|  |  |  |
| --- | --- | --- |
| Word | Stack | Notes |
| @ | (addr – data) | Read one 32 bit word. Must be on a 4 byte boundary.  *Addresses are all byte addresses.*  *Least Significant Byte first in word.* |
| ! | (data addr -) | Write one 32 bit word. Must be on a 4 byte boundary |
| c@ | (addr – data) | Read one 8 bit byte |
| c! | (data addr -) | Write one 8 bit byte |
| +! | (value addr -) | Add value into memory. Must be on a 4 byte boundary |
| + | (n1 n2 – n3) | Add two 32 bit values |
| - | (n1 n2 – n3) | Subtract two 32 bit values (result is n1 – n2) |
| \* | (n1 n2 – n3) | Multiply two 32 bit values |
| / | (n1 n2 – n3) | Divide two 32 bit values (result is n1 / n2). If n2 = 0 result is 0 |
| and | (n1 n2 – n3) | Bitwise and two 32 bit values |
| or | (n1 n2 – n3) | Bitwise or two 32 bit values |
| xor | (n1 n2 – n3) | Bitwise exclusive or two 32 bit values |
| not | (n1 – n2) | One’s complement top of stack. |
| 0= | (n1 – n2) | Push Truth(n1 = 0) on stack (False 0, True $FFFFFFFF) |
| 0> | (n1 – n2) | Push Truth (n1 > 0) on stack |
| 0< | (n1 – n2) | Push Truth (n1 < 0) on stack |
| 0- | (n1 – n2) | Two’s complement negate top of stack |
| 1+ | (n1 – n2) | Increment top of stack |
| 1- | (n1 – n2) | Decrement top of stack |
| 2\* | (n1 – n2) | Shift top of stack left |
| 2/ | (n1 – n2) | Shift top of stack right (most significant bit 0, e.g. not arithmetic) |
| dup | (n1 – n1 n1) | Duplicate top of stack value |
| drop | (n1 - ) | Drop value on top of stack |
| swap | (n1 n2 – n2 n1) | Swap values on top of stack |
| rot | (n1 n2 n3 – n3 n1 n2) | Rotate top three values |
| over | (n1 n2 – n1 n2 n1) | Duplicate 2nd value down on stack |
| ; | ( - ) | Return from subroutine |
| r> | ( - n1) | Pop top of return stack to data stack |
| >r | (n1 - ) | Pop top of data stack to return stack |
| rdrop | ( - ) | Drop top of return stack |
| if | (n1 - ) | if n1 is not zero advance PC to after next *then* or *;* |
| then | ( - ) | Marker for if |
| for | (n1 - ) | Push PC then n1 to return stack. |
| next | ( - ) | (Works with for) Decrement top of return stack.   * If <= 0 then drop counter and loop address * If > 0 then set PC to second value on stack. |
| dsp! | (n1 - ) | Set data stack address.  *All stacks are pre-decrement push, post-increment pull.* |
| dsp@ | ( - n1) | Get data stack address |
| rsp! | (n1 - ) | Set return stack address |
| rsp@ | ( - n1) | Get return stack address |