11-18 Lecture

0. Admin/Homework 2

- **Homework 2 Availability:**
  - Expected release: Around noon today.
  - Task involves revising the final solution to the Wordle problem.
  - Focus on implementing a hint generator for Wordle using **information theory**.
- **Key Tasks for Homework 2:**
  - **Understand Wordle-A and its modifications:**
    - Minor changes to improve compatibility and formatting.
    - Includes importing `logarithm` functions for theoretical purposes.
  - **Information Theory Component:**
    - Central concept: Shannon entropy.
    - Goal: Optimize Wordle guess strategies based on entropy calculations.
  - **Implementation:**
    - Write two methods:
      - **Expand Pattern:** Generates possible patterns based on feedback.
      - **Hint:** It finds the word with the highest entropy from the remaining set of words.
- **Important Notes:**
  - Entropy measures the surprise or uncertainty of an event (measured in bits).
  - Homework focuses on the recursive expansion of patterns and computing entropy for feedback optimization.
  - Emphasis on understanding over coding complexity.

1. Review: The One Address Machine (OAM)

- **OAM Overview:**
  - A simple computing model to understand foundational machine-level execution.
  - Components:
    1. **Registers:** Program Counter (PC), Accumulator, B Register, Address Register.
    2. **Memory:** Infinite with memory-mapped I/O for input/output operations.
- **Instruction Cycle:**
  - **Fetch:** Load instruction from memory to the Instruction Register.
  - **Increment:** Increment Program Counter (PC) to point to the next instruction.
  - **Execute:** Perform the operation specified by the instruction.
- **Instruction Types:**
  - Arithmetic Operations (e.g., Add, Subtract, Multiply, Divide).
  - Memory Access (e.g., Load, Store).
  - Control Operations:
    1. **Branch:** Jump to a specified address.

        2. **Branch Positive/Zero:** Conditional jumps based on accumulator values.
        3. **Halt:** Stop execution.
- **Key Example:**
    - **Program to Increment Input by 1:**
        1. Load value from memory location 0 (input).
        2. Increment accumulator.
        3. Store result back to memory location 0 (output).
        4. Halt.

2. OAM Execute Phase: Control Instructions

- **Control Instructions for Program Flow:**
    - **Branch (BR):** Unconditional jump to a specific memory address.
    - **Branch Positive (BP):** Jump if accumulator > 0.
    - **Branch Zero (BZ):** Jump if accumulator == 0.
    - **Halt:** Ends program execution.
- **Subroutine Handling:**
    - Introduced **Branch and Store (BRS)** and **Branch Indirect (BRI)** for managing subroutines:
        - **BRS:** Stores the return address before jumping to subroutine.
        - **BRI:** Returns to the stored address, allowing seamless subroutine calls.

3. OAM: Sample Program

- **Example Program with Subroutine:**
    - Subroutine to increment a number three times:
        1. Input a number.
        2. Call the subroutine thrice (using BRS).
        3. Output the result.
- **Execution Process:**
    - Program Counter shifts between main program and subroutine.
    - Subroutine increments the accumulator and returns to the main program.

4. OAM: Indirect Addressing

- **Branch Indirect (BRI):**
    - Uses the address stored in memory to determine the next instruction.
    - Allows dynamic program control based on memory values.
- **Use in Subroutines:**
    - Enables return to the exact point in the main program after executing the subroutine.

5. OAM: Using Labels

- **Simplifying Code with Labels:**

    - Labels replace raw memory addresses, making code more readable and maintainable.
    - Example: Replace numerical addresses like `7` with meaningful labels like `add_one`.
- **Advantages:**
    - Simplifies updates when program structure changes.
    - Easier to debug and understand.

6. OAM: Another Sample Program

- **Program: Prime Number Checker**
    - Logic:
        - Read input number xxx.
        - Check if x=2x = 2x=2 (prime).
        - Perform iterative primality testing for x>2x > 2x>2.
        - Output "prime" or "composite."
    - Implementation Details:
        - Use branching for decision-making (e.g., BZ for checking if x=2x = 2x=2).
        - Iterate through potential divisors for primality check.
- **Challenges:**
    - Efficiency concerns due to iterative approach.
    - Emphasis on understanding OAM's mechanics rather than optimizing algorithms.