**Microprocessor Simulator in Java**

**Part IV**

**Class project CSCI 4661**

**By:**

**Abhinav Verma G40993392**

**Mahsa Chenari G24466554**

**Yuchen Tian G21284146**

**Fall 2015**

**Program Counter:**

A program counter is a register in a computer processor that contains the address (location) of the instruction being executed at the current time. As each instruction get fetched, the program counter increases its stored value by 1. After each instruction is fetched, the program counter points to the next instruction in the sequence. When the computer restarts or is reset, the program counter normally reverts to 0. In computing, a program is a specific set of ordered operations for a computer to perform. An instruction is an order given to a computer processor by a program. Within a computer, an address is a specific location in memory or storage. A register is one of a small set of data holding places that the processor uses. Some engineers refer to a program counter as an instruction address register or an address pointer. As a simple describe, the program counter holds the address of the next instruction that is to be **fetched-decoded-executed**. This will increment automatically as the current instruction is being decoded.

**Memory Address Register (MAR):**

The Memory Address Register (MAR) holds the address of the current instruction being executed. It points to the relevant location in memory where the required instruction is.

[PC] MAR (contents of Program Counter copied to the Memory Address Register)

**Memory Data Register (MDR) or Memory Buffer Register (MBR):**

The Memory Data Register can contain both instructions and data. At this stage, an instruction has been fetched and is being stored here in route to the Current Instruction Register. The instruction is copied from the memory location pointed to by the MAR.

[Memory] Addressed MBR (Contents of addressed memory is copied to the memory buffer register)

**Instruction Register (IR):**

The **Current Instruction Register** is used to store the current instruction to be **decoded and executed** (copied from the MDR).

[MBR] 🡪 (IR) (If contents of MBR is an instruction then it is copied to the Instruction Register).

**Instruction Decoder:**

Instruction decoder reads the instruction in binary format and then generates control signals for the arithmetic logical unit (ALU). There are various control signals that are generated by the instruction decoder depending upon the instruction to process.

**Decoding and executing the instruction:**

The instruction in the CIR gets decoded. In this example, the instruction is telling the processor to load the value in memory location 1000 (03) to the accumulator (one of the general purpose registers are usually used for the accumulator). As this happens, the Program Counter automatically increments.

**Arithmetic Logic Unit (ALU):**

The Arithmetic Logic Unit carries out any arithmetic and logical operations (calculations and value comparisons) required by any instruction that is executed. For example instruction at 503 would require the Arithmetic Logical Unit to add the number in location 1001 to the value already in the accumulator.

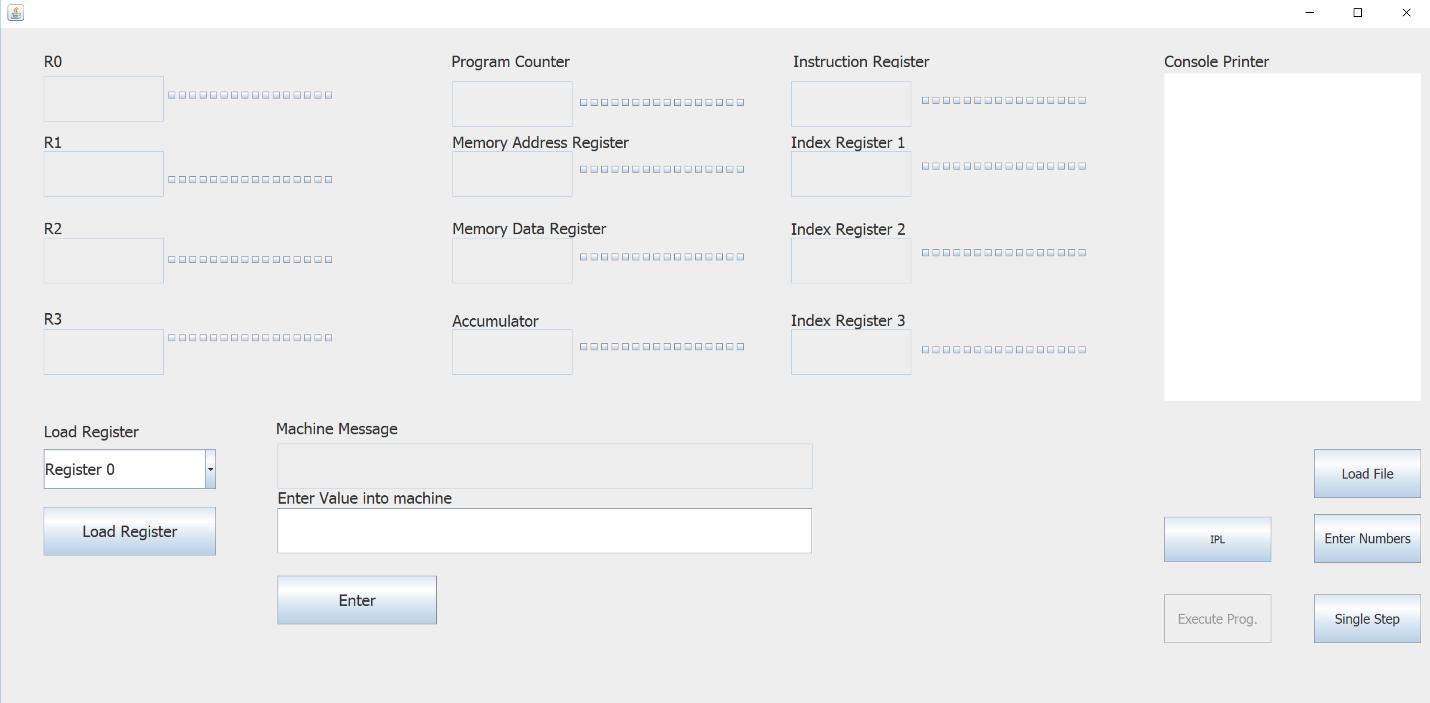
An arithmetic-logic unit (ALU) is the part of a computer [processor](http://searchcio-midmarket.techtarget.com/definition/processor) ([CPU](http://searchcio-midmarket.techtarget.com/definition/CPU)) that carries out arithmetic and logic operations on the [operand](http://whatis.techtarget.com/definition/operand)s in computer [instruction](http://searchcio-midmarket.techtarget.com/definition/instruction) [word](http://searchcio-midmarket.techtarget.com/definition/word)s. In some processors, the ALU is divided into two units, an arithmetic unit (AU) and a logic unit (LU). Some processors contain more than one AU - for example, one for *fixed-point* operations and another for floating-point operations. (In personal computers floating point operations are sometimes done by a [floating point unit](http://searchwinit.techtarget.com/definition/floating-point-unit) on a separate chip called a numeric coprocessor.)

Typically, the ALU has direct input and output access to the processor controller, main memory (random access memory or [RAM](http://searchmobilecomputing.techtarget.com/definition/RAM) in a personal computer), and input/output devices. Inputs and outputs flow along an electronic path that is called a [bus](http://searchstorage.techtarget.com/definition/bus). The input consists of an instruction word (sometimes called a machine instruction word) that contains an operation code (sometimes called an "op code"), one or more operands, and sometimes a format code. The operation code tells the ALU what operation to perform and the operands are used in the operation. (For example, two operands might be added together or compared logically.) The format may be combined with the op code and tells, for example, whether this is a fixed-point or a floating-point instruction. The output consists of a result that is placed in a storage *register* and settings that indicate whether the operation was performed successfully. (If it isn't, some sort of status will be stored in a permanent place that is sometimes called the machine status word.)

In general, the ALU includes storage places for input operands, operands that are being added, the accumulated result (stored in an *accumulator*), and shifted results. The flow of bits and the operations performed on them in the subunits of the ALU is controlled by gated circuits. The gates in these circuits are controlled by a sequence logic unit that uses a particular [algorithm](http://whatis.techtarget.com/definition/algorithm) or sequence for each operation code. In the arithmetic unit, multiplication and division are done by a series of adding or subtracting and shifting operations. There are several ways to represent negative numbers. In the logic unit, one of 16 possible logic operations can be performed - such as comparing two operands and identifying where bits don't match.

The design of the ALU is obviously a critical part of the processor and new approaches to speeding up instruction handling are continually being developed.

**Describing Our Project:**



After running the simulator:

1. Click “IPL” button:

* It loads all the values of the registers, and loads all the instructions in the instruction box for the user.
* It loads Program Counter (PC) value pointing to the first instruction.

1. Click “Enter Numbers”

* This button enters 20 numbers into the machine.

1. Click “Enter” button:

* This button enters the number in the textbox into the machine and asks the user for a next number till all the 20 numbers have been entered
* The last number is the number the user wants to compare all the numbers with.

1. “Load File”

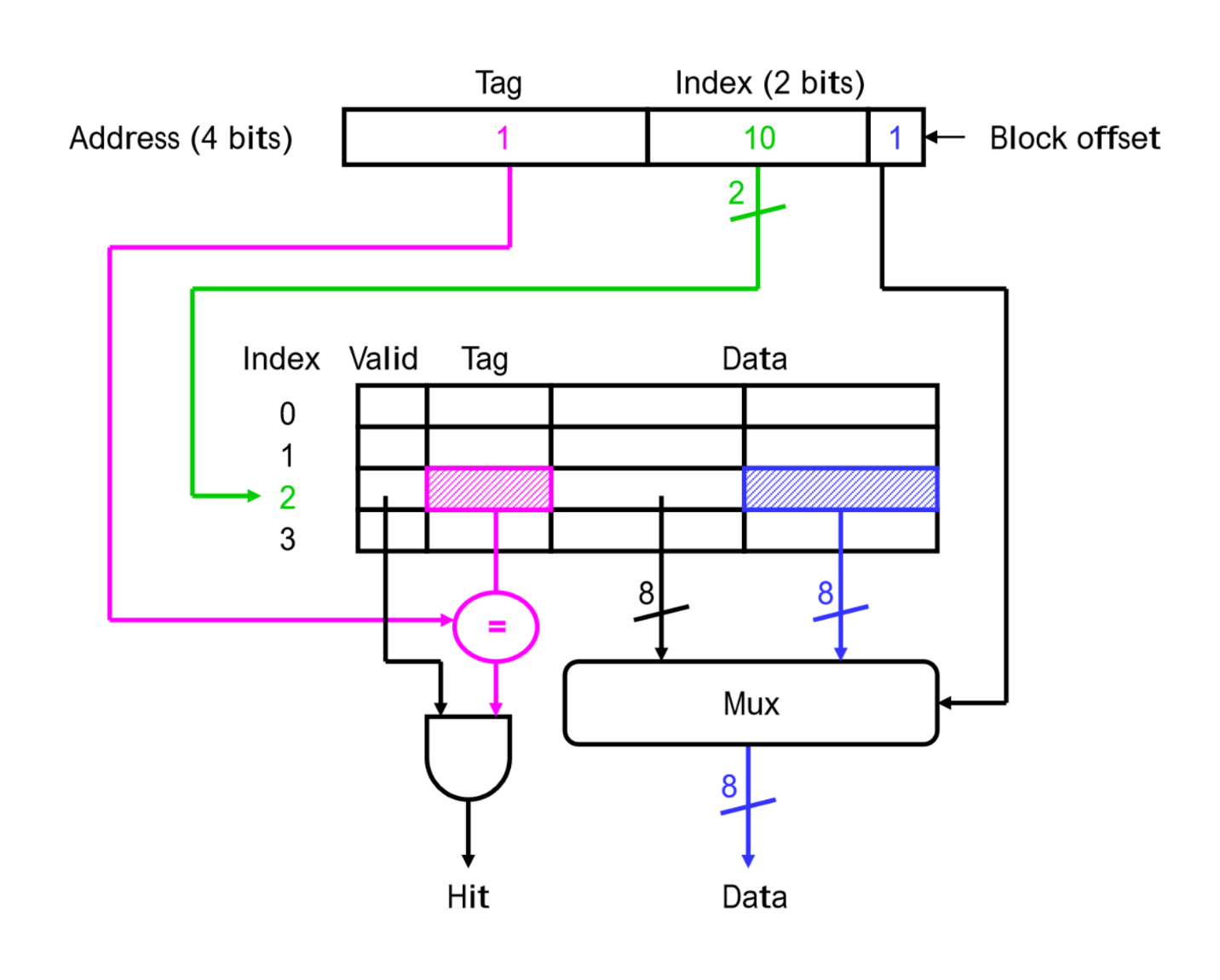
* After the number have been entered
* Load the program into the memory
* The program should be in binary format
* It fix up the instruction from the Instruction Register (IR);
* The Instruction Register (IR) sends the instructions to Instruction Decoder.
* Instruction Decoder generates control signals into the ALU.
* ALU processes information and updates the registers.

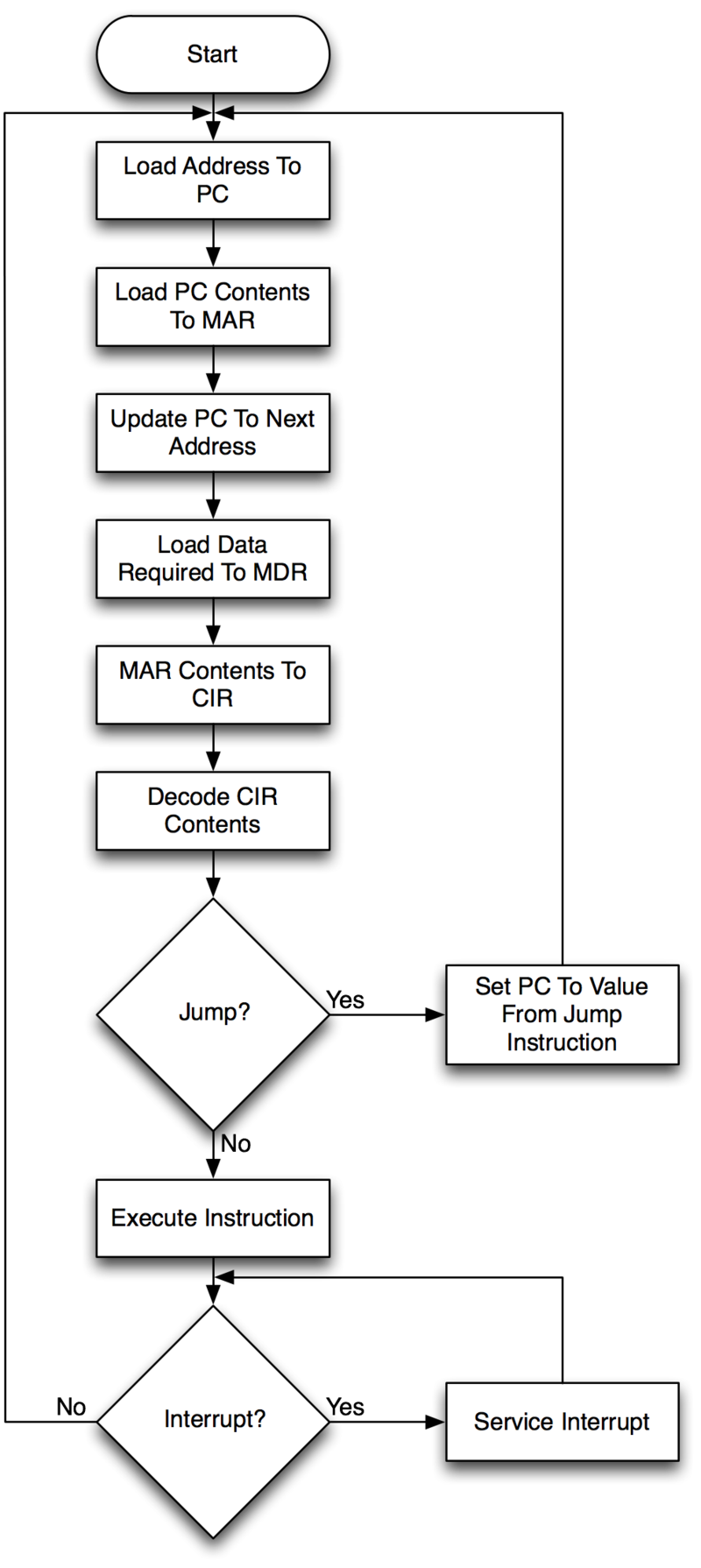
Another feature of the microprocessor is that it can load a binary value into the registers directly. This can be done by entering a valid 16 bit binary number in the ‘Load Register’ textbox. Select the register from the combo box and then press ‘Load Register’ button. Check the respective register to see the value you entered.

**Memory**

Directly Mapped Cache

Our Program implements directly mapped cache by dividing the memory into blocks s shown in the following image[[1]](#footnote-1)



Instruction Cycle

An instruction cycle (sometimes called fetch-decode-execute cycle) is the basic operation cycle of a computer. It is the process by which a computer retrieves a [program](https://en.wikipedia.org/wiki/Machine_code) [instruction](https://en.wikipedia.org/wiki/Instruction_%28computer_science%29) from its [memory](https://en.wikipedia.org/wiki/Computer_storage), determines what actions the instruction requires, and carries out those actions. This cycle is repeated continuously by the [central processing unit](https://en.wikipedia.org/wiki/Central_processing_unit) (CPU), from [boot up](https://en.wikipedia.org/wiki/Bootup) to when the computer is shut down.

In simpler CPUs, the instruction cycle is executed sequentially: each instruction is completely processed before the next one is started. In most modern CPUs, the instruction cycle is instead executed [concurrently](https://en.wikipedia.org/wiki/Concurrent_computing) in [parallel](https://en.wikipedia.org/wiki/Parallel_computing), as an [instruction pipeline](https://en.wikipedia.org/wiki/Instruction_pipeline): the next instruction starts being processed before the previous instruction is finished, which is possible because the cycle is broken up into separate steps.

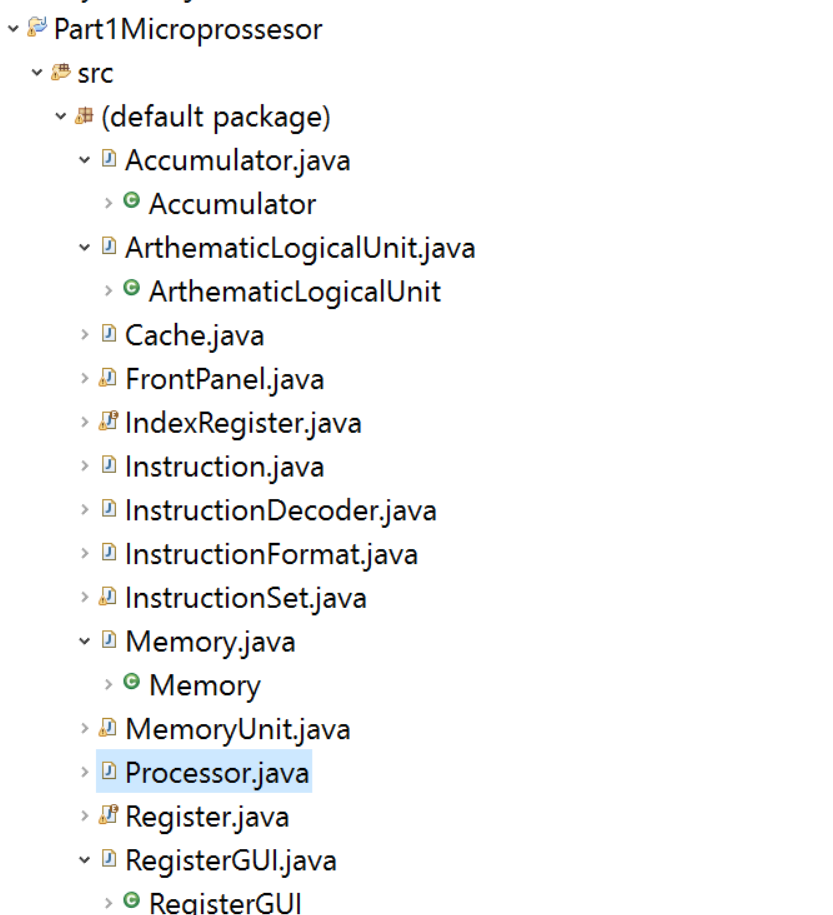
Information above are taken from: <https://en.wikipedia.org/wiki/Instruction_cycle>

Instruction Set

Our microprocessor implements all the instructions as specified by the class Project document



Program Structure



Program structure include all the necessary files:

Accumulator: it handles all the computations

ArthemeticLogicalUnit: all the instructions are executed here

Cache: It is a directly mapped cache

IndexRegister: it is used to calculate the effective address

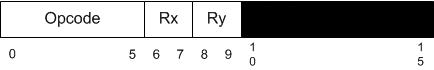
InstructionFormat: a class that uses OOPS concepts to read an instruction

RegisterGUI: it is a class that is used to help create a real word graphical image of the register

FrontPanel: Actual windows application class that contains all the elements like buttons and textbox, labels for the simulator.

InstructionDecoder: A class which reads the binary instructions

Instruction Register

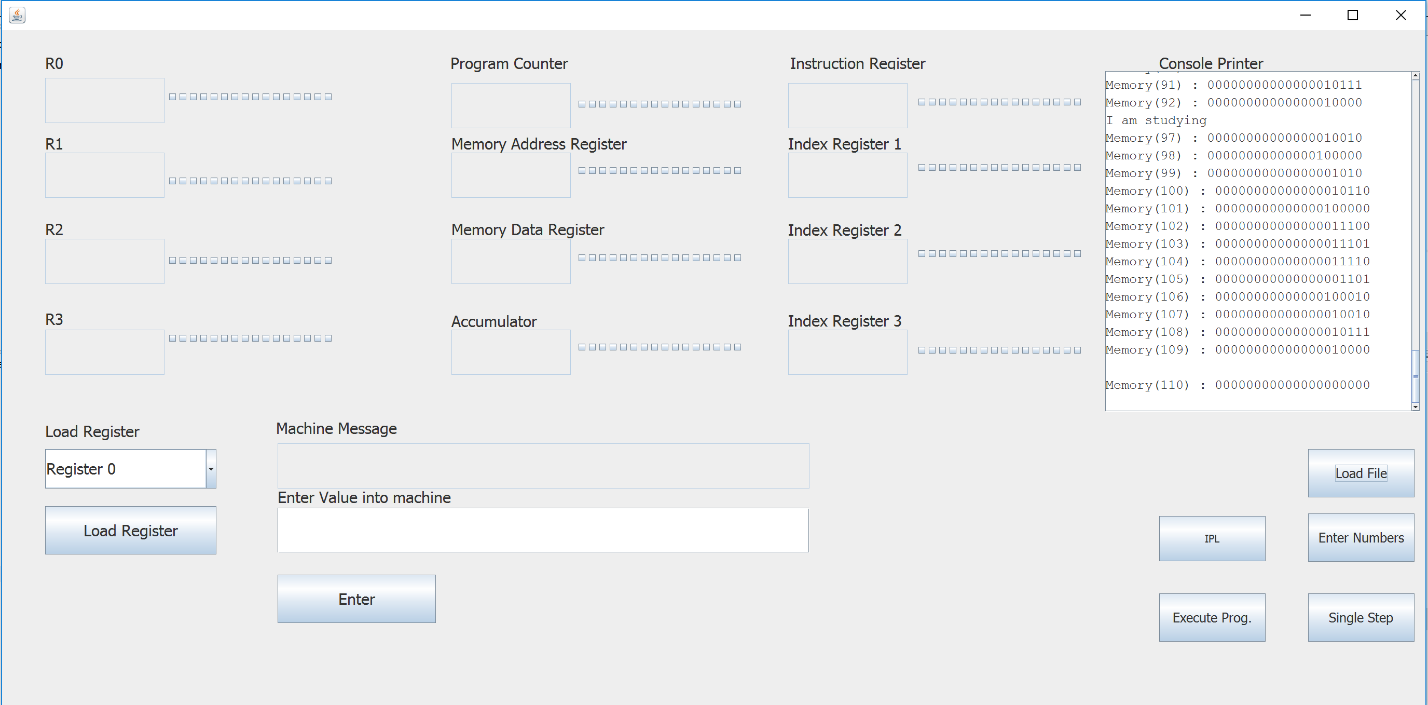


Steps to run program 2

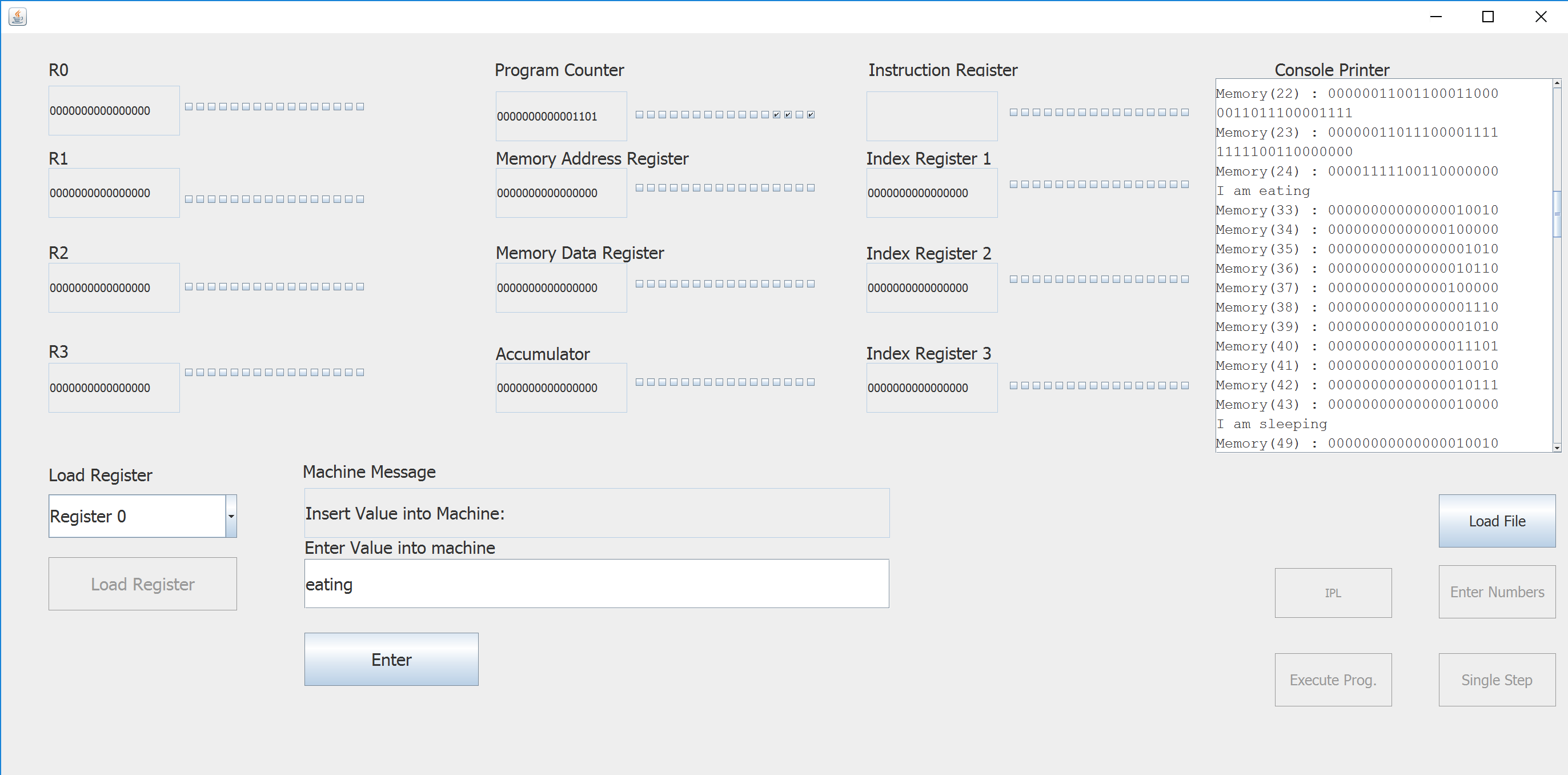
Load the file by clicking on the “Load File button”

Select the file supplied with the project

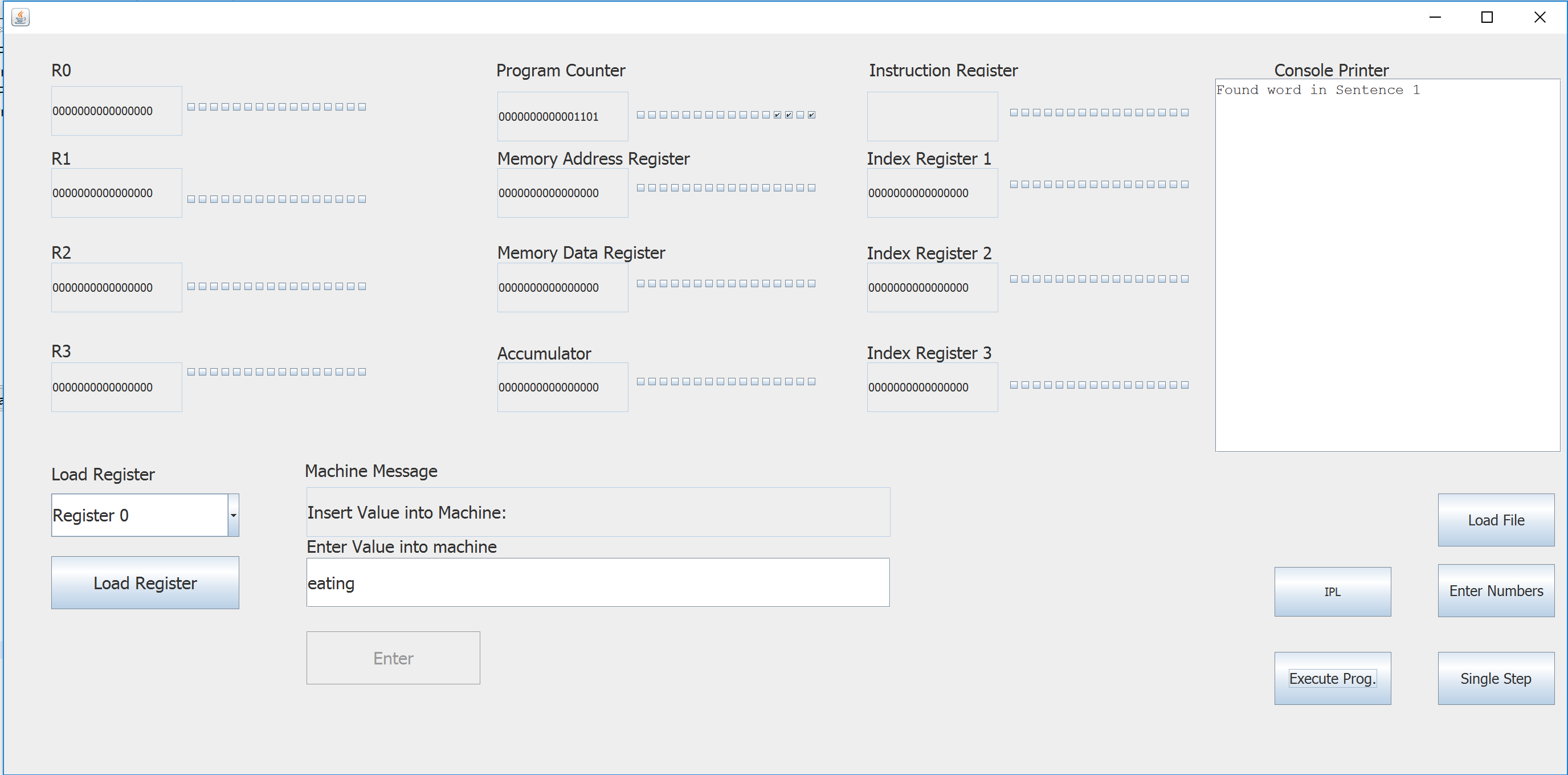
Allow the program to load



Execute the program that has been loaded in the memory by clicking on the “Execute Prog.” Button



Click on the “Enter” button



Vector Operations

Vector operations work on arrays as compared to scaler operations which operate on single scaler values.

The arrays used for vector operations is one dimensional.

Vector operations reduce the fetch and decode cycles used in single scaler value operations.

Reduced and stored latency is also reduced.

The program size is small because less number of instruction are required.

1. http://www.pitt.edu/~juy9/142/slides/L11-Cache2.pdf [↑](#footnote-ref-1)