**The LNM Institute of Information Technology, Jaipur**

**COA Laboratory (CSE216)**

**Assignment: 1**

**Programming Model-1**

Installation Link: Windows Supported Software Cpu Simulator (CPU Sim is a Java application and so requires that you have a Java runtime environment (JRE) installed on your computer)

<https://cpu-os-simulator.software.informer.com/7.5/>

# A. Introduction

## Objectives

At the end of this lab you should be able to:

* Use the CPU simulator to create basic CPU instructions
* Use the simulator to execute basic CPU instructions
* Use CPU instructions to move data to registers, compare values in registers, push data to the stack, pop data from the stack, jump to address locations and add values held in registers.
* Explain the functions of special CPU registers such as the PC, SR and SP registers.

# B. Processor (CPU) Simulators

The computer architecture tutorials are supported by simulators, which are created to underpin theoretical concepts normally covered during the lectures. The simulators provide visual and animated representation of mechanisms involved and enable the students to observe the hidden inner workings of systems, which would be difficult or impossible to do otherwise. The added advantage of using simulators is that they allow the students to experiment and explore different technological aspects of systems without having to install and configure the real systems.

## C. Basic Theory

The programming model of computer architecture defines those low‐level architectural components, which include the following

* CPU instruction set
* CPU registers
* Different ways of addressing instructions and data in instructions

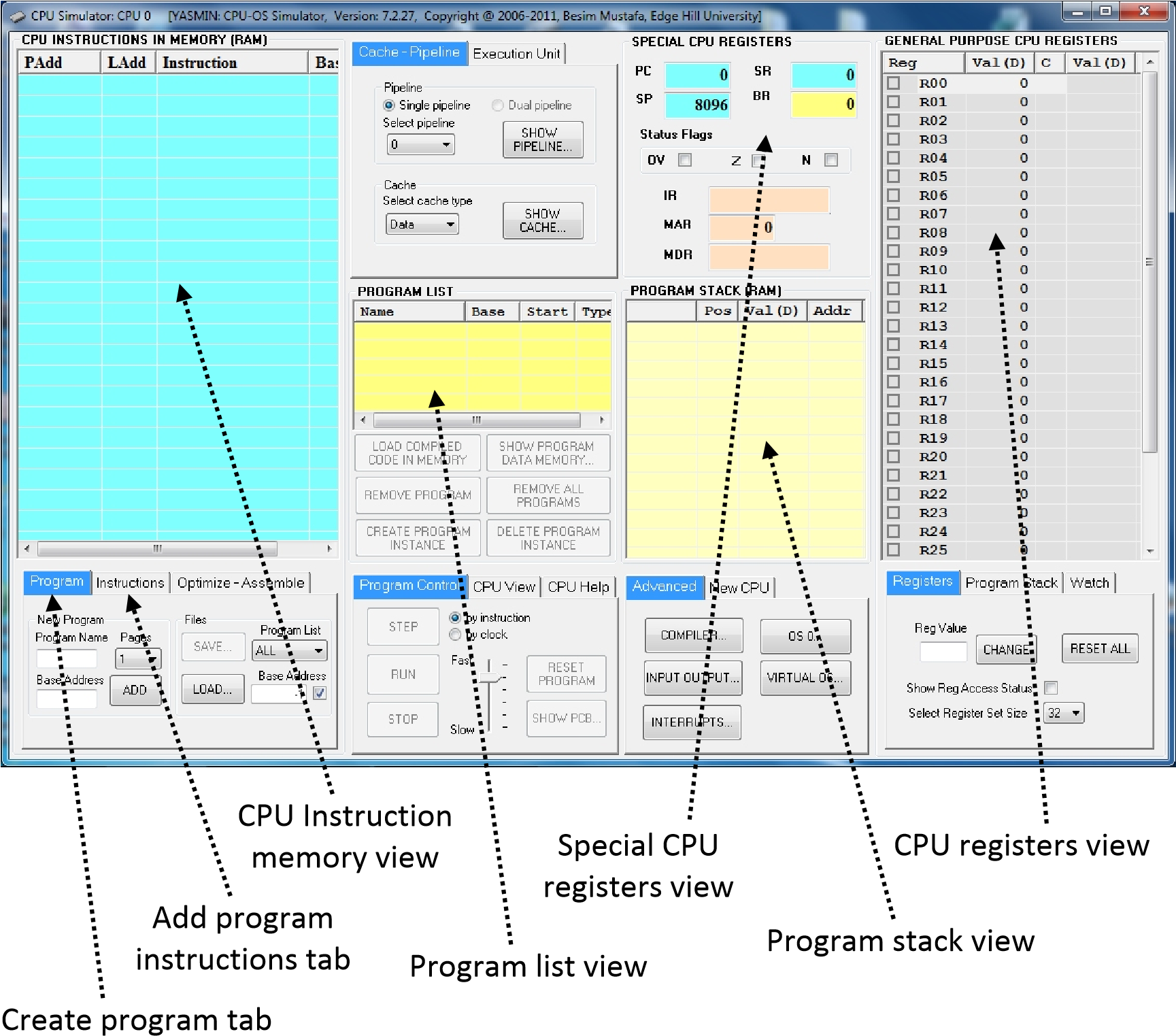
It also defines interaction between the above components. It is this low‐level programming model which makes programmed computations possible.

## D. Simulator Details

This section includes some basic information on the simulator, which should enable the students to use the simulator. The tutor(s) will be available to help anyone experiencing difficulty in using the simulator. The simulator for this lab is an application running on a PC running MS Windows operating system.

The main window is composed of several views, which represent different functional parts of the simulated processor. These are shown in Image 1 below and are composed of

* CPU Instruction memory
* Special CPU registers
* CPU (general purpose) registers
* Program stack
* Program creation and running features



### Image 1 – CPU Simulator window

The parts of the simulator relevant to this lab are described below. Please read this information carefully and try to identify the different parts on the CPU Simulator window **BEFORE** attempting the following exercises. Use this information in conjunction with the exercises that follow.

### 1. CPU instruction memory view

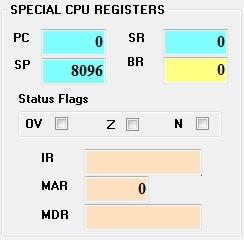
#### Image2 ‐ Instruction memory view

This view contains the program instructions. The instructions are displayed as sequences of low‐level instruction mnemonics (assembler‐level format) and not as binary code. This is done for clarity and makes code more readable by humans.

Each instruction is associated with two addresses: the physical address (**PAdd**) and the logical address (**LAdd**). This view also displays the base address (**Base**) against each instruction. The sequence of instructions belonging to the same program will have the same base address.

**2. Special CPU registers view** This view shows the set of CPU registers,

which have pre‐defined specialist functions:

**PC**: **Program Counter** contains the address of the next instruction to be executed. **IR**: **Instruction Register** contains the instruction currently being executed. **SR**: **Status Register** contains information pertaining to the result of the last executed instruction.

**SP**: **Stack Pointer** register points to the value maintained at the top of the program stack

(see below).

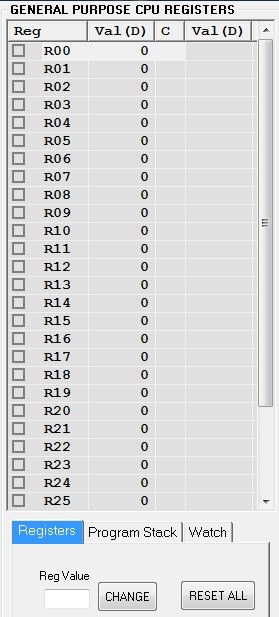
**BR**: **Base Register** contains current base address.

**MAR** : **Memory Address Register** contains the memory address currently being

**Image 3 ‐ Special CPU** accessed. **registers view Status bits: OV**: Overflow; **Z**: Zero; **N**:

Negative

### 3. CPU registers view

**Image 4 – CPU Registers view**

The register set view shows the contents of all the general‐purpose registers, which are used to maintain temporary values as the program's instructions are executed. Registers are very fast memories that hold temporary values while the CPU executes instructions.

This architecture supports from 8 to 64 registers. These registers are often used to hold values of a program's variables as defined in high‐level

languages.

Not all architectures have this many registers. Some have more (e.g. 128 register) and some others have less (e.g. 8 registers). In all cases, these

registers serve similar purposes.

This view displays each register's name (**Reg**), its current value (**Val**) and some additional values, which are reserved for program debugging. It can also be used to reset the individual register values manually which is often useful for advanced debugging. To manually change a register’s content, first select the register then enter the new value in the text box, **Reg Value**, and click on the **CHANGE** button in the **Registers** tab.

### 4. Program stack view

#### Image 5 ‐ Program stack view

The program stack is another area which maintains temporary values as the instructions are executed. The stack is a LIFO (last‐in‐first‐out) data structure. It is often used for efficient interrupt handling and sub‐routine calls. Each

program has its own individual stack.

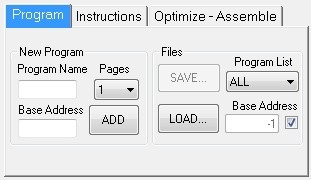
The CPU instructions PSH (push) and POP are used to store values on top of stack and pop values from top of stack respectively.

### 5. Program list view

#### Image 6 ‐ Program List View

Use the **REMOVE PROGRAM** button to remove the selected program from the list; use the **REMOVE ALL** **PROGRAMS** button to remove all the programs from the list. Note that when a program is removed, its instructions are also removed from the **Instruction Memory View** too.

### 6. Program creation

**Image 7 – Create program tab**

To create a new program enter its name in the **Program Name** box and its base address in the **Base Address** box then click on the **ADD** button. The new program’s name will appear in the Program List view (see Image 6).

#### Image 8 – Add program instructions tab

Use **ADD NEW…** button to add a new instruction; use **EDIT…** button to edit the selected instruction; use **MOVE DOWN**/ **MOVE UP** buttons to move the selected instruction down or up;

#### use INSERT ABOVE…/INSERT

**BELOW…** buttons to insert a new

instruction above or below the selected instruction respectively.

## E. Lab Exercises - Investigate and Explore the CPU Simulator

**Assignment 1:** Write a program to load the three numbers in registers, compute the average of numbers, and display the result at output port.

Find the **Appendix** for information on various CPU instructions. It also lists some of the instructions this simulator uses and also gives examples of their usage. **A word of caution**: Regularly save your code in a file in case the simulator crashes in which case you can restart the simulator and re‐load your file.

**\*\*\* End of exercises \*\*\***

**Appendix - Simulator Instruction Sub-set**

|  |  |
| --- | --- |
| **Instruction** | **Description** |
| **Data transfer instructions** | |
| MOV | Move data to register; move register to register e.g.  **MOV #2, R01** moves number 2 into register R01  **MOV R01, R03** moves contents of register R01 into register R03 |
| LDB | Load a byte from memory to register |
| LDW | Load a word (2 bytes) from memory to register |
| STB | Store a byte from register to memory |
| STW | Store a word (2 bytes) from register to memory |
| PSH | Push data to top of hardware stack (TOS); push register to TOS e.g.  **PSH #6** pushes number 6 on top of the stack  **PSH R03** pushes the contents of register R03 on top of the stack |
| POP | Pop data from top of hardware stack to register e.g.  **POP R05** pops contents of top of stack into register R05  **Note**: If you try to POP from an empty stack you will get the error message “Stack overflow”. |
| **Arithmetic instructions** | |
| ADD | Add number to register; add register to register e.g.  **ADD #3, R02** adds number 3 to contents of register R02 and stores the result in register R02.  **ADD R00, R01** adds contents of register R00 to contents of register R01 and stores the result in register R01. |
| SUB | Subtract number from register; subtract register from register |
| MUL | Multiply number with register; multiply register with register |
| DIV | Divide number with register; divide register with register |
| **Control transfer instructions** | |
| JMP | Jump to instruction address unconditionally |
|  | e.g.  **JMP 100** unconditionally jumps to address location 100 |
| JLT | Jump to instruction address if less than (after last comparison) |
| JGT | Jump to instruction address if greater than (after last comparison) |
| JEQ | Jump to instruction address if equal (after last comparison instruction) e.g.  **JEQ 200** jumps to address location 200 if the previous comparison instruction result indicates that the two numbers are equal, i.e. the Z status flag is set (the Z box will be checked in this case). |
| JNE | Jump to instruction address if not equal (after last comparison) |
| CAL | Jump to subroutine address |
| RET | Return from subroutine |
| SWI | Software interrupt (used to request OS help) |
| HLT | Halt simulation |
| **Comparison instruction** | |
| CMP | Compare number with register; compare register with register e.g.  **CMP #5, R02** compare number 5 with the contents of register R02 **CMP R01, R03** compare the contents of registers R01 and R03 Note:  If R01 = R03 then the status flag Z will be set, i.e. the Z box is checked.  If R03 > R01 then non of the status flags will be set, i.e. none of the status flag boxes are checked.  If R01 > R03 then the status flag N will be set, i.e. the N status box is checked. |
| **Input, output instructions** | |
| IN | Get input data (if available) from an external IO device |
| OUT | Output data to an external IO device |