Instruction	Syntax	Semantics
Set	`set <reg0> <imm0>`	Put the immediate value <imm0> in register <reg0>
Add	`add <reg0> <reg1> <reg2>`	Add the values in <reg1> and <reg2> together and store the result in <reg0>
Subtract	`sub <reg0> <reg1> <reg2>`	Subtract the value in <reg2> from the value in <reg1> and store the result in <reg0>
Multiply	`mul <reg0> <reg1> <reg2>`	Multiply the values in <reg1> and <reg2> together and store the result in <reg0>
Quotient	`quot <reg0> <reg1> <reg2>`	Divide the value in <reg1> by the value in <reg2> and store the quotient in <reg0>
Remainder	`rem <reg0> <reg1> <reg2>`	Divide the value in <reg1> by the value in <reg2> and store the remainder in <reg0>
Jump	`jmp <imm0>`	If the current instruction index (in the LR) is n, the next instruction to execute is at index n + imm0
Branch if equal	`beq <reg0> <reg1> <imm0>`	If the value in <reg0> is equal to the value in <reg1>, the next instruction to execute is at index n + imm0, otherwise it is at n + 1
Branch if not equal	`bne <reg0> <reg1> <imm0>`	If the value in <reg0> is not equal to the value in <reg1>, the next instruction to execute is at index n + imm0, otherwise it is at n + 1
Branch if greater or equal	`bge <reg0> <reg1> <imm0>`	If the value in <reg0> is greater than or equal to the value in <reg1>, the next instruction to execute is at index n + imm0, otherwise it is at n + 1
Branch if less or equal	`ble <reg0> <reg1> <imm0>`	If the value in <reg0> is less than or equal to the value in <reg1>, the next instruction to execute is at index n + imm0, otherwise it is at n + 1
Load from shared memory	`load <reg0> @input[<reg1>]`	Load the value at index "value of <reg1>" in the input array and store the value in <reg0>
Store to shared memory	`store <reg0> @output[<reg1>]`	Store the value of <reg0> in the output array at index "value of <reg1>"