Using the internal ladder analysis codes to handle networks that are not predefined into the system.

Each step in the analysis of a network must be worked out in advance and specified to the program by the connection codes following the part value in the same format previously described in chapter 5 for the predefined networks. These MUST be inserted using a text editor directly in the "dzn" file. These codes call the same internal functions of the PCFILT analysis module as the predefined network connection codes do internally.

The circuit editor will display the value of each part. The circuit editor may be used to change the values of true parts but the "Insert" utility can't be used to insert this type of codes.

A network is built up by the manipulation of several "registers". Data representing each part in the network is first read into the "holding register". This register is symbolized by the pound symbol "#". A "stack" of registers is used to hold intermediate calculations or data. These registers are symbolized as "Stk". The final results of the analysis is built up in the"Summation register" as the analysis proceeds through the "dzn" file. The summation register is symbolized by the Greek letter Sigma "Σ".

The following is a detailed description of each code, its function, and an example of its use:

==== The following codes use the data associated directly with them ====

```
      | #—Push—> Stk (92)     ** Store data in stack **
      Any part that is to become the beginning element of a floating branch. This
code tells the computer to push the part associated with it into the stack bottom
register to hold it for future use. This code may be repeated up to 9 times in
a row pushing each previous value up the stack. That is to say: the stack rises.
            5.00000000e+01 R 92
            (50 ohm resistor stored as a floating branch for later use)

      | #——Par—> Stk (93)     ** Parallel data into stack **
      Any part to be connected in parallel with the floating branch in the stack
bottom register (stack does not move).
            6.80000000e-06 L 93
            (6.8 uH inductor in parallel with stack bottom register)

      | #——Ser—> Stk (94)     ** Series data into stack **
      Any part connected series into the stack bottom register.
      (stack does not move)
            9.10000000e-11 C 94
            (91 pF Cap. in series with stack bottom register)
```

==== These codes manipulate the data in the stack ====

The "Part value" field in the file is ignored and may be zero.

The "ID type" field can be the vertical line character "|" or space “ ”.

```
      |  Stk—Par—> Σ (95)     ** Parallel stack into summation **
      Tells the computer to connect the entire floating branch in the stack
bottom register in Parallel with the main Summation register.
      (the stack drops)
            0.00000000e+00 | 95
```

| Stk—Ser—> Σ (96) ** Series stack into summation **

Series connect the entire floating branch in the stack bottom register into the main Summation register.

(the stack drops)

0.00000000e+00 | 96

| #——Par———> Σ (90) ** Parallel data into summation **

Any part to be connected in Series to the Summation register after the pop | Stk+1 ——> # (98) code.

2.50000000e-12 C 90

(2.5 pF Cap. in Parallel)

| #——Ser———> Σ (91) ** Series data into summation **

Any part connected from the Summation register to ground after the pop | Stk+1 ——> #(98) code.

2.000000000e-8 L 91

(20 nH inductor to ground)

| Stk+1 ——> # (98) ** Recall stack register 1 into hold register **

This code tells the program to take the information at 2nd stack register (1) and do with it what is specified by the NEXT code. The stack does NOT drop into the bottom register as usual but into register 1 from where the data was removed. The bottom register is register 0.

0.00000000e+00 | 98 (This is a trick to exchange the data in
0.00000000e+00 | 92 the bottom of the stack with the data in
the stack register 1 just above it.)

NOTE: Only codes (90) through (94) may be used to tell the program what to do with the data popped from stack register 1. Never follow (98) with code (95) or (96)!

For example, to connect the data from data stack register 1 in series with the Summation register:

Do this:

| Stk+1——> # | 0.00000000e+00 | 98
| #——Ser———> Σ | 0.00000000e+00 | 91

Data moves from the stack register 1 to the holding register (#), then it is connected in series into the Summation register.

NOT this:

| Stk+1——> # | 0.00000000e+00 | 98
| Stk—Ser—> Σ | 0.00000000e+00 | 96

Doing this will result in a "Code 98 error" message after the first frequency point is analyzed! The data moved to the holding register from the stack would be lost.

| D->Y in Stk (97) ** Delta to Y transformation **
| Y->D in Stk (100) ** Y to Delta transformation **

These codes causes a "delta" To "Y" or “Y” to “delta” transformation to be done to the parts or data previously stored in the bottom three registers of the stack.

See figure 1 (stack does not move)

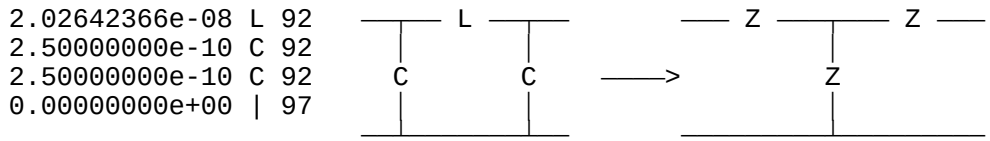
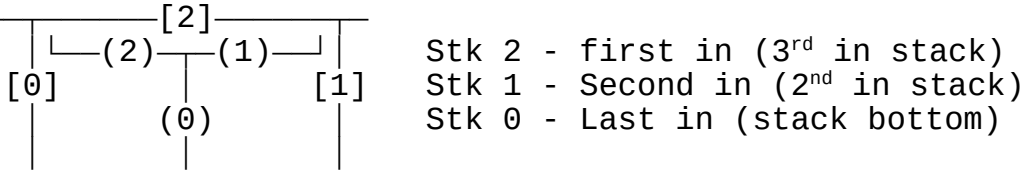


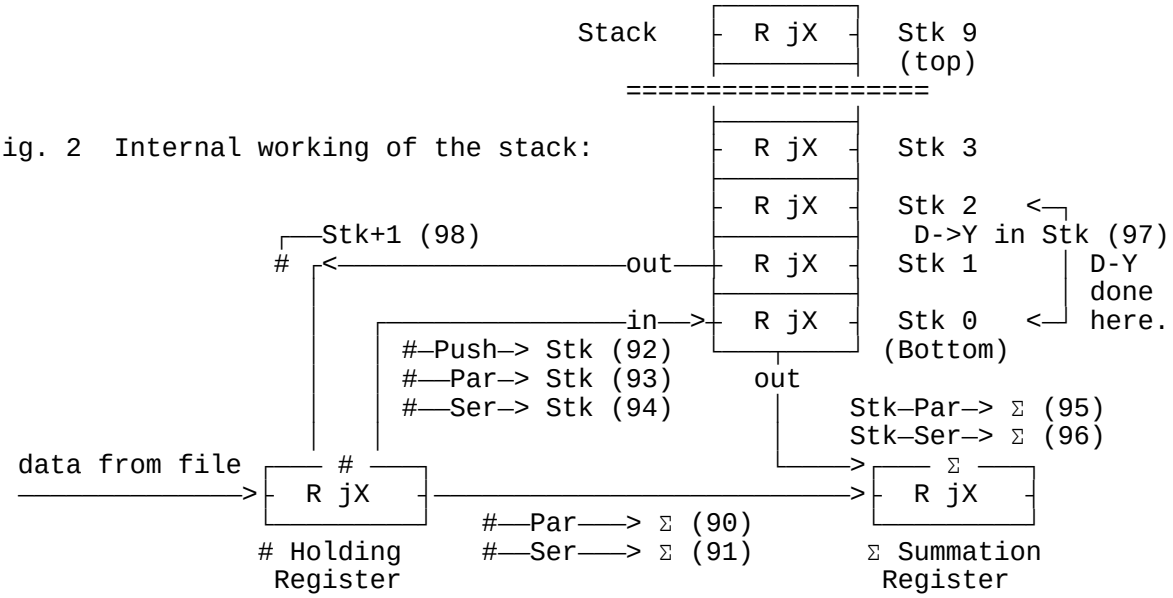
Fig. 1 Location of data for D->Y or Y->D in Sack:



Note: "in" makes reference to the order "Pushed":

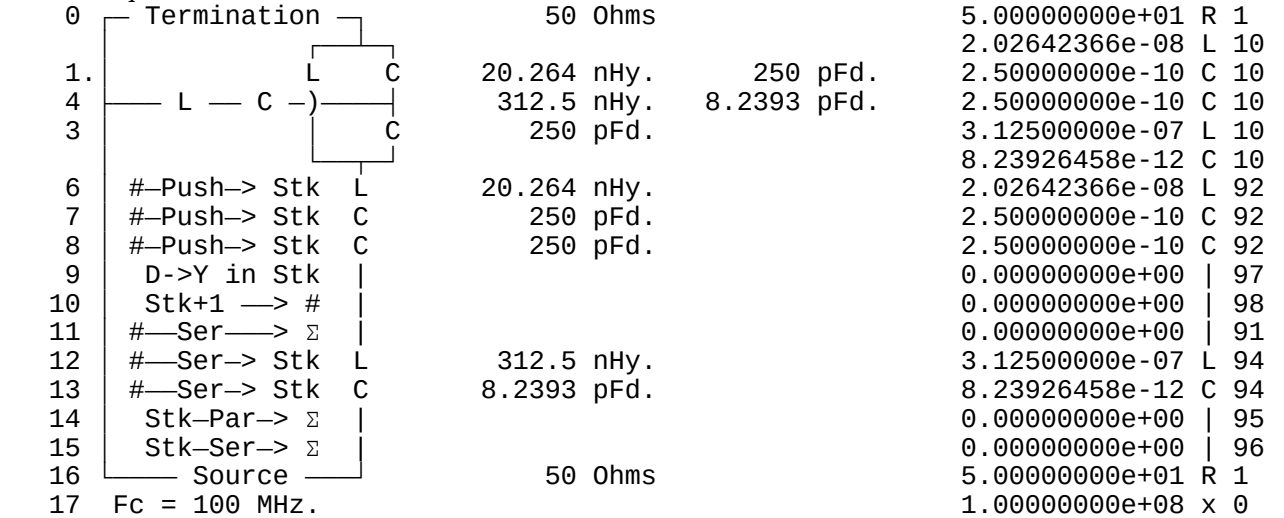
```
2.02642366e-08 L 92     first in   [2]
2.50000000e-10 C 92     second in  [1]
2.50000000e-10 C 92     last in    [0]
0.00000000e+00 | 97
```

Fig. 2 Internal working of the stack:

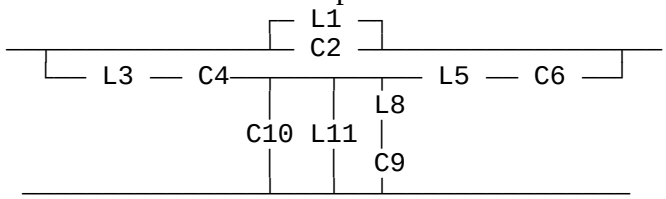


********** Examples **********

Below is a sample file showing two identical 2nd order all-pass group delay equalizers. The first uses the predefined code 10. The second uses the internal ladder network codes to define the same equalizer network.



This example is of a notch network that has no predefined code:




```
 0 ┌─ Termination ─┐          1000 Ohms                    1.00000000e+03 R 1
 1 │ #—Push—> Stk  L        1007.3 nHy.                    1.00730980e-06 L 92
 2 │ #——Par—> Stk  C        1.0073 pFd.                    1.00730980e-12 C 93
 3 │ #—Push—> Stk  L        2014.6 nHy.                    2.01461950e-06 L 92
 4 │ #——Ser—> Stk  C       0.50365 pFd.                    5.03654882e-13 C 94
 5 │ #—Push—> Stk  L        2014.6 nHy.                    2.01461950e-06 L 92
 6 │ #——Ser—> Stk  C       0.50365 pFd.                    5.03654882e-13 C 94
 7 │  D->Y in Stk  |                                       0.00000000e+00 | 97
 8 │ #—Push—> Stk  L         10000 nHy.                    1.00000000e-05 L 92
 9 │ #——Ser—> Stk  C       0.10147 pFd.                    1.01467000e-13 C 94
10 │ #——Par—> Stk  C        4.0292 pFd.                    4.02923912e-12 C 93
11 │ #——Par—> Stk  L        251.83 nHy.                    2.51827440e-07 L 93
12 │  Stk+1——> #   |                                       0.00000000e+00 | 98
13 │ #——Ser—> Stk  |                                       0.00000000e+00 | 94
14 │  Stk+1——> #   |                                       0.00000000e+00 | 98
15 │ #——Ser———> Σ  |                                       0.00000000e+00 | 91
16 │  Stk—Par—> Σ  |                                       0.00000000e+00 | 95
17 │  Stk—Ser—> Σ  |                                       0.00000000e+00 | 96
18 └──── Source ———┘          1000 Ohms                    1.00000000e+03 R 1
19  Fc = 158 MHz.                                         1.58000000e+08 x 1
```

********* Using the circuit editor with the specially coded networks *******

Networks using these codes have not been completely implemented into all functions of the circuit editor. You can change values, do impedance and frequency scaling. You can insert Q exception branches on any branch that specifies a part value. You can NOT insert any other type of branch within them but it is ok to join them with other predefined networks. A network coded this way should be considered as a single entity. You can also rotate multiplexer ports containing them.