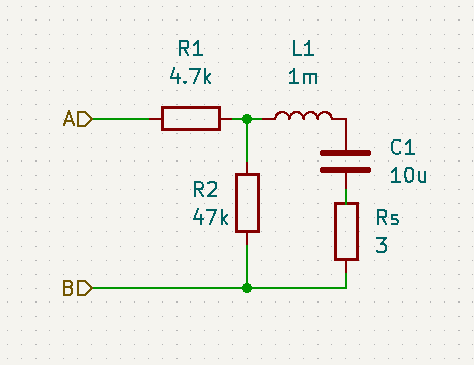
# Keywords

|  |  |  |
| --- | --- | --- |
| **Command** | **Type** | **Description** |
| ***any literal*** | data | Push a literal, the value of a variable or a string into the stack. If the literal is not a double precision number or a variable, it will be a bigint or a string (can be optionally placed inside double quotes). If it is a bigint (a long string possibly representing a decimal number), double quotes won’t be added. |
| ***num‘r’, ‘l’, ‘c’***  **e.g., *4.7e-3l*** | data | Any numeric literal followed by one of the characters‘r’, ‘l’ or ‘c’, indicating resistance, inductance or capacitance. The literal value will be stored on the stack but will be interpreted as an impedance value for a frequency in a variable called ‘f’. If variable ‘f’ doesn’t exist, a frequency value of 1Hz will be assumed.  For example, 4.7e-3l is the impedance of a 4.7mH inductor, or 29.531*j* ohms at 1000Hz. |
| **(*numR numI*)**  **or**  **(*numR,numI*)** | data | Push a double precision complex number with real part *numR* and imaginary part *numI* into the stack. |
| **(*numI*)** | data | Push a double precision imaginary number *numI* into the stack. |
| **a** | data | Push the value of the *acc* register into the stack. |
| **“a”** | data | Push a string, which can be an arbitrarily long decimal integer (a *bigint*) into the stack. If the string is a bigint, {+ - \* / max min > < >= <= = != } operators can be used. |
| **b** | data | Push the double precision or complex value of the *bak* register into the stack. |
| **va** | data | Push the vector in the vector register *vacc* into the stack. |
| **@ *or* @a** | operator | Pop the ToS element to *acc* register. This pops individual entries from a vector or matrix. |
| **@b** | operator | Pop the ToS element to *bak* register. This pops individual entries from a vector or matrix. |
| **@@** | operator | Pop *n* (must be a number ≥0) entities from the stack into the *acc* register where *n* is the value on the top of the stack. A vector or matrix is considered a single element and will be popped entirely. If the last entity is a vector, it will be popped into a variable that can be accessed using the variable name *vacc*. If *n* is 0, the entire stack will be cleared. If *n* is less than 0, no values will be popped and an error will result. The Execution Stack, which tracks conditionals will not be cleared (vector/matrices entry is also tracked using the execStack, will be adjusted after this operation). |
| **@*varname*** | operator | Pop ToS element into a variable named *varname*. This might result in a partial vector or matrix on the stack. |
| **?*varname*** | operator | Print the variable named *varname*. |
| **? *or* ?a** | operator | Print the *acc* register. |
| **?b** | operator | Print the *bak* register. |
| **?va** | operator | Print the vector register *vacc*. |
| **?”*string*”** | operator | Print the literal string *string* (printed without the double quotes). |
| **[ *or* ]** | data | Start or end a vector. |
| **[[ *or* ]]** | data | Start or end a matrix. |
| **( *or* )** | data | Encloses a complex number. |
| **dup** | operator | Duplicate the ToS. |
| **swp** | operator | Swap ToS with element prior to ToS. |
| **if** | conditional | Execute if the ToS element does not equal 0 or is a string. |
| **el** | conditional | Execute if the ToS element equals 0. |
| **fi** | conditional | End an **if-el** block |
| ***label*:** | label | A label used for jmp, jz or jnz. |
| **jmp** | unconditional jump | Jump to a label. |
| **jz** | conditional jump | Jump to a label if ToS is 0. |
| **jnz** | conditional jump | Jump to a label if ToS is not 0 or is a string. |
| **+ - \* / max min > < >= <= = !=** | operator | The usual arithmetic operators. Works for regular numbers (complex) or bigint strings. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Examples

Calculate the impedance between the terminals A and B in the diagram below at a frequency of 1kHz.



The full sequence:

1000 @f 4.7e3r 1e-3l 10e-6c + 3 + 47e3 // +

Explanation:

* Enter the frequency

1000 @f

* Compute the impedance for the last loop as (3 -9.6323) = 3 - 9.6323*j* ohm

4.7e3r 1e-3l 10e-6c + 3 + 47e3 // +

* Compute the impedance for the parallel loops as (3.0018 - 9.631) = 3.0018 - 9.631*j* ohm

4.7e3r (3 -9.6323) 47e3 // +

* Compute the impedance in series with the first resistor as (4703.0018 -9.631) = 4703.0018 -9.631*j* ohm

4.7e3r (3.0018 -9.6311) +

Answer is **4703.0018 - 9.631*j*** ohm