Bytecode

This document specifies the Bytecode used in the compiler. The compiler is a stack based system, with a program counter and a branch save register ; the branch instruction always copies the old program counter (address of next instruction) to branch save ; subroutines should push this on the data stack immediately.

|  |  |  |
| --- | --- | --- |
| **Byte Value(s)** | **Mnemonic/Usage** | **Notes** |
|  | 422 <n> | Push the given numeric constant on the stack. 422 is an example, it can be any numeric constant. All values are 32 bit signed integers. |
|  | <addr> @ <data> | Replace TOS with the 32 bit signed integer at (TOS). Must be on a four-byte boundary. Causes an exception if outside memory block. |
|  | <addr> C@ <data> | Replace TOS with the 8 bit integer at TOS. This reads non-sign extended, e.g. byte value $90 in memory returns 144 irrespective of anything else. Causes an exception if outside memory block. |
|  | <data> <addr> ! | Write the 32 bit signed integer TOS2 to the address TOS. Must be on a four byte boundary. Both values are lost. Causes an exception if outside memory block. |
|  | <data> <addr> C! | Write the 8 bit integer TOS2 to the address TOS. Both values are lost. The value written to memory is simply truncated. Causes an exception if outside memory block. |
|  | <n1> NEG <n2> | Negate the value at the top of the stack. |
|  | <n1> NOT <n2> | One’s complement (e.g. invert all bits) of the value at the top of the stack. |
|  | <n1> <n2> + <n3> | Calculate n1 + n2 and replace those values with the result. Overflow ignored. |
|  | <n1> <n2> - <n3> | Calculate n1 - n2 and replace those values with the result. Overflow ignored. |
|  | <n1> <n2> \* <n3> | Calculate n1 \* n2 and replace those values with the result. Overflow ignored. |
|  | <n1> <n2> / <n3> | Calculate n1 / n2 and replace those values with the result. Division by zero causes a logged error. |
|  | <n1> <n2> MOD <n3> | Calculate n1 mod n2 and replace those values with the result. Division by zero causes a logged error. |
|  | <n1> <n2> BITAND <n3> | Calculate the bitwise and of n1 and n2 and replace those values with the result |
|  | <n1> <n2> BITOR <n3> | Calculate the bitwise or of n1 and n2 and replace those values with the result |
|  | <n1> <n2> BITXOR <n3> | Calculate the bitwise xor of n1 and n2 and replace those values with the result |
|  | <n1> <n2> == <n3> | Compare n1 and n2. If they are equal, push -1 on the stack, otherwise push zero. N1 and n2 are lost. |
|  | <n1> <n2> > <n3> | Compare n1 and n2. If n1 is greater than n2 push -1 on the stack, otherwise push zero. N1 and n2 are lost. This comparison is a *signed* one, hence 0x00000001 (1) is greater than 0xFFFFFFFF (-1). |
|  | <n1> <n2> < <n3> | Compare n1 and n2. If n1 is less than n2 push -1 on the stack, otherwise push zero. N1 and n2 are lost. This comparison is a *signed* one, hence 0x00000001 (1) is less than 0xFFFFFFFF (-1). |
|  | $$(Function) | Call a system function, which may or may not return a value. See list of system functions later on. |

System Functions

|  |  |  |
| --- | --- | --- |
| **Number** | **Name** | **Function** |
| 0 | LEN | Calculate the length of the ASCIIZ string at address TOS. Crashes if bad address or off the end of memory. |
| 1 | ABS | Set TOS to the absolute value of TOS |
| 2 | RND | Set TOS to a random value 1 – TOS ; if TOS < 1 then 32 bit random integer |
| 3 | SGN | Set TOS to the sign -1,0,1 of TOS |
| 4 | MEM | Allocate TOS bytes of memory, crashes if bad amount. |
| 5 | PADX | Return Joypad X, -1 0 1 in TOS, parameter must be zero |
| 6 | PADY | Return Joypad Y, -1 0 1 in TOS, parameter must be zero |
| 7 | FIRE | Return state of fire button (-1 or 0) for A (TOS = 0) B(TOS = 1), all other values return 0. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |