Computer System Architecture
Programming Project Part II

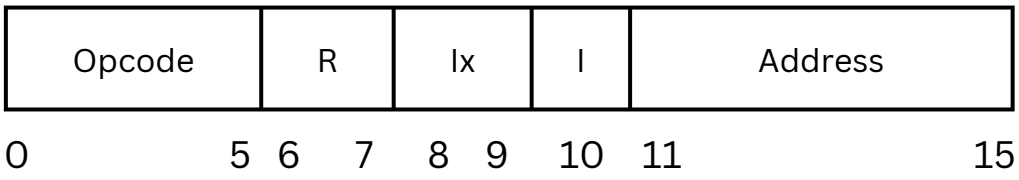
Project Description

By **Team5** :
Sitong Liu
Jahnavi Ramagiri
Reeya Gupta
Maryam Toushehjoo

Project Description

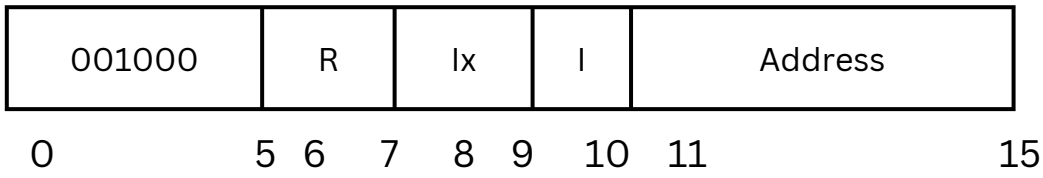
In the first part of the project we implemented the Load/Store instructions and a simple memory, for this part of the project at the beginning, we have started to code for the rest of the instructions, except for CHK, floating point/vector operations ,and trap. Then we expanded the User Interface to demonstrate the 1st program.
Starting with the instructions:

Transfer Instructions:
All of the instructions are implemented in the **Simulator.java and their opcodes in Opcodes.java.**
The transfer instructions move data between memory and the general-purpose and processor registers, processor registers, and I/O devices ,from one processor register to another ,and perform operations such as conditional moves. The conditional transfer test the values in the registers.
The illustration below shows the binary instruction code format of transfer instructions:

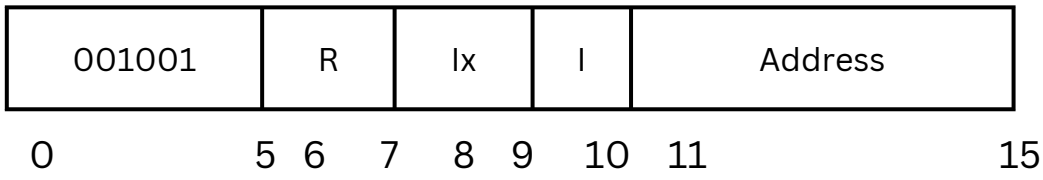


Field	Bits	Note
Opcode	6	Specifies one of 64 possible instructions; Not all may be defined in this project
R	2	Specifies the General-Purpose Register GPR (R0-R3)
IX	2	Index Register (IXR0 - IXR3)
I	1	If I =0, It is indirect addressing O.W no addressing
Addr	5	Specifies one of 32 locations

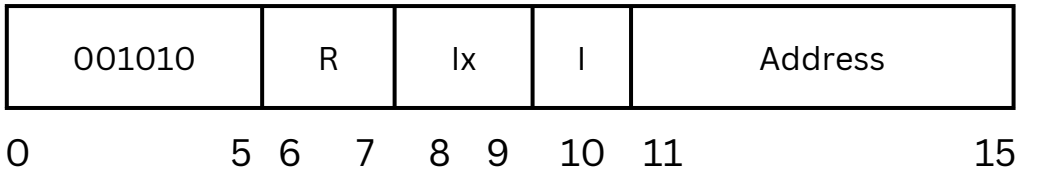
Opcode	Instruction	Description
10	JZ r, x, address[,I]	Jump If Zero: If c(r) = 0, then PC <- EA Else PC <- PC+1



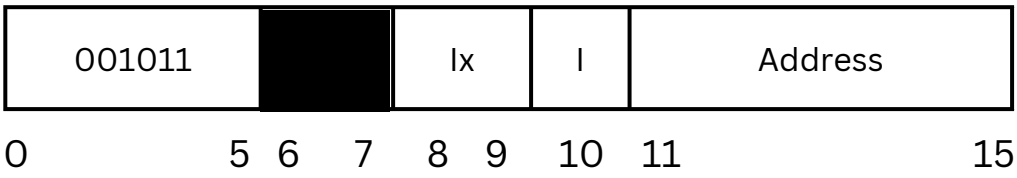
11	JNE r, x, address[,I]	Jump If Not Equal: If c(r) != 0, then PC <-- EA Else PC <- PC + 1



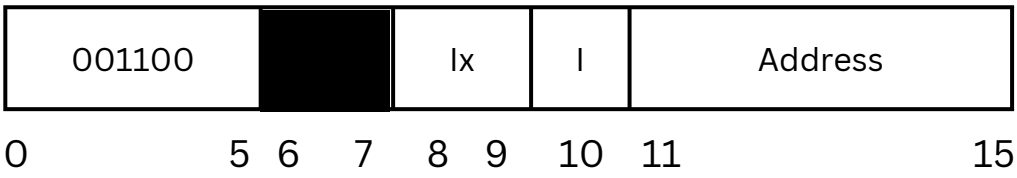
12	JCC cc, x, address[,I]	Jump If Condition Code cc **replaces** r for this instruction cc takes values 0, 1, 2, 3 0 Overflow, 1 Underflow, 2 Divzero, 3 Equalornot Register is to check; If cc bit = 1, PC <- EA Else PC <- PC + 1



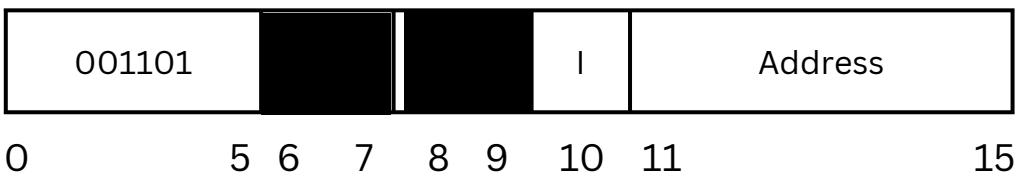
13	JMA x, address[,I]	Unconditional Jump To Address PC <- EA, Note: r is **ignored** in this instruction



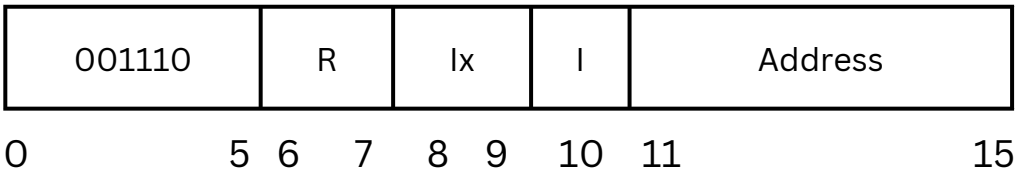
14	JSR x, address[,I]	Jump and Save Return Address: R3 <- PC+1; PC <- EA R0 should contain pointer to arguments Argument list should end with –1 (all 1s) value



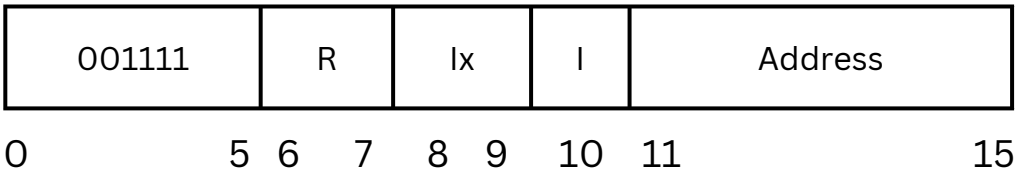
15	RFS Immed	Return From Subroutine w/ return code as Immed portion (optional) stored in the instruction’s address field. R0 <- Immed; PC <- c(R3) IX, I fields are ignored.



16	SOB r, x, address[,I]	Subtract One and Branch. R = 0..3 r <- c(r) – 1 If c(r) > 0, PC <- EA; Else PC <- PC + 1



17	JGE r,x, address[,I]	Jump Greater Than or Equal To: If c(r) >= 0, then PC <- EA Else PC <- PC + 1

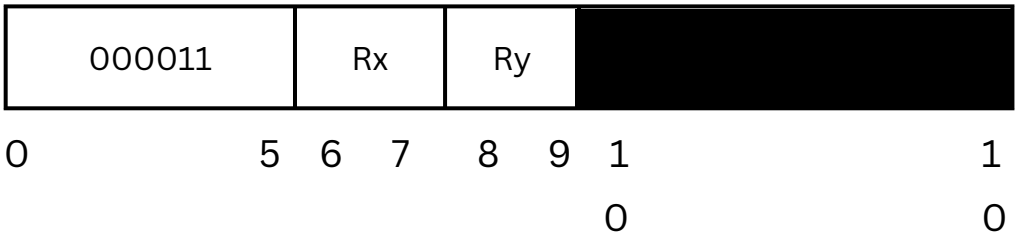


c(r) shows the content of register r.
All these transfer codes are specified in Simulator.java after the Load and Store instructions.

Arithmetic and Logical Instructions:
Logical instructions support and, or operations. Arithmetic instructions support addition, subtraction, multiplication, and division operations.

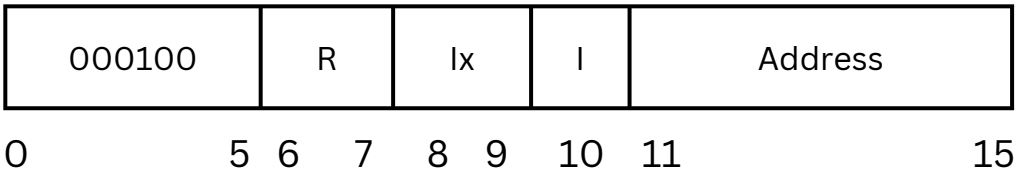
Arithmetic :

Certain arithmetic and logical instructions are register to register operations. The format of these instructions is:

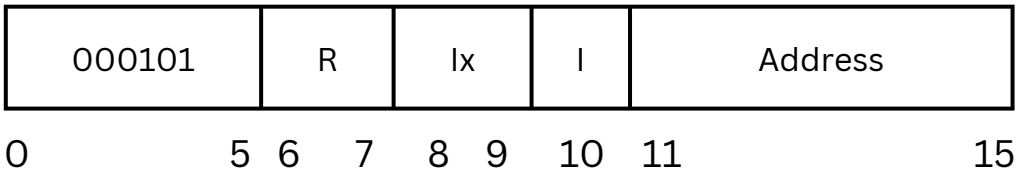


The blacked out portion means that portion of the instruction is ignored. Rx and Ry refer to one of R0-R3.

04	AMR r, x, address[,I]	Add Memory To Register, r = 0..3 r<- c(r) + c(EA)



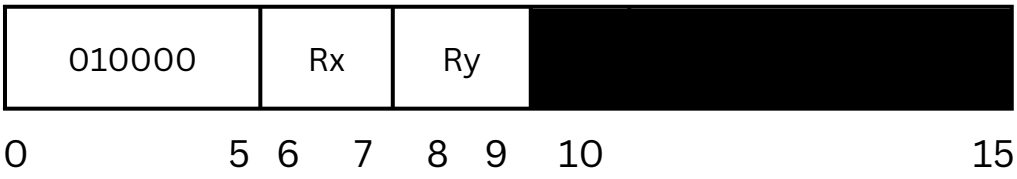
05	SMR r, x, address[,I]	Subtract Memory From Register, r = 0..3 r<- c(r) – c(EA)



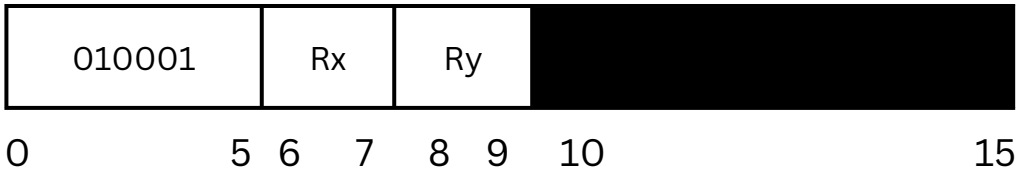
06	AIR r, immed	Add Immediate to Register, r = 0..3 r <- c(r) + Immed Note: 1. if Immed = 0, does nothing 2. if c(r) = 0, loads r with Immed IX and I are ignored in this instruction
		For immediate instructions, the Address portion is considered to be the Immediate value.

07	SIR r, immed	Subtract Immediatefrom Register, r = 0..3 r <- c(r) - Immed Note: 1. if Immed = 0, does nothing 2. if c(r) = 0, loads r1 with –(Immed) IX and I are ignored in this instruction

20	MLT rx,ry	Multiply Register by Register rx, rx+1 <- c(rx) * c(ry) rx must be 0 or 2 ry must be 0 or 2 rx contains the high order bits, rx+1 contains the low order bits of the result Set OVERFLOW flag, if overflow



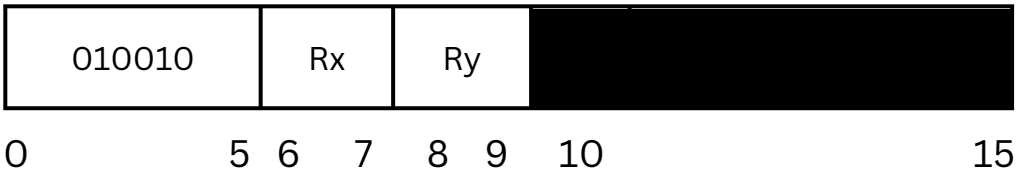
21	DVD rx,ry	Divide Register by Register rx, rx+1 <- c(rx)/ c(ry) rx must be 0 or 2 rx contains the quotient; rx+1 contains the remainder ry must be 0 or 2 If c(ry) = 0, set cc(3) to 1 (set DIVZERO flag)



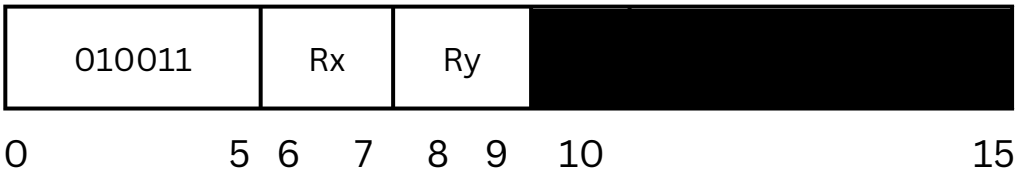
Logical :

The logical instructions perform bitwise operations.

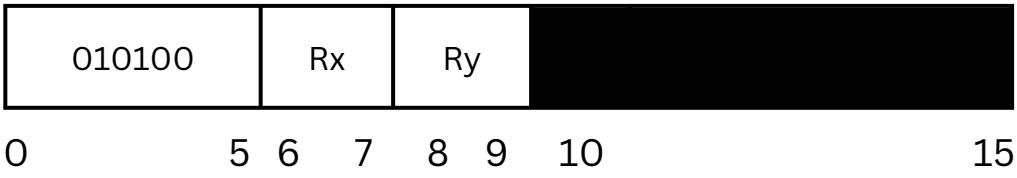
22	TRR rx, ry	Test the Equality of Register and Register If c(rx) = c(ry), set cc(4) <- 1; else, cc(4) <- 0



23	AND rx, ry	Logical And of Register and Register c(rx) <- c(rx) AND c(ry)



24	ORR rx, ry	Logical Or of Register and Register c(rx) <- c(rx) OR c(ry)

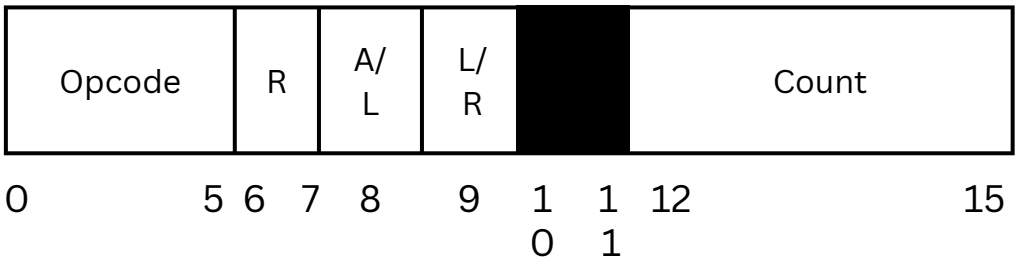


25	NOT rx	Logical Not of Register To Register C(rx) <- NOT c(rx)

Shift/Rotate Operations:

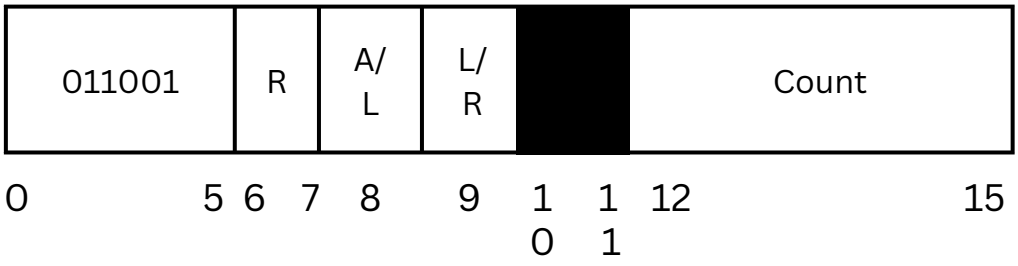
Shift and Rotate instructions control a datum in a sign in. they're used to shift the bits in the destination operand via one or more positions either to the left or proper.

The binary instruction code format of Shift and Rotate commands is as follows:

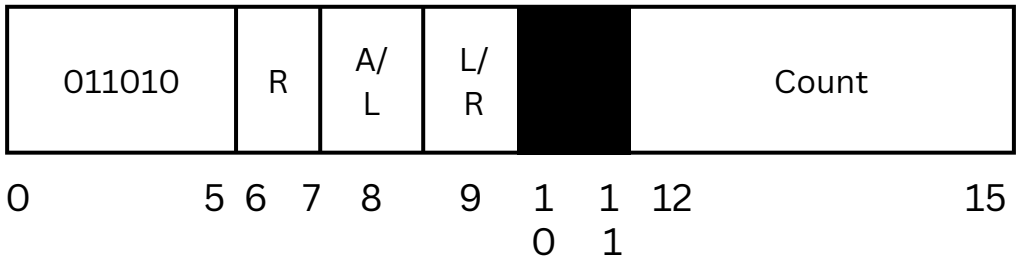


Field	Bits	Note
Opcode	6	Specifies one of 64 possible instructions; Not all may be defined in this project
R	2	Specifies the General-Purpose Register GPR (R0-R3)
A/L	2	Arithmetic Shift (A/L = 0); Logical Shift (A/L = 1)
L/R	2	Logical Rotate (L/R = 1)
Count	4	Specifies the Count for Operation

31	SRC r, count, L/R, A/L	Shift Register by Count c(r) is shifted left (L/R =1) or right (L/R = 0) either logically (A/L = 1) or arithmetically (A/L = 0) XX, XXX are ignored Count = 0...15 If Count = 0, no shift occurs



32	RRC r, count, L/R, A/L	Shift Register by Count c(r) is shifted left (L/R =1) or right (L/R = 0) either logically (A/L = 1) or arithmetically (A/L = 0) XX, XXX are ignored Count = 0...15 If Count = 0, no shift occurs



I/O Operations

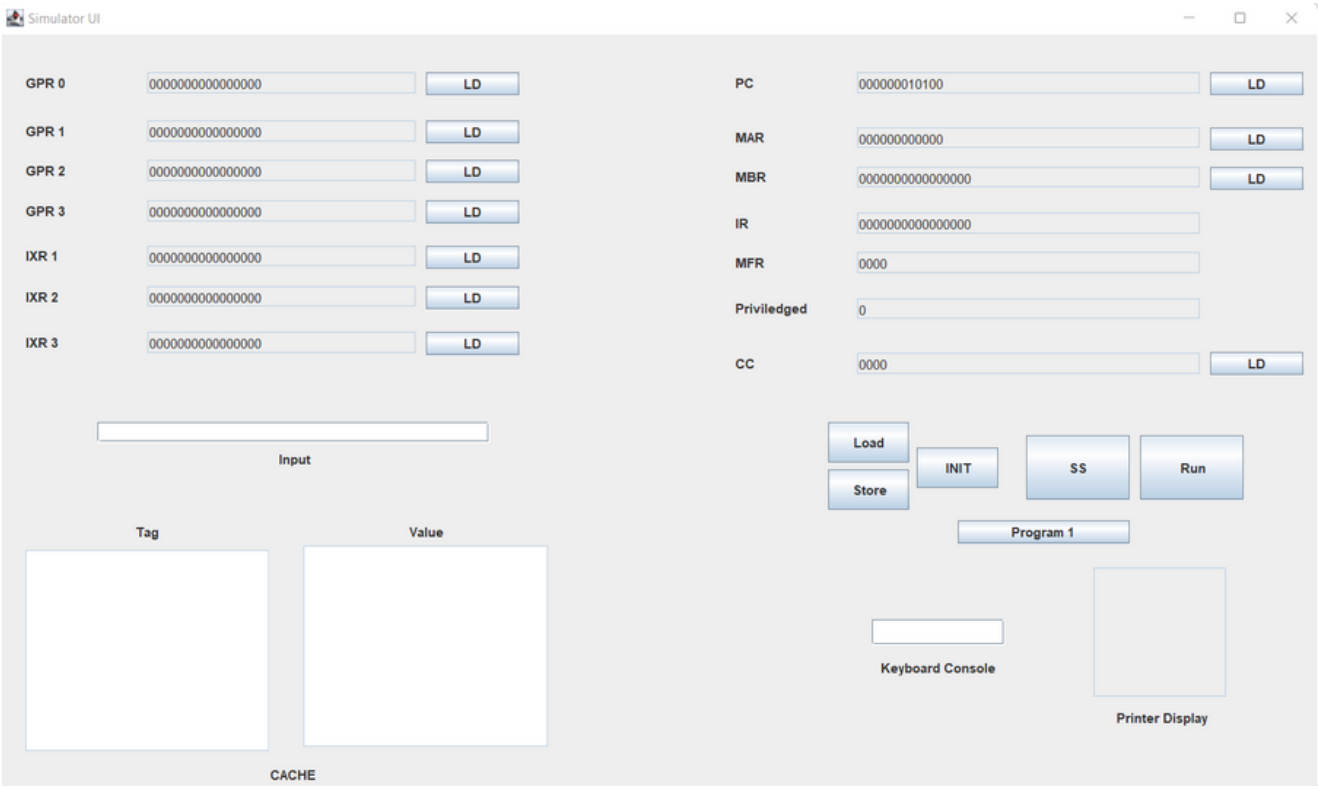
I/O operations permits the imperative gadget of the computer to talk with the outdoor environment i.e., peripheral gadgets. For man or woman I/O, the instruction layout is:

DEVID	Device
0	Consule Keyboard
1	Console Printer
2	Card Reader
3-31	Console Registers, switches, etc

61	IN r, devid	Input Character To Register from Device, r = 0..3

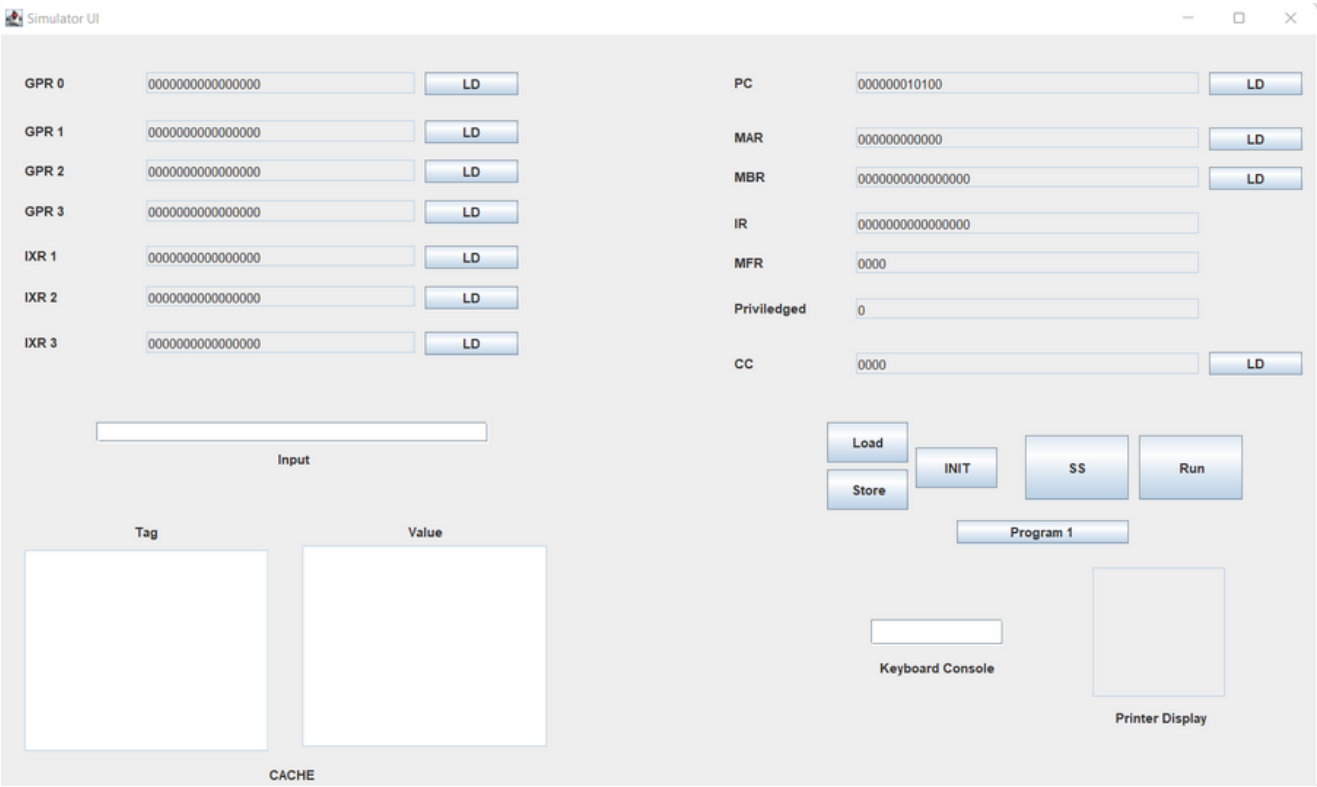
62	OUT r, devid	Output Character to Device from Register, r = 0..3

UI Display



- The **GPR0-3 are 4**, 16-bit General Purpose Registers. These can store both memory address and data.
- The **IXR**1-0 are 3, 16- bit Index Registers, they hold the current offset of a memory location. These are used for pointing to operand addresses when running the program.
- **PC: PC** is a 12-bit program counter which has the address of the next instruction to be executed from memory. It is a digital counter needed for faster execution of tasks as well as for tracking the current execution point.
- **MAR: Memory Address Register** is a 12-bit register which is used to access data and instructions from memory during the execution phase of instruction. MAR holds the memory location of data that needs to be accessed. When reading from memory, data addressed by MAR is fed into the MBR (memory buffer register).
- **MFR:** It is a 4-bit Machine Fault Register.

UI Display



- **MBR: Memory Buffer Register** is a 16-bit register which is used to store the data being transferred to and from the immediate access store. It contains the copy of designated memory locations specified by MAR. It acts as a buffer allowing the processor and memory units to act independently without being affected by minor differences in operation.
- **IR: IR** is a 16- bit register that holds the instruction currently being executed or decoded. Each instruction to be executed is loaded into the instruction register, which holds it while it is decoded, prepared and ultimately executed, which can take several steps.
- **CC:** The overflow and divided by zero flag will be shown in the CC
- **SS: The SS** is the Single Step button, it is used to execute the one step at a time in order to determine functioning.
- **Run:** This button is used to execute all of the instructions specified in the input and produces the final output.
- **INIT:** Initializing PC, Instruction, and memory
- **Program1**: This button is used to run the program 1