CSCI 6461 Project Part 1

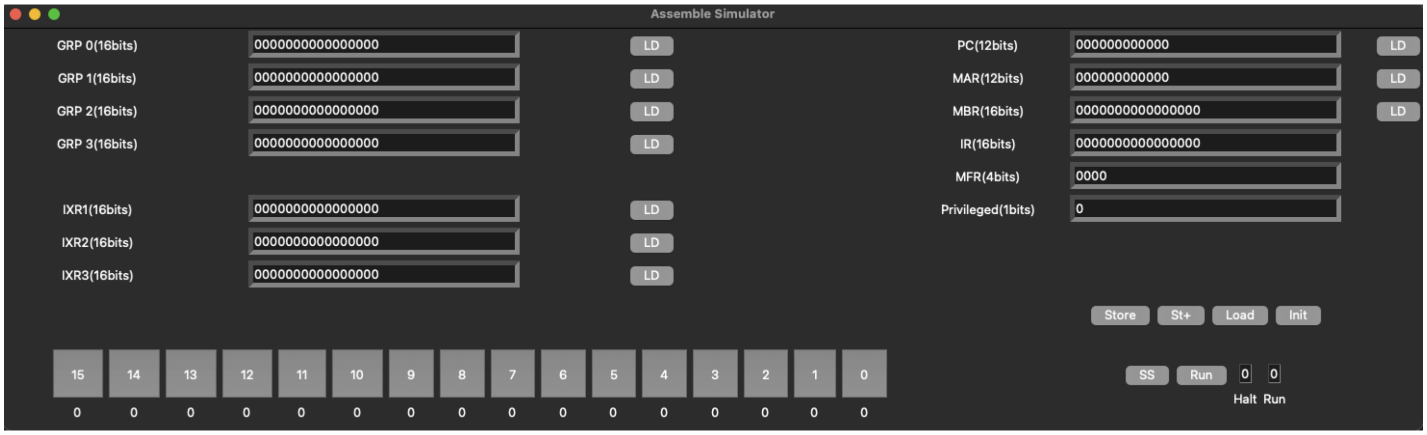
Monica Kodwani, Vishesh Javangula, Yejin Kim

Overview

Our simulator mainly has three functional modules. First, A graphical user interface (GUI) shows the layout for the operating program and interacts with the CPU module when actions activate. Second, the CPU role the arithmetic calculations for address and contents based on instructions. Lastly, the memory module occupies eligible space for running programs and manages the predefined instructions from the IPL.txt file.

Modules

1. Graphical user interface (GUI)
    a. Layout



GRP 0(16bits) 0000000000000000
GRP 1(16bits) 0000000000000000
GRP 2(16bits) 0000000000000000
GRP 3(16bits) 0000000000000000
IXR1(16bits) 0000000000000000
IXR2(16bits) 0000000000000000
IXR3(16bits) 0000000000000000
PC(12bits) 000000000000
MAR(12bits) 000000000000
MBR(16bits) 0000000000000000
IR(16bits) 0000000000000000
MFR(4bits) 0000
Privileged(1bits) 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Halt 0
Run 0

b. Library
   The user interface (UI) is built using the Python Tkinter tool

c. Main idea behind setting up a graphical user interface is to define a widget and place that widget on the grid.

d. Widgets: Frames (Organizes the grid)
   i. Frames (Organizes the grid)
      - frameswitches (frame for the 16 switches)
      - frameregister (frame for all the registers)
      - frameoperation (frame for the operation buttons - Store, St+, Load, Init)
      - framerun (SS, Run, Halt Light, Run Light)
   ii. Labels
      - Labels for all registers (GPR0-3, IXR1-3, PC, MAR, MBR, IR, MFR, Privileged, Halt, Run)
   iii. Buttons

- All LD Buttons, Store, St+, Load, Init, SS, Run, All Switches

e. Methods:
   i. Click*:
      - Users can toggle between 0 and 1 to set value for the switches
   ii. LD_*:
      - Command functions when the corresponding buttons are pushed.
      - Users can input the bits through the bottom numbers buttons (0 - 15).
      - Load numbers from user input.
   iii. reset(): Reset all registers and memory contents.
   iv. Load(): Load contents of MAR into MBR and show in MBR field on GUI.
   v. store(): Store MBR contents to memory in MAR .
   vi. storeplus(): Call store() and increases PC by 1
   vii. singlestep(): Process one instruction that the program counter is currently indicating.
   viii. run(): Run all instructions in IPL.txt file. After completion of each single step, pause for three seconds to show the intermediate result on GUI.
   ix. init(): Initialize memory and load instructions in IPL.txt file.
   x. show_general_register(general_register): Show contents of the given general register.
   xi. show_index_register(index_register): Show contents of the given index register.

2. CPU
   a. Member variables
      i. PC: Program counter to address the next instruction to be executed.
      ii. GRs: Four general purpose registers.
      iii. IndexRegisters: Three index registers to contain a base address that supports base register addressing of memory.
      iv. MAR: Memory address register to hold the address of the word to be fetched from memory.
      v. MBR: Memory buffer register to hold the word just fetched from or the word to be or last stored into memory.
      vi. MFR: Machine fault register to contain the ID code if a machine fault after it occurs.
      vii. IR: Managing instructions to store, read or decode.
      viii. memsize: Predefined memory size.
      ix. Memory: Reserve size of (memsize) memory and store the IPL.txt instructions.
   b. Methods
      i. __init__(memsize): Initialize all member variables
      ii. reset(): Reset all registered and program counter to default values when hitting HALT instruction.
      iii. single_step(operand, index_register, mode, general_register): Process one instruction that the program counter is currently indicating.

        iv. get_effective_addr(operand, index_register, mode): Calculate the effective address for every instruction.
        v. check_addr(addr): Verify the effective address is valid.
        vi. LDR(operand, index_register, mode, general_register): Load data to general register from effective address.
        vii. STR(operand, index_register, mode, general_register): store data from general register to memory.
        viii. LDA(operand, index_register, mode, general_register): load effective address into general register.
        ix. LDX(operand, index_register, mode, general_register): load data into index register.
        x. STX(operand, index_register, mode, general_register): store data from index register into memory.
        xi. HALT(): Stop the machine.

3. Memory/Register
    a. Methods for Registers
        i. set_instruction(val): Store instruction to memory.
        ii. get_instruction(): Get instruction from memory.
        iii. decode(): Decode the instruction. The following figure shows our instruction format.

Address (5bits)	I (1bits)	IX (2bits)	R (2bits)	Opcode (6bits)

    b. Methods for Memory
        i. __init__(size): Dictionary data structure holds data in corresponding address.
        ii. read_mem(fileName): Read file IPL.txt and store all instructions to memory.
4. Utils
    a. Converter
        i. hex_to_decimal(hex): Convert hexadecimal number to decimal.
        ii. decimal_to_binary(dec, bit): Convert decimal number to binary in a given number of bits.
        iii. hex_to_binary(hex): Convert hexadecimal number to binary.
        iv. binary_string_to_decimal(bin_string): Convert string type of binary to decimal number.
        v. binary_string_to_hex(bin_string): Convert string type of binary to hexadecimal number.
    b. Program Counter
        i. increment_addr(): Increase the program counter by one.