**INSTRUCTION SET**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mnemonic | ID | Instruction | Arguments | Action |
| HALT | 00 | Nebula will exit |  | System.exit(0); |
|  |  |  |  |  |
| STACK COMMANDS |  |  |  |  |
| PUSH @Value | 01 | Pushes @Value to the top of the stack. | @Value | Stack[sp] = @Value |
| POP | 02 | Pops the top value off of the stack. |  | Stack[sp--] = 0; |
| CLR | 03 | Clear all values from the stack, revert to 0 |  | For I = 0; I < stack.length; i++{  Stack[i] = 0  } |
|  |  |  |  |  |
| REGISTER COMMANDS |  |  |  |  |
| RPOP @Addr | 04 | Pops the top value of the ‘Stack’ and places it at *register[Addr]* | @Addr (Register Address) | Register[addr] = stack[sp]  Stack[sp--] = 0 |
| RCOP @Addr | 05 | Copys top value of the ‘Stack’ and places it at *register[Addr]*. | @Addr (Register Address) | Register[addr] = stack[sp] |
| RPUSH @Value @Addr | 06 | Places @Value on the register at *Addr (register[Addr])* | @Value (Integer)  @Addr (Register Address) | Register[addr] = @Value; |
|  |  |  |  |  |
| ARITHMETIC COMMANDS |  |  |  |  |
| DIV | 07 | Divide value at top of stack, and return it to the same point. | Stack[sp] & Stack[sp-1] | Stack[sp-1] = Stack[sp]/Stack[sp-1] |
| MLT | 08 | Multiply value at top of stack, and return it to the same point. | Stack[sp] & Stack[sp-1] | Stack[sp-1] = Stack[sp]\*Stack[sp-1]  Sp--; |
| ADD | 09 | Add value the two top values on the stack. | Stack[sp] & Stack[sp-1] | Stack[sp-1] = Stack[sp]+Stack[sp-1]  Sp--; |
| SUB | 10 | Subtracts the value at the top of the stack, from the second’s value on the stack. | Stack[sp] & Stack[sp-1] | Stack[sp-1] = Stack[sp]-Stack[sp-1]  Sp--; |
|  |  |  |  |  |
| IO COMMANDS |  |  |  |  |
| CPRNT | 11 | Prints ASCI Letter represented by the value at top of the Stack | Stack[sp] | Syso((char) stack[sp]) |
| IPRNT | 12 | Prints the Integer value at the top of the Stack. | Stack[sp] | Syso(stack[sp]`1) |
|  |  |  |  |  |
| FLOW COMMANDS |  |  |  |  |
| GOTO @Line | 13 | Jump to @Line in the program. | @Line (Integer) | pp = @Line; |
| GIE @Line | 14 | GOTO If equal | @Line @Value1 stack[sp] @Value2 stack[sp-1] | If(stack[sp] == stack[sp-1]{  pp = @line  } |
| GIL @Line | 15 | GOTO If less than | @Line @Value1 stack[sp] @Value2 stack[sp-1] | If(stack[sp] < stack[sp-1]{  pp = @line  } |
| GIG @Line | 16 | GOTO If greater than | @Line @Value1 stack[sp] @Value2 stack[sp-1] | If(stack[sp] > stack[sp-1]{  pp = @line  } |
|  |  |  |  |  |