Instruction	Syntax	Semantics
Assign	`%name = assign() : <i32>`	Put the immediate value <i32> in variable %name
Add	`%result = add(%op1, %op2)`	Add the values of variables %op1 and %op2 and store the result in variable %result
Subtract	`%result = sub(%op1, %op2)`	Subtract the value of variable %op2 from the value of variable %op1 and store the result in variable %result
Multiply constant	`%result = mult_const(%op) : <i32>`	Multiply the value of variable %op with the immediate <i32> and store the result in %result
Bit-conditional multiply constant	`%result = bcond_mult_const(%op1, %op2) : <i32>`	• If %op2 is 0, do nothing • If %op2 is 1, multiply the value of variable %op1 with the immediate <i32> and store the result in %result • If %op2 is any other value, Undefined Behavior
Jump	`jump() : #name`	Jump to the Block with name #name
Branch if equal	`beq(%op1, %op2) : #name`	Jump to the Block with name #name only if the values of variables %op1 and %op2 are equal
Branch if not equal	`bne(%op1, %op2) : #name`	Jump to the Block with name #name only if the values of variables %op1 and %op2 are not equal
Branch if greather than	`bgt(%op1, %op2) : #name`	Jump to the Block with name #name only if the value of variable %op1 is greater than the value of variable %op2
Branch if less than	`blt(%op1, %op2) : #name`	Jump to the Block with name #name only if the value of variable %op1 is less than the value of variable %op2
Send message	`send_msg(%csck, %op)`	Send the value in variable %op over the classical socket to another node
Receive message	`%msg = recv_msg(%csck)`	Block and wait until a message is received over the classical socket and store its value in the variable %msg
Run Local Routine	`%result = run_routine(%args) : #name`	Run the Local Routine named #name with arguments %args and store results in %result
Run Request Routine	`%result = run_request(%args) : #name`	Run the Request Routine named #name with arguments %args and store result in %result
Submit Routines	`submit_routines(%args) : [#name]`	Submit a batch of Local and/or Request Routines with names [#name] for execution, with arguments %args
Join on Routine results	`%results = join_routines() : [#name]`	Block until the results are available for all Local and/or Request Routines with names [#name] and store the results in %results