EE5311: Digital IC Design

SPICE syntax aids:
https://ngspice.sourceforge.io/docs/ngspice-manual.pdf

11.3.2 .DC: DC Transfer Function

General form:

```
.dc srcnam vstart vstop vincr [src2 start2 stop2 incr2]
```

Examples:

```
.dc VIN 0.25 5.0 0.25
.dc VDS 0 10 .5 VGS 0 5 1
.dc VCE 0 10 .25 IB 0 10u 1u
.dc RLoad 1k 2k 100
.dc TEMP -15 75 5
```

The `.dc` line defines the dc transfer curve source and sweep limits (with capacitors open and inductors shorted). **`srcnam`** is the name of an independent voltage or current source, a resistor, or the circuit temperature. **`vstart`**, **`vstop`**, and **`vincr`** are the starting, final, and incrementing values, respectively. The first example causes the value of the voltage source V_{IN} to be swept from 0.25 Volts to 5.0 Volts with steps of 0.25 Volt. A second source (**`src2`**) may optionally be specified with its own associated sweep parameters. In such a case the first source is swept over its own range for each value of the second source. This option is useful for obtaining semiconductor device output characteristics. See the example on transistor characterization (17.3).

SPICE Analysis mode: Transient

11.3.10 .TRAN: Transient Analysis

General form:

```
.tran tstep tstop <tstart <tmax>> <uic>
```

Examples:

```
.tran 1ns 100ns
.tran 1ns 1000ns 500ns
.tran 10ns 1us
```

tstep is the printing or plotting increment for line-printer output. For use with the post-processor, **tstep** is the suggested computing increment. **tstop** is the final time, and **tstart** is the initial time. If **tstart** is omitted, it is assumed to be zero. The transient analysis always begins at time zero. In the interval [zero, **tstart**), the circuit is analyzed (to reach a steady state), but no outputs are stored. In the interval [**tstart**, **tstop**], the circuit is analyzed and outputs are stored. **tmax** is the maximum stepsize that ngspice uses; for default, the program chooses either **tstep** or (**tstop**-**tstart**)/50.0, whichever is smaller. **tmax** is useful when one wishes to guarantee a computing interval that is smaller than the printer increment, **tstep**.

An initial transient operating point at time zero is calculated according to the following procedure: all independent voltages and currents are applied with their time zero values, all capacitances are opened, inductances are shorted, the non linear device equations are solved iteratively.

Measurement for various analysis modes

13.5.50 Meas: Measurements on simulation data

General Form (example):

```
MEAS {DC|AC|TRAN|SP} result TRIG trig_variable VAL=val <TD=td>
<CROSS=# | CROSS=LAST> <RISE=#|RISE=LAST> <FALL=#|FALL=LAST>
<TRIG AT=time> TARG targ_variable VAL=val <TD=td>
<CROSS=# | CROSS=LAST> <RISE=#|RISE=LAST>
<FALL=#|FALL=LAST> <TRIG AT=time>
```

Most of the input forms found in 11.4 may be used here with the command `meas` instead of `.meas(ure)`. Using meas inside the `.control` ... `.endc` section offers additional features compared to the `.meas` use. `meas` will print the results as usual, but in addition will store its measurement result (typically the token **`result`** given in the command line) in a vector.

Control structures - while

13.6.1 While - End

General Form:

```
while condition
statement
...
end
```

While condition, an arbitrary algebraic expression, is true, execute the statements.

Example:

```
let loopindex = 0
while loopindex < 5
  echo index is $&loopindex
  let loopindex = loopindex + 1
end
```

Comment: `let` creates a vector. Convert vector `loopindex` to number (as required by `echo`) by `$&loopindex`. The condition statement compares vectors.

Control structures - dowhile

13.6.3 Dowhile - End

General Form:

```
dowhile condition
statement
...
end
```

The same as `while`, except that the condition is tested after the statements are executed.

Example:

```
let loopindex = 0
dowhile loopindex <> 5
  echo index is $&loopindex
  let loopindex = loopindex + 1
end
```

Control structures - foreach

13.6.4 Foreach - End

General Form:

```
foreach var value ...
statement
...
end
```

The statements are executed once for each of the values, each time with the variable **`var`** set to the current value. (**`var`** can be accessed by the **`$var`** notation - see below).

Examples:

```
foreach val -40 -20 0 20 40
  echo var is $val
end
echo
set myvariable = ( -4 -2 0 2 4 )
foreach var $myvariable
  echo var is $var
end
echo
let myvec = vector(5)
foreach var $&myvec
  echo var is $var
end
```

The values themselves may be set by a variable like `myvariable` or a vector like `myvec`.

Control structures - repeat

13.6.2 Repeat - End

General Form:

```
repeat [number]
statement
...
end
```

Execute the statements number times, or forever if no argument is given.

Examples: Comment:

```
* plain number
repeat 3
  echo How many loops? Count yourself!
end
echo
* variable
set loops = 7
repeat $loops
  echo How many loops? $loops
end
echo
* vector
let loopvec = 4
repeat $&loopvec
  echo How many loops? $&loopvec
end
```

`set` creates a variable. `repeat` requires a number as parameter, either a plain number or converted from vector by `$&loopvec` or converted from variable by `$loops`.

13.6.5 If - Then - Else

General Form:

```
if condition
statement
...
else
statement
...
end
```

If the condition is non-zero then the first set of statements are executed, otherwise the second set. The `else` and the second set of statements may be omitted.

Example:

```
foreach val -40 -20 0 20 40
  if $val < 0
    echo variable $val is less than 0
  else
    echo variable $val is greater than or equal to 0
  end
end
echo
let vec = 1
if vec = 1 ; if $&vec = 1   is possible as well
  echo vec is $&vec
end
```

Comment: The condition may be evaluated by numbers or vectors. Variables have to be parsed to numbers like `$val`.