Design and Analysis

Sunday, April 17, 2011
5:44 PM

Design

CPU Class

I) powerOn()
  a. Sets the starting point in memory and reads the corresponding instructions
  b. Begins the processor cycle
  c. Once the cylce is over, it prints the result

II) beginCycle()
  a. Prints out the heading for the data table
  b. Begins processing the instructions, and prints out the data for each step as it goes
    i. It does this by checking what the instruction is, then called the appropriate helper function to finish the job

III) JANZ()
  a. Announces itself
  b. Checks if the current accumulator is zero
    i. If true: move to the next instruction
    ii. If false: jump to the instruction given as an argument

IV) StAM()
  a. Announces itself
  b. Loads a given address location into the accumulator and moves to the next instruction

V) AddM()
  a. Announces itself
  b. Adds the value at a specified address to the accumulator and moves to the next instruction

VI) AddI()
  a. Announces itself
  b. Adds the value of the argument to the accumulator and moves to the next instruction

VII) SubM()
  a. Announces itself
  b. Subtracts the value at a specified address from the accumulator and moves to the next instruction

VIII) SubI()
  a. Announces itself
  b. Subtracts the value of the argument from the accumulator and moves to the next instruction

IX) nextLocation()
  a. Simply returns the address for the next consecutive instruction to be processed

X) Print()
  a. Shorthand method for printing stuff
  b. Unique because it make sure any lines printed are longer than 18 characters

XI) opCodeReader()
  a. Reads the current instructions and prints out the results in a more human readable format
  b. Also prints the numerical argument in the instruction

Memory Class

I) Store(String binaryLocation, String content)

    a. Checks if the content given is usable
        i. If true: saves the content in the specified address
        ii. If false: Gives error
- II) Read(String location)
    a. Reads a location in memory and returns its contents
- III) binaryToDecimal(String binary)
    a. Recursively checks if each character is 0 or 1 and multiplies it by its appropriate factor to get a decimal number equivilent
- IV) decimalToBinary(int decimal)
    a. Divides by two, adding the remainder to a string whick culminates in a binary representation of the input decimal
- V) isNext(int I)
    a. Checks if the spot after array[I] exists.

Analysis

- I) powerOn()
    - I) **Input**: None
    - II) **Output**: None
    - III) **Constraints**: None
    - IV) **Assumptions**: The memory location at 0 exists
    - V) **Modifies:** PC, insRegister
- II) beginCycle()
    - I) **Input**: None
    - II) **Output**:None
    - III) **Constraints**: None
    - IV) **Assumptions**: None
    - V) **Modifies:** insRegister
- III) JANZ()
    - I) **Input**: None
    - II) **Output**: None
    - III) **Constraints**: None
    - IV) **Assumptions**: All of the variables used are binary
    - V) **Modifies:** PC
- IV) StAM()
    - I) **Input**: None
    - II) **Output**: None
    - III) **Constraints**: None
    - IV) **Assumptions**: All of the variables used are binary
    - V) **Modifies:** PC, memory
- V) AddM()
    - I) **Input**: None
    - II) **Output**: None
    - III) **Constraints**: None
    - IV) **Assumptions**: All of the variables used are binary
    - V) **Modifies:**PC, accumulator
- VI) AddI()
    - I) **Input**: None
    - II) **Output**: None
    - III) **Constraints**: None

IV) **Assumptions**: All of the variables used are binary
V) **Modifies:** PC, Accumulator

VII) SubM()
   I) **Input**: None
   II) **Output**: None
   III) **Constraints**: None
   IV) **Assumptions**: All of the variables used are binary
   V) **Modifies:** PC, Accumulator

VIII) SubI()
   I) **Input**: None
   II) **Output**: None
   III) **Constraints**: None
   IV) **Assumptions**: All of the variables used are binary
   V) **Modifies:** PC, Accumulator

IX) nextLocation()
   I) **Input**: String representing an address location
   II) **Output**: String representing next address location
   III) **Constraints**: PC must not be greater than 32
   IV) **Assumptions**: None
   V) **Modifies:** None

X) Print()
   I) **Input**: A string to be printed
   II) **Output**: None
   III) **Constraints**: None
   IV) **Assumptions**: None
   V) **Modifies:** None

XI) opCodeReader()
   I) **Input**: None
   II) **Output**: None
   III) **Constraints**: None
   IV) **Assumptions**: There is an OPCode
   V) **Modifies:** None

<u>Memory Class</u>

I) Store(String binaryLocation, String content)
   I) **Input**: String representing a memory address and new content to be assigned to it
   II) **Output**: None
   III) **Constraints**: The new content is 8 characters or less
   IV) **Assumptions**: The address given is valid
   V) **Modifies:** A given memory location

II) Read(String location)
   I) **Input**: String representing an address location
   II) **Output**: A string containing the content of the address location
   III) **Constraints**: None
   IV) **Assumptions**: There is a valid memory location given
   V) **Modifies:** None

III) binaryToDecimal(String binary)
   a. **Input**: A string representing a number in binary
   b. **Output**: A decimal int value of the number
   c. **Constraints**: None
   d. **Assumptions**: Input is binary
   e. **Modifies:** None

IV) decimalToBinary(int decimal)

   a. **Input**: A decimal integer
   b. **Output**: The binary representation of the number
   c. **Constraints**: None
   d. **Assumptions**: Input is decimal
   e. **Modifies:** None

V) isNext(int I)

   a. **Input**: an address location
   b. **Outsput**: a boolean describing whether the next address location exists or not
   c. **Constraints**: None
   d. **Assumptions**: Input location exists
   e. **Modifies:** None