| | Banano CPU Hex Instructions | | | | | Notes |
|---|---|---|---|---|---|---|
| | Instruction Hex Code | *Arguments* | ASM Instruction | Psuedo Code | Summary | "REGID" is simply a shorthand for a single value from 0-5 representing each register. **(The register IDs correspond to the following, in order: [Program Counter, A, B, C, D, X, Y, Z, E, F, G, H, I, J])** Arguments are leading values in memory, these get used and skipped when executing an instruction. |
| | 0x0001 | *N\A* | hlt | Halt | Stops program counter from ticking | |
| | 0x1010 | *X* | goto | Jump to X | Jumps program pointer to address at literal | |
| | 0x1011 | *REGID* | goto | Jump to REG | Jumps program pointer to address at register value | |
| Storing Data | 0x0010 | *REGID, X* | ldr | Reg = X | Stores literal into register | |
| | 0x0011 | *REGID, REGID* | ldr | Reg1 = Reg2 | Stores value of register into another register | |
| | 0x0110 | *REGID, X* | sto | Mem[Reg] = X | Stores literal into memory at register | |
| | 0x0111 | *REGID, REGID* | sto | Mem[Reg1] = Reg2 | Stores value of register into memory at register | |
| | 0x0210 | *X, Y* | sto | Mem[X] = Y | Stores literal into memory at literal | |
| | 0x0211 | *X, REGID* | sto | Mem[X] = Reg | Stores value of register into memory at literal | |
| Arithematic | 0x0a00 | *OPID, X* | calc | RegC = OP(RegC, X) | Computes register C and literal into register C | |
| | 0x0a01 | *OPID, REGID* | calc | RegC = OP(RegC, Reg) | Computes register C and register into register C | |
| | | *N\A* | | | | |
| | | *N\A* | | | | |
| | 0x1a00 | *REGID* | inc | Reg++ | Increments register | |
| | 0x1a10 | *REGID* | dec | Reg-- | Decrements register | |
| | 0x0ab0 | *X* | push | Prg.push(X) | Pushes literal into program stack | |
| | 0x0ab1 | *REGID* | push | Prg.push(Reg) | Pushes register value into program stack | |
| | 0x1ab0 | *REGID* | pop | Reg = Prg.pop() | Pops top of program stack onto register | |
| | 0xfab0 | *N\A* | rev | Reverse PRG | Reverses the program stack | |
| | 0x0b10 | *X* | gosub | Subroutine at X | Starts a subroutine at X, adds position to callstack | |
| | 0x0b11 | *REGID* | gosub | Subroutine at Reg | Starts a subroutine at reg., adds position to callstack | |
| | 0x1001 | *N\A* | ret | Return | Resumes program pointer from position in callstack | |
| | 0x0c00 | *REGID, X* | jmc | Jump X if REG != 0 | Jumps to X if register value is not equal to 0 | |
| | 0x0c10 | *REGID, X* | jmz | Jump X if REG == 0 | Jumps to X if register value is equal to 0 | |
| | 0x1210 | *REGID, X* | lod | Reg = Mem[X] | Loads memory from X into a register | |
| | 0x1211 | *REGID, REGID* | lod | Reg1 = Mem[Reg2] | Loads memory from register into a register | |
| | 0xfa00 | *N\A* | syscall | IoPort() | Sends a request through IoPort - Awaits it to finish. | |
| | 0x2ab0 | *X* | jms | Jump if Prg > 0 | Jumps to X if program stack has any items | |
| | 0x2ab1 | *X* | jsz | Jump if Prg == 0 | Jumps to X if program stack has no items | |
| | | *N\A* | | | | |
| | | *N\A* | | | | |

| CPU CALC Instructions (Operations) | |
|---|---|
| OPID | *Operation* |
| 0x0000 | *A + B* |
| 0x0001 | *A - B* |
| 0x0002 | *A * B* |
| 0x0003 | *A / B* |
| 0x0004 | *A && B* |
| 0x0005 | *A \|\| B* |
| 0x0006 | *A XOR B* |
| 0x0007 | *A >> B* |
| 0x0008 | *A << B* |
| 0x0009 | *A > B* |
| 0x000a | *A < B* |
| 0x000b | *A == B* |
| 0x000c | *A != B* |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |