The N.I.G.E. Machine

# 1. CPU instruction set reference

|  |  |  |  |
| --- | --- | --- | --- |
| **Assembler mnemonic** | **Instruction length (bytes)** | **Encoding** | **Duration (cycles)** |
| **Description** | | **Parameter stack effect** ( before -- after)  ( 3rd 2nd 1st on stack --) | **Return stack effect** |

|  |  |  |  |
| --- | --- | --- | --- |
| **NOP** | 1 byte | 0x00 | 1 cycle |
| No operation | | ( --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **DROP** | 1 byte | 0x01 | 1 cycle |
| Remove top item from parameter stack | | ( x --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **DUP** | 1 byte | 0x02 | 1 cycle |
| Duplicate the top stack item | | ( x -- x x) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **?DUP** | 1 byte | 0x03 | 2 cycles |
| Duplicate the top stack item only if non zero | | (x -- x x | x) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **SWAP** | 1 byte | 0x04 | 1 cycle |
| Exchange the two top stack items | | ( x y -- y x) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **OVER** | 1 byte | 0x05 | 1 cycle |
| Make a copy of the second item on the stack | | ( x y -- x y x) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **NIP** | 1 byte | 0x06 | 1 cycle |
| Dispose of the second item on the stack | | ( x y -- y) | ( --) |
| **ROT** | 1 byte | 0x07 | 1 cycle |
| Rotate the top three stack times so that the second item becomes top | | (x y z -- z x y) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **>R** | 1 byte | 0x08 | 1 cycle |
| Remove the top item from the parameter stack and place it on the return stack | | ( x --) | ( -- x) |

|  |  |  |  |
| --- | --- | --- | --- |
| **R@** | 1 byte | 0x09 | 1 cycle |
| Copy the top item from the return stack to the parameter stack | | ( -- x) | ( x -- x) |

|  |  |  |  |
| --- | --- | --- | --- |
| **R>** | 1 byte | 0x0A | 1 cycle |
| Remove the top item from the return stack and place it on the parameter stack | | ( -- x) | ( x --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **PSP@** | 1 byte | 0x0B | 1 cycle |
| Load the parameter stack with the current value of the parameter stack pointer. The stack pointer is the count of items currently on the stack and also directs the CPU datapath to the first stack item held in SRAM | | ( -- PSP) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **RSP@** | 1 byte | 0x0C | 1 cycle |
| Load the parameter stack with the current value of the return stack pointer. The stack pointer is the count of items currently on the stack and also directs the CPU datapath to the first stack item held in SRAM | | ( -- RSP) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **PSP!** | 1 byte | 0x0D | 1 cycle |
| Save the top item from the stack as the current parameter stack pointer | | ( PSP --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **RSP!** | 1 byte | 0x0E | 1 cycle |
| Save the top item from the stack as the current return stack pointer | | ( RSP --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **+** | 1 byte | 0x0F | 1 cycle |
| Add two 32 bit integer numbers. x3 = x1 + x2 | | ( x1 x2 -- x3) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **-** | 1 byte | 0x10 | 1 cycle |
| Subtract two 32 bit integer numbers.   x3 = x1 - x2 | | ( x1 x2 -- x3) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **NEGATE** | 1 byte | 0x11 | 1 cycle |
| Negate a 32 bit integer in two’s complement format | | (x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **1+** | 1 byte | 0x12 | 1 cycle |
| Add 1 | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **1-** | 1 byte | 0x13 | 1 cycle |
| Subtract 1 | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **2\*** | 1 byte | 0x14 | 1 cycle |
| Arithmetic shift left | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **2/** | 1 byte | 0x15 | 1 cycle |
| Arithmetic shift right | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **MULTS** | 1 byte | 0x16 | 5 cycles |
| Multiply two signed 32 bit integers to produce a 64-bit integer that is held in the top two stack positions, highest part top of stack | | ( x1 x2 -- d3) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **MULTU** | 1 byte | 0x17 | 1 cycle |
| Multiply two unsigned 32 bit integers to produce a 64-bit integer that is held in the top two stack positions, highest part top of stack | | ( u1 u2 -- ud3) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **ADDX** | 1 byte | 0x18 | 1 cycle |
| Add two integers with extend flag as carry. The extend flag resides within the datapath and is not otherwise accessible to software. The flag is only affected by arithmetic instructions. | | ( x1 x2 -- x3) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **SUBX** | 1 byte | 0x19 | 1 cycle |
| Subtraction with extend flag as borrow. The extend flag resides within the datapath and is not otherwise accessible to software. The flag is only affected by arithmetic instructions. | | ( x1 x2 -- x3) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **DIVS** | 1 byte | 0x1A | 42 cycles |
| Divide two 32-bit signed numbers to produce a 32-bit quotient (top of stack) and a 32-bit remainder (next on stack) | | (x1 x2 -- u-rem u-quot) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **DIVU** | 1 byte | 0x1B | 41 cycles |
| Divide two 32-bit unsigned numbers to produce a 32-bit quotient (top of stack) and a 32-bit remainder (next on stack) | | (x1 x2 -- rem quot) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **=** | 1 byte | 0x1C | 1 cycle |
| Returns -1 (true) if x1 = x2 | | ( x1 x2 -- flag) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **<>** | 1 byte | 0x1D | 1 cycle |
| Returns -1 (true) if x1 <> x2 | | ( x1 x2 -- flag) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **<** | 1 byte | 0x1E | 1 cycle |
| Returns -1 (true) if x1 < x2 | | ( x1 x2 -- flag) | ( --) |
| **>** | 1 byte | 0x1F | 1 cycle |
| Returns -1 (true) if x1 > x2 | | ( x1 x2 -- flag) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **U<** | 1 byte | 0x20 | 1 cycle |
| Returns -1 (true) if u1 < u2, where u is unsigned | | ( u1 u2 -- flag) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **U>** | 1 byte | 0x21 | 1 cycle |
| Returns -1 (true) if u1 > u2, where u is unsigned | | ( u1 u2 -- flag) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **0=** | 1 byte | 0x22 | 1 cycle |
| Returns -1 (true) if x1 = 0. Equivalent to Boolean NOT | | ( x1 -- flag) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **0<>** | 1 byte | 0x23 | 1 cycle |
| Returns -1 (true) if x1 <> 0 | | ( x1 -- flag) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **0<** | 1 byte | 0x24 | 1 cycle |
| Returns -1 (true) if x1 < 0 | | ( x1 -- flag) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **0>** | 1 byte | 0x25 | 1 cycle |
| Returns -1 (true) if x1 > 0 | | ( x1 -- flag) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **FALSE** | 1 byte | 0x26 | 1 cycle |
| Place zero (false) on the stack. Equivalent to ZERO | | ( -- 0) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **AND** | 1 byte | 0x27 | 1 cycle |
| Bitwise AND | | ( x1 x2 -- x3) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **OR** | 1 byte | 0x28 | 1 cycle |
| Bitwise OR | | ( x1 x2 -- x3) | ( --) |
| **INVERT** | 1 byte | 0x29 | 1 cycle |
| Bitwise NOT | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **XOR** | 1 byte | 0x2A | 1 cycle |
| Bitwise XOR | | ( x1 x2 -- x3) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **LSL** | 1 byte | 0x2B | 1 cycle |
| Logical shift left | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **LSR** | 1 byte | 0x2C | 1 cycle |
| Logical shift right | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **XBYTE** | 1 byte | 0x2D | 1 cycle |
| Sign extend a byte to 32 bits | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **XWORD** | 1 byte | 0x2E | 1 cycle |
| Sign extend a word to 32 bits | | ( x1 -- x2) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **FETCH.L** | 1 byte | 0x2F | 5 cycles in SRAM |
| Fetch a longword from memory, big endian | | ( addr -- n) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **STORE.L** | 1 byte | 0x30 | 5 cycles in SRAM |
| Store a longword in memory, big endian | | ( n addr --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **FETCH.W** | 1 byte | 0x31 | 3 cycles in SRAM |
| Fetch a word from memory, big endian | | ( addr -- n) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **STORE.W** | 1 byte | 0x32 | 3 cycles in SRAM |
| Store a word in memory, big endian | | ( n addr --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **FETCH.B** | 1 byte | 0x33 | 2 cycles in SRAM |
| Fetch a byte from memory | | ( addr -- n) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **STORE.B** | 1 byte | 0x34 | 2 cycles in SRAM |
| Store a byte in memory | | ( n addr --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **LOAD.B** or **#.B** | 2 bytes | 0x35, x1 | 2 cycles |
| Fetch inline byte to stack and zero extend | | ( -- x) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **LOAD.W** or **#.W** | 3 bytes | 0x36, x2, x1 | 3 cycles |
| Fetch inline word to stack and zero extend. High byte first | | ( -- x) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **LOAD.L** or **#.L** | 5 bytes | 0x37, x4, x3, x2, x1 | 5 cycles |
| Fetch inline longword to stack. Highest byte first | | ( -- x) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **JMP** | 1 byte | 0x38 | 2 cycles |
| Redirect program execution to the address on the parameter stack | | ( addr --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **BSR** | 1 byte | 0x39 | 2 cycles |
| Redirect program execution by the address offset on the parameter stack and place the original next following instruction address on the return stack | | ( offset --) | ( -- addr) |

|  |  |  |  |
| --- | --- | --- | --- |
| **JSR** | 1 byte | 0x3A | 2 cycles |
| Redirect program execution to the address on the parameter stack and place the original next following instruction address on the return stack | | ( addr --) | ( -- addr) |

|  |  |  |  |
| --- | --- | --- | --- |
| **TRAP** | 1 byte | 0x3B | 2 cycles |
| Jump to the trap vector (address 0x02) and place the original next following instruction address on the return stack. Used for breakpoint debugging. | | ( --) | ( -- addr) |

|  |  |  |  |
| --- | --- | --- | --- |
| **RTS\_TRAP** | 1 byte | 0x3C | 2 cycles |
| Return from subroutine, execute one program instruction and trap again. Used for single step debugging | | ( --) | ( addr --)  ( -- addr) |

|  |  |  |  |
| --- | --- | --- | --- |
| **RTI** | 1 byte | 0x3D | 2 cycles |
| Return from an interrupt routine. Similar to RTS but also changes the interrupt controller state to unblocks further interrupts | | ( --) | (addr --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **RTS** | 1 byte | 0x40 | 2 cycles |
| Return from a subroutine that was entered via a JSR or BSR instruction | | ( --) | ( addr --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **,RTS** | 1 byte | (0x40 OR opcode) | 1 cycle |
| As RTS but is a compound for any single-cycle instruction that does not itself reference or impact the return stack. The compound instruction saves one cycle and one byte on each subroutine return (e.g. DROP,RTS). | | ( --) | ( addr --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **BEQ** | 2 bytes | (0x80 OR hi), lo | 3 cycles |
| Branch if the top of stack item is zero. The top 6 bits of the branch offset are in the first instruction byte, the bottom 8 bits of the branch address follow in a second instruction byte. The branch offset is calculated from the address of the second byte | | ( flag --) | ( --) |

|  |  |  |  |
| --- | --- | --- | --- |
| **BRA** | 2 bytes | (0xC0 OR hi), lo | 3 cycles |
| Branch. The top 6 bits of the branch offset are in the first instruction byte, the bottom 8 bits of the branch address follow in a second instruction byte. The branch offset is calculated from the address of the second byte | | ( --) | ( --) |

# 2. FORTH system software

The following reference is organized according to the categories of the ANSI FORTH documentation. Status indicates whether the word has been implemented on the N.I.G.E. Machine. Y indicates yes, N no, and that Y\* that the word is implemented by with some limitation or difference as compared with ANSI FORTH.

### CORE words

|  |  |  |
| --- | --- | --- |
| **Word** | **Status** | **Notes** |
| ! | Y | See also W! |
| # | N | Use U# instead. Division with a 64-bit dividend is not implemented on the N.I.G.E. Machine. |
| #> | N | Use U#> instead. See #. |
| #S | N | Use U#S instead. See #. |
| ‘ | Y |  |
| ( | Y |  |
| \* | Y |  |
| \*/ | Y\* | The intermediate value is single (32-bit) precision only |
| \*/MOD | Y\* | The intermediate value is single (32-bit) precision only |
| + | Y |  |
| +! | Y |  |
| +LOOP | Y |  |
| , | Y | See also W, M, and $, |
| - | Y |  |
| . | Y |  |
| .” | Y |  |
| / | Y |  |
| /MOD | Y |  |
| 0< | Y |  |
| 0= | Y |  |
| 1+ | Y |  |
| 1- | Y |  |
| 2! | N | Less suitable for a big-endian processor |
| 2\* | Y |  |
| 2/ | Y |  |
| 2@ | N | Less suitable for a big-endian processor |
| 2DROP | Y |  |
| 2DUP | Y |  |
| 2OVER | Y |  |
| 2SWAP | Y |  |
| : | Y |  |
| ; | Y |  |
| < | Y |  |
| <# | Y |  |
| = | Y |  |
| > | Y |  |
| >BODY | N | Would be a no operation in the N.I.G.E. Machine FORTH environment. Will not be implemented for space and efficiency reasons |
| >IN | Y |  |
| >NUMBER | Y |  |
| >R | Y |  |
| ?DUP | Y |  |
| @ | Y |  |
| ABORT | Y |  |
| ABORT” | N | Will not be implemented for space and efficiency reasons |
| ABS | Y |  |
| ACCEPT | Y |  |
| ALIGN | N | Would be a no operation in the N.I.G.E. Machine FORTH environment. Will not be implemented for space and efficiency reasons |
| ALIGNED | N | Would be a no operation in the N.I.G.E. Machine FORTH environment. Will not be implemented for space and efficiency reasons |
| ALLOT | Y |  |
| AND | Y |  |
| BASE | Y |  |
| BEGIN | Y |  |
| BL | Y |  |
| C! | Y | See also W! |
| C, | Y | See also W, |
| C@ | Y | See also W@ |
| CELL+ | Y |  |
| CELLS | Y |  |
| CHARS | N | Would be a no operation in the N.I.G.E. Machine FORTH environment. Will not be implemented for space and efficiency reasons |
| CONSTANT | Y |  |
| COUNT | Y |  |
| CR | Y |  |
| CREATE | Y |  |
| DECIMAL | Y |  |
| DEPTH | Y |  |
| DO | Y |  |
| DOES> | Y |  |
| DROP | Y |  |
| DUP | Y |  |
| ELSE | Y |  |
| EMIT | Y |  |
| ENVIRONMENT? | N | Will not be implemented for space and efficiency reasons |
| EVALUATE | Y |  |
| EXECUTE | Y |  |
| EXIT | Y |  |
| FILL | Y | See also FILL.W |
| FIND | Y |  |
| FM/M | N | Will not be implemented for space and efficiency reasons |
| HERE | Y |  |
| HOLD | Y |  |
| I | Y |  |
| IF | Y |  |
| IMMEDIATE | Y |  |
| INVERT | Y |  |
| J | Y |  |
| KEY | Y |  |
| LEAVE | Y |  |
| LOOP | Y |  |
| LSHIFT | Y |  |
| M\* | Y |  |
| MAX | Y |  |
| MIN | Y |  |
| MOD | Y |  |
| MOVE | Y |  |
| NEGATE | Y |  |
| OR | Y |  |
| OVER | Y |  |
| POSTPONE | Y |  |
| QUIT | Y |  |
| R> | Y |  |
| R@ | Y |  |
| RECURSE | Y |  |
| REPEAT | Y |  |
| ROT | Y |  |
| RSHIFT | Y |  |
| S” | Y | See also C” and ,” |
| S>D | N | Equivalent to FALSE on the N.I.G.E. Machine. Will not be implemented for space and efficiency reasons. |
| SIGN | Y |  |
| SM/REM | N | Division with a 64-bit dividend is not supported |
| SOURCE | Y |  |
| SPACE | Y |  |
| SPACES | Y |  |
| STATE | Y |  |
| SWAP | Y |  |
| THEN | Y |  |
| U. | Y |  |
| U< | Y |  |
| UM\* | Y |  |
| UM/MOD | N | Division with a 64-bit dividend is not supported |
| UNLOOP | Y |  |
| UNTIL | Y |  |
| VARIABLE | Y |  |
| WHILE | Y |  |
| WORD | Y |  |
| XOR | Y |  |
| [ | Y |  |
| [‘] | Y |  |
| [CHAR] | Y |  |
| ] | Y |  |

### CORE EXTENSION words

|  |  |  |
| --- | --- | --- |
| .( | Y |  |
| .R | Y |  |
| 0<> | Y |  |
| 0> | Y |  |
| <> | Y |  |
| ?DO | Y |  |
| AGAIN | Y |  |
| BUFFER: | Y | Allocates space in PSDRAM. Suitable for larger data blocks |
| C” | Y | In the interpretation state C” will copy the text until the following “ to the PAD as a counted string and return its address |
| CASE | Y |  |
| COMPILE, | Y |  |
| DEFER | Y |  |
| ENDCASE | Y |  |
| ENDOF | Y |  |
| FALSE | Y | Returns 0 |
| HEX | Y |  |
| INTERPRET | Y | Interpret a line from the input buffer |
| IS | Y |  |
| MARKER | Y |  |
| NIP | Y |  |
| OF | Y |  |
| PAD | Y |  |
| PARSE | Y |  |
| PICK | Y |  |
| RESTORE-INPUT | Y | SAVE-INPUT and RESTORE-INPUT use internal variables to store the input source specification. RESTORE-INPUT does not return a flag |
| SAVE-INPUT | Y | See RESTORE-INPUT |
| TRUE | Y |  |
| U.R | Y |  |
| U> | Y |  |
| UNUSED | Y |  |
| WITHIN | Y |  |
| \ | Y |  |

### 6.3.3 FACILITY words

|  |  |  |
| --- | --- | --- |
| KEY? | Y | See also KKEY?, S0KEY? And S1KEY? |

### FILE ACCESS words

|  |  |  |
| --- | --- | --- |
| INCLUDED | Y |  |

### PROGRAMMING TOOLS words

|  |  |  |
| --- | --- | --- |
| .S | Y |  |
| ? | Y |  |
| DUMP | Y |  |
| WORDS | Y |  |
| STATE | Y |  |

### STRING words

|  |  |  |
| --- | --- | --- |
| COMPARE | Y |  |
| SLITERAL | Y | See also CLITERAL |

### N.I.G.E. Machine specific FORTH words

#### General screen control

|  |  |  |
| --- | --- | --- |
| TAB | ( -- addr) | VARIABLE pointing to the current size of tab stops. The default is 3 |
| ACOL | ( -- addr) | Byte length VARIABLE pointing to the color code for user input keystrokes |
| BCOL | ( -- addr) | Byte length VARIABLE pointing to the color code for output from the FORTH environment |
| TCOL | ( -- addr) | Byte length VARIABLE pointing to the color code for current output by VEMIT or CSR-PLOT |
| CLS | ( --) | Clear the screen |
| COLOR-TABLE | ( --) | Display a table of the 256 available character/text colors |

#### Programming the text/character display

|  |  |  |
| --- | --- | --- |
| VEMIT | ( n --) | Emit a character to the VDU |
| VTYPE | ( c-addr n --) | Type a string to the VDU |
| CSR-X | ( -- addr) | VARIABLE pointing to the current column position of the cursor |
| CSR-Y | ( -- addr) | VARIABLE pointing to the current row position of the cursor |
| CSR-ADDR | ( -- addr) | Return the memory address of the current cursor position within the character/text buffer in PSDRAM |
| CSR-PLOT | ( x --) | Plot the ASCII character x at the current cursor position |
| CSR-ON | ( --) | Plot the cursor symbol at the current cursor position. The character at that position is saved in an internal variable |
| CSR-OFF | ( --) | Unplot the cursor symbol from the current cursor position and restore the character which was previously there |
| CSR-FWD | ( --) | Advance the cursor by one character |
| CSR-BACK | ( --) | Move back the cursor by one character |
| CSR-TAB | ( --) | Advance the cursor to the next tab stop |
| SCROLL | ( n -- flag) | Scroll the screen fwd or back n lines within the 120 line frame buffer. Returns true if out of range, or false otherwise |
| NEWLINE | ( --) | Scroll the screen downwards by one line of text and return the cursor to the first column of the blank line below |

#### External I/O

|  |  |  |
| --- | --- | --- |
| **Word** | **Stack effect** | **Description** |
| S0EMIT | ( x --) | Emit a character to RS232 port 0 |
| S0KEY? | ( -- flag) | Check if a character is waiting to be read from the 256 byte circular buffer maintained for RS232 port 0 |
| S0KEY | ( -- n) | Wait for and read the next character available at RS232 port 0 |
| S0TYPE | ( c-addr n --) | Type a string to RS232 port 0, asynchronous. |
| S1EMIT | ( x --) | Emit a character to RS232 port 1 |
| S1KEY? | ( -- flag) | Check if a character is waiting to be read from the 256 byte circular buffer maintained for RS232 port 1 |
| S1KEY | ( -- n) | Wait for and read the next character available at RS232 port 1 |
| S1TYPE | ( c-addr n --) | Type a string to RS232 port 1, asynchronous. |
| S1ZERO | ( --) | Reset the 256 byte circular buffer maintained for RS232 port 1 and abandon characters currently waiting to be read from there |
| KKEY? | ( -- flag) | Check if a character is waiting to be read from the 256 byte circular buffer maintained for the PS/2 keyboard |
| KKEY | ( -- n) | Wait for and read the next character available from the PS/2 keyboard |
| PS2DECODE | ( n -- n) | Decode a PS/2 scan code into ASCII. Returns 0 if there is no valid ASCII match. PS2DECODE is called directly by the PS/2 interrupt controller. |

#### I/O redirection

|  |  |  |
| --- | --- | --- |
| >REMOTE | ( --) | Redirect FORTH environment output to RS232 port 0 |
| >LOCAL | ( --) | Redirect FORTH environment output to the screen |
| <REMOTE | ( --) | Receive FORTH environment input from RS232 port 0 |
| <LOCAL | ( --) | Receive FORTH environment input from the keyboard |

#### Compiler extensions

|  |  |  |
| --- | --- | --- |
| HERE1 | ( -- addr) | VARIABLE pointing to the dictionary pointer for the PSDRAM dictionary space. Only used by BUFFER: |
| INLINESIZE | ( -- addr) | VARIABLE pointing to the maximum code-length in bytes that the compiler will compile inline rather than as a subroutine call. The default value is 10 and the minimum allowable is 9 since certain code, such as LOOP code, much be compiled inline |
| W, | (w -- ) | Allocate 2 bytes in the dictionary and store a word from the stack |
| M, | (addr u --) | Allocate and store u bytes from addr into the dictionary. u is not saved in the dictionary. Compiles a string or other block of data from memory |
| $, | ( addr u --) | Allocate and store u bytes from addr into the dictionary. u is is compiled as the first byte. Compiles a counted string. |
| LITERAL | ( n --) | Compile a literal to the dictionary |
| CLITERAL | ( addr u --) | Compile to the dictionary a string literal as an executable that will be re-presented at run time as a counted string c-addr |

#### Other common FORTH words

|  |  |  |
| --- | --- | --- |
| BINARY | ( --) | Set BASE = 2 |
| WAIT | ( u --) | Wait for u milliseconds |
| RESET | ( --) | Reset the N.I.G.E. Machine to power on configuration but otherwise preserve memory contents |
| NOT | ( n – n) | Equivalent to 0= |
| XBYTE | ( n – n) | Sign extend a byte on the stack to 32 bits |
| XWORD | ( n –n) | Sign extend a word on the stack to 32 bits |
| W@ | ( addr – n) | Fetch a word from memory |
| W! | ( n addr --) | Store a word in memory |
| FILL.W | ( addr n w --) | Fill a region of memory with n words w. FILL.W utilizes the STORE.W machine language instruction and will be faster than FILL in accessing PSDRAM |
| UPPER | (x -- X) | Convert one ASCII character to uppercase |
| DIGIT | ( char base -- n true | char false ) | Convert a single ASCII character to a number in the given base |
| NUMBER? | ( c-addr u - false | n true ,) | Convert an ASCII string to a number and return with a success or failure flag |
| COMP | ( n1 n2 – n) | Return -1 if n1<n2, +1 if n1>2, 0 if n1=n2 |
| $= | ( c-addr1 u1 c-addr2 u2 -- flag) | Test two strings for equality. Case insensitive. |
| ERROR | ( n --) | Print “ERROR n” and ABORT |

#### SD card reader/writer

|  |  |  |
| --- | --- | --- |
| LOAD | ( c-addr n -- len true | false) | Load a file with the name given by c-addr n into the file buffer from an SD card reader/writer attached to RS232 port 1 |
| SAVE | (c-addr n) | Save the file currently in the file buffer to an SD card reader/writer attached to RS232 port 1 with the name given by c-addr n |
| DOS | ( --) | Enter terminal mode for direct communication with an SD card reader/writer attached to RS232 port 1. Exit terminal mode by pressing ESC |