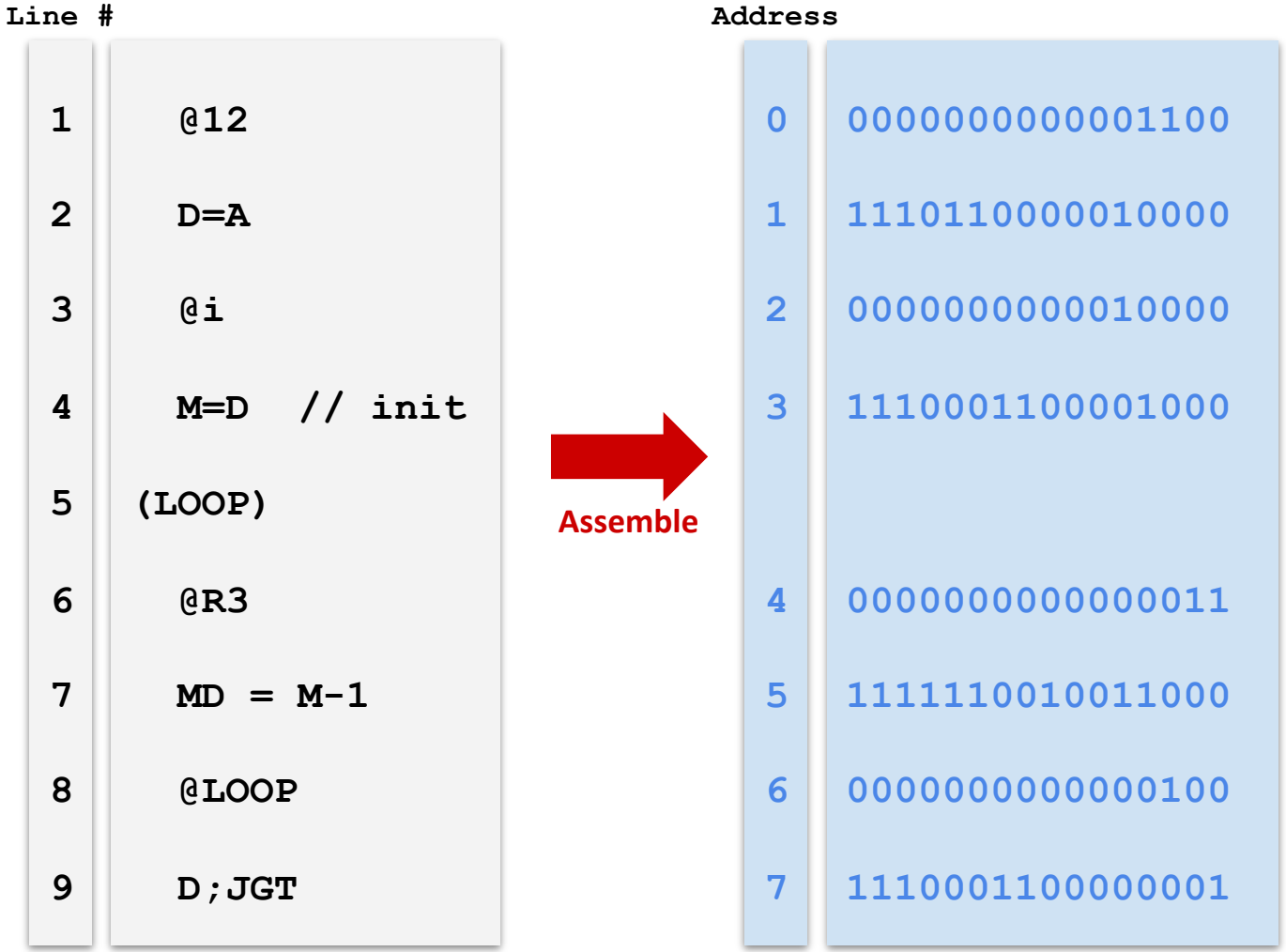
- Look up each value in the corresponding table

j1 (*out* < 0)	j2 (*out* = 0)	j3 (*out* > 0)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If *out* > 0 jump
0	1	0	JEQ	If *out* = 0 jump
0	1	1	JGE	If *out* ≥ 0 jump
1	0	0	JLT	If *out* < 0 jump
1	0	1	JNE	If *out* ≠ 0 jump
1	1	0	JLE	If *out* ≤ 0 jump
1	1	1	JMP	Jump

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

(when a=0) *comp* mnemonic	c1	c2	c3	c4	c5	c6	(when a=1) *comp* mnemonic
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

Difficulties for the Assembler



Difficulties for the Assembler

- Three broad concerns:

Parsing	Recognizing type of each instruction and label, extracting relevant fields, skipping whitespace & comments
Symbols	Mapping from labels to instruction addresses, mapping from code symbols to RAM addresses, creating new symbols, corresponding line numbers to instruction addresses
Encoding	Converting relevant fields to binary values, converting symbol values to binary values

Bells and Whistles… Why Bother?

- Tradeoff: Adding convenience for programmer makes it harder to build the Assembler
  - E.g., removing symbols from Hack would make Assembler much simpler, still possible to write all the same programs
  - But language would be far more annoying to use

- **Don't underestimate the importance of convenience**
  - Put another way: Adding these extra features makes programmers more productive

Parsing

- Source code is just a giant string: we need to go character-by-character to understand that string

- Parser presents iterator-like interface:
  - To “advance” one instruction:
    - Move cursor forward, skipping whitespace and comments, until next non-empty line (ending on a newline)
  - To “read” current instruction:
    - Throw away whitespace & comments
    - Determine what type of instruction
    - Pull relevant fields out

Symbols: Labels

- Keep symbol table, mapping symbols (strings) to their values (integers)
  - Initialize with built-in symbols

SYMBOL	VALUE
R0	0
R1	1
...	...
R15	15
SCREEN	16384
KBD	24576

Symbols: Labels

- Keep symbol table, mapping symbols (strings) to their values (integers)
  - Initialize with built-in symbols

SYMBOL	VALUE
R0	0
R1	1
...	...
R15	15
SCREEN	16384
KBD	24576

- Run through instructions, using this pseudocode:

```
If current line is (LABEL):
    Add LABEL → next address to
    symbol table
If current line is @LABEL:
    Lookup LABEL in symbol table,
    insert value into A-instruction
```

Symbols: Labels

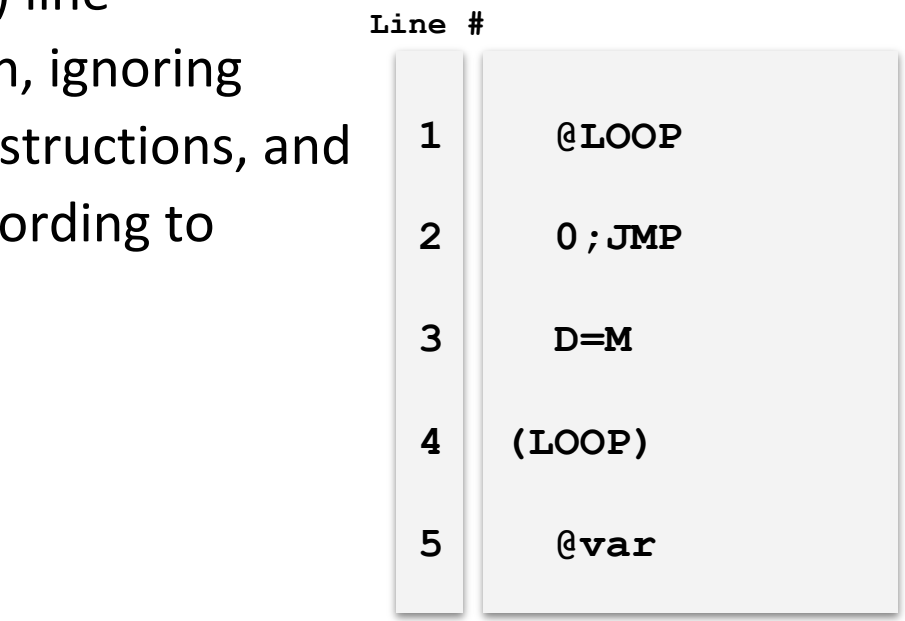
- Problem: what if a label's use comes before its definition?

```
Line #
1     @LOOP
2     0;JMP
3     D=M
4   (LOOP)
5     @var
```

Symbols: Labels

- Problem: what if a label's use comes before its definition?

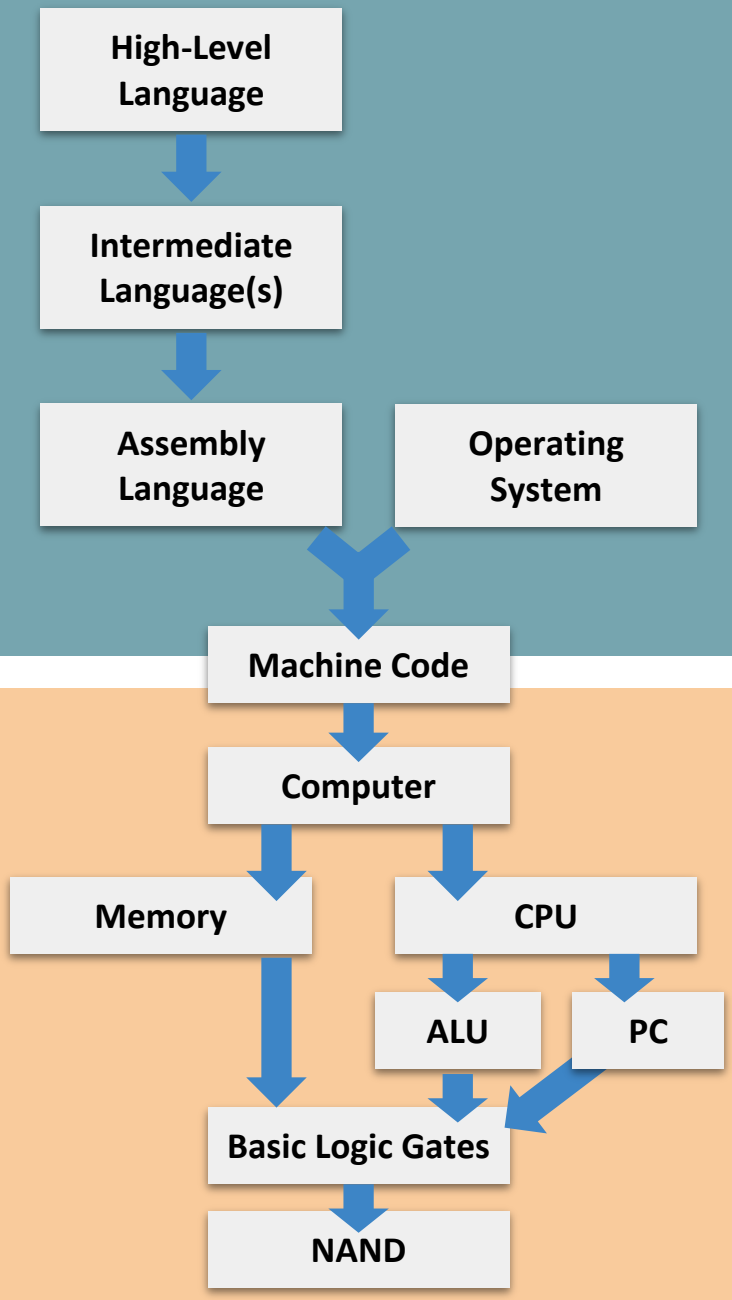
- Solution: Two passes
  - Pass 1: Populate symbol table by moving through file and ignoring anything that isn't a (LABEL) line
  - Pass 2: Go through file again, ignoring (LABEL) lines, encoding C-instructions, and encoding A-instructions according to symbol table lookup



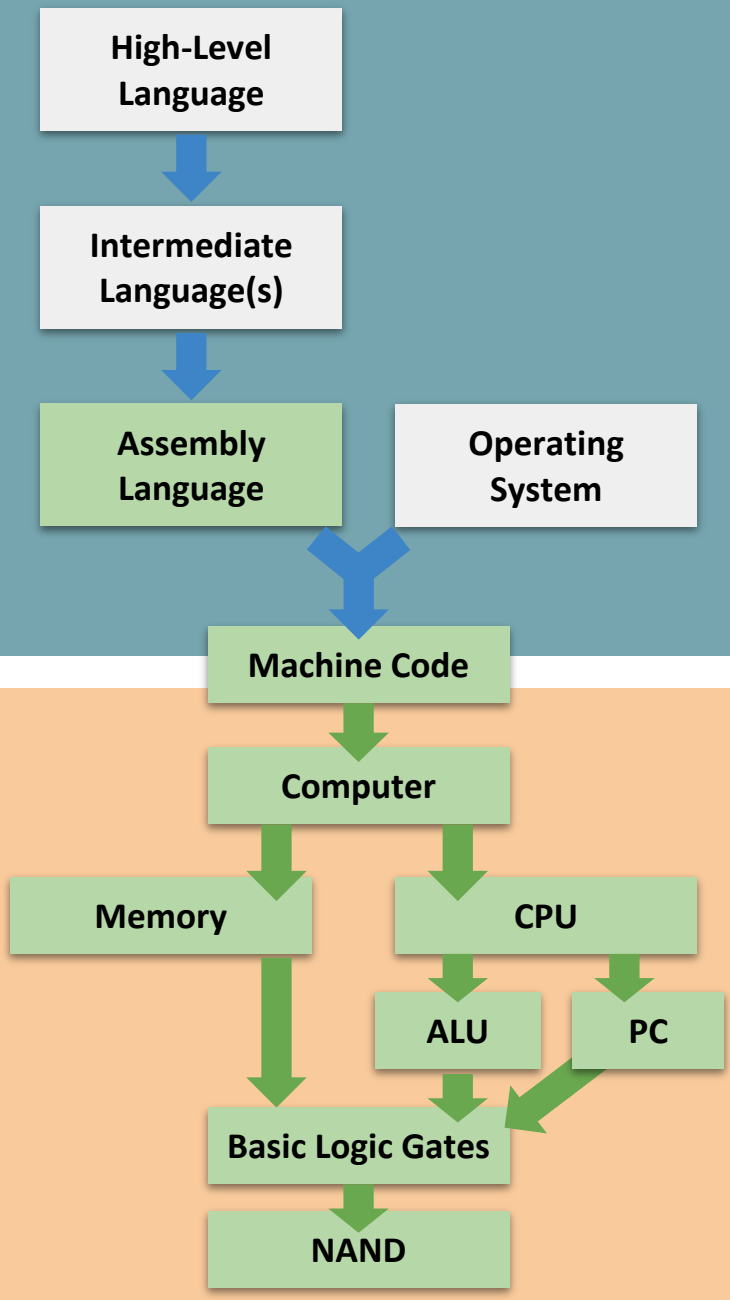
Lecture Outline

- Stress and Student Wellness
  - Investing Time for Self-care and Well-being
- Inside the Assembler
  - Assembler Motivations and Challenges
  - Parsing, Symbols, Encoding
- **Compilers and The Software Stack**
  - **Steps for Compiling Software**
- Hack CPU Logic Example: writeM
  - Project 6 CPU Logic Exercise

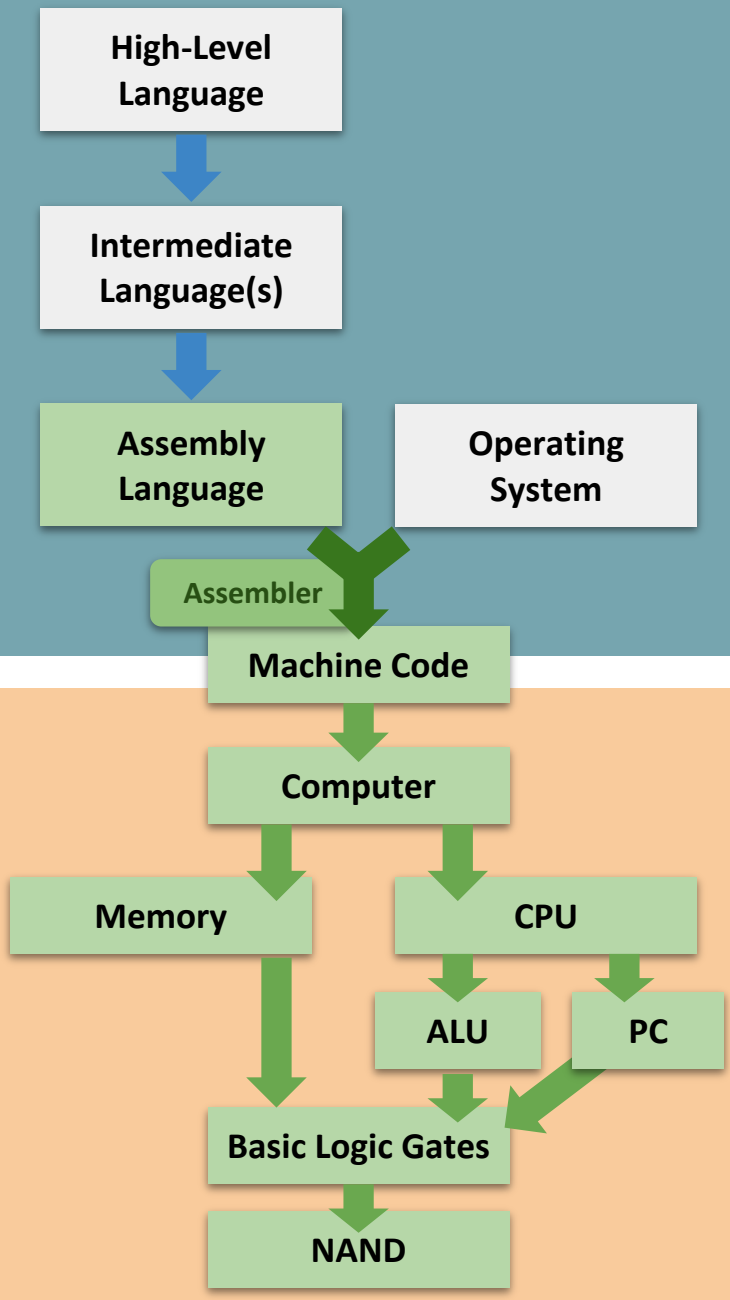
Roadmap


High-Level Language
Intermediate Language(s)
Assembly Language
Operating System
SOFTWARE
Machine Code
HARDWARE
Computer
Memory
CPU
ALU
PC
Basic Logic Gates
NAND

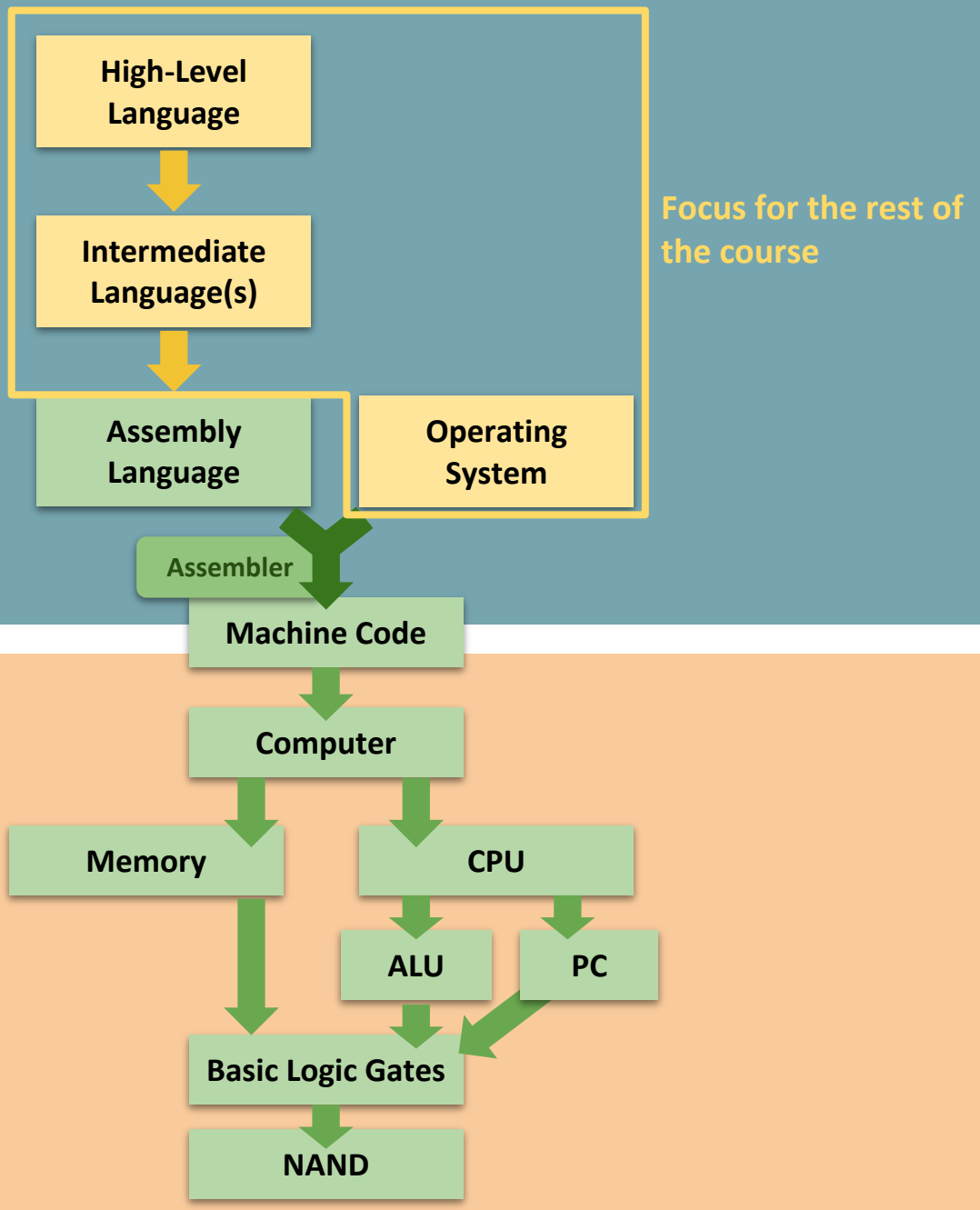
Roadmap


High-Level Language
Intermediate Language(s)
Assembly Language
Operating System
SOFTWARE
Machine Code
HARDWARE
Computer
Memory
CPU
ALU
PC
Basic Logic Gates
NAND

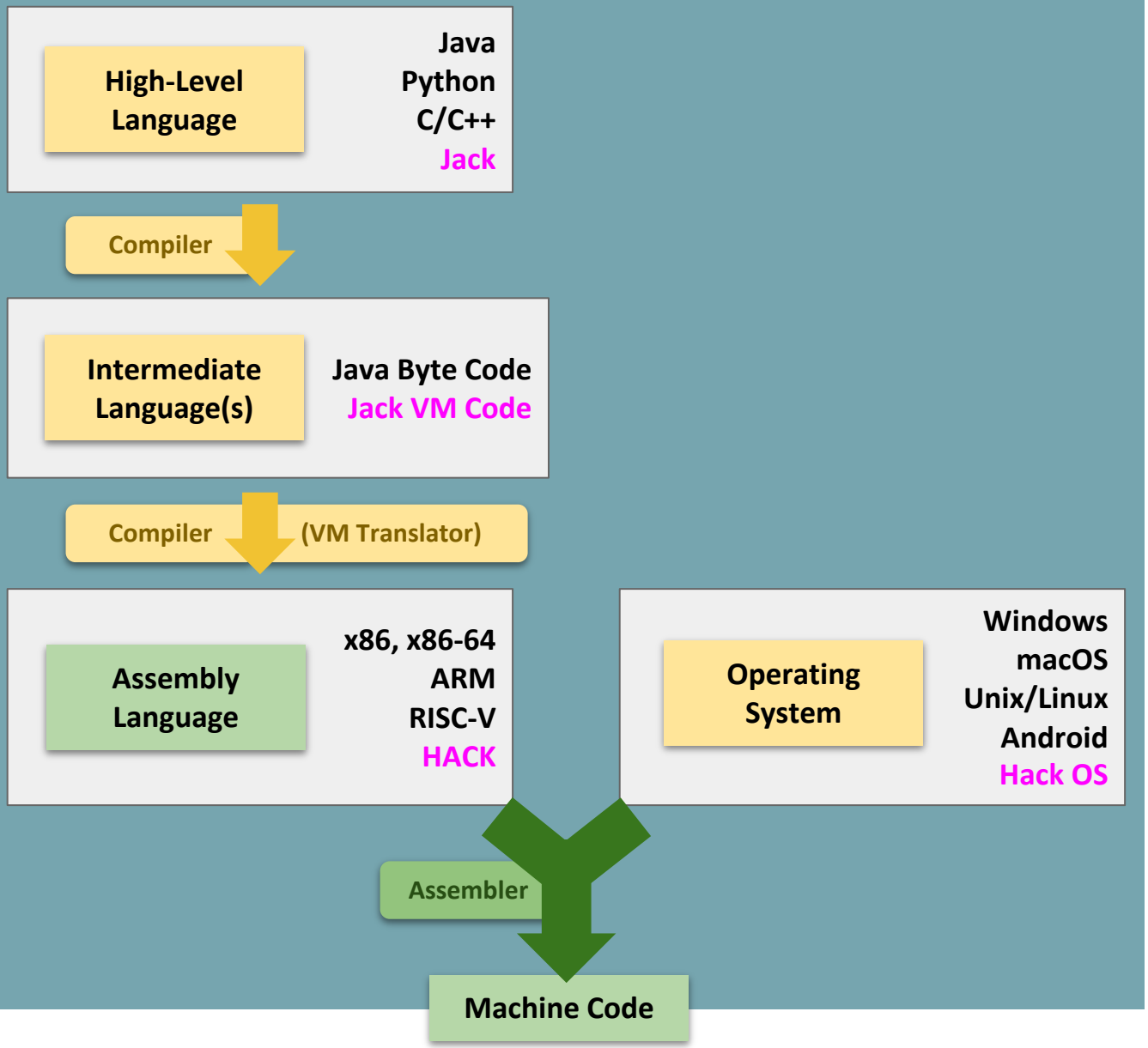
Roadmap


High-Level Language
Intermediate Language(s)
Assembly Language
Operating System
SOFTWARE
Assembler
Machine Code
HARDWARE
Computer
Memory
CPU
ALU
PC
Basic Logic Gates
NAND

Roadmap

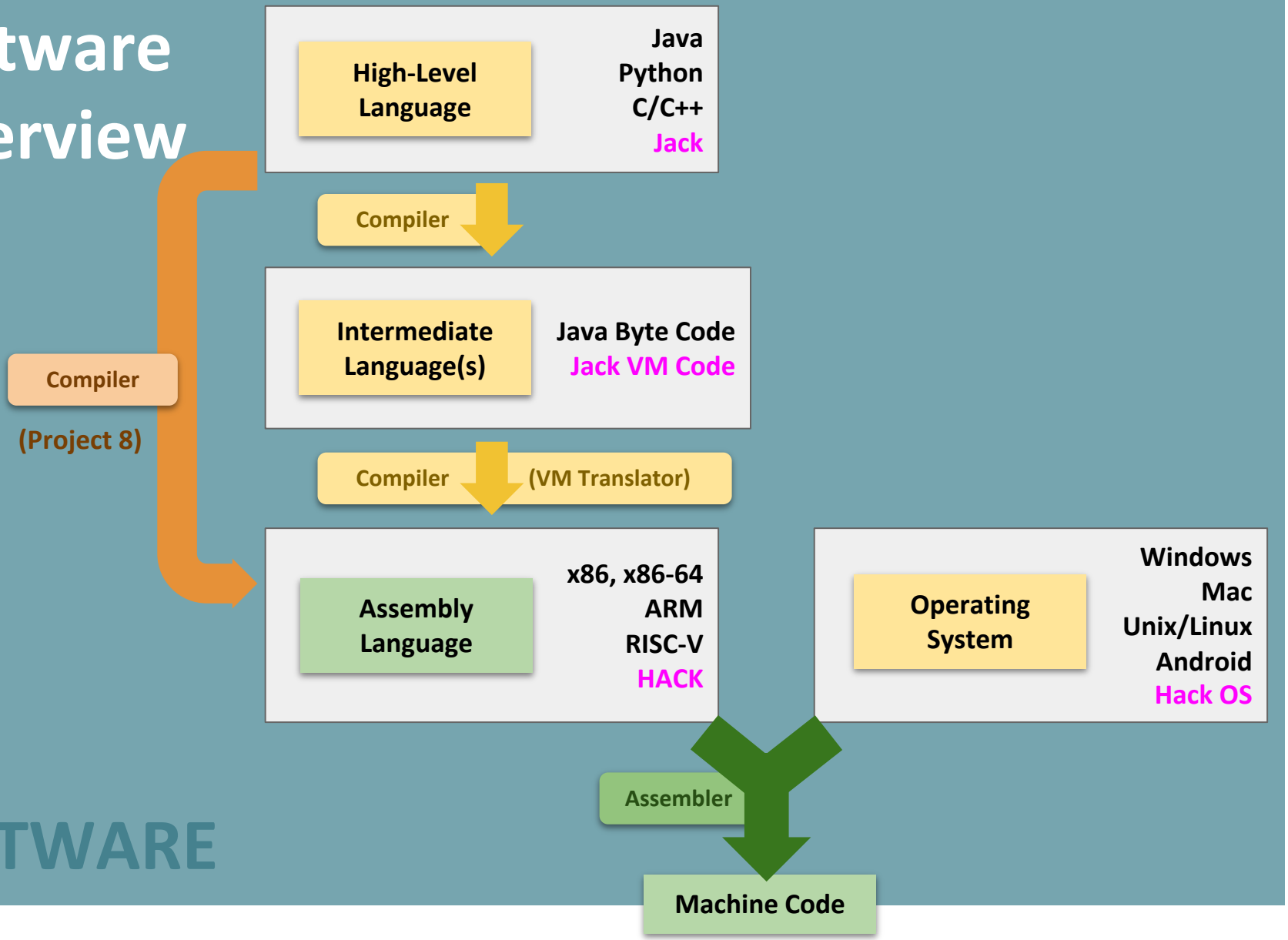

High-Level Language
Intermediate Language(s)
Assembly Language
Operating System
Focus for the rest of the course
SOFTWARE
Assembler
Machine Code
HARDWARE
Computer
Memory
CPU
ALU
PC
Basic Logic Gates
NAND

Software Overview



SOFTWARE

Software Overview



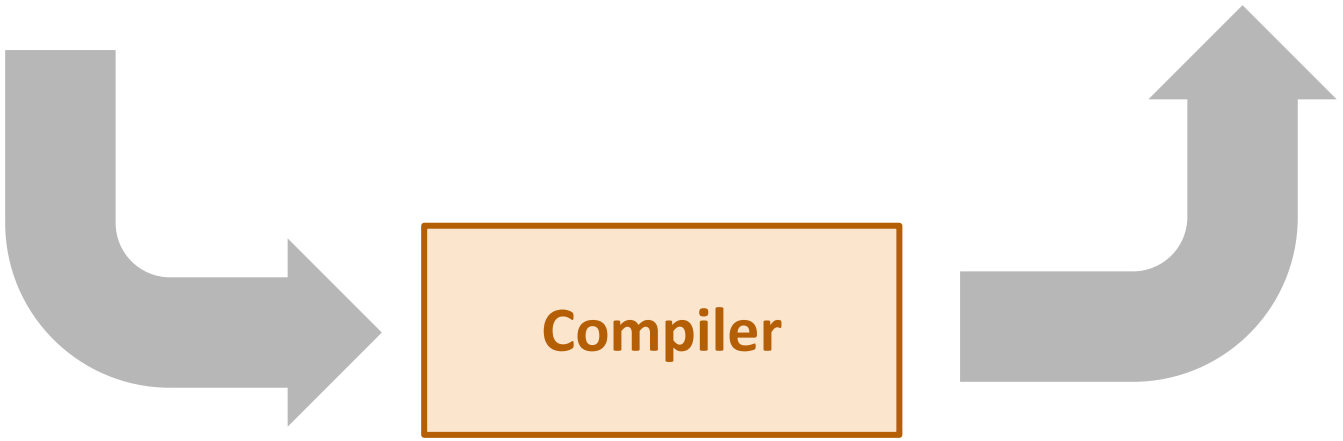
The Compiler: Goal

```
public int fact(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * fact(n - 1);
  }
}
```

High-Level Language

```
(fact)
  @R0
  M=M+1
  @R1
  D=A
  @ifbranch
  D;JEQ
```

Assembly Language



The Compiler: Goal

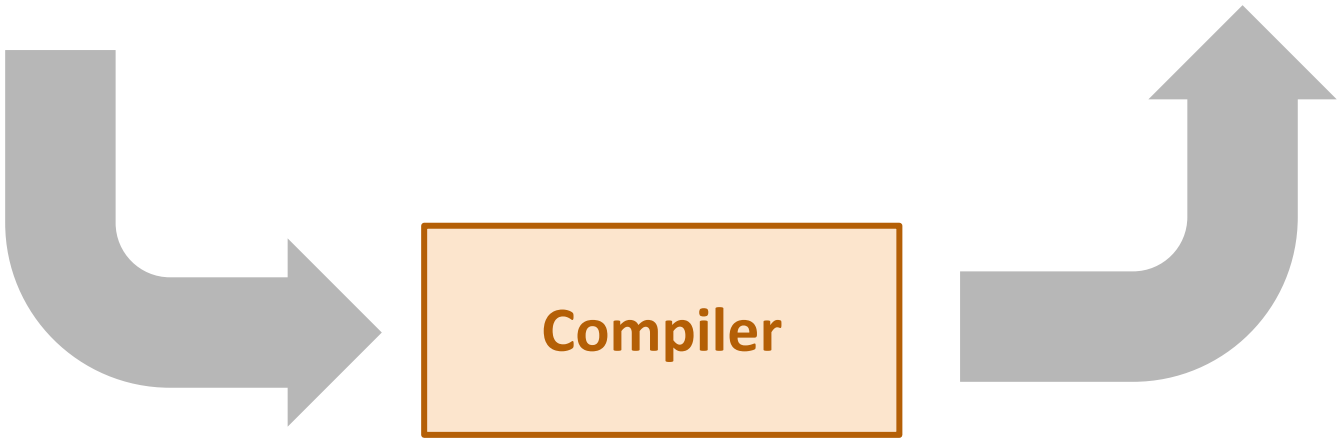
```
public int fact(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * fact(n - 1);
  }
}
```

High-Level Language

Theory Definition: a string, from the set of strings making up a language

```
(fact)
  @R0
  M=M+1
  @R1
  D=A
  @ifbranch
  D;JEQ
```

Assembly Language



The Compiler: Goal

```
public int fact(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * fact(n - 1);
  }
}
```

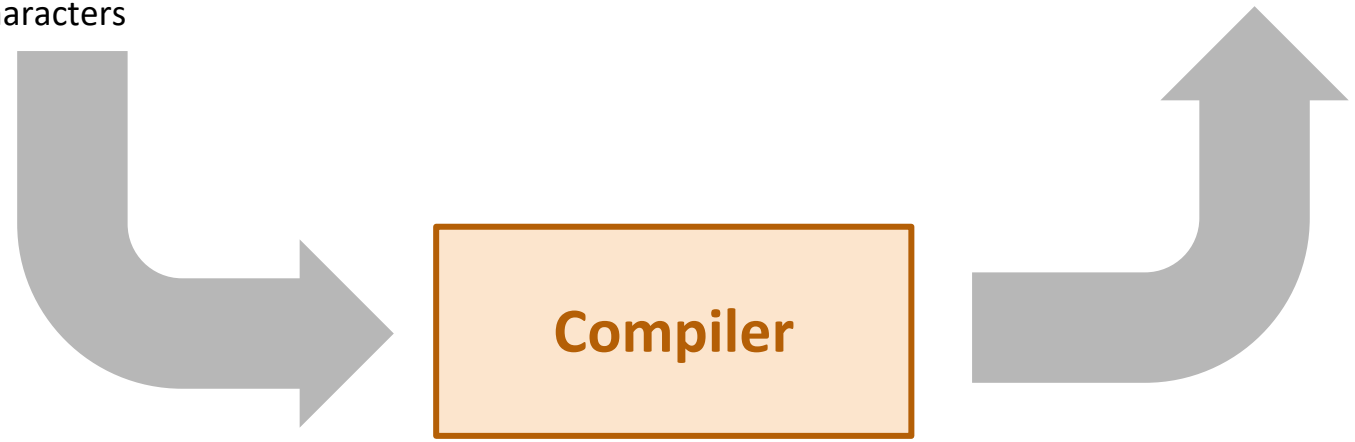
High-Level Language

Theory Definition: a string, from the set of strings making up a language

Practical Definition: a file containing a bunch of characters

```
(fact)
  @R0
  M=M+1
  @R1
  D=A
  @ifbranch
  D;JEQ
```

Assembly Language



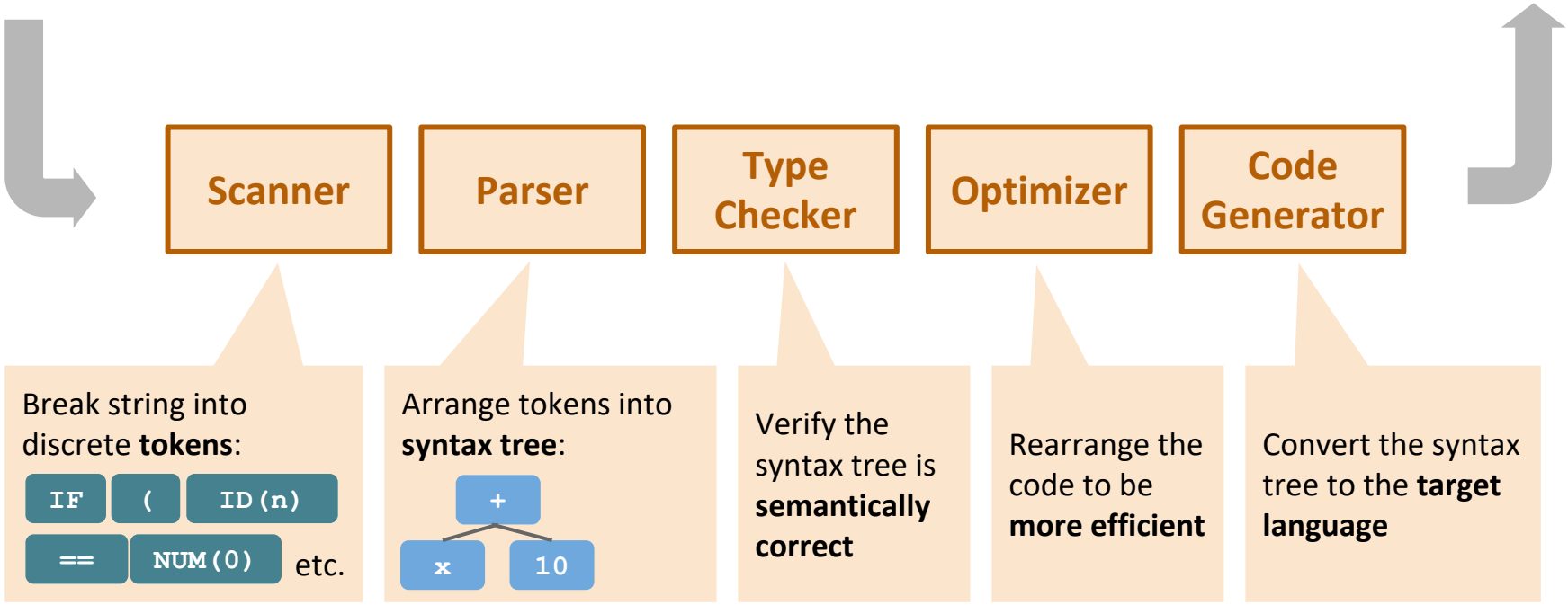
The Compiler: Implementation

```
public int fact(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * fact(n - 1);
  }
}
```

High-Level Language

```
(fact)
  @R0
  M=M+1
  @R1
  D=A
  @ifbranch
  D;JEQ
```

Assembly Language



Lecture Outline

- Stress and Student Wellness
  - Investing Time for Self-care and Well-being
- Inside the Assembler
  - Assembler Motivations and Challenges
  - Parsing, Symbols, Encoding
- Compilers and The Software Stack
  - Steps for Compiling Software
- **Hack CPU Logic Example: writeM**
  - **Project 6 CPU Logic Exercise**

Hack CPU Logic Example: writeM

- Example: Determine when **`writeM`** should be set to 1

- Step 1: What do we pay attention to?
  - **`writeM`** is related to whether we write to memory or not
  - We need to look up the destination bits specification from Chapter 4

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

Figure 4.4 The *dest* field of the *C*-instruction.

Hack CPU Logic Example: writeM

- Example: Determine when **`writeM`** should be set to 1

- Step 2: Determine logic for specification
  - Read the "Destination Specification" section of Chapter 4
  - Instruction bits:

```
1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3
```

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

Figure 4.4 The *dest* field of the *C*-instruction.

Hack CPU Logic Example: writeM

- Example: Determine when **`writeM`** should be set to 1

- Step 2: Determine logic for specification
  - Read the "Destination Specification" section of Chapter 4
  - Instruction bits:

```
1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3
```

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

Figure 4.4 The *dest* field of the *C*-instruction.

Hack CPU Logic Example: writeM

- Example: Determine when **`writeM`** should be set to 1

- Step 2: Determine logic for specification
  - Read the "Destination Specification" section of Chapter 4
  - Instruction bits:

    `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`
  - d3 determines if the output should be written to memory
  - Which bit of our instruction is that?
  - So **`writeM = instruction[3]`**?

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register

Hack CPU Logic Example: writeM

- Example: Determine when **`writeM`** should be set to 1

- What's wrong with **`writeM = instruction[3]`**?
  - What happens if we have an A-instruction?
  - We only write to destinations in the case of a C-instruction
  - So, **`writeM = C-instruction & instruction[3]`**
  - Certain actions only occur on certain instruction types
  - You may have to include a check for instruction type in your logic

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register

Hack CPU Implementation: Logic Sub Chips

- We provide three sub chips and tests that implement the control logic for the A Register, D Register, and PC
  - **`LoadAReg`** contains logic for loading the A Register
  - **`LoadDReg`** contains logic for loading the D Register
  - **`JumpLogic`** contains logic for determining if the PC should load, jump, or increment

- Implement and test these first, then use them in your CPU implementation
  - Intended to help you narrow the scope of bugs

Lecture 13 Reminders

- Midterm grades and feedback will be released by Friday
- **Project 6 (Mock Exam Problem & Building a Computer) due this Friday (5/10) at 11:59pm**
- Project 7, Part I (Midterm Corrections) released, due next Friday (5/17) at 11:59pm **(no late days may be used)**
- Course Staff Support
  - Ray has office hours today after class in CSE2 151
  - Amy has office hours tomorrow at 10:30am in CSE2 153
  - Feel free to post your questions on the Ed discussion board