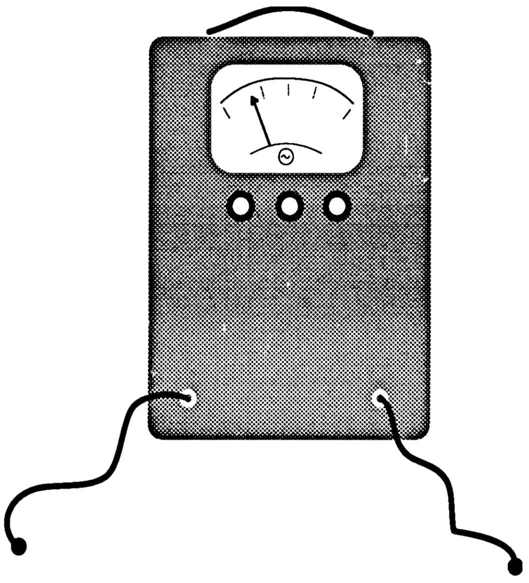
HP-28S

Software Power Tools

Electrical Circuits



A Product of
Solve and Integrate Corporation

A GRAPEVINE PUBLICATION

HP-28S
Software Power Tools:

ELECTRICAL CIRCUITS

A Product of
Solve and Integrate Corporation

Grapevine Publications, Inc.
P.O. Box 118
Corvallis, Oregon 97339-0118 U.S.A.

Acknowledgements

Thanks and appreciation go once again to Hewlett-Packard Company for continuing to produce such top-quality products and documentation – and for the inspiration and encouragement for this project. The programs for the objects herein called `INODE`, `PUTX` and `VMESH` (listed on pages 130, 170 and 190, respectively) were adapted from similar materials copyrighted by Hewlett-Packard Company and used herein by permission of Hewlett-Packard Company.

Printed in the United States of America
First Printing – March, 1989
ISBN 0-931011-25-6

Contents

Introduction To This Book

There's *a lot* contained in these pages, so *take the time right now* to read this little orientation on what is (and isn't) expected of you.

What You "Gotta" Do

- *You "gotta" know something about the HP-28S* – how to do arithmetic and stack manipulations – and how to key in, name, store, and invoke (i.e. "call" or evaluate) program and data objects. This book is not a primer on the HP-28S (see the list of other products on page 254 if you want a real tutorial on the machine itself). However, if you need "just a little refresher" on some of these basics, then see Appendix B (page 244).

- *You "gotta" know the basics of sinusoidal electrical circuits.* This book is not a primer on that topic, either. It's just a description of some tools that can help you solve a variety of common problems. The HP-28S can't correct you if you ask wrong questions or come to wrong conclusions. It'll do the calculating, but *you* have to do the thinking. Among other things, that means you need to have a firm grasp of the meanings and uses of **complex numbers, impedance, voltage, current, resistance, capacitance and inductance.**

- *You "gotta" key in, name, and test some code (programs).* How much code? That depends on what you want to do. This book has *two large collections of small routines.* In the first collection (called Circuit Reduction Tools) *each routine can be a useful tool all by itself.* In the second collection (called Circuit Analysis Tools), *you'll need many routines* working in concert to form a working tool.

- *You "gotta" invest a little time, read the instructions, and practice.* There are good reasons for the large number of pages you see here. It's just not realistic to start with this book on the night before the big exam or design proposal, so *don't* expect to be able to look up a topic in the index, flip to that page and instantly find the keystrokes necessary to solve a given problem. It doesn't work like that *until* you've keyed in the tools and practiced with them enough to know what you're doing. *Even to decide where to start or which tools you need – read the instructions.* The little introductory chapter before each of the two collections of tools is designed solely to help you decide which tools you need.

What You "Don't Gotta" Do

- *You "don't gotta" read everything in the book.* Just *start* at Chapter 1, and then follow the instructions.

- *You "don't gotta" key in everything in the book;* you may never use some of this stuff. Rest assured – after you decide what you want to do, you'll be told which routines are necessary to get the job done.

- *You "don't gotta" become an HP-28S-whiz-programmer to understand and use these tools* – so don't be put off by the descriptions of flags and errors, etc. That information is there just in case you do want to expand upon these tools for your own purposes. If not, just follow the instructions and use the examples as your guides.

- *You "don't gotta" be limited by this book.* If you *are* a proficient HP-28S programmer, then you can use this core idea to expand upon for your own circuit analysis. If so, then you might find Appendix A (page 230) interesting.

Chapter 1

CIRCUIT REDUCTION TOOLS:

Introduction

This Chapter is meant to help you decide whether you need the Circuit *Reduction* Tools – and then decide where to read next.

What These Tools Can – And Cannot – Do For You

Circuit *Reductio*n Tools are just that: They are little routines that help to simplify or reduce a circuit's elements or configuration.

For example, with the help of these tools, you can:

- Key in and convert between different formats of complex numbers;
- Convert from passive circuit elements (R, L, and C) to generalized impedances (Z);
- Find equivalent single impedances for series or parallel combinations of impedances;
- Find the distributions of currents flowing through 2 or more parallel impedances;
- Find the distributions of voltages across 2 or more series impedances;
- Perform Δ-Y Conversions (either way);
- Find Thevinin's and Norton's equivalent source-impedance combinations.

However, you *cannot* use these tools very conveniently to:

- Calculate voltages at points in a network of circuit elements;
- Calculate currents through branches in a network of circuit elements;
- Calculate transfer ratios of ladder networks.

For such calculations, you'll need the Circuit *Analysis* Tools.

The Circuit Reduction tools are all *separate and simple;* they don't "call" one another or otherwise operate "automatically" inside any larger context or structure.

They're much like the built-in stack and arithmetic commands in the CATALOG of your HP-28S.

Anytime you want to use one of these tools, you key in the appropriate values to the stack, invoke the routines's name, and zap – there's your result on the stack. Then you have to store it or otherwise manipulate it yourself – with more keystrokes and commands. It's all just numbers to the HP-28S.

That's different than the kinds of programs you may be more familiar with on, say, personal computers. Many of those are big, friendly, menu-driven applications that automatically store your data into meaningful structures, prompt you with options, trap errors, etc. If you prefer such "smarter programs," you may find the Circuit *Analysis* Tools more to your liking (assuming they do what you need).

With these Circuit Reduction tools, you must do everything "manually." And if you want to combine these tools into programs – to save yourself keystrokes or to do "smarter," automatic combinations of operations, that's fine – *but you have write those programs yourself.* This book is not going to help you in that respect.

Where To Go Next

At this point, you're probably of one of the following minds:

- The Circuit Reduction Tools are definitely *not* what you're after right now. In that case, skip immediately over to Chapter 4 (page 88);

- The Circuit Reduction Tools *are* definitely what you want – and you want to key them all in, to have them handy for your work. In that case, turn the page and dig right into Chapter 2;

- There are probably *some* of the Circuit Reduction Tools that you might like to "have around the house," but you're not yet sure which ones – you'd like to get to know them a little better first. In that case, you have two choices:

  – You can simply browse on through Chapter 2 (there are brief examples included there) and pick out what you need, based upon that cursory inspection;

  – You can skip over to Chapter 3 (page 68). There, you'll get a little more in-depth discussion of the uses, ideas, and the "feel" of these tools, and you can return more knowledgeably to Chapter 2 later, to select what you need.

Chapter 2

CIRCUIT REDUCTION TOOLS:

Reference

This Chapter is presented as a convenient, alphabetized listing of all the Circuit Reduction Tools – including brief examples illustrating the mechanics of their uses.

For more in-depth examples and discussion of these tools and the ideas behind them, see Chapter 3 (page 68).

Before you key in anything, **read these important preliminaries:**

First of all, as is usual with the HP-28S, there are many ways to key in solutions to problems, and it's just impossible to show every method. This book simply cannot "read your mind" to know which menu or directory you're currently using when you want to call one of these tools – so it can't give you the most convenient set of keystrokes for your particular case.

In all these examples, therefore, rather than specifically telling you to press a *key* (e.g. ■[PURGE] or [DROP]) or to select a *menu item* (e.g. R→C P→R), you'll see all commands in a generic form (spelled out as if you had typed them in):

```
PURGE DROP
 R→C P→R
```

etc.

So just keep in mind that, depending upon what you're doing, maybe it would be more convenient for you to use special keys or menu items than to type in the commands character by character.

A sample program description is shown on the opposite page. This is the general format for the description of each object in this chapter....

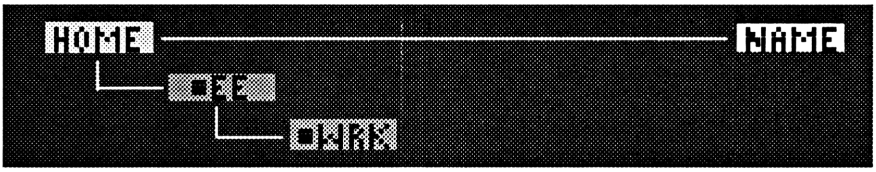
Notice especially the memory diagram at the top of the sample format (you'll see such a diagram for each routine). This shows the *recommended directory location* for the object – in relation to other directories that are superordinate to ("above") or subordinate to ("below") it. Hopefully, this diagram will also make it easier to find what you're looking for when flipping through the book (the routines are presented alphabetically, and there's also an index in the back, if you prefer). And it may help to orient you and put into perspective exactly what you're doing and what this program does in relation to the "big picture."

However, these diagrams are *recommendations* only. You could certainly store any or all of these Circuit Reduction Tools in the `HOME` directory or any other directory you want. Just bear in mind that *when you invoke them, you must do so from a directory that is at the same level or subordinate to ("below") where you have stored them.*

If you do use these recommended locations, then right now – before you key in any of these tools, **you should perform the following commands** to create the appropriate directories (of course, you need to do this only if you haven't already done so):

```
HOME
'▪EE' CRDIR
▪EE '▪WRK' CRDIR
```

To help you check the accuracy of your keystrokes when inputting of these Circuit Reduction Tools, each routine is listed with its *checksum,* which is an integer can be generated with the help of the `CKSM` routine (the first routine listed). `CKSM` is also listed in the Circuit Analysis Tools.



Title:

A phrase that briefly tells you what the routine does.

Name (Checksum)

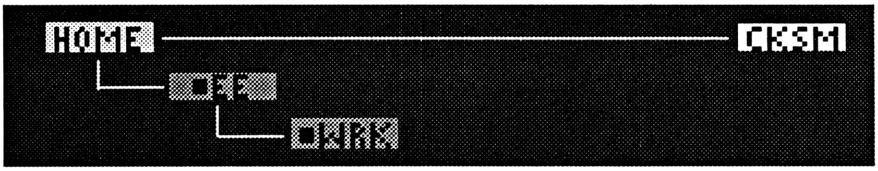
The object name identifying the routine, followed by an integer that can be used to help "proofread" the object after you have keyed it in.

```
« OBJECT »
```

The program "code" itself, as it appears if you `RCL` it in `STD` display mode.

Summary:	A brief description of the purpose and logic of the routine, giving the equation(s) the routine is based on – and the proper states of the stack and flags.
Example(s):	One or more quick, simple examples to give you the general idea of how to use it.
Inputs:	A list of all acceptable types of input objects.
Outputs:	A list of the various possible types of output objects.
Units:	Any assumptions made about physical units.
Errors:	A list of things that could go wrong if you give bad inputs, have incorrect flag settings, etc.
Notes:	Anything else you might need to know.

Notes (Yours)



Proofread A Named Object:

CKSM (1040278)

```
« 2 32 ^ RCLF → N F
S « N RCL STD HEX 64
STWS 45 SF 48 CF
→STR N →STR + 0 1 3
PICK SIZE FOR I OVER
I DUP SUB NUM I * +
DUP F MOD SWAP F /
IP + NEXT SWAP DROP
S STOF » »
```

Summary: CKSM (checksum) checks for "typos" by computing a unique integer (which should match the value given) for a named object.

Inputs: Level 1 – the name of an object.

Outputs: Level 1 – an integer – the checksum.

Errors: Bad Argument Type occurs if the input is not a name. Undefined Name occurs if the input name is undefined.

Notes: CKSM is *optional but highly recommended,* and is most generally useful in the HOME directory.

Convert Capacitance To Impedance:

C→Z (18662)

```
« → C '-i/(ω*C)' »
```

Summary: This routine uses the equation

$$Z = R + \mathbf{i}\left[\omega L - \frac{1}{\omega C}\right]$$

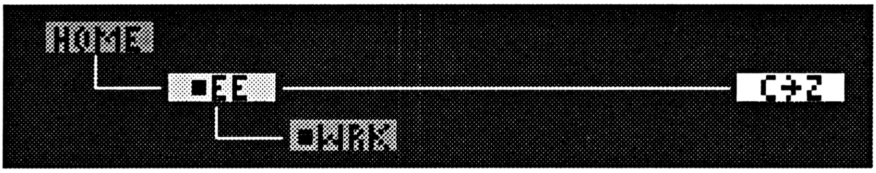
where $\mathbf{i} = \sqrt{-1}$, and ω is the sinusoidal, angular frequency in radians/second. C→Z returns the impedance, Z, for the specified capacitance, C, which it takes from Level 1 of the stack.

Examples:

Problem: Convert a 5-pf capacitance to an impedance.
Solution: `2 SCI 'ω' PURGE 5E-12 C→Z`
Result: `'-(i/(ω*5.00E-12))'` (if flags 35 and 36 are both set).

Problem: What is the impedance of the capacitor in the previous problem if the circuit has an angular, radian frequency of 377 radians/second?
Solution: Assuming the result of the previous example is on the stack: `377 'ω' STO EVAL.`
Result: `'-(i/1.89E-9)'`

Problem: Convert the symbolic capacitor value `'CA'` to symbolic impedance.
Solution: `'ω' PURGE 35 SF 36 SF 'CA' C→Z`
Result: `'-(i/(ω*CA))'`



Inputs: Level 1 – a real or complex number (in rectangular format only), an algebraic object, or a name object – the capacitance value, C.

Outputs: Level 1 – a complex number or an algebraic expression – the impedance value, Z. The type of object output depends on four factors: (i) the type of the input object; (ii) the status of flag 35 (constants mode); (iii) the status of flag 36 (results mode); (iv) the contents of `'w'`.

Units: ω must be given in radians/second. C given in farads (f), will return a Z in ohms (Ω). Any other coherent, internally consistent unit system will work, also.

Errors: `Undefined Name` will occur if either an input name or `'w'` is undefined when flag 36 is clear. `Infinite Result` will occur if C = `0` (or `(0,0)`) and flag 59 (infinite result action) is set.

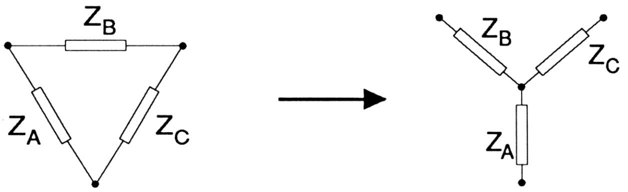
Notes: Since the HP-28S has no ω character, lowercase `w` is used instead. If a strictly numeric result is desired, `'w'` must contain the appropriate angular frequency for the circuit element. Note that if `'w'` is accessible (i.e. at the same level or "upward" on the memory diagram) from the current directory, the defined value of `w` will then be used in the calculations. Flag 45 (multi-line display mode) will affect the appearance of complex and algebraic results. If flag 35 (constants mode) is clear, `i` will be replaced by `(0,1)` in symbolic results.

Convert A "Δ" Impedance Configuration To A "Y" Impedance Configuration:

D→Y (123730)

```
« 3 DUPN + + → A B C
T « A C * T / A B *
T / B C * T / » »
```

Summary: D→Y converts a three-element "Δ" network to an equivalent "Y" network, as shown here:



D→Y expects the three "Δ" impedances to be put onto the stack in *clockwise order*. The resultant "Y" impedances will be returned in clockwise order, also.

Example: Problem: Given a "Δ" network with impedances $Z_A = 4$, $Z_B = 2+i5$, and $Z_C = i3$, find an equivalent "Y" configuration.

Solution: `2 FIX 4 '2+i*5' 'i*3' D→Y →NUM`

Result: `(-0.42,1.56)` (this is Z_{YC})

Then: `DROP →NUM`

Result: `(2.08,0.56)` (this is Z_{YB})

Then: `DROP →NUM`

Result: `(0.96,0.72)` (this is Z_{YA})

Inputs: Level 3 – a real or complex number (in rectangular format), or an algebraic or a name object – the impedance value, Z_A.
Level 2 – any type allowed for Level 3 – the impedance value, Z_B.
Level 1 – any type allowed for Level 3 – the impedance value, Z_C.

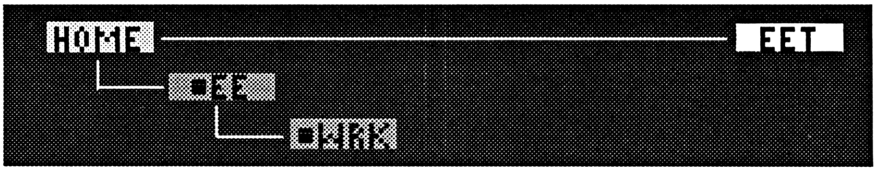
Outputs: Level 1 – the impedance value, Z_{YC}.
Level 2 – the impedance value, Z_{YB}.
Level 3 – the impedance value, Z_{YA}.

The types of the output objects depends on the types of input objects and the states of flag 35 (constants mode) and flag 36 (results mode).

Units: Output units will match the input units only if input units are all identical. Ohms are conventional, of course.

Errors: `Infinite Result` will occur if the sum of the input impedances is zero and flag 59 (infinite result action) is set. `Undefined Result` will occur if *all* inputs are zero. `Undefined Name` will occur if an input name is undefined and flag 36 (results mode) is clear.

Notes: None to speak of.



Create Circuit Reduction Tools Menu:

EET (19328)

```
« HOME ▪EE TOOLS »
```

Summary: EET moves to the ▪WRK directory and it creates a custom menu containing the Circuit Reduction Tools. See TOOLS for more details.

Example: None necessary.

Inputs: None.

Outputs: None.

Units: Not applicable.

Errors: EET will not work properly unless the directory ▪WRK has already been created (▪WRK is subordinate to ▪EE, which, in turn, is subordinate to HOME, as shown in the diagram above).

Notes: Put EET into the HOME directory ("EET phone HOME"), so that you can always begin using your Circuit Reduction Tools on a moment's notice.

Notes (Yours)

Convert Hz. To Radians/Second:

F→ω (15381)

```
« → F '2*F*π' »
```

Summary: This routine uses the equation $\omega = 2\pi F$

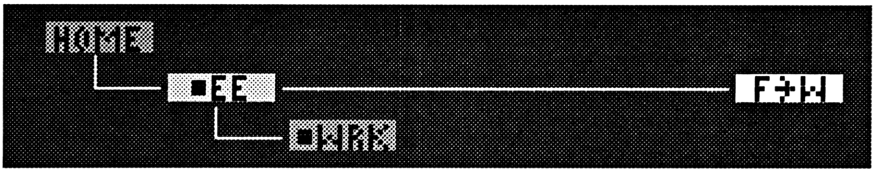
where ω is the angular frequency in radians/second, corresponding to F, which is the sinusoidal frequency in cycles/second (Hz.). F→ω returns ω for the specified F, which it takes from Level 1 of the stack.

Example: Problem: Convert 100 Hz to radians/second.
Solution: `STD 100 F→ω`
Result: `'200*π'` (if flags 35 and 36 are both set.)
Then: `→NUM` (unnecessary if flag 35 were clear.)
Result: `628.318530718`

Inputs: Level 1 – a real number, an algebraic object, or a name object – the frequency value, F, in Hertz. An algebraic or name-type input must be ultimately reducible to a real number.

Outputs: Level 1 – a real number or an algebraic expression – the angular frequency, ω, in radians/second.

The type of output object generated depends on three factors: (i) the type of input object; (ii) the status of flag



35 (constants mode); (iii) the status of flag 36 (results mode). Depending on these conditions, F→ω will return different outputs, as follows:

If flag 35 is set, a resultant algebraic expression will contain 'π'. If not, the result uses a numerical approximation of π.

If flag 36 is set and the input object containes name objects, the resulting object will also contain name objects — that is, the input name objects will not be evaluated. But if flag 36 is clear, any input name objects are evaluated and replaced with their numeric contents in the output object.

Units: If F is given in Hertz, ω will be returned in radians/second. Any other internally consistent set of units will work, also.

Errors: `Undefined Name` will occur if an input name is undefined and flag 36 (results mode) is clear.

Notes: Since the HP-28S has no ω character, use ω instead.

Distribute Current Through Two Parallel Impedances:

IDIST (95390)

```
« → A B I « A B + B
I * OVER / A I * ROT
/ » »
```

Summary: Given two parallel impedances and an input current, IDIST finds the current through each impedance.

Examples: Problem: A total current of 2∠62° flows through two parallel impedances, Z_A = (10,37.7) and Z_B = (0,-265). Find the current through each impedance.

Solution: `DEG 2 FIX (10,37.7) (0,-265) (2,62) P→R IDIST R→P`

Result: `(0.34,-135.37)` (I_B = 0.34∠135.37°)

Then: `SWAP R→P`

Result: `(2.33,59.48)` (I_A = 2.33∠59.48°)

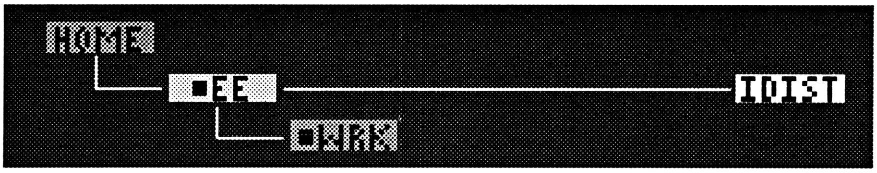
Problem: A current of I amperes flows through a network of 2 parallel impedances, Z_A and Z_B. Find the current through each impedance.

Solution: `'ZA' 'ZB' 'I' IDIST`

Result: `'ZA*I/(ZA+ZB)'` (This is I_B)

Then: `SWAP`

Result: `'ZB*I/(ZA+ZB)'` (This is I_A)



Inputs: Level 3 – a real or complex number (rectangular format only), an algebraic object or a name object – the first impedance value, Z_A.
Level 2 – any type allowable at Level 3 – the second impedance value, Z_B.
Level 1 – any type allowable at Level 3 – the total current value, I.

Outputs: Level 1 – a complex number (rectangular format only) or an algebraic expression – the current value, I_B.
Level 2 – a complex number (rectangular format only) or an algebraic expression – the current value, I_A.

The types of output objects generated depend on: (i) the input object types; (ii) the status of flag 35 (constants mode); (iii) the status of flag 36 (results mode).

Units: Use internally consistent units (e.g. ohms for impedances and amperes for currents).

Errors: `Infinite Result` will occur if the sum of the impedances is `0` or `(0,0)` and flag 59 (infinite result action) is set. `Undefined Name` will occur if an input name is undefined and flag 36 (results mode) is clear.

Notes: None to speak of.

Distribute Current Through N Parallel Impedances:

IDSTN (454624)

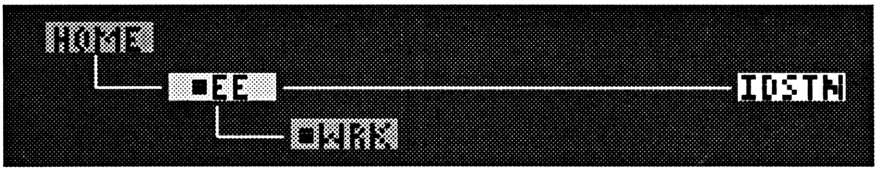
```
« → I N « N DUPN 1 N
1 - START ZADDP NEXT
I * → T « 1 N START
T SWAP / N ROLL NEXT
» » »
```

Summary: Given N parallel impedances and an input current, IDSTN finds the current through each impedance.

Examples: Problem: A total of 10 amperes of current flows through a network of 4 parallel impedances with the following values: Z_A = 1 ohm; Z_B = 2 ohms; Z_C = 3 ohms; and Z_D = 4 ohms. Find the current through each of these impedances.

Solution: `2 FIX 1 2 3 4 10 4 IDSTN`
Result: `1.20` (This is I_D)
Then: `DROP`
Result: `1.60` (This is I_C)
Then: `DROP`
Result: `2.40` (This is I_B)
Then: `DROP`
Result: `4.80` (This is I_A)

Inputs: Levels N+2 through Level 3 – real or complex numbers (rectangular format only), or algebraic or name objects – the impedance values, Z_A through Z_N, respectively.



Level 2 – a real or complex number (rectangular format only), or an algebraic or name object – the total current value, I.
Level 1 – a real number – the number of impedances, N.

Outputs: Levels 1 through N – complex numbers (rectangular format) or algebraic expressions – the current values, I_N through I_A, respectively.

The types of output objects generated depend on: (i) the types of the input objects; (ii) the status of flag 35 (constants mode); (iii) the status of flag 36 (results mode).

Units: Use internally consistent sets of units (e.g. ohms for impedances and amperes for currents).

Errors: `Infinite Result` will occur if any of the impedances is `0` (or `(0,0)`) and flag 59 (infinite results action) is set. `Undefined Name` will occur if an input name is undefined and flag 36 (results mode) is clear.

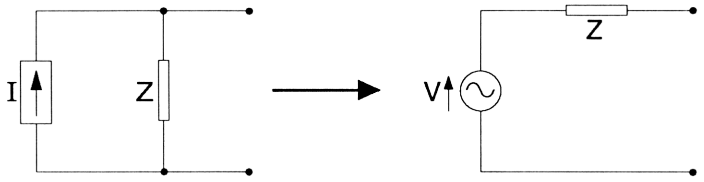
Notes: `IDSTN` uses (and therefore requires) `ZADDP`.

Convert A Parallel IZ Source To A Series VZ Source:

IZ→VZ (35969)

```
« → I Z « I Z * Z »
»
```

Summary: This routine converts from a parallel-connected current source and impedance to a series-connected voltage source and impedance, as shown below:



Example: Problem: Convert a current source, I, and a parallel impedance, Z, to an equivalent voltage source with a series impedance.

Solution: `'I' 'Z' IZ→VZ`

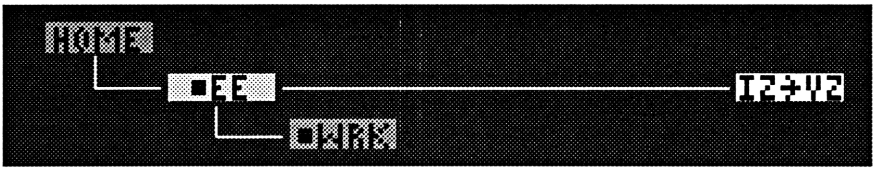
Result: `'Z'` (the series impedance value)

Then: `SWAP`

Result: `'I*Z'` (the voltage source value)

Inputs: Level 2 – a real or complex number (rectangular format), or an algebraic or name object – The current source value, I.

Level 1 – a real or complex number (rectangular format), or an algebraic or name object – the parallel impedance value, Z.



Outputs: Level 1 – a complex number (rectangular format), or an algebraic expression – the series impedance value, Z.
Level 2 – a complex number (rectangular format), or an algebraic expression – the voltage source value, V.

The types of the output objects depend on the types of input objects and the states of flag 35 (constants mode) and flag 36 (results mode).

Units: Use internally consistent units (e.g. amperes for currents, ohms for impedances, and volts for voltages).

Errors: `Undefined Name` will occur if an input name is undefined and flag 36 (results mode) is clear.

Notes: None to speak of.

Convert Inductance To Impedance:

L→Z (17557)

```
« → L 'i*(w*L)' »
```

Summary: This routine uses the equation

$$Z = R + \mathbf{i}\left[\omega L - \frac{1}{\omega C}\right]$$

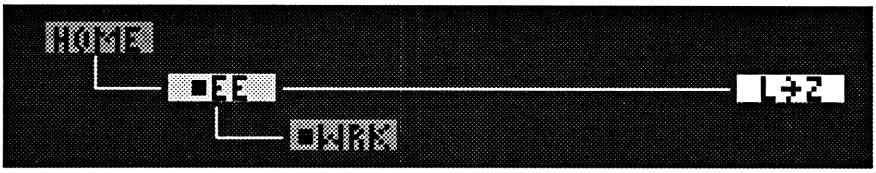
where $\mathbf{i} = \sqrt{-1}$, and ω is the sinusoidal, angular frequency in radians/second. L→Z returns the impedance, Z, for the specified inductance, L, which it takes from Level 1 of the stack.

Examples:

Problem: Convert a 0.1-h inductor to an impedance.
Solution: `STD 'w' PURGE .1 L→Z`
Result: `'i*(w*.1)'` (if flags 35 and 36 are both set).

Problem: What is the impedance of the inductor in the previous problem if the circuit has an angular, radian frequency of 377 radians/second?
Solution: Assuming the result of the previous example is on the stack: `377 'w' STO EVAL`.
Result: `'i*37.7'`

Problem: Convert the symbolic inductor value L_A to a symbolic impedance.
Solution: `'w' PURGE 35 SF 36 SF 'LA' L→Z`
Result: `'i*(w*LA)'`



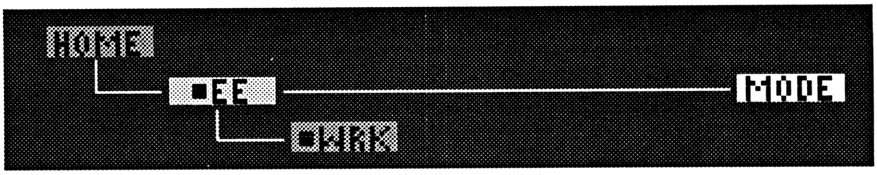
Inputs: Level 1 – a real or complex number (in rectangular format only), an algebraic object, or a name object – the inductance value, L.

Outputs: Level 1 – a complex number or an algebraic expression – the impedance value, Z. The type of object output depends on four factors: (i) the type of the input object; (ii) the status of flag 35 (constants mode); (iii) the status of flag 36 (results mode); (iv) the contents of 'w'.

Units: ω must be given in radians/second. L given in henrys (h), will return a Z in ohms (Ω). Any other coherent, internally consistent unit system will work, also.

Errors: `Undefined Name` will occur if either an input name or 'w' is undefined when flag 36 (results mode) is clear.

Notes: Since the HP-28S has no ω character, lowercase w is used instead. If a strictly numeric result is desired, 'w' must contain the appropriate angular frequency for the circuit element. Note that if 'w' is accessible (i.e. at the same level or "upward" on the memory diagram) from the current directory, the defined value of w will be then used in the calculations. Flag 45 (multiline display mode) will affect the appearance of complex and algebraic results. If flag 35 (constants mode) is clear, i will be replaced by (0,1) in symbolic results.



Generate the MODE Menu:

MODE (578548)

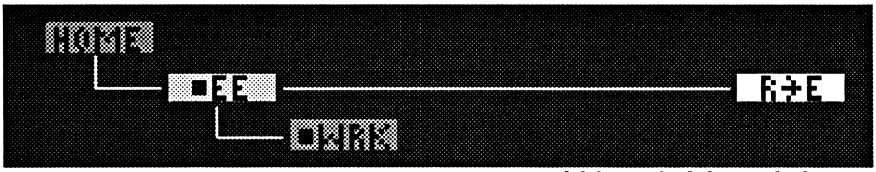
```
« 36 FS? { SRES■ } {
SRES } IFTE 35 FS? {
SCON■ } { SCON }
IFTE + 34 FS? { SSOL
} { SSOL■ } IFTE +
'TOOLS' + MENU ■EE
■WRK »
```

Summary: MODE creates a small custom utility menu useful for setting certain HP-28S system modes that affect the tools described in this book. Since it is a menu-creation routine, it is not meaningful to talk about inputs, outputs, or units.

Example: None needed.

Errors: None.

Notes: MODE uses (and therefore requires the presences of) SRES■, SRES, SCON■, SCON, SSOL, SSOL■, and the ■EE and ■WRK menus.



Convert Two Real Numbers To 'M*e^(i*α)':

R→e (266109)

```
« →NUM SWAP →NUM
SWAP R→C C→R RCLF 36
SF 'i' ROT * 'e'
SWAP ^ ROT SWAP *
SWAP STOF »
```

Summary: R→e converts two real numbers into an expression of the form 'M*e^(i*α)', where M is the magnitude and α is the angle of the complex vector.

Example: Problem: Key in the number $5e^{i0.93}$.
Solution: 2 FIX 5 .93 R→e
Result: '5*e^(i*0.93)'

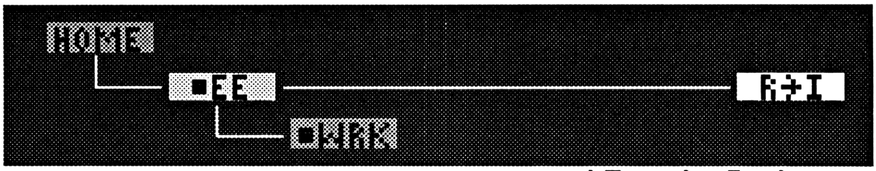
Inputs: Level 2 – a real number – the magnitude, M.
Level 1 – a real number – the angle, α, *in radians.*

Outputs: Level 1 – an algebraic object – the complex expression.

Units: Real numbers themselves are unitless.

Errors: Bad Argument Type occurs for non-real inputs.

Notes: The angle (α) is *always* expressed in radians.



Convert Two Real Numbers To 'Re+i*Im':

R→i (293698)

```
« →NUM SWAP →NUM
SWAP R→C C→R RCLF 36
SF ROT ROT 'i' OVER
SIGN * SWAP ABS * +
SWAP STOF »
```

Summary: R→i converts two real numbers into an expression of the form 'Re+i*Im', where Re is the real portion and Im is the imaginary portion of the complex vector.

Example: Problem: Key in the number 3+i4.
Solution: 2 FIX 3 4 R→i
Result: '3+i*4'

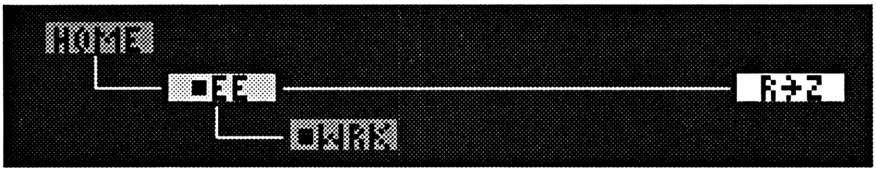
Inputs: Level 2 – a real number – the real portion, Re.
Level 1 – a real number – the imaginary portion, Im.

Outputs: Level 1 – an algebraic object – the complex expression.

Units: Real numbers themselves are unitless.

Errors: Bad Argument Type occurs for non-real inputs.

Notes: None to speak of.



Convert Resistance To Impedance:

R→Z (9842)

```
« → R 'R' »
```

Summary: R→Z uses the equation $Z = R + \mathbf{i}\left[\omega L - \frac{1}{\omega C}\right]$

(where $\mathbf{i} = \sqrt{-1}$, and ω is the sinusoidal, angular frequency in radians/second) to return the impedance, Z, for the specified resistance, R.

Example: Problem: Convert a 100Ω resistance to an impedance.
Solution: `STD 100 R→Z`
Result: `100`

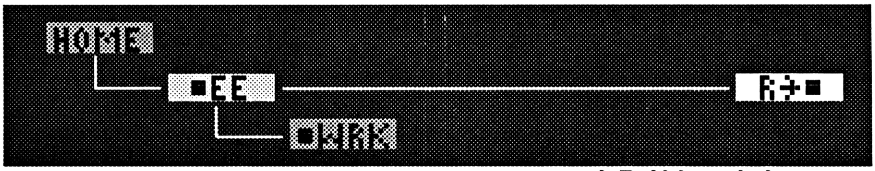
Inputs: Level 1 – any type of data object – the resistance, R.

Outputs: Level 1 – the input object returns unchanged.

Units: Z will be returned in the same units as the input, R.

Errors: None.

Notes: R→Z performs no real function; it is included here for completeness.



Convert Two Real Numbers To '∠(M,α)':

R→∠ (192694)

```
« →NUM SWAP →NUM
SWAP R→C RCLF STD
"'∠" ROT →STR + STR→
SWAP STOF »
```

Summary: R→∠ converts two real numbers to a complex expression – in the polar form '∠(M,α)', where M is the magnitude and α is the angle of the complex vector.

Example: Problem: Key in the number 5∠53.13°.
Solution: `2 FIX 5 53.13 R→∠`
Result: `'∠(5,53.13)'`

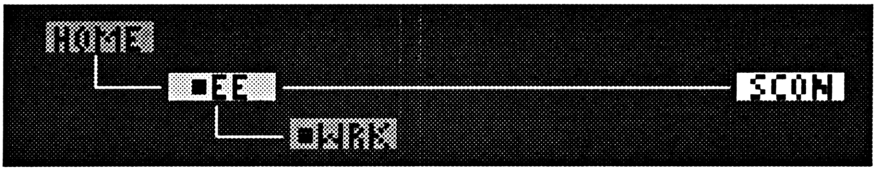
Inputs: Level 2 – a real number – the magnitude, M.
Level 1 – a real number – the angle, α, *in degrees.*

Outputs: Level 1 – an algebraic object – the complex expression.

Units: Real numbers themselves are unitless.

Errors: `Bad Argument Type` occurs for non-real inputs.

Notes: The angle (α) is *always* in degrees. R→∠ uses (and therefore requires the presence of) the function ∠.



Set Symbolic Constants Mode:

SCON (22677)

```
« 35 SF 36 SF MODE »
```

Summary: SCON is a selection on the MODE custom utility menu. SCON is useful for quickly setting the HP-28S symbolic constants (and symbolic results) mode, which affects the tools described in this book. The setting/clearing of this mode shows in the custom MODE menu by the appearance/disappearance of the ▪ character after the name of the mode – just as in the HP-28's built-in MODE menu.

Example: None needed.

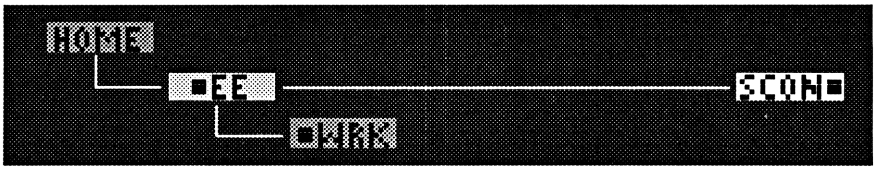
Inputs: None.

Outputs: None.

Units: Not applicable.

Errors: None.

Notes: SCON uses (and therefore requires the presence of) MODE. Since symbolic constants mode requires symbolic results mode, both of those flags are set.



Clear Symbolic Constants Mode:

SCON▫ (16805)

```
« 35 CF MODE »
```

Summary: SCON▫ is a selection on the MODE custom utility menu. SCON▫ is useful for quickly clearing the HP-28S symbolic constants mode, which affects the tools described in this book. The setting/clearing of this mode shows in the custom MODE menu by the appearance/disappearance of the ▫ character after the name of the mode – just as in the HP-28's built-in MODE menu.

Example: None needed.

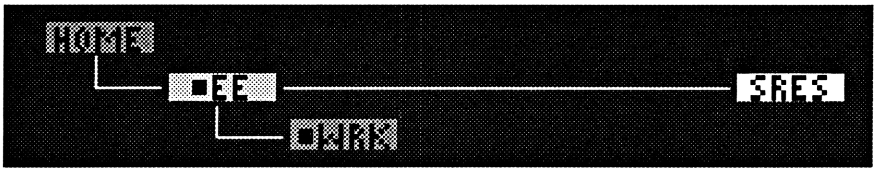
Inputs: None.

Outputs: None.

Units: Not applicable.

Errors: None.

Notes: SCON▫ uses (and therefore requires the presence of) MODE.



Set Symbolic Results Mode:

SRES (14136)

```
« 36 SF MODE »
```

Summary: SRES is a selection on the MODE custom utility menu. SRES is useful for quickly setting the HP-28S symbolic results mode, which affects the tools described in this book. The setting/clearing of this mode shows in the custom MODE menu by the appearance/disappearance of the ▫ character after the name of the mode – just as in the HP-28's built-in MODE menu.

Example: None needed.

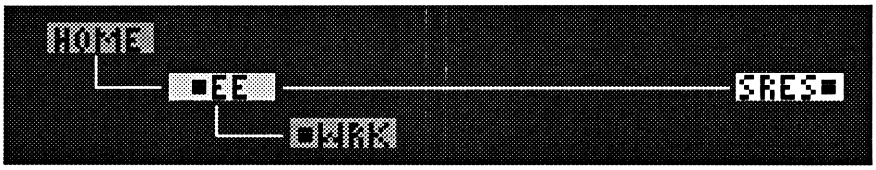
Inputs: None.

Outputs: None.

Units: Not applicable.

Errors: None.

Notes: SRES uses (and therefore requires the presence of) MODE.



Clear Symbolic Results Mode:

SRES▫ (16979)

```
« 36 CF MODE »
```

Summary: SRES▫ is a selection on the MODE custom utility menu. SRES▫ is useful for quickly clearing the HP-28S symbolic results mode, which affects the tools described in this book. The setting/clearing of this mode shows in the custom MODE menu by the appearance/disappearance of the ▫ character after the name of the mode – just as in the HP-28's built-in MODE menu.

Example: None needed.

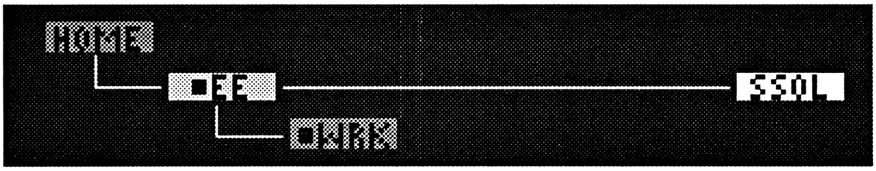
Inputs: None.

Outputs: None.

Units: Not applicable.

Errors: None.

Notes: SRES▫ uses (and therefore requires the presence of) MODE.



Set Symbolic Solutions Mode:

SSOL (14096)

```
« 34 CF MODE »
```

Summary: SSOL is a selection on the MODE custom utility menu. SSOL is useful for quickly setting the HP-28S symbolic solutions mode, which affects the tools described in this book. The setting/clearing of this mode shows in the custom MODE menu by the appearance/disappearance of the ▫ character after the name of the mode – just as in the HP-28's built-in MODE menu.

Example: None needed.

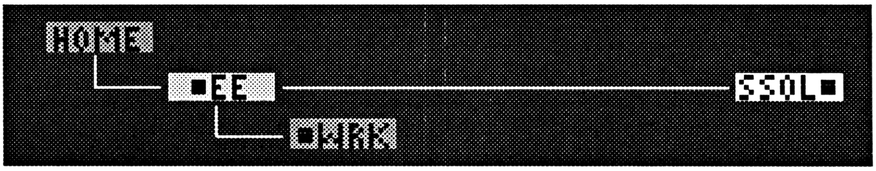
Inputs: None.

Outputs: None.

Units: Not applicable.

Errors: None.

Notes: SSOL uses (and therefore requires the presence of) MODE.



Clear Symbolic Constants Mode:

SSOL▫ (17131)

```
« 34 SF MODE »
```

Summary: SSOL▫ is a selection on the MODE custom utility menu. SSOL▫ is useful for quickly clearing the HP-28S symbolic constants mode, which affects the tools described in this book. The setting/clearing of this mode shows in the custom MODE menu by the appearance/disappearance of the ▫ character after the name of the mode – just as in the HP-28's built-in MODE menu.

Example: None needed.

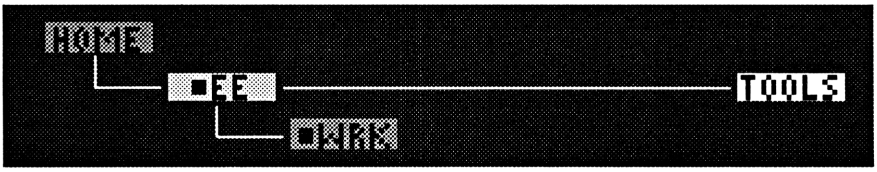
Inputs: None.

Outputs: None.

Units: Not applicable.

Errors: None.

Notes: SSOL▫ uses (and therefore requires the presence of) MODE.



Generate the TOOLS Menu:

TOOLS (665789)

```
« { R→i R→° R→e →i
→° →e R→Z L→Z C→Z
F→ω ω→F ω ZADDS
ZADDP IDIST VDIST
IDSTN VDSTN VZ→IZ
IZ→VZ Y→D D→Y MODE }
MENU °EE °WRK »
```

Summary: TOOLS creates a menu of all the basic Circuit Reduction Tools, then moves to the °WRK menu, which is subordinate to ("below") the °EE menu. Thus °WRK has access to TOOLS.

Example: None needed.

Errors: None.

Notes: TOOLS uses (and therefore requires the presences of) R→i, R→°, R→e, →i, →°, →e, R→Z, L→Z, C→Z, F→ω, ω→F, ω, ZADDS, ZADDP, IDIST, VDIST, IDSTN, VDSTN, VZ→IZ, IZ→VZ, Y→D, D→Y, MODE, and the °EE and °WRK menus. Note that ω itself is an item in this menu, to allow convenient storage into and purging of that variable.

Distribute Voltage Across Two Series Impedances:

VDIST (96963)

```
« → A B V « A B + A
V * OVER / B V * ROT
/ » »
```

Summary: Given two series impedances and an input voltage, VDIST finds the voltage across each impedance.

Examples:

Problem: A total voltage of 2∠62° is applied across two impedances connected in series, Z_A = (10,37.7) and Z_B = (0,-265). Find the voltage across each impedance.

Solution: `DEG 2 FIX (10,37.7) (0,-265) (2,62) P→R VDIST R→P`

Result: `(2.33,59.48)` (V_B = 2.33∠59.48°)

Then: `SWAP R→P`

Result: `(0.34,-135.37)` (V_A = 0.34∠-135.37°)

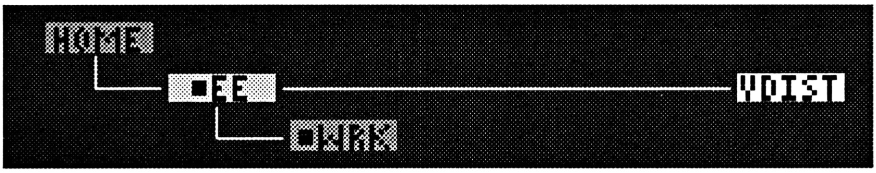
Problem: A voltage of V volts is applied across a series of 2 impedances, Z_A and Z_B. What is the voltage across each impedance?

Solution: `'ZA' 'ZB' 'V' VDIST`

Result: `'ZB*V/(ZA+ZB)'` (This is V_B)

Then: `SWAP`

Result: `'ZA*V/(ZA+ZB)'` (This is V_A)



Inputs: Level 3 – a real or complex number (rectangular format only), an algebraic object or a name object – the first impedance value, Z_A.
Level 2 – any type allowable at Level 3 – the second impedance value, Z_B.
Level 1 – any type allowable at Level 3 – the total voltage value, V.

Outputs: Level 1 – a complex number (rectangular format only) or an algebraic expression – the voltage value, V_B.
Level 2 – a complex number (rectangular format only) or an algebraic expression – the voltage value, V_A.

The types of output objects generated depend on: (i) the input object types; (ii) the status of flag 35 (constants mode); (iii) the status of flag 36 (results mode).

Units: Use internally consistent units (e.g. ohms for impedances and amperes for currents).

Errors: `Infinite Result` will occur if the sum of the impedances is `0` or `(0,0)` and flag 59 (infinite result action) is set. `Undefined Name` will occur if an input name is undefined and flag 36 (results mode) is clear.

Notes: None to speak of.

Distribute Voltage Across N Series Impedances:

VDSTN (421047)

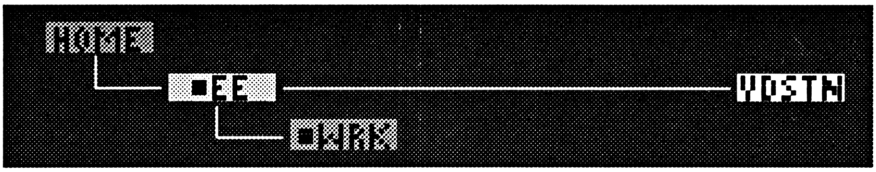
```
« → V N « N DUPN 1 N
1 - START + NEXT V
SWAP / → T « 1 N
START T * N ROLL
NEXT » » »
```

Summary: Given N series impedances and an input voltage, VDSTN finds the voltage across each impedance.

Examples: Problem: A total of 15 volts is applied across a series of 4 series impedances with these values: Z_A = 1 ohm; Z_B = 2 ohms; Z_C = 3 ohms; and Z_D = 4 ohms. Find the voltage across each of these impedances.

Solution:	`2 FIX 1 2 3 4 15 4 VDSTN`
Result:	`6.00` (This is V_D)
Then:	`DROP`
Result:	`4.50` (This is V_C)
Then:	`DROP`
Result:	`3.00` (This is V_B)
Then:	`DROP`
Result:	`1.50` (This is V_A)

Inputs: Levels N+2 through Level 3 – real or complex numbers (rectangular format only), or algebraic or name objects – the impedance values, Z_A through Z_N, respectively.



Level 2 – a real or complex number (rectangular format only), or an algebraic or name object – the total voltage value, V.
Level 1 – a real number – the number of impedances, N.

Outputs: Levels 1 through N – complex numbers (rectangular format) or algebraic expressions – the voltage values, V_N through V_A, respectively.

The types of output objects generated depend on: (i) the types of the input objects; (ii) the status of flag 35 (constants mode); (iii) the status of flag 36 (results mode).

Units: Use internally consistent sets of units (e.g. ohms for impedances and amperes for currents).

Errors: `Infinite Result` will occur if any of the impedances is `0` (or `(0,0)`) and flag 59 (infinite results action) is set. `Undefined Name` will occur if an input name is undefined and flag 36 (results mode) is clear.

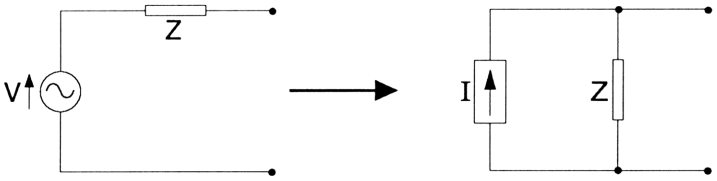
Notes: None to speak of.

Convert A Series VZ Source To A Parallel IZ Source:

VZ→IZ (36249)

```
« → V Z « V Z / Z »
»
```

Summary: This routine converts from a series-connected voltage source and impedance to a parallel-connected current source and impedance, as shown below:



Example: Problem: Convert a 50-volt voltage source, in series with an impedance of (1,-5), to an equivalent current source with a parallel impedance.

Solution: `2 FIX DEG 50 (1,-5) VZ→IZ`

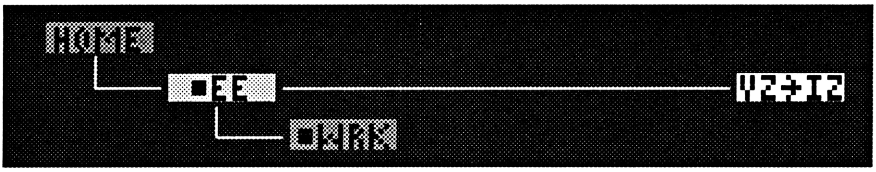
Result: `(1.00,-5.00)` (the parallel impedance)

Then: `SWAP R→P`

Result: `(9.81,78.69)` (I = 9.81∠78.69°)

Inputs: Level 2 – a real or complex number (rectangular format), or an algebraic or name object – the voltage source value, V.

Level 1 – a real or complex number (rectangular format), or an algebraic or name object – the series impedance value, Z.



Outputs: Level 1 – a complex number (rectangular format), or an algebraic expression – the parallel impedance value, Z. Level 2 – a complex number (rectangular format), or an algebraic expression – the current source value, I.

The types of the output objects depend on the types of input objects and the states of flag 35 (constants mode) and flag 36 (results mode).

Units: Use internally consistent units (e.g. amperes for currents, ohms for impedances, and volts for voltages).

Errors: `Undefined Name` will occur if an input name is undefined and flag 36 (results mode) is clear. `Infinite result` will occur if the input Z is zero and flag 59 (infinite result action) is clear.

Notes: None to speak of.

Convert Radians/Second To Hz:

ω→F (15980)

```
« → ω 'ω/2/π' »
```

Summary: This routine uses the equation $F = \omega / 2\pi$

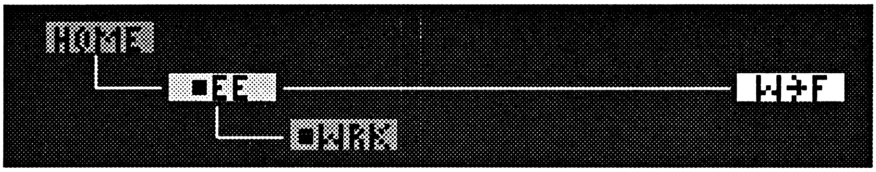
where ω is the angular frequency in radians/second, corresponding to F, which is the sinusoidal frequency in cycles/second (Hz.). ω→F returns F for the specified ω, which it takes from Level 1 of the stack.

Example: Problem: Convert 3π/2 radians/second to Hz.
Solution: `STD 3 π * 2 / ω→F`
Result: `'3*π/2/2/π'` (flags 35 and 36 both set.)
Then: `COLCT`
Result: `.75`

Alternate Solution: `3 π * 2 / ω→F →NUM`
Result: `.749999999999`

Problem: Convert (n-2)π radians/second to Hz.
Solution: `'N' 2 - π * ω→F`
Result: `'(N-2)*π/2/π'` (flags 35 and 36 set.)
Then: `COLCT`
Result: `'.5*(-2+N)'`

Inputs: Level 1 – a real number or complex number (in rectangular format only), an algebraic object, or a name object – the angular frequency value, ω, in radians/second.



An algebraic or name-type input must be ultimately reducible to a complex or real number.

Outputs: Level 1 – a real number or an algebraic expression – the frequency, F, in Hz. (cycles/second).

The type of output object generated depends on three factors: (i) the type of input object; (ii) the status of flag 35 (constants mode); (iii) the status of flag 36 (results mode). Depending on these conditions, F→W will return different outputs, as follows:

If flag 35 is set, a resultant algebraic expression will contain 'π'. If not, the result uses a numerical approximation of π.

If flag 36 is set and the input object contains name objects, the resulting object will also contain name objects — that is, the input name objects will not be evaluated. But if flag 36 is clear, any input name objects are evaluated and replaced with their numeric contents in the output object.

Units: If ω is given in radians/second, F will be returned in Hz. Any other internally consistent units will work, also.

Errors: Undefined Name will occur if an input name is undefined and flag 36 (results mode) is clear.

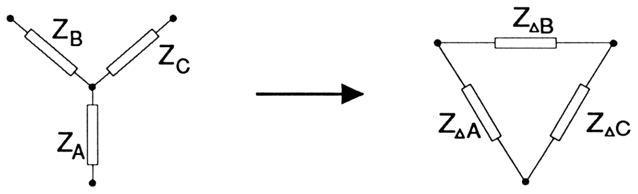
Notes: Since the HP-28S has no ω character, use W instead.

Convert A "Y" Impedance Configuration To A "Δ" Impedance Configuration:

Y→D (174067)

```
« → A B C « A B * B
C * C A * + + → T «
T C / T A / T B / »
» »
```

Summary: Y→D converts a three-element "Y" network to an equivalent "Δ" network, as shown here:



Y→D expects the three "Y" impedances to be put onto the stack in *clockwise order.* The resultant "Δ" impedances will be returned in clockwise order, also.

Example: Problem: Given a "Y" network with impedances Z_A, Z_B, and Z_C, find an equivalent "Δ" configuration.

Solution: `'ZA' 'ZB' 'ZC' Y→D`

Result: `'(ZA*ZB+(ZB*ZC+ZC*ZA))/ZB'`

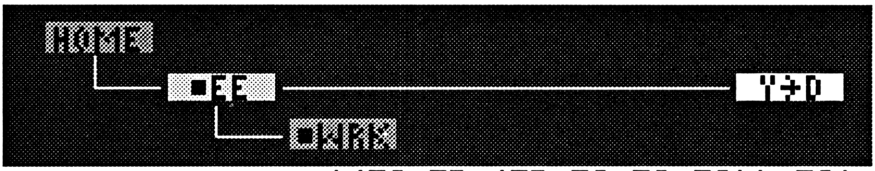
(this is $Z_{\Delta C}$)

Then: `DROP`

Result: `'(ZA*ZB+(ZB*ZC+ZC*ZA))/ZA'`

(this is $Z_{\Delta B}$)

Then: `DROP`



Result: `'(ZA*ZB+(ZB*ZC+ZC*ZA))/ZC'`
(this is $Z_{\Delta A}$)

Inputs: Level 3 – a real or complex number (in rectangular format), or an algebraic or a name object – the impedance value, Z_A.
Level 2 – any type allowed for Level 3 – the impedance value, Z_B.
Level 1 – any type allowed for Level 3 – the impedance value, Z_C.

Outputs: Level 1 – the impedance value, $Z_{\Delta C}$.
Level 2 – the impedance value, $Z_{\Delta B}$.
Level 3 – the impedance value, $Z_{\Delta A}$.

The types of the output objects depends on the types of input objects and the states of flag 35 (constants mode) and flag 36 (results mode).

Units: Output units will match the input units only if input units are all identical. Ohms are conventional, of course.

Errors: `Infinite Result` will occur if any input impedance is zero and flag 59 (infinite result action) is set. `Undefined Result` will occur if *all* inputs are zero. `Undefined Name` will occur if an input name is undefined and flag 36 (results mode) is clear.

Notes: None to speak of.

Add Two Parallel Impedances:

ZADDP (44875)

```
« → A B 'INV(INV(A)+
INV(B))' »
```

Summary: 'ZADDP' adds two parallel impedances together, yielding a single equivalent impedance, using the equation

$$Z_T = \frac{1}{\frac{1}{Z_A} + \frac{1}{Z_B}}$$

Examples:

Problem: Find the single impedance equivalent to the parallel impedances (2,3) and (1.2,4.5).

Solution: `2 FIX (2,3) (1.2,4.5) ZADDP`

Result: `(0.89,1.86)`

Problem: Find the single equivalent impedance for parallel impedances Z_A and Z_B.

Solution: `'ZA' 'ZB' ZADDP`

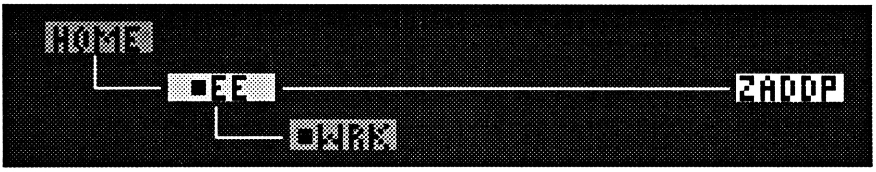
Result: `'INV(INV(ZA)+INV(ZB))'`

Problem: If, in the previous problem, Z_A were (4.3,2), and the total (equivalent) impedance were (2.99,1.25), find Z_B.

Solution: `(4.3,2) 'ZA' STO (2.99,1.25)`
`=`

Result: `'INV(INV(ZA)+INV(ZB))=`
`(2.99,1.25)'`

Then: `'ZB' ISOL EVAL`



Result: (9.69,3.12)

Inputs: Levels 2 and 1 – real or complex numbers (rectangular format only), algebraic objects, or name objects – the two parallel impedance values.

All inputs should be ultimately reducible to complex or real numbers.

Outputs: Level 1 – a complex number (rectangular format) or an algebraic expression – the total equivalent impedance value.

The output object type depends on the input object types and on the status of flag 36 (results mode): If flag 36 is set when name-type input objects are used, the output object will also contain names. Otherwise, any input names are replaced with their numeric contents in the output object.

Units: Output units will match input units (assuming that all input units are identical).

Errors: Infinite Result will occur if both impedances are 0 (or (0,0)) and flag 59 (infinite result action) is set. Undefined Name will occur if an input name is undefined and flag 36 (results mode) is clear.

Notes: None to speak of.

Add Two Series Impedances:

ZADDS (15691)

```
« → A B 'A+B' »
```

Summary: 'ZADDS' adds two series impedances together, yielding a single equivalent impedance, using the equation $Z_T=Z_A+Z_B$.

Examples:

Problem: Find the single impedance equivalent to the series impedances (2,3) and (1.2,4.5).
Solution: `1 FIX (2,3) (1.2,4.5) ZADDS`
Result: `(3.2,7.5)`

Problem: Find the single equivalent impedance for series impedances Z_A and Z_B.
Solution: `'ZA' 'ZB' ZADDS`
Result: `'ZA+ZB'`

Problem: If, in the previous problem, Z_A were (4,2), and the total (equivalent) impedance were (4.1,5), find Z_B.
Solution: `(4,2) 'ZA' STO (4.1,5) =`
Result: `'ZA+ZB=(4.1,5)'`
Then: `'ZB' ISOL EVAL`
Result: `(0.1,3.0)`

Inputs: Levels 2 and 1 – real or complex numbers (rectangular format only), algebraic objects, or name objects – the two series impedance values.

HOME
■EE
■WRK
ZADDS

All inputs should be ultimately reducible to complex or real numbers.

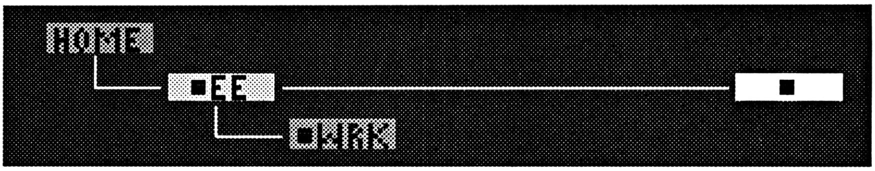
Outputs: Level 1 – a complex number (rectangular format) or an algebraic expression – the total equivalent impedance value.

The output object type depends on the input object types and on the status of flag 36 (results mode): If flag 36 is set when name-type input objects are used, the output object will also contain names. Otherwise, any input names are replaced with their numeric contents in the output object.

Units: Output units will match input units (assuming that all input units are identical).

Errors: `Undefined Name` will occur if an input name is undefined and flag 36 (results mode) is clear.

Notes: None to speak of.



Polar Format Function:

■ (91127)

```
« → M A « RCLF DEG M
A R→C P→R SWAP STOF
» »
```

Summary: ■ is an *auxiliary function* that converts two real numbers into a polar complex number of the form (M, α), where M is the magnitude and α is the angle of the complex vector. This function is used by R→■ and →■, so you'd probably use those routines rather than this low-level function directly.

Example: None needed.

Inputs: Level 2 – a real number – the magnitude, M.
Level 1 – a real number – the angle, α, *in degrees.*

Outputs: Level 1 – a complex number – in polar format.

Units: Real and complex numbers themselves are unitless.

Errors: Bad Argument Type occurs for non-real inputs.

Notes: The angle (α) is *always* in degrees. ■ must be placed in a directory at least as "high" as →■ and R→■.

Notes (Yours)

Convert (Re, Im) To 'M*e^(i*α)':

→e (27112)

```
« (1,0) * →NUM C→R
R→C RCLF 36 SF RAD
SWAP R→P C→R 'i'
SWAP * 'e' SWAP ^ *
SWAP STOF »
```

Summary: →e converts a complex number from any evaluable format to this *exponential* format: `'M*e^i*α'`, where M is the magnitude of the vector in the complex plane and α is the directional angle, *in radians.*

Examples: Problem: Find $4e^{i\pi/4} + 1.5e^{i0.32}$ – in exponential format.

Solution: `2 FIX π 4 / i * EXP 4 * RAD (1.5,.32) P→R + →e`

Result: `'5.38*e^(i*0.66)'`

Problem: Find (-1.3+**i**0.5) + (4.4-**i**2.3). Again, express the answer in `'M*e^i*α'` format.

Solution: `(-1.3,.5) '4.4-i*2.3' + →e`

Result: `'3.58*e^(i*(-0.53))'`

Inputs: Level 1 – any object that can be reduced to a real or complex number by →NUM – the number to be converted to an exponential format.

Outputs: Level 1 – an algebraic expression – the formatted exponential.

HOME
■EE
■WRK
→E

→e generates an algebraic expression that contains no unevaluated variables (`'e'` and `'i'` are symbolic constants). Thus it behaves as a *symbolic complex number,* whose angle (α) is *always* given in radians.

Certain conversions involving combinations of →e and other conversion routines will produce odd results because of rounding errors. For example, the sequence `(0,2) →e →i` returns `'8.73323846264E-12+i*2'`, instead of `'i*2'`, because →e returns `'2*e^(i*1.57079632679)'`, instead of the more accurate `'2*e^(i*π/2)'`. You can regard numbers smaller than 10^{-10} as 0; often, in fact, you can use `RND` and a proper display mode (`2 FIX` or `4 SCI`, etc.) to actually round to `0`.

Units: Complex numbers themselves are unitless.

Errors: `Bad Argument Type` will occur if an input is not reducible to a real or complex number.

Notes: `R→P` and `P→R` are other conversion commands that may be useful (and more familiar) to you in working with the numbers you frequently encounter in circuit analysis. But be careful! HP-28S system commands (except `P→R` and some of these conversion tools) expect complex numbers to be in – or reduce to – *rectangular* form. If you use any of those system commands on complex numbers in *polar* form, you'll get incorrect results!

Convert (Re, Im) To 'Re+i*Im':

→i (224825)

```
« (1,0) * →NUM C→R
RCLF 36 SF ROT ROT '
i' OVER SIGN * SWAP
ABS * + SWAP STOF »
```

Summary: →i converts a complex number from any evaluable format to a *rectangular algebraic* format: 'Re+i*Im', where Re is the real portion and Im is the imaginary portion of the complex vector.

Examples: Problem: Find $4e^{i\pi/4} + 1.5e^{i0.32}$ – in rectangular algebraic format.

Solution: `2 FIX π 4 / i * EXP 4 * RAD (1.5,.32) P→R + →i`

Result: `'4.25+i*3.30'`

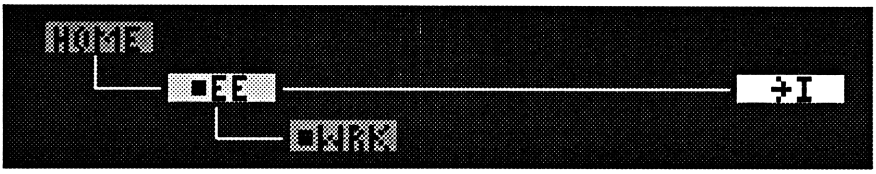
Problem: Find (-1.3+i0.5) + (4.4-i2.3). Again, express the answer in 'Re+i*Im' format.

Solution: `(-1.3,.5) '4.4-i*2.3' + →i`

Result: `'3.10-i*1.80'`

Inputs: Level 1 – any object that can be reduced to a real or complex number by →NUM – the number to be converted to an exponential format.

Outputs: Level 1 – an algebraic expression – the rectangular algebraic format.



→i generates an algebraic expression that contains no unevaluated variables (' i ' is a symbolic constant). It behaves, therefore, as a *symbolic complex number*.

Certain conversions involving combinations of →i and other conversion routines will produce odd results because of rounding errors. For example, the sequence (0,2) →e →i returns '8.73323846264E-12+i*2', instead of 'i*2', because →e returns '2*e^(i*1.57079632679)', instead of the more accurate '2*e^(i*π/2)'. You can regard numbers smaller than 10^{-10} as 0; often, in fact, you can use RND and a proper display mode (2 FIX or 4 SCI, etc.) to actually round to 0.

Units: Complex numbers themselves are unitless.

Errors: Bad Argument Type will occur if an input is not reducible to a real or complex number.

Notes: R→P and P→R are other conversion commands that may be useful (and more familiar) to you in working with the numbers you encounter in circuit analysis. But be careful! HP-28S system commands (except P→R and some of these conversion tools) expect complex numbers to be in – or reduce to – *rectangular* form. If you use any of those system commands on complex numbers in *polar* form, you'll get incorrect results!

Convert (Re, Im) To '∡(M, α)':

→∡ (244542)

```
« (1,0) * →NUM C→R
R→C RCLF SWAP DEG
STD R→P →STR "'∡"
SWAP + STR→ SWAP
STOF »
```

Summary: →∡ converts a complex number from any evaluable format to this *polar* format: '∡(M, α)', where M is the magnitude of the vector in the complex plane and α is the directional angle, in degrees.

Examples: Problem: Find $4e^{i\pi/4} + 1.5e^{i0.32}$ – in this polar format.

Solution: `2 FIX π 4 / i * EXP 4 * RAD (1.5,.32) P→R + →∡`

Result: `'∡(5.38,37.82)'`

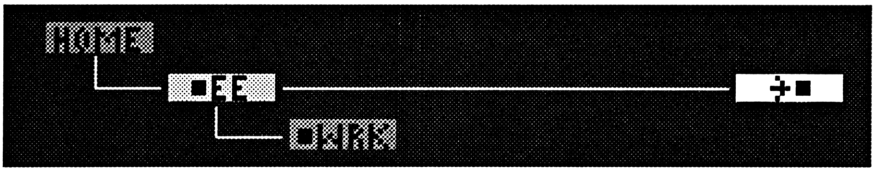
Problem: Find (-1.3+i0.5) + (4.4-i2.3). Again, express the answer in '∡(Z, α)' format.

Solution: `(-1.3,.5) '4.4-i*2.3' + →∡`

Result: `'∡(3.58,-30.14)'`

Inputs: Level 1 – any object that can be reduced to a real or complex number by →NUM – the number to be converted to a polar format.

Outputs: Level 1 – an algebraic expression – the polar format.



→▫ generates an algebraic expression that contains no unevaluated variables. Thus it behaves as a *symbolic complex number,* whose angle (α) is *always* given in degrees.

Certain conversions involving combinations of →▫ and other conversion routines will produce odd results because of rounding errors. For example, the sequence `(0,2) →e →▫` gives `'▫(2,89.9999999997)'`, instead of `'▫(2,90)'`. This is because `→e` returns `'2*e^(i*1.57079632679)'`, instead of the expected `'2*e^(i*π/2)'`.

Units: Complex numbers themselves are unitless.

Errors: `Bad Argument Type` will occur if an input is not reducible to a real or complex number.

Notes: `R→P` and `P→R` are other conversion commands that may be useful (and more familiar) to you in working with the numbers you frequently encounter in circuit analysis. But be careful! HP-28S system commands (except `P→R` and some of these conversion tools) expect complex numbers to be in – or reduce to – *rectangular* form. If you use any of those system commands on complex numbers in *polar* form, you'll get incorrect results!

Chapter 3

CIRCUIT REDUCTION TOOLS:

Discussion

This Chapter contains more examples, background and detailed discussions on using the Circuit Reduction Tools.

For the actual listings of these program tools, see Chapter 2 (page 12).

A reminder about the keystroke solutions you'll see in this section and throughout this book:

As usual with the HP-28S, there are many ways to key in solutions to problems, and it's just impossible to show every method. This book simply cannot "read your mind" to know which menu or directory you're currently using when you want to use one of these tools – so it can't give you the most convenient set of keystrokes for your particular case.

In all these solutions, therefore, rather than specifically telling you to press a *key* (e.g. ■[PURGE] or [DROP]) or to select a *menu item* (e.g. R→C or P→R), you'll see all commands in their "generic" form (spelled out as if you had typed them in):

```
PURGE DROP

R→C P→R
```

etc.

So just keep in mind that, depending upon what you're doing, it may well be that instead of typing, you could more conveniently use a key or a menu item instead.

Now then – on with the discussions....

Symbolic vs. Numeric Calculations

The HP-28S is not just a calculator. If you wanted merely a machine to perform arithmetic, you probably wouldn't spend the time or money on something so sophisticated. This machine is also a *symbolic equation solver.* That means you must often consider not only how to *perform* a calculation, but how the *forms* of the input and output and the current *mode* of calculation will affect your result.

For example, there's no mystery about what should happen when you do want to do simple arithmetic (say, add 1 and 2, like so: 1 2 +). You should get one real-valued result, 3. But what if you want to add the number 5 to the number Z_A (like so: ZA 5 +)? What kind of answer do you want – a *numeric* or a *symbolic* result? Does ZA have a specific value, or is it some arbitrary variable?

To fully illustrate these questions, consider some examples:

- You want to build a symbolic expression 'A*B/(A+B)'. How do you do this?
- You have two variables 'A' and 'B', containing numbers with which to calculate. How would you calculate A x B / (A + B)?
- You have an expression, 'A*B/(A+B)', which you want to evaluate for many different values of 'A' and 'B'. Whaddya do?
- You want to find the numeric value of π+4. How?

Gad, the mind boggles with all the possibilities. But these are all quite common and valid needs you might have – *and they are perfectly reasonable things to expect your HP-28S to do.*

You must simply know the recipes.

To work confidently with your HP-28S, you need to be familiar with two of its *system flags* (internal status indicators which you can vary back and forth between two opposite states). They are: **Flag 35** (constants mode) and **Flag 36** (results mode).

Flag 35 (constants mode) determines what kind of result you'll get when any of the system constants (i, e, MAXR, MINR and π) are evaluated.

For example, when flag 35 is *set* (which you do like this: 35 SF), and you perform 'π' EVAL, you'll get the *symbolic* result: 'π'. On the other hand, when flag 35 is *clear* (35 CF) performing 'π' EVAL will give you 3.14159265359, the *nearest numerical equivalent.*

Flag 36 (results mode) determines whether functions will reduce name objects to their numeric contents.

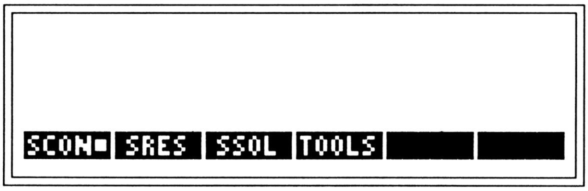
For example, when flag 36 is *set* (via 36 SF), the commands 'A' 1 + give the result 'A+1'. Setting flag 36 effectively tells functions to "leave all name objects alone." But when this flag is *clear* (36 CF) functions *will* evaluate all name objects so as to return numeric results: 2 'A' STO 'A' 1 + yields 3, for instance. Thus, if you're working with flag 36 clear, you should remember that name objects will then function as places in which to store numbers – not as abstract variables. Indeed, names that *don't* contain numeric objects will cause errors when operated on under this flag setting.

Note also that symbolic constants are simply name objects with specially reserved names. Therefore, if flag 36 is clear, the machine "has permission" when using functions to reduce *all* names to their numeric equivalents – including the specially reserved names. In effect, then, when you clear flag 36, you are overriding the setting of flag 35.

The settings of these two flags is, of course, up to you, but *you'll have the most flexibility if you generally leave them both set,* thus preserving all names and constants in your results. Remember that when you have a symbolic expression that you need to reduce to its numeric equivalent, you can always do so easily with EVAL or →NUM.

Of course, even when you do want to change the settings of flags 35 and 36, it's not always convenient to be messing about with SF and CF commands (and trying to remember the flag numbers, etc.).

So just remember that the MODE selection on the TOOLS menu will send you to a menu similar in function and appearance to the machine's built-in MODE menu, where each menu key toggles a mode on and off:



The two modes are SRES (Symbolic RESults) and SCON (Symbolic CONstants) --- flags 36 and 35 respectively.

If the name in the menu has a little box to the right of it, like this, SCON■, that mode is *set.* Pressing that menu's key will therefore *clear* the mode and the box: SCON; pressing the key when there's no box will *set* the mode and reinstate the box.

By pressing TOOLS, you'll leave the MODE menu and return to the TOOLS menu.

Complex Number Calculations

The six routines '→i', '→e', '→∠', 'R→i', 'R→e', and 'R→∠' extend the HP-28S complex number commands by allowing you to construct *complex number expressions.* These six programs *always* return *symbolic* results; they were designed to do so – to allow you to generate complex numbers in alternate, *symbolic* formats. However, just as with any other arithmetic, the results mode (flag 36) does affect how these expressions are *combined.* Watch:

```
2 FIX SRES∠   (turns off symbolic results)
1 2 R→i       Result: '1+i*2'
5 53 R→∠      Result: '∠(5,53)'
+             Result: (4.01,5.99)
```

That's what you'd get with symbolic results turned *off* (and you would then need to use →i, →e, or →∠ to put it back into a symbolic format, if that's what you prefer). Now do the same thing again, but with symbolic results set:

```
SRES          (turns on symbolic results)
1 2 R→i       Result: '1+i*2'
5 53 R→∠      Result: '∠(5,53)'
+             Result: '1+i*2+∠(5,53)'
```

As you can see, all symbolic expressions and constants are preserved – just as you'd expect with flag 36 set. Indeed, all six of these routines produce results that behave in this internally consistent way. Then, if you wanted the actual numeric value, you would use →NUM.

So in keeping with the recommendation (on page 72) that flags 35 and 36 both be kept set (generally a more flexible status) this book will *assume* such in all its results – that *symbolic results and symbolic constants modes are both set* – unless otherwise stated.

Keying In Complex Numbers

As you may know, there are several different complex number formats in widespread use among different disciplines. This is irrelevant, of course, if you never encounter any format other than the one you now know and love. But in case you do meet with a different format, it's good to be able to convert back and forth – and to combine numbers in different formats, reducing the results to your preferred format.

The first thing to realize with the HP-28S, of course, is that it has its own formats and conventions. The basic complex number on the HP-28S is the Cartesian-coordinate (also called *rectangular*) form: `(3,4)`, where the `3` is the real component and the `4` is the imaginary component. *All the HP-28S's calculations assume that a complex number is in this rectangular form* (except those operations specifically intended to convert from another form to this one).

All would be well if this rectangular form were the only one anybody ever used, but life is never that simple. There's also a *polar* form often used in electrical engineering and it is sometimes acceptable to the HP-28S: `(5,53.13)`, where the `5` is the magnitude of the complex vector, and the `53.13` is the angle it makes with the real axis. This angle may be measured in degrees or radians (the HP-28S will assume one or the other according to the current angle mode).

The problem is, there's no way to tell by looking at your calculator's display in which form a complex number is being expressed. Indeed, experienced users often develop the habit of using one form or the other, converting between them only when necessary.

But the HP-28S makes life even easier than that. It can create and use *symbolic expressions* (including the symbolic constants `'e'` and `'i'`) as easily as numbers, thus allowing the creation of other common complex number formats: `'3+i*4'` and `'5*e^(i*0.93)'`

These are the machine's renderings of the *algebraic rectangular* form, R+iC, and the *exponential* form, $Me^{i\alpha}$; they will reduce to numeric values if you use →NUM. Note that the HP-28S uses the mathematician's **i**, rather than the engineer's **j**, to represent $\sqrt{-1}$. Note also that the exponential form is valid only for α in *radians.*

Unfortunately, there's yet *another* common complex number format – the *polar degree* format (the engineers' favorite), which is M∠α, with α in degrees. The problem with this is that it's not at all convenient on the HP-28S, because the machine lacks the ∠ character. However, with a slightly different format, you can still present the same information: '▫(5,53.13)'. Here the 5 is the magnitude, and the 53.13 is the angle, α, *in degrees.* This is the polar degree format in this book.

Of course, all three of these alternate complex number forms would be merely interesting novelties without some convenient methods for creating and using them on your HP-28S. So this book provides three commands that exactly parallel the HP-28S's R→C ("real-to-complex") command – corresponding to each of these three symbolic formats:

```
'Re+i*IM'     'M*e^(i*α)'     '▫(M,α)'
   R→i            R→e            R→▫
```

Just like R→C, each of these commands takes its components from stack levels 1 and 2 and leaves the resulting expression on level 1:

```
2 FIX 1 1 R→i     Result: '1+i'
2 √ π 4 / R→e     Result: '1.41*e^(i*0.79)'
2 √ 45 R→▫        Result: '▫(1.41,45)'
```

Of course, these commands are only for your convenience; you can always key in these expressions directly. For example, to create the expression '1+i', you would press ['] [1] [+] [LC] [I] [ENTER], etc.

Math With Mixed Complex Formats

The real beauty of these three complex formats is that they're *fundamentally equivalent.* That is, they'll all reduce to the HP-28S's Cartesian rectangular form when you use →NUM (indeed, if you apply →NUM to the expressions on page 75, you'll get (1.00,1.00) for each one of them – try it)!

This means that you can enter complex numbers in *any* of the three alternate formats, perform calculations on them, then reduce the final result with one simple →NUM!

For example:

```
1 1 R→i          Result: '1+i'
3 34 R→▫         Result: '▫(3,34)'
+ 2 /            Result: '(1+i+▫(3,34))/2'
```
That's getting ugly.
```
2 FIX →NUM       Result: (1.74,1.34)
```
That's much better.

Yes, but what's this result in polar-degrees format? After all, if you're used to working in a certain format, it would be ideal to get the final result in that format, too – no?

```
→▫               Result: '▫(2.20,37.52)'
```

Note that this command is *not* R→▫ but →▫. Keep in mind that R→▫ (along with R→i and R→e) will always combine real numbers to *form* a complex expression where there was none before. This is useful mainly for entering complex numbers.

By contrast, →▫ (and →i and →e) will actually convert an *existing* complex number from any other allowable format to the desired format.

Thus, you have *four* basic complex number conversion routines: →i, →▪, →e, and →NUM. The built-in →NUM command is included, because after all, it too will convert any of the other allowable formats to a certain desired format – the HP-28S's own Cartesian rectangular format!

So, if in that last example, you really wanted to see the results in, say, algebraic rectangular format, it wasn't necessary to use →NUM at all:

```
1 1 R→i 3
34 R→▪ +
2 /           Result: '(1+i+▪(3,34))/2'
→i            Result: '1.74+i*1.34'
```

Try another one:

```
1 45 R→▪      Result: '▪(1,45)'
LN            Result: 'LN(▪(1,45))'
→i            Result: '6.40E-13+i*0.79'
```

→i (like any of the other conversion routines) automatically evaluates the expression before converting it to Re+iIm format. So any function such as LN, that can take a complex expression as an argument, will be evaluated by the conversion routines.

Notice that the result of LN(1∠45°) has a very small real portion – small enough to be considered rounding error and replaced by 0. To do this, you could use IM to return the imaginary portion as a real number. Then reënter the result as the imaginary portion of a complex number with a real portion of 0:

```
IM            Result: 'IM(6.40E-13+i*0.79)'
0 SWAP
R→i           Result: 'i*0.79'
```

Solving A Complex Expression For A Complex Result

Sometimes you'll need to do algebra with complex numbers, forming complex expressions containing *variables:* `'A+i*B-∠(1,45)'`

You cannot build such an object just by using the conversion tools; you must key in the variable names by hand. Here's one method:

```
'A' 'i' 'B' * +      Result: 'A+i*B'
1 45 R→∠ -           Result: 'A+i*B-∠(1,45)'
```

Of course, you can always perform further operations to build a more complicated expression, but *only after the variables have been given values* can you get its numeric result or convert its format:

```
8 'A' STO 12 'B' STO →i      Result: '7.29+i*11.29'
```

You can also use the SOLV menu to conveniently store values into the variables in such an expression and then evaluate the expression (with the $\boxed{\text{EXPR=}}$, $\boxed{\text{LEFT=}}$ or $\boxed{\text{RT=}}$ commands).

To evaluate this expression: `'INV(INV(ZA)+INV(ZB))'`
with these pairs of variable values:

ZA	1	'2-i*3'	'∠(2,45)'
ZB	3	2.3	'∠(100,36)'

...try this: `'INV(INV(ZA)+INV(ZB))'` $\boxed{\text{SOLV}}$ STEQ SOLVR

Then: 1 $\boxed{\text{ZA}}$ 3 $\boxed{\text{ZB}}$ $\boxed{\text{EXPR=}}$ Result: `0.75`

'2-i*3' $\boxed{\text{ZA}}$ 2.3 $\boxed{\text{ZB}}$ $\boxed{\text{EXPR=}}$ →i

Result: '1.47-i*0.58'

'ZA' PURGE '∡(2,45)' $\boxed{\text{ZA}}$ '∡(100,36)' $\boxed{\text{ZB}}$ $\boxed{\text{EXPR=}}$ →∡

Result: '∡(1.96,44.82)'

Note, however, that since the "SOLVer" capability itself does not extend to complex numbers, you cannot generally use it directly to *solve* for the values of complex variables in complex expressions. Instead, you must use ISOL – and probably some other algebraic rearrangement tools.

Before using ISOL, you should become comfortable with the use of the solution mode flag (flag 34):

```
34 SF 'A^2=9'  'A'  ISOL          Result: 3.00
34 CF 'A^2=9'  'A'  ISOL          Result: 's1*3'

34 SF 'A^3=9'  'A'  ISOL          Result: 2.08
34 CF 'A^3=9'  'A'  ISOL
                        Result: 'EXP(2*π*i*n1/3)*2.08'
```

With flag 34 *clear,* the expression for the *general solution* in each case is given. Arbitrary integers are represented by n1, n2, etc. Thus, replacing n1 above with any integer will yield a valid answer. Similarly, s1, s2, etc., represent arbitrary sign multipliers (±1), so that replacing s1 above with either 1 or -1 will yield valid results.

With flag 34 set, you get the *principal value* as a result. This value is what you get when you substitute 0 for all arbitrary integers and 1 for all arbitrary signs.

As always, the choice of solution mode (flag 34) is up to you – and the symbolic solutions switch, SSOL / SSOL■ (in the MODE menu – page 72), helps you to easily flip back and forth. Just be aware that this book will assume flag 34 to be *clear* in all examples unless otherwise indicated.

Conversions From Passive Circuit Elements To Impedances

Since the idea behind circuit reduction is to combine circuit elements, this book gives you tools to convert any combinations of resistances (R), capacitances (C), and inductances (L), to general impedances (Z).

Of course, since resistance is the real component of impedance, the conversion of resistance to impedance is trivial; `R→Z` returns the original input value: `75 R→Z` Result: `75`

However, the impedances in capacitances and inductances depend on the angular (radian) frequency of the circuit (ω), represented by the HP-28S variable `'w'`. If this frequency is undefined, the results of impedance conversions will therefore contain its unevaluated name, `w`:

```
'w' PURGE STD
.001 C→Z          Result: '-(i/(w*.001))'
.1 L→Z            Result: 'i*(w*.1)'
```

These results are entirely valid algebraic expressions, and you can thus build more complicated expressions from them, but you cannot obtain any numeric results from them without first assigning a value to `w`:

```
377 'w' STO 2 FIX
.1 L→Z →i         Result: 'i*37.70'
.001 C→Z →i       Result: '-(i*2.65)'
```

Note that once `w` is defined, it remains so defined for all future calculations, until it is redefined or undefined (PURGEd). However, if you've requested strictly numeric results (i.e. the symbolic results mode – flag 36 – is off), you *must* define `w` before you perform conversions. If you don't, you'll get an error: `Undefined Name`

These options give you a lot of flexibility in your conversions and calculations: You can certainly define `w` before doing anything else (as you just saw), so that this same frequency will be applied to all subsequent impedance calculations.

Alternatively, you can *undefine* `w` (with `'w' PURGE`) before doing any such calculations. This lets you build up and store complicated impedance expressions, then define `w` later. In this way, it's easy to evaluate the same impedance at different frequencies.

With that in mind, because AC circuit frequencies are often expressed in Hertz (cycles/second) rather than radians/second, a pair of conversion tools have been provided to make it easier for you to vary and specify frequencies.

For example, to convert from a 60-Hz frequency to ω:

```
60 F→w          Result: '120*π'
→NUM            Result: 376.99 (radians/second)
```

And back again:

```
w→F             Result: '188.50/π'
→NUM            Result: 60.00 (Hz.)
```

Putting this together with the impedance conversions, you can see how easy it is to convert, say, a 10-μf capacitor to an impedance value in a 60-Hz. circuit:

```
60 F→w 'w' STO
1E-5 C→Z      Result: '-(i/(120*π*1.00E-5))'
→i            Result: '-(i*265.26)'
```

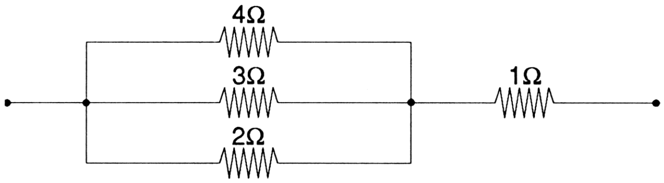
Impedance Element Combinations

There are two different ways to combine circuit impedances – in series and in parallel – and you have some tools to suit these needs. For example, to add two impedances...

in series: `'ZA' 'ZB' ZADDS` <u>Result</u>: `'ZA+ZB'`
in parallel: `'ZA' 'ZB' ZADDP`
<u>Result</u>: `'INV(INV(ZA)+INV(ZB))'`

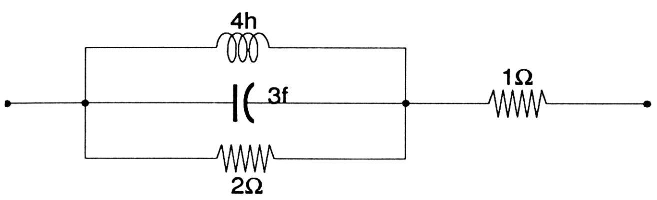
Think of these impedance "addition" tools just like addition operations – invoked the same way you use **+** in simple arithmetic. Simple? Yes, but here's a set of examples to illustrate the power of "simple" tools:

Problem: Given the circuit below, find the equivalent impedance.



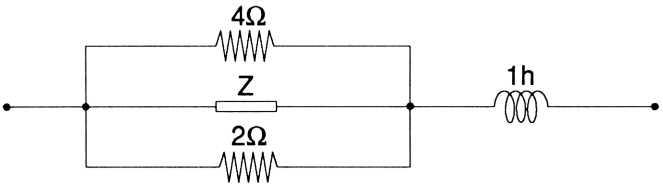
Solution:
```
2 FIX 2 R→Z 3 R→Z 4 R→Z ZADDP ZADDP
1 R→Z ZADDS
```
Answer: `1.92` (ohms)

Problem: Find the equivalent impedance at 40 Hz.:



Solution:
```
2 FIX 40 F→w 'w' STO
2 R→Z 3 C→Z 4 L→Z ZADDP ZADDP
1 R→Z ZADDS →i
```
Answer: `'1.00-i*1.33E-3'` (ohms)

Problem: The circuit below has an equivalent impedance of 0.78+j94.44 at 15 Hz. Find Z. Then find the equivalent impedance if Z = 2.5+j0.3.



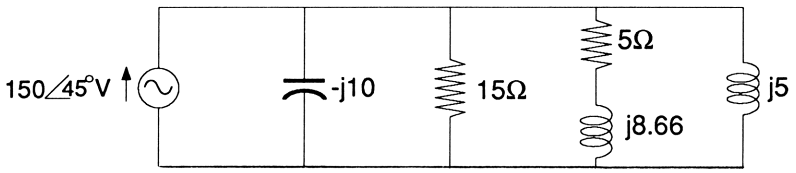
Solution: `2 FIX 15 F→w 'w' STO 2 R→Z 'Z' 4 R→Z`
`ZADDP ZADDP 1 L→Z ZADDS`
`ENTER 'EQ' STO` (saves this expression.)
`.78 94.44 R→i = 'Z' ISOL →i`

Answer: `'1.53+i*1.00'` (Z, in ohms)

Then: `2.5 .3 R→i 'Z' STO EQ →i`

Answer: `'0.87+i*94.28'` (the equivalent impedance)

Problem: Find the total current through this circuit:



Solution: `0 -10 R→i 15 ZADDP`
`5 0 8.66 R→i ZADDS ZADDP`
`0 5 R→i ZADDP 150 45 R→▫ SWAP / →▫`

Answer: `'▫(33.01,-12.99)'` (amps)

Problem: Now, in the RL branch of the circuit in the previous problem, find the current through the inductor and the voltage across the resistor.

Solution: `150 45 R→▫ 5 8.66 R→i / →▫`

Answer: `'▫(15.00,-15.00)'` (amps)

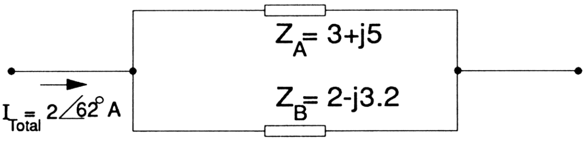
Then: `5 * →▫`

Answer: `'▫(75.00,-15.00)'` (volts)

Current Distribution Through Parallel Impedances

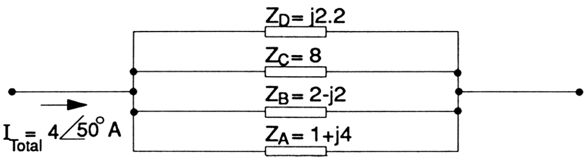
You have two current distribution tools to help you calculate the current through each element of a parallel network – once you know the total current (very handy, for example, in the previous problem).

The first of these two tools, IDIST, distributes a total input current over two specified impedances (and *notice* that the outputs appear on the stack in the same positions as the corresponding inputs):



```
2 FIX 3 5 R→i 2 -3.2 R→i 2 62 R→∠ IDIST →▫
                Result: '∠(2.19,101.24)'  (this is I_B)
SWAP →▫         Result: '∠(1.42,-15.79)'  (this is I_A)
+ →▫            Result: '∠(2.00,62.00)'   (I_TOTAL again)
```

The second tool, IDSTN, is just like the first except that it allows current distribution over *any number* of parallel impedances. Of course, this means you must provide an additional parameter specifying the number of impedances required:

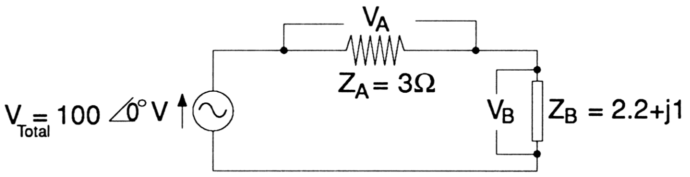


```
1 4 R→i 2 -2 R→i 8 0 2.2 R→i 4 50 R→∠
4 IDSTN →▫      Result: '∠(2.94,5.39)'     (I_D)
4 ROLL →▫       Result: '∠(1.57,19.43)'    (I_C)
4 ROLL →▫       Result: '∠(2.29,140.39)'   (I_B)
4 ROLL →▫       Result: '∠(0.81,95.39)'    (I_A)
+ + + →▫        Result: '∠(4.00,50)'       (I_TOTAL again)
```

Voltage Distribution Across Series Impedances

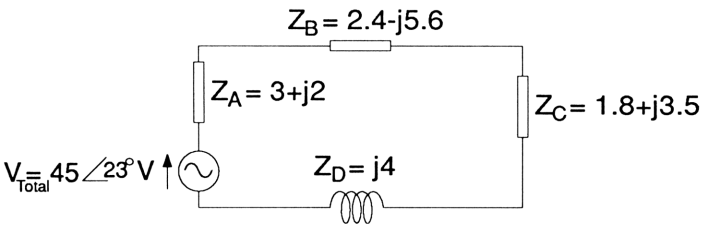
The two voltage distribution tools, `VDIST` and `VDSTN`, are much like the current distribution tools, except, of course, that they calculate the *voltage* drop across each element of a *series* of impedances.

For example, use `VDIST` on this circuit (and *notice* that the outputs appear on the stack in the same positions as the corresponding inputs):



```
3 2.2 1 R→i 100
VDIST →▫           Result: '▫(45.64,13.56)'     (V_B)
SWAP →▫            Result: '▫(56.65,-10.89)'    (V_A)
+ →▫               Result: '▫(100,0)'           (V_TOTAL)
```

And to use `VDSTN` on this circuit:

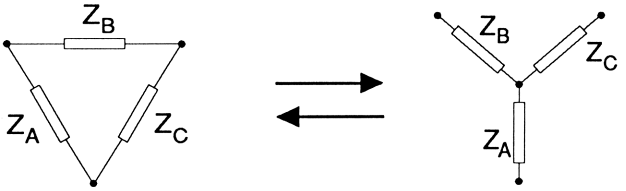


```
3 2 R→i 2.4 -5.6 R→i 1.8 3.5 R→i 0 4 R→i
45 23 R→▫

4 VDSTN →▫         Result: '▫(21.98,84.56)'     (V_D)
4 ROLL →▫          Result: '▫(19.81,28.25)'     (V_C)
4 ROLL →▫          Result: '▫(33.48,-72.24)'    (V_B)
4 ROLL →▫          Result: '▫(21.63,57.34)'     (V_A)
+ + + →▫           Result: '▫(45,23)'           (V_TOTAL)
```

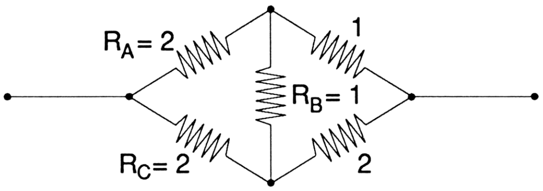
Δ-Y Conversions

There is also this handy pair of conversion routines: Δ-to-Y (`D→Y`) and Y-to-Δ (`Y→D`). These commands convert between these configurations:

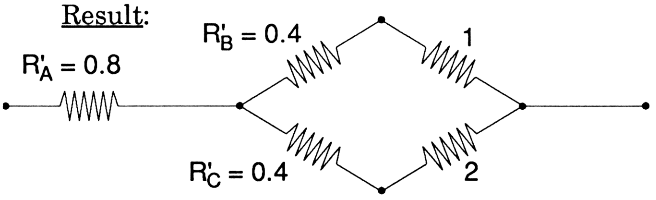


In converting, you must note the *clockwise* order of the impedances as they appear in the above diagram. Keep in mind that *this order is reflected on the stack – with input* *<u>and</u>* *output:* No matter which way you're converting, if input is `ZA ENTER ZB ENTER ZC`,
then output is `ZC DROP ZB DROP ZA`

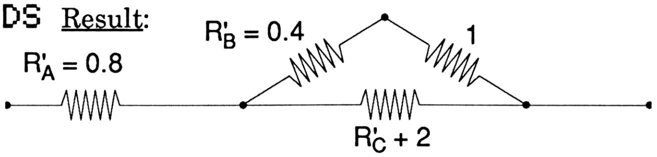
For example, reduce the following network to its equivalent impedance:



`2 1 2 D→Y` <u>Result</u>:



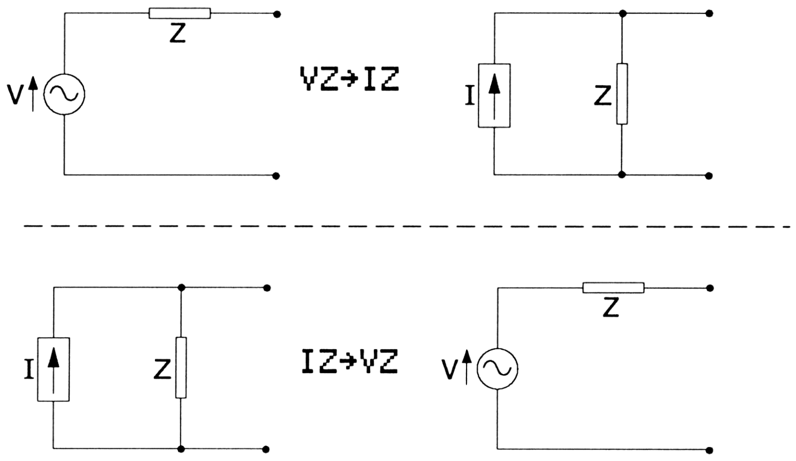
Next: `2 ZADDS` <u>Result</u>:



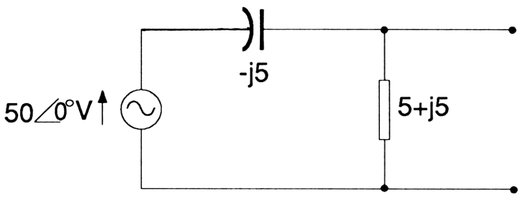
Now reduce the rest of it: `SWAP 1 ZADDS ZADDP ZADDS`
<u>Result</u>: `1.68` (ohms)

Active Element Conversions

Alas, most simple circuits contain not only passive elements (impedances), they also have active elements – usually independent sources of voltage or current – which make these simple conversions handy to have:



The conversions are trivial but useful for converting source configurations between series and parallel. For example, to reduce the following parallel-voltage circuit to its equivalent series-voltage and impedance:



```
50 0 -5 R→i VZ→IZ 5 5 R→i ZADDP
```

`IZ→VZ →i` Result: `'5-i*5'` (resultant Z)
`SWAP →∠` Result: `'∠(70.71,45)'` (resultant V)

Chapter 4

CIRCUIT ANALYSIS TOOLS:

Introduction

This Chapter is to help you decide which (if any) of the Circuit Analysis Tools you need.

What These Tools Can – And Cannot – Do For You

The Circuit *Analysis* Tools are three large collections of routines that analyze voltages, currents, or input-output ratios in circuits. With the help of these tools, you can build three kinds of circuit descriptions in your HP-28S. Then, depending upon the type of circuit you have built, you can:

- Calculate voltages at points in a network of circuit elements (that's Node-Voltage Network Analysis). Any circuit to be so analyzed may contain resistors (R), capacitors (C), inductors (L), impedances (Z), and independent active current sources (I) – *but it may not contain any independent active voltage sources (V);*

- Calculate currents through branches in a network of circuit elements (that's Mesh-Current Network Analysis). Any circuit to be so analyzed may contain resistors (R), capacitors (C), inductors (L), impedances (Z), and independent active voltage sources (V) – *but it may not contain any independent active current sources (I);*

- Calculate transfer ratios of ladder networks (that's Ladder Network Transfer Analysis). Any ladder circuit to be so analyzed may contain parallel or series resistors (RP or RS), capacitors (CP or CS), inductors (LP or LS), and impedances (ZP or ZS), but it may not contain any active elements. You can then calculate the input impedance (ZIN) and various transfer ratios, including Voltage and Current Transfer Ratios, Forward Transfer Impedance and Admittance, and Power Transfer Ratio (Gain).

Note that once you specify the type of circuit (one of the above three types) that you're going to analyze – and then build it – you cannot then simply change the type; you must rebuild a similar description with the new type.

You *cannot* conveniently use the Circuit Analysis Tools to:

- Solve for an unknown element value or an unknown frequency value in a circuit;
- Key in and convert between different formats of complex numbers;
- Convert from passive circuit elements (R, L, and C) to generalized impedances (Z);
- Find equivalent single impedances for series or parallel combinations of impedances;
- Perform Δ-Y Conversions (either way);
- Find Thevinin's and Norton's equivalent source-impedance combinations.

For these sorts of calculations, you'll find the Circuit *Reduction* Tools to be much more helpful.

What To Do Next

- If none of these Circuit Analysis Tools seems to be what you're after, you might try the Circuit Reduction Tools (Chapter 1, page 8).

- If you wish to have *all* the Circuit Analysis Tools available to you, then turn to page 94 and work through Chapter 5, keying in every routine. *Don't neglect to read all the preliminaries in Chapter 5!*

- If at this point, you want to pick and choose among the three kinds of analysis tools available, then you can work through Chapter 5, keying in only those routines necessary for the desired analysis. The checklists on the next two pages (and the information provided in Chapter 5) will tell you which routines are required.

  Even with these checklists, though, *don't neglect to read all the preliminaries in Chapter 5!*

Checklists For The Three Kinds Of Circuit Analysis

The following lists give the page numbers and names of all the objects *required* for the 3 types of circuit analysis – but it's not as much as it seems; there's a lot of overlap. A * preceding an object name means it's needed for *all three* kinds of analysis; a ‡ preceding the namemeans it's needed only for Mesh-Current and Node-Voltage analysis; a • preceding the name means *"optional but recommended."*

Required For Mesh-Current Network Analysis

97 * ADD	125 * GETK	170 * PUTX
98 * AI	126 * GOTO	171 * QUIT
99 ‡ C	128 * INIT	172 ‡ R
100 * CALC	129 ‡ INODE	173 RMESH
101 * CDSP	131 ‡ L	179 • SAVE
102 * CELS	138 LMESH	181 * Show
103 * CHNG	140 • LOAD	182 * SIP
104 * CINIT	146 * MAIN	183 * Size
105 * CIRCS	147 MESH	184 * SLDEL
106 • CKSM	148 MESHCALC	185 * SZCHK
107 CMESH	150 MESHELS	186 * SZDSP
114 * DEL	151 MESHSHOW	187 * TPDSP
115 * EDIT	153 * Next	188 * Type
116 * EEA	155 NODECALC	189 V
117 * ERRBP	160 * PAUSE	190 VMESH
118 * EXIT	161 * PREV	192 * YES?
119 * FRDSP	162 • PRINT	194 ‡ Z
120 * FREQ	165 * PUTE	195 ZMESH
121 * Freq	168 PUTMESH	196 ‡ ZNODE
124 * GETE	169 ‡ PUTNODE	

Required For Node-Voltage Network Analysis

97	*	ADD	124	*	GETE	169	‡	PUTNODE
98	*	AI	125	*	GETK	170	*	PUTX
99	‡	C	126	*	GOTO	171	*	QUIT
100	*	CALC	127		I	172	‡	R
101	*	CDSP	128	*	INIT	174		RNODE
102	*	CELS	129	‡	INODE	179	•	SAVE
103	*	CHNG	131	‡	L	181	*	Show
104	*	CINIT	139		LNODE	182	*	SIP
105	*	CIRCS	140	•	LOAD	183	*	Size
106	•	CKSM	146	*	MAIN	184	*	SLDEL
108		CNODE	153	*	Next	185	*	SZCHK
114	*	DEL	154		NODE	186	*	SZDSP
115	*	EDIT	155		NODECALC	187	*	TPDSP
116	*	EEA	157		NODEELS	188	*	Type
117	*	ERRBP	158		NODESHOW	192	*	YES?
118	*	EXIT	160	*	PAUSE	194	‡	Z
119	*	FRDSP	161	*	PREV	196		ZNODE
120	*	FREQ	162	•	PRINT			
121	*	Freq	165	*	PUTE			

Required For Ladder Network Transfer Analysis

97	*	ADD	123	•	FTZ	167		PUTLADR
98	*	AI	124	*	GETE	170	*	PUTX
100	*	CALC	125	*	GETK	171	*	QUIT
101	*	CDSP	126	*	GOTO	175		RP
102	*	CELS	128	*	INIT	176		RPLADR
103	*	CHNG	130	•	INZ	177		RS
104	*	CINIT	132		LADR	178		RSLADR
105	*	CIRCS	133		LADRCALC	179	•	SAVE
106	•	CKSM	134		LADRELS	181	*	Show
109		CP	135		LADRENGN	182	*	SIP
110		CPLADR	137		LADRSHOW	183	*	Size
111		CS	140	•	LOAD	184	*	SLDEL
112		CSLADR	142		LP	185	*	SZCHK
113	•	CTR	143		LPLADR	186	*	SZDSP
114	*	DEL	144		LS	187	*	TPDSP
115	*	EDIT	145		LSLADR	188	*	Type
116	*	EEA	146	*	MAIN	191	•	VTR
117	*	ERRBP	153	*	Next	192	*	YES?
118	*	EXIT	160	*	PAUSE	197		ZP
119	*	FRDSP	161	*	PREV	198		ZPLADR
120	*	FREQ	162	•	PRINT	199		ZS
121	*	Freq	164	•	PTR	200		ZSLADR
122	•	FTA	165	*	PUTE			

CHAPTER 5

CIRCUIT ANALYSIS TOOLS: Reference

This chapter gives the program listings and brief descriptions of the larger network analysis tools. They are presented in this brief, ordered format for your convenience as you initially key them in. *For examples of their uses, see Chapter 6 (page 202).*

When loading and using these tools, you must already understand the basics of object creation (including menus and directories) on the HP-28S. If you don't feel comfortable with your level of understanding, turn to Appendix B ("Some HP-28S Basics" – page 244) *before* you begin to key in these tools.

Before you key in anything, **read these important preliminaries:**

First of all, with the HP-28S, there are many ways to key in solutions to problems; it's impossible to show every method. This book cannot "read your mind" to know which menu or directory you're currently using when you want to call one of these tools – so it can't necessarily give you the quickest set of keystrokes for your situation. Any set of commands in this book, therefore – rather than specifically telling you to press a *key* (e.g. ■ PURGE or DROP) or to select a *menu item* (e.g. R→C P→R) – will be given instead in their generic form (spelled out as if you had *typed* them in):

```
PURGE DROP
 R→C P→R
```

etc.

So just keep in mind that it might be more convenient for you to use keys or menu items than to type the commands.

Next: You might never use any of these routines for anything other than their designed purposes in this book. However, many of these routines can be very useful to you in your own programming – and it would certainly pay you to consider them. Therefore, each routine has been documented in some detail. Just don't get bogged down in those details unless you need them for your own purposes.

No matter what: To prepare your HP-28S to do any kind of circuit analysis – before you key in any of these tools – you must give the following commands to create these directories (if you haven't already done so):

```
HOME
'°EE' CRDIR
°EE '°WRK' CRDIR
'°CIRC' CRDIR
```

(°CIRC is technically *optional but recommended*; you'll need it to SAVE circuits.)

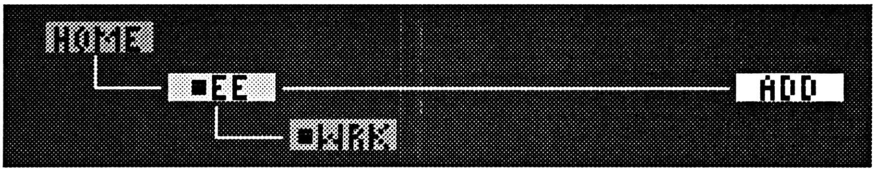
All program tools in this reference chapter are described in a consistent format, as illustrated by the first entry on the opposite page....

Simply proceed through the routines in the order presented here (it's alphabetized according to the name of the routine). For each routine, check the **Notes** item (at the bottom) to see if you really need it for what you want to do (or use the appropriate checklist from pages 91-93). If you do need it, then key it in, name it and store it in the appropriate directory, and continue on to the next one.

Unless specifically noted otherwise under **Notes**, you should assume that each routine is *required* for successful operation of *any* of the three types of circuit analysis (Node Voltage, Mesh Current, and Ladder Network Transfer Analysis) offered here.

Each program description will be accompanied by a **memory diagram** similar to the diagram on the opposite page. This is to help you find the routine more easily when you're flipping through this section of the book – and it also reminds you where (in which directory) this routine must be located (and the **Notes** will remind you of this, also).

To help you check the accuracy of your keystrokes when inputting these Circuit Analysis Tools, each routine is listed with its *checksum,* which can be generated with the help of the CKSM routine (listed on page 106). Therefore, you may actually want to key in CKSM first, to help you check the rest of your keystrokes. CKSM is also listed with the Circuit Reduction Tools.



Add An Element To The Current Circuit:

ADD (17026)

```
« CELS MENU 2 CF »
```

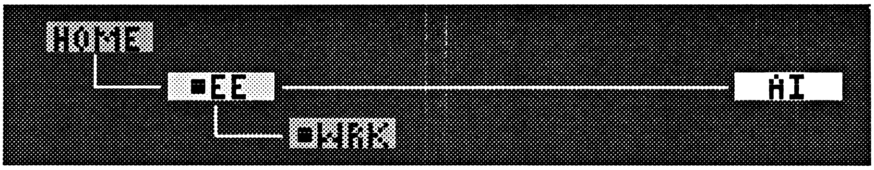
Summary: ADD sets up a menu for adding to a circuit, using CELS to provide the list of menu items. Flag 2 is cleared to tell other routines that the mode is ADD rather than CHNG.

Inputs: None

Outputs: None

Errors: Bad Argument Type will occur if the current circuit (stored in 'CIRC') is an invalid data object for the circuit editor. Undefined Name will occur if 'CIRC' is undefined; or, if it is improperly initialized, execution will halt with an undefined name left on the stack.

Notes: ADD uses (and therefore requires access to) CELS. Put ADD into the ■EE directory.



Take The Integer Portion Of The Absolute Value Of The Object At The Specified Stack Level:

AI (53699)

```
« → L « L ROLL ABS
IP L ROLLD » »
```

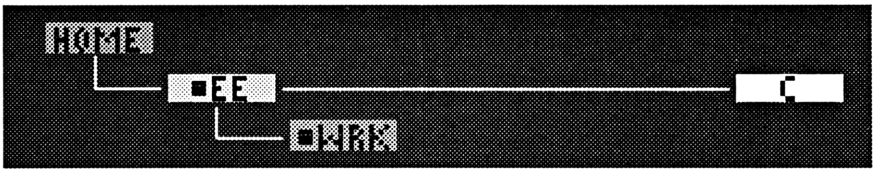
Summary: AI modifies the object at the specified stack level by replacing it with the integer portion of its absolute value. AI works similarly to the HP-28S's ROLL command: the Level number specified must be the target object's Level *before* the Level number was input.

Inputs: Level 1 – the Level number, n, of the target object.

Outputs: The target object at Level n is modified.

Errors: Too Few Arguments will occur if the stack does not contain the Level specified. Bad Argument Type will occur if Level 1 does not contain a real number.

Notes: Put AI into the ■EE directory.



Add A Capacitance Element To The Circuit List:

C (7532)

```
« "C" PUTE »
```

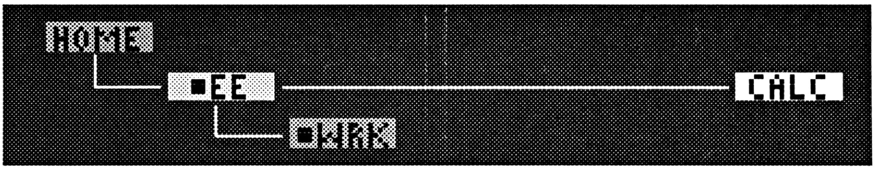
Summary: C invokes PUTE (PUT Element) in order to add a capacitance-type element ("C") to the current circuit ('CIRC').

Inputs: Level 3 – an integer – the "from" node or forward mesh.
Level 2 – an integer – the "to" node or reverse mesh.
Level 1 – a real number – the capacitance value (C).

Outputs: None.

Errors: If the circuit is empty when C is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: C is *necessary for both* Node-Voltage *and* Mesh-Current circuit analyses. If you need it, put C into the ■EE directory. C uses PUTE.



Calculate A Circuit's Unknowns According To The Circuit's Type:

CALC (27300)

```
« Type "CALC" + STR→
»
```

Summary: CALC performs the big calculation command specific to the current circuit (either NODECALC, MESHCALC, or LADRCALC). If 'CIRC' is improperly initialized so that its type has no corresponding calculation routine, the name of the undefined routine will be left on the stack:

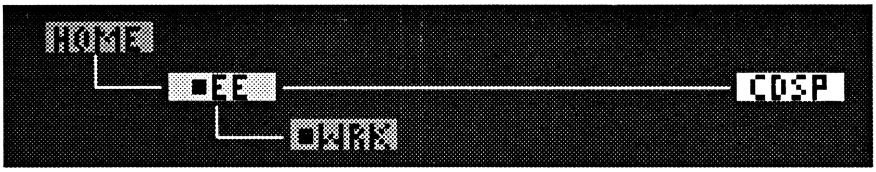
"FRED" CINIT CALC (stack: 'FREDCALC')

Inputs: The routine uses to the circuit data stored in 'CALC'.

Outputs: The appropriate calculation routine is invoked.

Errors: Bad Argument Type will occur if Type returns something other than a string.

Notes: Put CALC into the ▪EE directory. CALC uses (requires) Type.



Display Summary Information For The Current Circuit:

CDSP (53730)

```
« CLLCD TPDSP SZDSP
FRDSP PAUSE »
```

Summary: CDSP displays a synopsis of the current circuit, in the following format:

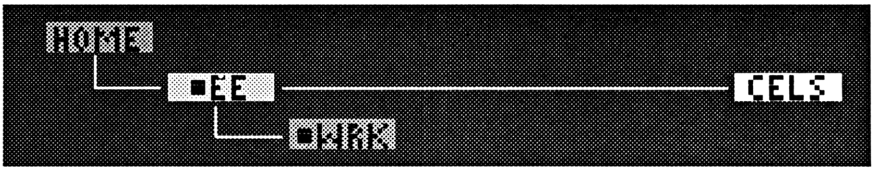
```
<type> Analysis
<n>Element Circuit
Frequency= <frequency>
```

Inputs: None.

Outputs: None.

Errors: Undefined Name will occur if the current circuit is undefined. Bad Argument Type or other unpredictable errors will occur if the circuit data list is damaged or improperly initialized.

Notes: CDSP uses TPDSP, SZDSP, FRDSP and PAUSE. Put CDSP into the ■EE directory.



List The Valid Elements For The Current Circuit Type:

CELS (26172)

```
« Type "ELS" + STR→
»
```

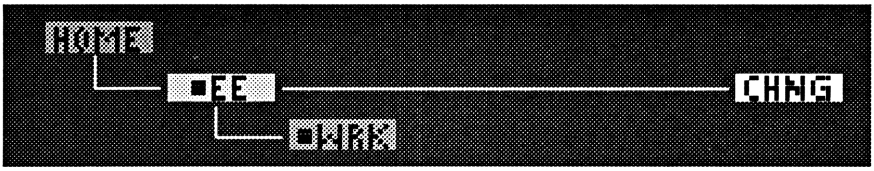
Summary: CELS uses the Type of the current circuit (stored in 'CIRC') to build and invoke the name of a list – either NODEELS, MESHELS, or LADRELS – of the valid elements for that circuit type.

Inputs: None.

Outputs: None.

Errors: Bad Argument Type will occur if the current circuit ('CIRC') is an invalid data object for the circuit editor. Undefined Name will occur if 'CIRC' is undefined; or, if it is improperly initialized, execution will halt with an undefined name left on the stack.

Notes: CELS requires Type. Put CELS into the ▪EE directory.



Change The Specified Circuit Element:

CHNG (38051)

```
« GETE DROP CELS
MENU 2 SF »
```

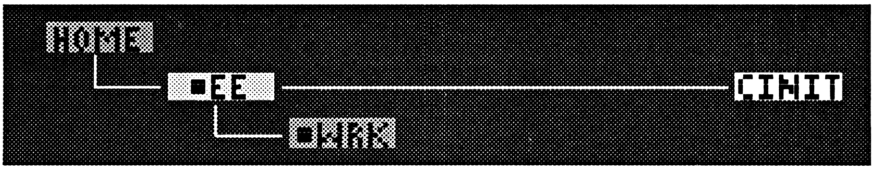
Summary: CHNG gets the current element (indicated by the value in 'CIDX') of the current circuit and places it on the stack.

Inputs: CHNG takes values from 'CIDX' and 'CIRC'.

Outputs: Level 3 – an integer (the "from" node or forward loop);
Level 2 – an integer (the "to" node or reverse loop);
Level 1 – a real or complex number (the element value).

Errors: Bad Argument Type will occur if the current circuit (named 'CIRC') is an invalid data object for the circuit editor. Undefined Name or other unpredictable errors (such as halting of execution) will occur if 'CIRC' is undefined or improperly initialized.

Notes: CHNG requires GETE and CELS. Put CHNG into the ▪EE directory.



Initialize The Circuit To The Specified Type:

CINIT (88162)

```
« 0 2 →LIST 1 →LIST
'CIRC' STO 0 'CIDX'
STO »
```

Summary: CINIT expects the circuit type as a character string at stack Level 1. It then creates the "master" circuit description list, named 'CIRC', whose first element is another list: { "<type>" 0 }. The 0 is the initial frequency of the circuit and this value is also stored in 'CIDX', the circuit index.

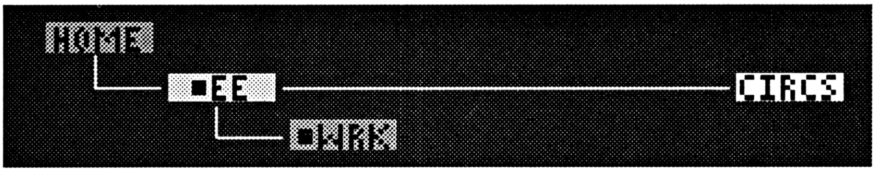
CINIT does not check for valid input; it simply creates this list, overwriting any previous contents of 'CIRC' and 'CIDX'.

Inputs: Level 1 – a string ("NODE", "MESH" or "LADR").

Outputs: None.

Errors: Too Few Arguments will occur if the circuit type is not on the stack.

Notes: Put CINIT into the ■EE directory.



The List Of Valid Circuit Types:

CIRCS (21873)

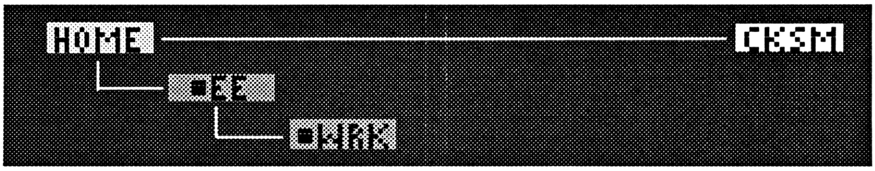
```
{ MESH NODE LADR }
```

Summary: CIRCS is just a simple list – not a program object (and so it's meaningless to speak of **Inputs**, **Outputs**, or **Errors**).

This list is a series of all currently valid circuit types (the ones made valid by the programs in this book are the three types shown above). If other types are made valid by the later development of other "circuit-crunching engines," then the names of those types could simply be added to the above names.

The list is used to (i) determine whether or not an input value is a valid circuit type; and (ii) generate menus containing the currently valid circuit types.

Notes: Put CIRCS into the ■EE directory.



Proofread A Named Object:

CKSM (1040278)

```
« 2 32 ^ RCLF → N F
S « N RCL STD HEX 64
STWS 45 SF 48 CF
→STR N →STR + 0 1 3
PICK SIZE FOR I OVER
I DUP SUB NUM I * +
DUP F MOD SWAP F /
IP + NEXT SWAP DROP
S STOF » »
```

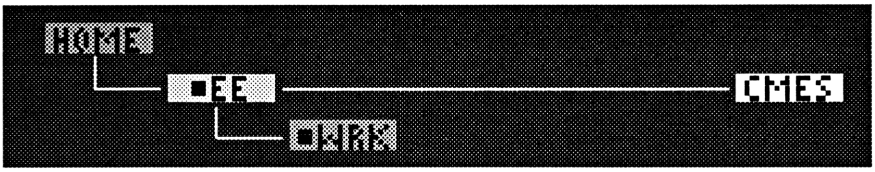
Summary: CKSM (checksum) checks for "typos" by computing a unique integer (which should match the value given) for the named object.

Inputs: Level 1 – the name of an object.

Outputs: Level 1 – an integer – the checksum.

Errors: Bad Argument Type occurs if the input is not a name. Undefined Name occurs if the input name is undefined.

Notes: CKSM is *optional but highly recommended,* and is most generally useful in the HOME directory.



Store A Capacitance Into The Impedance Array:

CMESH (34410)

```
« ω * INV NEG i *
ZMESH »
```

Summary: CMESH stores a capacitance as an impedance (calculated by Z = -**i**/(ω C)) into the impedance array, 'ZA'.

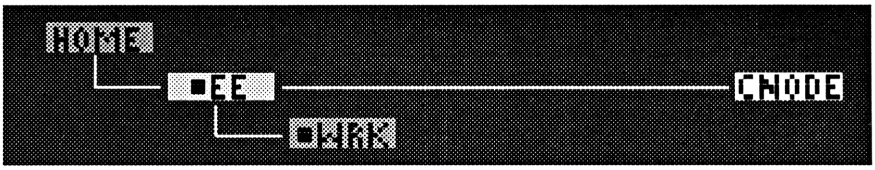
Inputs: Level 3 – an integer – the "from" loop.
Level 2 – an integer – the "to" loop.
Level 1 – a real number – the capacitance value (C).

Outputs: None.

Errors: Infinite Result will occur if the capacitance or frequency value is zero and flag 59 (infinite result action) is set.

Notes: CMESH is necessary *only* for Mesh-Current circuit analysis. If you need it, put it into the ▪EE directory.

CMESH uses ZMESH.



Store A Capacitance Into The Admittance Array:

CNODE (34016)

```
« ω * INV NEG i *
ZNODE »
```

Summary: CNODE stores a capacitance as an admittance (calculated by Y = **i**ω C) into the admittance array, 'ZA'.

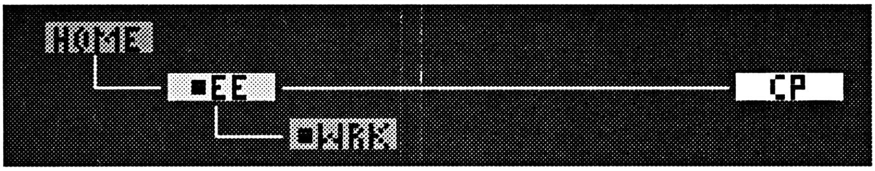
Inputs: Level 3 – an integer – the "from" node.
Level 2 – an integer – the "to" node.
Level 1 – a real number – the capacitance value (C).

Outputs: None.

Errors: Infinite Result will occur if the capacitance or frequency value is zero and flag 59 (infinite result action) is set.

Notes: CNODE is necessary *only* for Node-Voltage circuit analysis. If you need it, put it into the ▪EE directory.

CNODE uses ZNODE.



Add A Parallel-Capacitance Element To The Circuit List:

CP (9937)

```
« "CP" PUTE »
```

Summary: CP invokes PUTE (PUT Element) to add a parallel-capacitance element ("CP") to the current circuit ('CIRC').

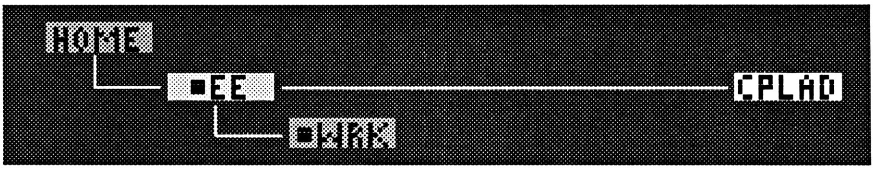
Inputs: Level 1 – a real number – the parallel capacitance value (CP).

Outputs: None.

Errors: If the circuit is empty when PUTE is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: CP is necessary *only* for Ladder Network Transfer analysis. If you need it, put CP into the ▪EE directory.

CP uses PUTE.



Create A Parallel Capacitance Transfer Matrix:

CPLADR (41489)

```
« ω * i SWAP / NEG
RPLADR »
```

Summary: CPLADR creates a parallel capacitance transfer matrix of this form:

$$\begin{bmatrix} 1 & 0 \\ i\omega C & 1 \end{bmatrix}$$

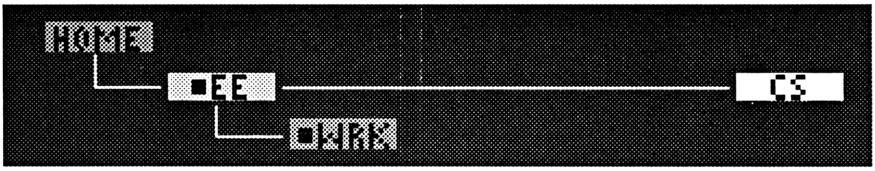
Inputs: Level 1 – a real number – the capacitance value.

Outputs: Level 1 – a two-by-two array – the parallel capacitance transfer matrix.

Errors: Bad Argument Type will occur if the input capacitance is not a real number or if 'ω' is undefined.

Notes: CPLADR is necessary only for Ladder Network Transfer analysis. If you need it, put it into the ▪EE directory.

CPLADR uses RPLADR.



Add A Series-Capacitance Element To The Circuit List:

CS (10000)

```
« "CS" PUTE »
```

Summary: CS invokes PUTE (PUT Element) to add a series-capacitance element ("CS") to the current circuit ('CIRC').

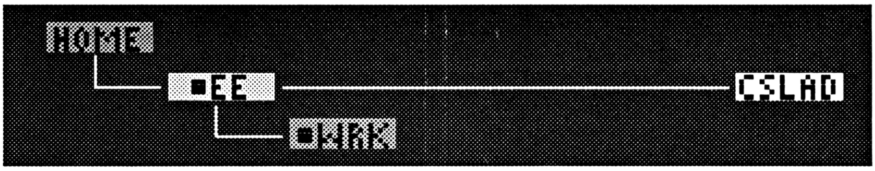
Inputs: Level 1 – a real number – the series capacitance value (CS).

Outputs: None.

Errors: If the circuit is empty when CS is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: CS is necessary *only* for Ladder Network Transfer analysis. If you need it, put CS into the ■EE directory.

CS uses PUTE.



Create A Series Capacitance Transfer Matrix:

CSLADR (41642)

```
« ω * i SWAP / NEG
RSLADR »
```

Summary: CSLADR creates a series capacitance transfer matrix of this form:

$$\begin{bmatrix} 1 & \frac{-i}{\omega C} \\ 0 & 1 \end{bmatrix}$$

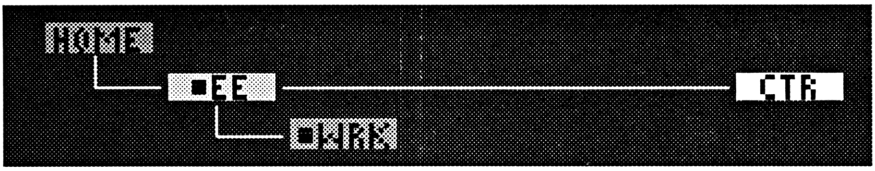
Inputs: Level 1 – a real number – the capacitance value.

Outputs: Level 1 – a two-by-two array – the series capacitance transfer matrix.

Errors: Bad Argument Type will occur if the input capacitance is not a real number or if 'ω' is undefined.

Notes: CSLADR is necessary only for Ladder Network Transfer analysis. If you need it, put it into the ■EE directory.

CSLADR uses RSLADR.



Calculate The Ladder Network Current Transfer Ratio:

CTR (70691)

```
« LADRENGN '-1/(ZA(2
,1)*ZL+ZA(2,2))'
EVAL »
```

Summary: CTR calculates the value of the Current Transfer Ratio for a "LADR"-type circuit.

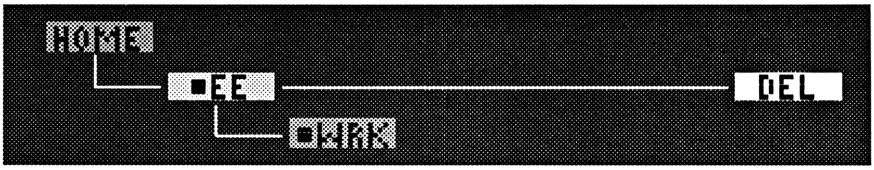
Inputs: None – data is taken from ZL and ZA.

Outputs: Level 1 – a real or complex number – the ratio of output current to input current.

Errors: Infinite Result will occur if 'ZA(2,1)*ZL+ZA(2,2)' is (0,0) and flag 59 (infinite result action) is set. Unpredictable errors may occur if the LADR circuit is not properly initialized or contains invalid data.

Notes: CTR is necessary only for Ladder Network Transfer analysis. If you need it, put CTR into the ▪EE directory.

CTR uses LADRENGN.



Delete The Current Circuit Element:

DEL (636935)

```
« SZCHK IF
"DELETE Element "
CIDX SIP + YES? THEN
CIRC CIDX 1 + DUP
SLDEL 'CIRC' STO
Size CIDX DUP2 < ROT
ROT IFTE 'CIDX' STO
END »
```

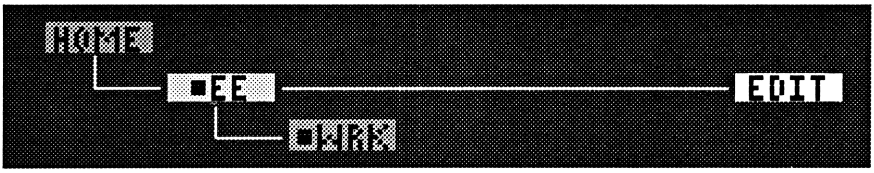
Summary: DEL deletes the circuit element given by 'CIDX', after ascertaining the element's existence and your intent to so delete. If this was the last element in the circuit list, 'CIDX' is decremented.

Inputs: None.

Outputs: None.

Errors: None.

Notes: DEL requires SZCHK, SIP, YES?, SLDEL and Size. Put DEL into the ▪EE directory.



Generate A Menu Of Circuit Editing Commands:

EDIT (138179)

```
« { ADD DEL PREV
Next Show CHNG GOTO
PRINT EXIT } MENU »
```

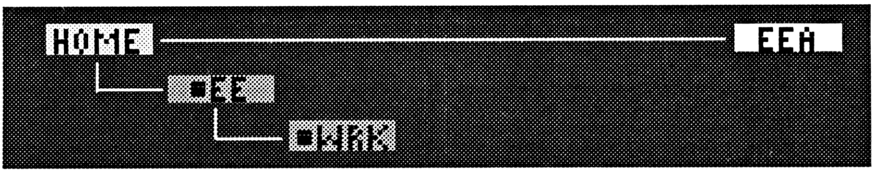
Summary: EDIT generates the menu of editing commands for building and altering the current circuit.

Inputs: None.

Outputs: None.

Errors: None.

Notes: Put EDIT into the ■EE directory.



Enter Circuit Analysis Tools:

EEA (517653)

```
« ▪EE ▪WRK RCLF 31
CF IFERR Type THEN
"x" END SWAP STOF
CIRCS "'" ROT + STR→
POS NOT « CIRCS MENU
» 'MAIN' IFTE »
```

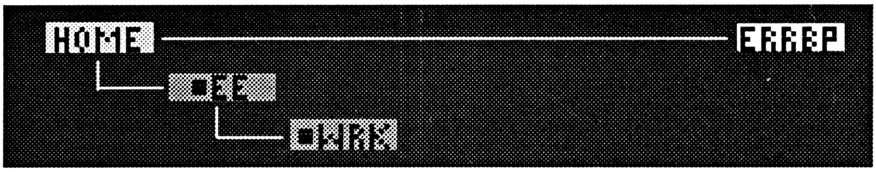
Summary: EEA starts the circuit editor by moving to the ▪WRK menu and attempting to access the current circuit ('CIRC') with the Type tool. If Type returns an error or a type not found in 'CIRCS', a menu appears with the valid circuit types, and the user selects the desired type. If 'CIRC' passes the tests, the MAIN menu is generated.

Input: None – EEA looks for a valid circuit, named 'CIRC'.

Output: None.

Errors: Bad Argument Type will occur if 'CIRC' contains a non-list or a damaged list.

Notes: EEA uses ▪WRK (a directory), Type, CIRCS, and MAIN. Put EEA into the HOME directory.



Generate An Error Beep:

ERRBP (20649)

```
« 1400 .075 BEEP »
```

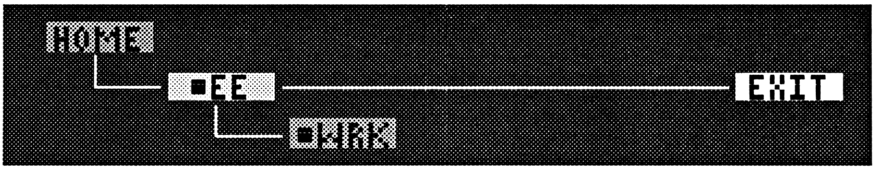
Summary: This routine generates a beep of the same frequency and duration as the HP-28S's error beep. This beep is used by programs in this book to signal user errors.

Inputs: None.

Outputs: None.

Errors: None.

Notes: Since ERRBP could be generally useful for many HP-28S routines, put it in the HOME directory. For the purposes of this book, however, it's also OK in the ■EE directory.



Exit To The Main Menu:

EXIT (7303)

```
« MAIN »
```

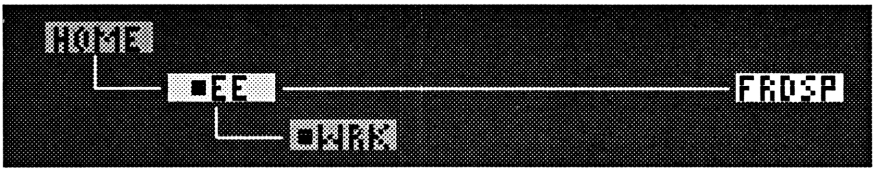
Summary: EXIT appears in the EDIT menu and allows the user to leave the current circuit by exiting to the MAIN menu.

Inputs: None.

Outputs: None.

Errors: None.

Notes: EXIT uses MAIN. Put EXIT in the ■EE directory.



Display Circuit Frequency:

FRDSP (66350)

```
« "Frequency= " Freq
→STR + 3 DISP »
```

Summary: FRDSP shows the frequency stored in the current circuit, – on line 3 of the display, as follows:

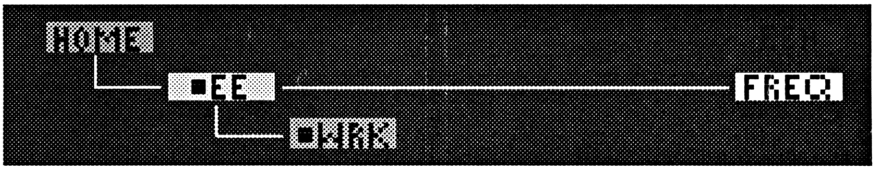
Frequency= <frequency>

Inputs: None (data is taken from 'CIRC').

Outputs: The frequency display shown above.

Errors: Unpredictable errors will occur if the circuit has not been properly initialized.

Notes: FRDSP uses Freq. Put FRDSP into the ▪EE directory.



Store The Circuit's Frequency:

FREQ (16030)

```
« → F « CIRC DUP 1
GET 2 F PUT 1 SWAP
PUT 'CIRC' STO MAIN
» »
```

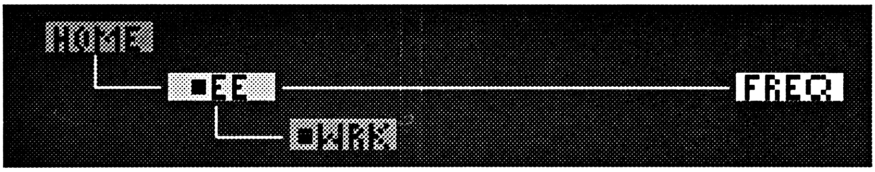
Summary: FREQ stores the sinusoidal steady-state frequency (in Hz.) in the current circuit. It assumes that 'CIRC' has been properly initialized.

Inputs: Level 1 – a real number – the steady-state, sinusoidal frequency of the circuit.

Outputs: None.

Errors: Unpredictable errors will occur if the circuit has not been properly initialized.

Notes: FREQ uses MAIN. Put FREQ into the ▪EE directory.



Recall The Circuit's Frequency:

Freq (24687)

```
« CIRC 1 GET 2 GET »
```

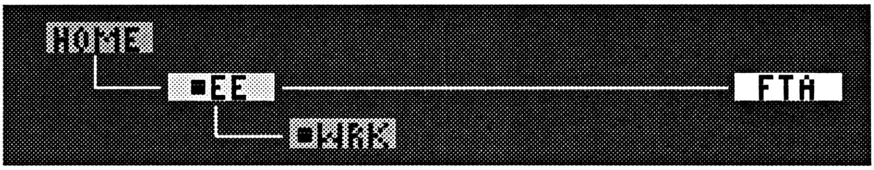
Summary: Freq recalls the steady-state, sinusoidal frequency from the current circuit. Freq assumes that the circuit has already been properly initialized.

Inputs: None.

Outputs: Level 1 – a real number – the steady-state, sinusoidal frequency of the circuit.

Errors: Unpredictable errors will occur if the circuit has not been properly initialized (e.g. the machine may Freq out).

Notes: Put Freq into the ■EE directory. Notice that Freq will *appear* the same as FREQ in your menu display, since menu items can only display upper-case letters.



Calculate The Ladder Network Forward Transfer Admittance:

FTA (69961)

```
« LADRENGN '-1/(ZA(1
,1)*ZL+ZA(1,2))'
EVAL »
```

Summary: FTA calculates the value of the Forward Transfer Admittance for a "LADR"-type circuit.

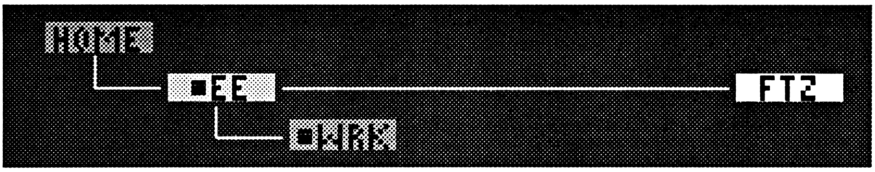
Inputs: None – data is taken from ZL and ZA.

Outputs: Level 1 – a real or complex number – the Forward Transfer Admittance

Errors: Infinite Result will occur if 'ZA(1,1)*ZL+ZA(1,2)' is (0,0) and flag 59 (infinite result action) is set. Unpredictable errors may occur if the circuit has not been properly initialized or if it contains invalid data.

Notes: FTA is necessary only for Ladder Network Transfer analysis. If you need it, put FTA into the ▪EE directory.

FTA uses LADRENGN.



Calculate The Ladder Network Forward Transfer Impedance:

FTZ (14952)

```
« CTR NEG ZL * »
```

Summary: FTZ calculates the value of the Forward Transfer Impedance for a "LADR"-type circuit.

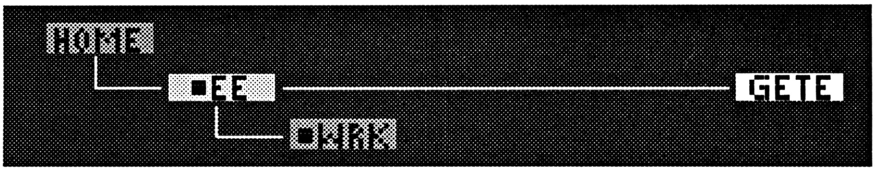
Inputs: None – data is taken from ZL and ZA.

Outputs: Level 1 – a real or complex number – the Forward Transfer Impedance

Errors: Infinite Result will occur if 'ZA(2,1)*ZL+ZA(2,2)' is (0,0) and flag 59 (infinite result action) is set. Unpredictable errors may occur if the circuit has not been properly initialized or if it contains invalid data.

Notes: FTZ is necessary only for Ladder Network Transfer analysis. If you need it, put FTZ into the ■EE directory.

FTZ uses CTR.



Get The Current Element From The Circuit:

GETE (66426)

```
« SZCHK CIRC CIDX 1
+ GET LIST→ DROP »
```

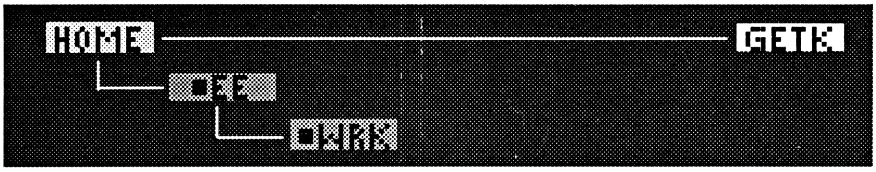
Summary: GETE recalls the current circuit element (as specified by 'CIDX') from the current circuit ('CIRC'). SZCHK is used to determine whether or not the circuit is empty.

Inputs: None – data is taken from 'CIRC'.

Outputs: Level 4 – an integer – the "from" mesh or node.
Level 3 – an integer – the "to" mesh or node.
Level 2 – a real or complex number – the element value.
Level 1 – a character string – the element type.

Errors: Unpredictable errors will occur if the circuit has not been properly initialized.

Notes: GETE uses SZCHK. Put GETE into the ▪EE directory.



Get A Keystroke And Return Its Name:

GETK (32655)

```
« DO UNTIL KEY END »
```

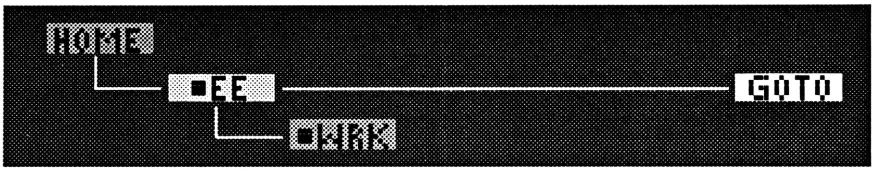
Summary: GETK is useful for getting a single keystroke input from the user. The routine causes the HP-28S to pause and wait for the keystroke. Then it will return that key's name to Level 1 of the stack. GETK will run indefinitely if no key is pressed.

Inputs: Any keystroke except ATTN (ON).

Outputs: Level 1 – the key's name.

Errors: Pressing ATTN will halt the program and not return a key name – possibly leaving a 0 on the stack if so halted.

Notes: Since GETK could be useful for many HP-28S routines, put it in the HOME directory (for use with this book, the ▪EE directory is also OK).



Go To The Specified Circuit Element:

GOTO (93364)

```
« ABS IP 1 MAX Size
MIN →NUM 'CIDX' STO
EDIT »
```

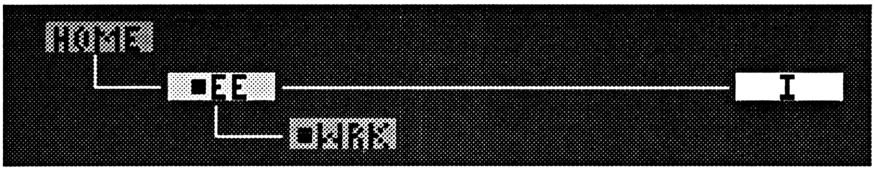
Summary: GOTO stores the integer value in Level 1 into 'CIDX', thus moving the circuit pointer to the specified value. If this index value is less than 1, it will default to 1. Input values greater than the size of the circuit will default to the number of the last circuit element. Complex numbers and vectors/arrays will not cause error messages, but their magnitudes (ABS) will be used as the input index. GOTO returns to the EDIT menu when done.

Inputs: Level 1 – an integer – the index.

Outputs: None.

Errors: Undefined Name will occur if the input value is an undefined name. Bad Argument Type will occur if the input value is a list, binary integer or program object.

Notes: GOTO uses Size and EDIT. Put GOTO into the ■EE directory.



Add A Current Element To The Circuit List:

I (7640)

```
« "I" PUTE »
```

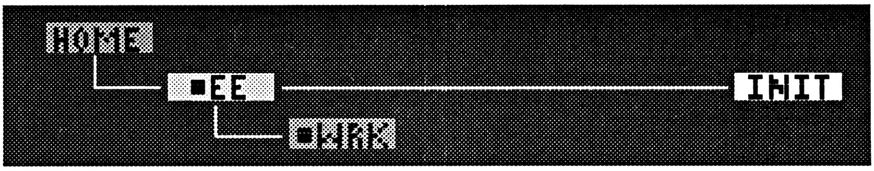
Summary: I invokes PUTE (PUT Element) in order to add a current-type element ("I") to the current circuit ('CIRC').

Inputs: Level 3 – an integer – the "from" node.
Level 2 – an integer – the "to" node.
Level 1 – a real or complex number – the current value (I).

Outputs: None.

Errors: If the circuit is empty when I is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: I is necessary *only* for Node-Voltage circuit analysis. If you need it, put I into the ■EE directory. I uses PUTE.



Initialize The Current Circuit:

INIT (144451)

```
« IF "INITIALIZE"
YES? THEN CIRCS MENU
ELSE MAIN END »
```

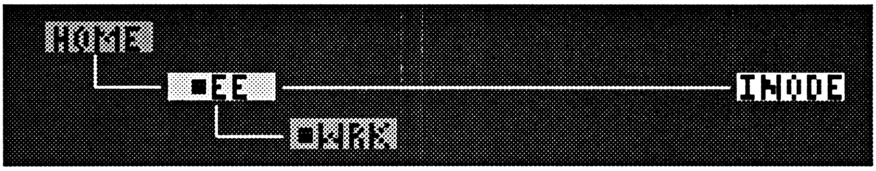
Summary: INIT asks the user whether or not the circuit should be initialized. If the response is yes, a menu of possible circuit types (actually, initializing routines) is presented and the program exits. Otherwise (if the response is no), control is passed back to the MAIN menu.

Inputs: None.

Outputs: None.

Errors: None.

Notes: INIT uses YES? and MAIN. Put INIT into the ▪EE directory.



Store A Current Into The Current Array:

INODE (443609)

```
« NEG → V « IF DUP2
* NOT THEN - DUP
SIGN V * 'V' STO ABS
1 1 ELSE 1 ROT 1 -1
END 1 'VA' V PUTX »
»
```

Summary: INODE stores a current into the current array.

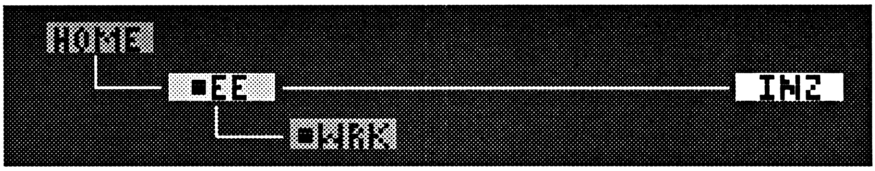
Inputs: Level 3 – an integer – the "from" node.
Level 2 – an integer – the "to" node.
Level 1 – a real or complex number – the current value (I).

Outputs: None.

Errors: Infinite Result will occur if the current value is zero and flag 59 (infinite result action) is set.

Notes: INODE is *required for both* Node-Voltage *and* Mesh-Current analyses – but not Ladder Network analysis. If you need it, put it into the ■EE directory.

INODE uses PUTX.



Calculate The Input Impedance:

INZ (133435)

```
« LADRENGN '(ZA(1,1)
*ZL+ZA(1,2))/(ZA(2,1
)*ZL+ZA(2,2))' EVAL
»
```

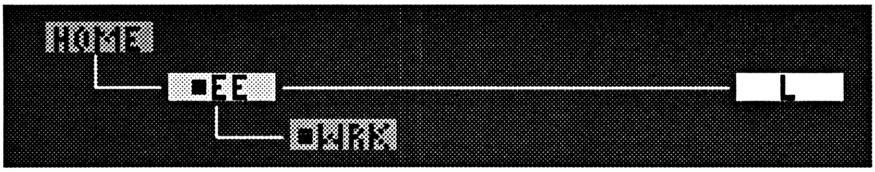
Summary: INZ calculates the input impedance for the current LADR-type network.

Inputs: None (data is taken from existing objects).

Outputs: Level 1 – a complex number – the input impedance value (INZ).

Errors: Infinite Result will occur if is (0,0) and flag 59 (infinite result action) is set. Unpredictable errors may occur if the circuit is not properly initialized or if it contains invalid data.

Notes: INZ uses LADRENGN. Put INZ into the ■EE directory.



Add An Inductance Element To The Circuit List:

L (7694)

```
« "L" PUTE »
```

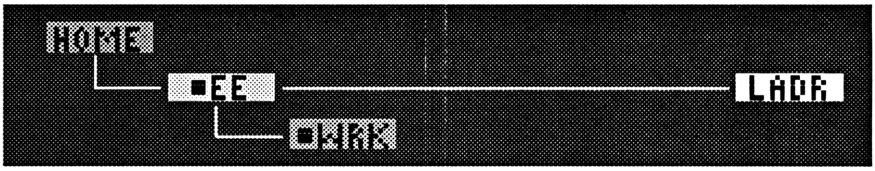
Summary: L invokes PUTE (PUT Element) in order to add an inductance-type element ("L") to the current circuit ('CIRC').

Inputs: Level 3 – an integer – the "from" node or forward mesh.
Level 2 – an integer – the "to" node or reverse mesh.
Level 1 – a real number – the inductance value (L).

Outputs: None.

Errors: If the circuit is empty when L is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: L is *necessary for both* Node-Voltage *and* Mesh-Current circuit analyses. If you need it, put L into the ■EE directory. L uses PUTE.



Initialize A New LADR-Type Circuit:

LADR (24968)

```
« "LADR" CINIT MAIN
»
```

Summary: LADR initializes the current circuit and gives it the type "LADR". All information previously in the current circuit is lost.

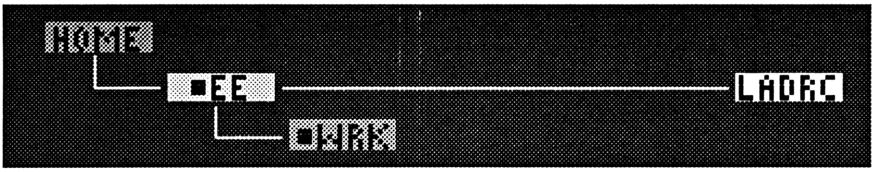
Inputs: None.

Outputs: None – initial values are written into CIRC.

Errors: None.

Notes: LADR is necessary *only* for Ladder Network Transfer analysis; it is useless for the other two kinds of circuit analysis.

LADR uses CINIT and MAIN. Put LADR into the ■EE directory.



Prepare A Menu For LADR Calculations:

LADRCALC (90909)

```
« { INZ FTA PTR CTR
VTR FTZ EXIT } MENU
»
```

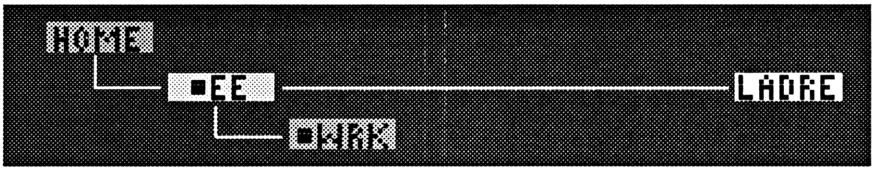
Summary: LADRCALC creates a menu of the possible ladder network calculations. The extra item, 'EXIT', is included to allow the user to leave the menu without necessarily performing a calculation.

Inputs: None.

Outputs: None.

Errors: None.

Notes: LADRCALC is necessary *only* for Ladder Network Transfer analysis. If you need it, put it into the ▫EE directory.



List Of Valid Elements In A LADR-Type Circuit:

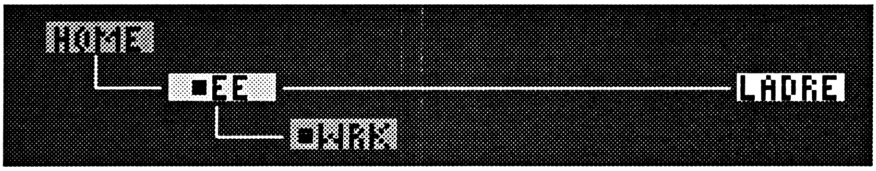
LADRELS (57943)

```
{ RS RP CS CP LS LP
ZS ZP EXIT }
```

Summary: LADRELS is just a simple list – not a program object (and so it's meaningless to speak of **Inputs**, **Outputs**, or **Errors**).

LADRELS is the list of element types permissible in a "LADR" type circuit. This list is used to create the ADD and CHNG menus. If other types of elements are made valid by later modification of the appropriate calculations routines, then the names of those new element types could simply be added to the above names.

Notes: LADRELS is necessary *only* for Ladder Network Transfer analysis. If you need it, then put it into the ▪EE directory.

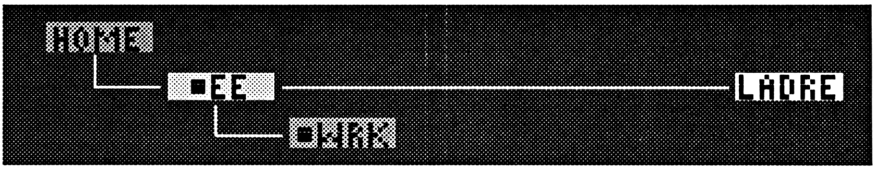


Main Ladder Calculation Engine:

LADRENGN (3745032)

```
« IF 3 FC? THEN 2
IDN DUP 0 CON R→C
'ZA' STO Freq 2 * π
* →NUM 'ω' STO CIRC
2 GET LIST→ DROP NUM
CHR IF "LC" SWAP POS
DUP THEN IF 1 ==
THEN ω * ELSE ω *
NEG INV END i * →NUM
ELSE DROP END 'ZL'
STO CIRC 3 OVER SIZE
SUB LIST→ 1 SWAP
START LIST→ DROP
"LADR" + STR→ 'ZA'
SWAP STO* NEXT 3 SF
END »
```

Summary: LADRENGN is the main calculation routine for Ladder Network Transfer analysis. First, it tests flag 3 to see if the impedance array, 'ZA', needs calculating. If so, it creates the complex impedance array, initializes the circuit radian frequency 'ω', stores the value of the last im-



pedance in 'ZL', and then interprets the circuit list to calculate the value of 'ZA'.

At the end of the routine, flag 3 is set to indicate that the impedance array is current and does not need to be recalculated in order to perform other ladder network calculations.

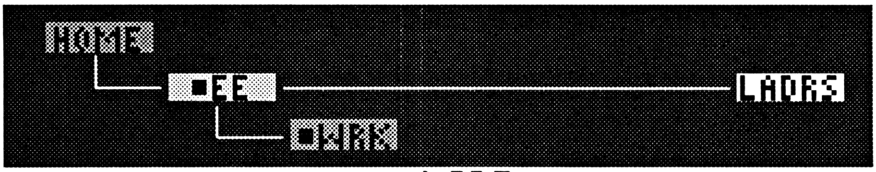
Inputs: None – data is taken from the CIRC and other objects.

Outputs: None – results are stored in the array, ZA.

Errors: Unpredictable errors may occur if the circuit has not been properly initialized or if it contains invalid data.

Notes: LADRENGN is necessary *only* for Ladder Network Transfer analysis. If you need it, put it into the ■EE directory. Note that LADRENGN will *appear* identical to LADRELS in your menu display, since both names are longer than the menu can display.

LADRENGN uses Freq, RSLADR, RPLADR, CSLADR, CPLADR, LSLADR, LPLADR, ZSLADR, and ZPLADR.



Display The Current LADR-Circuit Element:

LADRSHOW (795599)

```
« GETE → V Y « "#"
CIDX SIP + 58 CHR +
"  " + Y + "■" + 134
CHR " " + V →STR +
IF DUP "■" POS NOT
THEN "■" + END + 1
FC? « 1 DISP » IFT »
»
```

(■'s here are NEWLINE characters.)

Summary: LADRSHOW displays the current ("nth") LADR element:

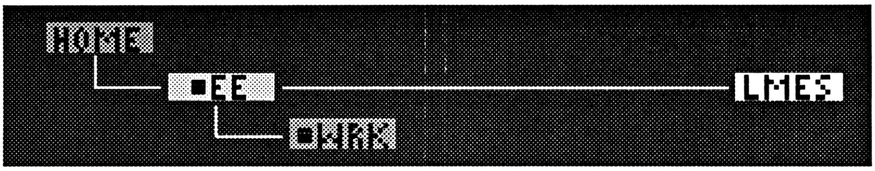
#<n>: <type>
▶ <value>

Inputs: None.

Outputs: The display described above.

Errors: Unpredictable errors occur with an uninitialized circuit.

Notes: LADRSHOW is necessary *only* for Ladder Network Transfer analysis. If you need it, put it into the ■EE directory. LADRSHOW uses GETE and SIP.



Store An Inductance Into The Impedance Array:

LMESH (20051)

```
« w * i * ZMESH »
```

Summary: LMESH stores an inductance as an impedance (calculated by Z = **i**ω L)) into the impedance array, 'ZA'.

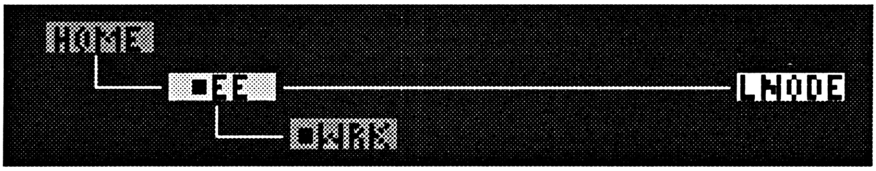
Inputs: Level 3 – an integer – the forward loop.
Level 2 – an integer – the reverse loop.
Level 1 – a real number – the inductance value (L).

Outputs: None.

Errors: Infinite Result will occur if the inductance or frequency value is zero and flag 59 (infinite result action) is set.

Notes: LMESH is necessary *only* for Mesh-Current circuit analysis. If you need it, put it into the ■EE directory.

LMESH uses ZMESH.



Store An Inductance Into The Admittance Array:

LNODE (19769)

```
« w * i * ZNODE »
```

Summary: LNODE stores an inductance as an admittance (calculated by $Y = -i/\omega L$) into the admittance array, 'ZA'.

Inputs: Level 3 – an integer – the "from" node.
Level 2 – an integer – the "to" node.
Level 1 – a real number – the inductance value (L).

Outputs: None.

Errors: Infinite Result will occur if the inductance or frequency value is zero and flag 59 (infinite result action) is set.

Notes: LNODE is necessary *only* for Node-Voltage circuit analysis. If you need it, put it into the ▪EE directory.

LNODE uses ZNODE.

Load A Saved Circuit:

LOAD (370746)

```
« IF Size THEN
"OVERWRITE Current■Circuit"
YES? NOT 'MAIN' IFT
END ■CIRC VARS
'EXIT' + MENU »
```

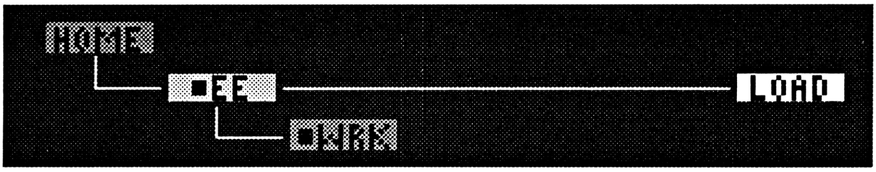
(The ■ here is the NEWLINE character.)

Summary: LOAD presents a menu of SAVE'd circuits (from the ■CIRC directory) from which to choose. Pressing one of the named keys will make that circuit the current circuit. If the current circuit is non-empty, LOAD will prompt the user with:

```
OVERWRITE Current
Circuit?
(Y or N)
```

A "yes" response (see YES?) will cause the chosen circuit to overwrite the current circuit. Otherwise control is passed back to MAIN and the current circuit is retained.

An additional, non-circuit entry is provided in the menu: EXIT allows you to abort the LOAD operation without loading or overwriting.



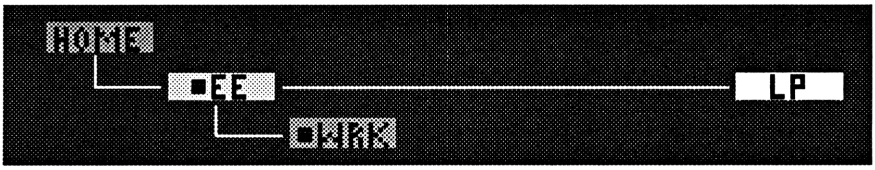
Inputs: Keyboard – a "yes" (either Y or ENTER) or "no" (any other) keystroke – to confirm overwriting of the current circuit with the one specified.

Outputs: None.

Errors: Garbage will be left in the stack if ATTN is pressed in response to the (Y or N) prompt. Undefined Name will occur if the current circuit is undefined. Bad Argument Type or other unpredictable errors will occur if the newly-loaded circuit's data list is damaged or improperly initialized.

Notes: LOAD is an *optional* routine. You do *not* need it in order to perform any of the three kinds of circuit analysis *unless* you wish to name and save different circuits in the HP-28S (which is certainly very handy). If you do use LOAD, put it into the ▪EE directory.

LOAD uses Size, YES?, MAIN, and EXIT.



Add A Parallel-Inductance Element To The Circuit List:

LP (10108)

```
« "LP" PUTE »
```

Summary: LP invokes PUTE (PUT Element) to add a parallel-inductance element ("LP") to the current circuit ('CIRC').

Inputs: Level 1 – a real number – the parallel inductance value (LP).

Outputs: None.

Errors: If the circuit is empty when LP is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: LP is necessary *only* for Ladder Network Transfer analysis. If you need it, put LP into the ■EE directory.

LP uses PUTE.

HOME
EE
LPLAD

Create A Parallel Inductance Transfer Matrix:

LPLADR (23498)

```
« w * i * RPLADR »
```

Summary: LPLADR creates a parallel inductance transfer matrix of this form:

$$\begin{bmatrix} 1 & 0 \\ \frac{-i}{\omega L} & 1 \end{bmatrix}$$

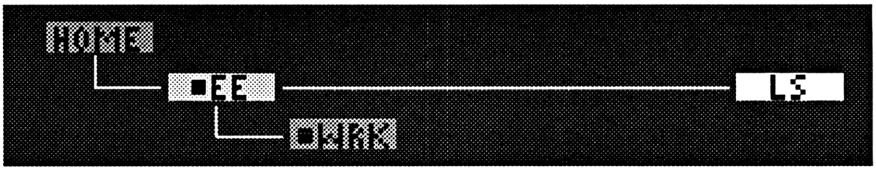
Inputs: Level 1 – a real number – the inductance value.

Outputs: Level 1 – a two-by-two array – the parallel inductance transfer matrix.

Errors: Bad Argument Type will occur if the input inductance is not a real number or if 'w' is undefined.

Notes: LPLADR is necessary only for Ladder Network Transfer analysis. If you need it, put it into the ▪EE directory.

LPLADR uses RPLADR.



Add A Series-Inductance Element To The Circuit List:

LS (10171)

```
« "LS" PUTE »
```

Summary: LS invokes PUTE (PUT Element) to add a series-inductance element ("LS") to the current circuit ('CIRC').

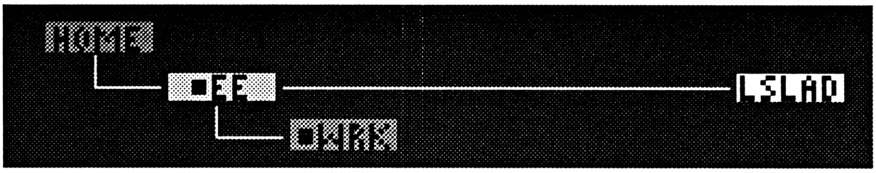
Inputs: Level 1 – a real number – the series inductance value (LS).

Outputs: None.

Errors: If the circuit is empty when LS is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: LS is necessary *only* for Ladder Network Transfer analysis. If you need it, put LS into the ■EE directory.

LS uses PUTE.



Create A Series Inductance Transfer Matrix:

LSLADR (23597)

```
« w * i * RSLADR »
```

Summary: LSLADR creates a series inductance transfer matrix of this form:

$$\begin{bmatrix} 1 & i\omega L \\ 0 & 1 \end{bmatrix}$$

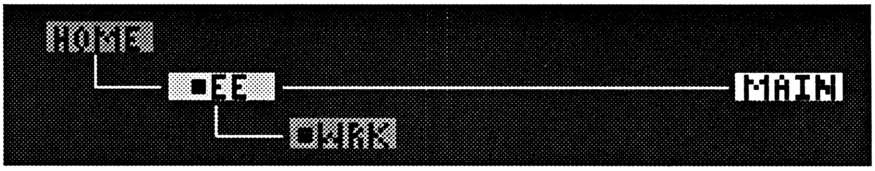
Inputs: Level 1 – a real number – the inductance value.

Outputs: Level 1 – a two-by-two array – the series inductance transfer matrix.

Errors: Bad Argument Type will occur if the input inductance is not a real number or if 'w' is undefined.

Notes: LSLADR is necessary only for Ladder Network Transfer analysis. If you need it, put it into the ■EE directory.

LSLADR uses RSLADR.



Display The Main Circuit-Editor Menu:

MAIN (224264)

```
« ■WRK { INIT EDIT
LOAD SAVE CALC FREQ
QUIT } MENU 1 CF 2
CF 3 CF CDSP ABORT »
```

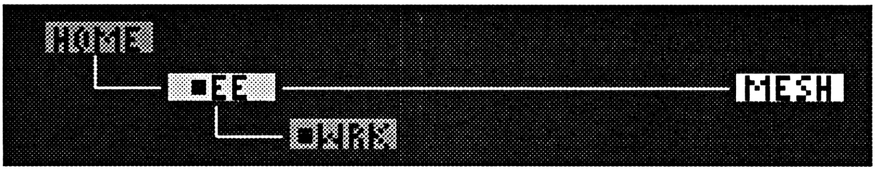
Summary: MAIN creates the main circuit editor menu, clears flags 1, 2, and 3 (which are the printing, ADD/CHNG, and LADR calculation flags respectively), displays the current circuit information and then aborts, ensuring that the program does not return control to any calling program.

Inputs: None.

Outputs: None.

Errors: Undefined Name will occur if the current circuit is undefined. Bad Argument Type or other unpredictable errors will occur if the circuit data list is damaged or improperly initialized.

Notes: MAIN uses CDSP. Put MAIN into the ■EE directory.



Initialize A New MESH-Type Circuit:

MESH (25246)

```
« "MESH" CINIT MAIN
»
```

Summary: MESH initializes the current circuit and gives it the type "MESH". All information previously in the current circuit is lost.

Inputs: None.

Outputs: None – initial values are written into CIRC.

Errors: None.

Notes: MESH is necessary *only* for Mesh-Current circuit analysis; it is useless for the other two kinds of circuit analysis.

MESH uses CINIT and MAIN. Put MESH into the ■EE directory.

Perform The Mesh-Current Calculation:

MESHCALC (17129)

```
« NODECALC »
```

Summary: MESHCALC calls NODECALC, which interprets the current circuit list and creates the two complex arrays, 'ZA' (the impedance array) and 'VA' (the voltage source array). It then divides 'VA' by 'ZA' to give the array of mesh currents.

First, SZCHK is called to test for an empty circuit. If all is well, the cyclic frequency is then converted to a radian frequency and stored in 'w'. Next, the maximum mesh number ("n") is found and used to set the dimensions of 'ZA' and 'VA'. Then it the mesh storage routine ("ZMESH") is called for each element .

Inputs: None – all data is taken from CIRC and other objects.

Outputs: Level 1 – an array of complex numbers – the mesh currents for the current circuit (with "n" meshes), in this format:

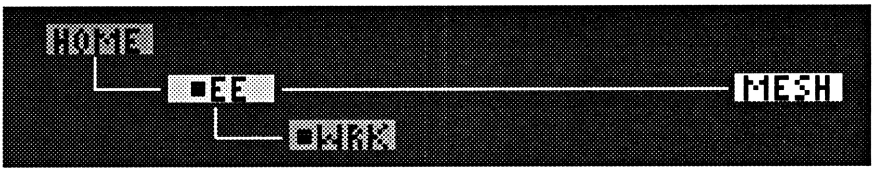
$$
\begin{array}{l}
[\,[\,<I_1>\,] \\
\;\;[\,<I_2>\,] \\
\;\;[\,<I_3>\,] \\
\;\;\;\;. \\
\;\;\;\;. \\
\;\;\;\;. \\
\;\;[\,<I_n>\,]\,]
\end{array}
$$

HOME
■EE
■CIRC
MESH

Errors: Undefined Name will occur if CIRC is undefined. Bad Argument Type will occur if CIRC contains a non-list or a damaged list. Unpredictable errors will occur if the circuit has not been properly initialized.

Notes: MESHCALC is necessary *only* for Mesh-Current circuit analysis. If you need it, put it into the ■EE directory. Notice that MESHCALC will *appear* the same as MESH in your menu display, since both names are as long or longer than menu items can display.

MESHCALC uses NODECALC.



List Of Valid Elements In A MESH-Type Circuit:

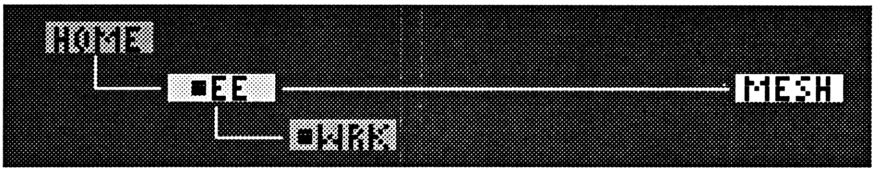
MESHELS (25677)

```
{ R C L Z V EXIT }
```

Summary: MESHELS is just a simple list – not a program object (and so it's meaningless to speak of **Inputs**, **Outputs**, or **Errors**).

MESHELS is the list of element types permissible in a "MESH" type circuit. This list is used to create the ADD and CHNG menus. If other types of elements are made valid by later modification of the appropriate calculations routines, then the names of those new element types could simply be added to the above names.

Notes: MESHELS is necessary *only* for Mesh-Current circuit analysis. If you need it, then put it into the ■EE directory. Notice that MESHELS will *appear* the same as MESH in your menu display, since both names are as long or longer than menu items can display.



Display The Current MESH-Circuit Element:

MESHSHOW (1316849)

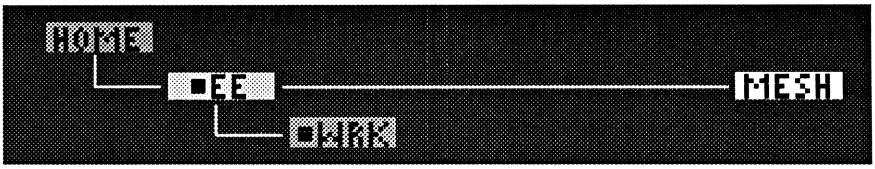
```
« GETE → F T V Y «
"#" CIDX SIP + 58
CHR + "  " + F SIP +
" → " Y " " 142 CHR
OVER + + + + + T SIP
+ "■" + 134 CHR " "
+ V →STR + IF DUP
"■" POS NOT THEN "■"
+ END + 1 FC? « 1
DISP » IFT » »
```

(The ■'s appearing here are all NEWLINE characters.)

Summary: MESHSHOW displays the current ("nth") element (as indicated by 'CIDX') of the current circuit (contained in 'CIRC') in the following format:

#<n>: <forward> → <type> ← <reverse>
▶ <value>

Inputs: None.

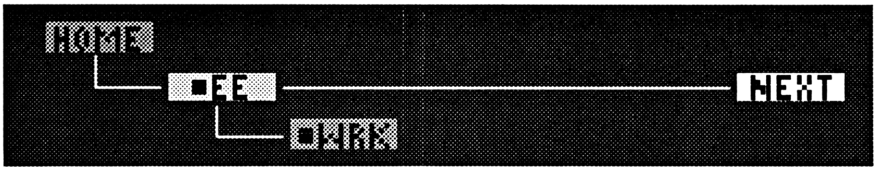


Outputs: The display described on the previous page.

Errors: `Bad Argument Type` will occur if `SIP` encounters any object which does not evaluate to a real number (or, `Undefined Name` will occur if that object is an undefined name). Unpredictable errors will occur if the circuit has not been properly initialized.

Notes: `MESHSHOW` is necessary *only* for Mesh-Current circuit analysis. If you need it, put it into the `■EE` directory. Notice that `MESHSHOW` will *appear* the same as `MESH` in your menu display, since both names are as long or longer than menu items can display.

`MESHSHOW` uses `GETE` and `SIP`.



Move To And Display The Next Circuit Element:

Next (128502)

```
« SZCHK Size CIDX ==
1 'CIDX+1' IFTE
'CIDX' STO Show »
```

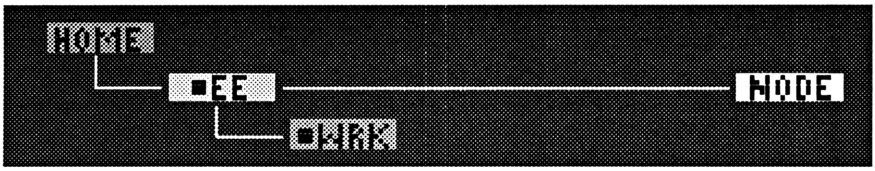
Summary: Next increments by one the the circuit index (CIDX), except under two conditions: If the circuit is empty (i.e. if CIDX is 0) SZCHK reports the condition and passes control to MAIN. If incrementing 'CIDX' would make it greater than the Size of the circuit, 'CIDX' is set to 1, thus "wrapping" around to the circuit's first element.

Inputs: None.

Outputs: None.

Errors: Bad Argument Type will occur if 'CIDX' is undefined or contains a non-real value, or if 'CIRC' contains a non-list or a damaged list. Undefined Name will occur if 'CIRC' is undefined. If Type returns a string that gives a nonexistent name when combined with "SHOW", that name will be left on the stack.

Notes: Next uses SZCHK, Size, and Show. Put Next into the ■EE directory.



Initialize A New NODE-Type Circuit:

NODE (24999)

```
« "NODE" CINIT MAIN
»
```

Summary: NODE initializes the current circuit and gives it the type "NODE". All information previously in the current circuit is lost.

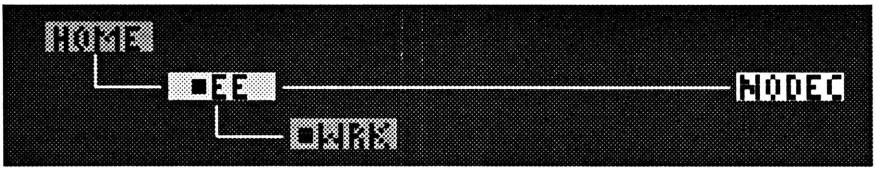
Inputs: None.

Outputs: None – initial values are written into CIRC.

Errors: None.

Notes: NODE is necessary *only* for Node-Voltage circuit analysis; it is useless for the other two kinds of circuit analysis.

NODE uses CINIT and MAIN. Put NODE into the ■EE directory.



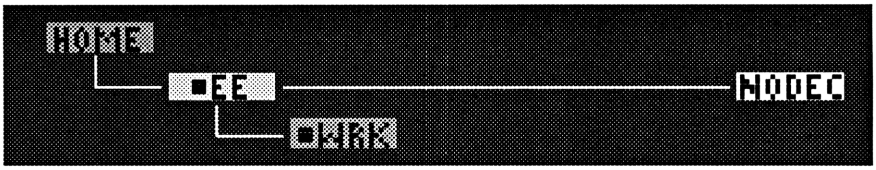
Perform The Node-Voltage Calculation:

NODECALC (2377587)

```
« SZCHK Freq 2 * π *
→NUM 'w' STO CIDX 0
1 Size FOR I I
'CIDX' STO GETE
DROP2 MAX MAX NEXT
SWAP 'CIDX' STO
(0,0) DUP2 OVER ROT
2 →LIST SWAP CON
'ZA' STO SWAP 1 2
→LIST SWAP CON 'VA'
STO CIRC LIST→ 2
SWAP START LIST→
DROP Type + STR→
NEXT DROP VA ZA / »
```

Summary: NODECALC interprets the current circuit list and creates the two complex arrays, 'ZA' (the impedance array) and 'VA' (the current source array). It then divides 'VA' by 'ZA' to give the array of node voltages.

First, SZCHK is called to test for an empty circuit. If all is well, the cyclic frequency is then converted to a radian frequency and stored in 'w'. Next, the maximum node number is found and used to set the dimensions of 'ZA'



and 'VA'. Then the node storage routine is called for each element. NODECALC is used for both node and mesh calculations.

Inputs: None – all data is taken from CIRC and other objects.

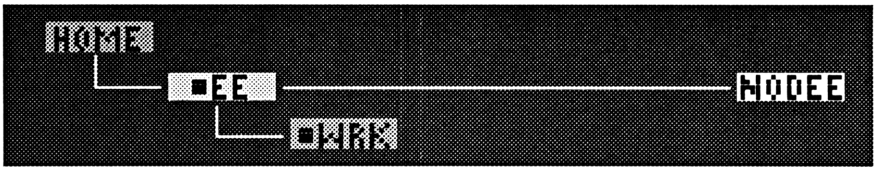
Outputs: Level 1 – an array of complex numbers – the node voltages for the current circuit (with "n" nodes), in this format:

[[<V_1>]
 [<V_2>]
 [<V_3>]
 .
 .
 .
 [<V_n>]]

Errors: Undefined Name will occur if CIRC is undefined. Bad Argument Type will occur if CIRC contains a non-list or a damaged list. Unpredictable errors will occur if the circuit has not been properly initialized.

Notes: NODECALC is *necessary for both* Node-Voltage *and* Mesh-Current circuit analysis – but it's unnecessary for Ladder Network analysis. If you need NODECALC, put it into the ■EE directory.

NODECALC uses SZCHK, Freq, Size, GETE, and Type.



List Of Valid Elements In A NODE-Type Circuit:

NODEELS (25365)

```
{ R C L Z I EXIT }
```

Summary: NODEELS is just a simple list – not a program object (and so it's meaningless to speak of **Inputs**, **Outputs**, or **Errors**).

NODEELS is the list of element types permissible in a NODE-type circuit. This list is used to create the ADD and CHNG menus. If other types of elements are made valid by later modification of the appropriate calculations routines," then the names of those new element types could simply be added to the above names.

Notes: NODEELS is necessary *only* for Node-Voltage circuit analysis. If you need it, then put it into the ■EE directory.

Display The Current NODE-Circuit Element:

NODESHOW (1143846)

```
« GETE → F T V Y «
"#" CIDX SIP + 58
CHR + "  " + F SIP +
" → " Y OVER + + + T
SIP + "▪" + 134 CHR
" " + V →STR + IF
DUP "▪" POS NOT THEN
"▪" + END + 1 FC? «
1 DISP » IFT » »
```

(The ▪'s appearing here are all NEWLINE characters.)

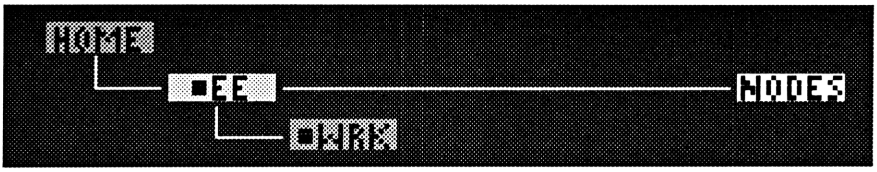
Summary: NODESHOW displays the current ("nth") element (as indicated by 'CIDX') of the current circuit (contained in 'CIRC') in the following format:

#<n>: <from> → <type> → <to>
▶ <value>

Inputs: None.

Outputs: The display described above.

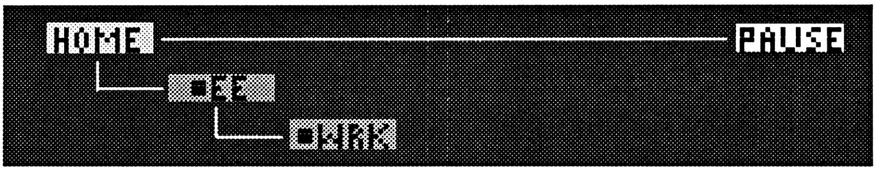
Errors: Bad Argument Type will occur if SIP encounters any object which does not evaluate to a real number (or,



Undefined Name will occur if that object is an undefined name). Unpredictable errors will occur if the circuit has not been properly initialized.

Notes: NODESHOW is necessary *only* for Node-Voltage circuit analysis. If you need it, put it into the ▪EE directory.

NODESHOW uses GETE and SIP.



Pause A Program For Two Seconds:

PAUSE (10492)

```
« 2 WAIT »
```

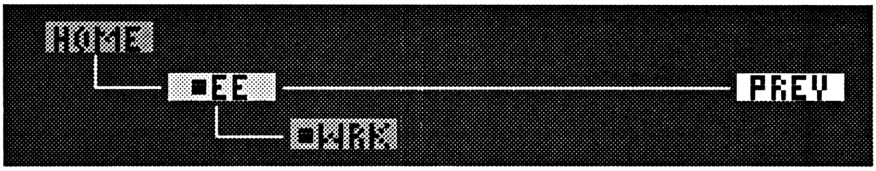
Summary: PAUSE suspends program execution for two seconds – useful for prolonging the time that a message is displayed.

Inputs: None.

Outputs: None.

Errors: None.

Notes: Since PAUSE could be generally useful to other HP-28S routines, put it in the HOME directory. For the purposes of this book, however, the ▪EE directory is OK, too.



Move To And Display The Previous Circuit Element:

PREV (132360)

```
« SZCHK CIDX 1 > '
CIDX-1' 'Size' IFTE
'CIDX' STO Show »
```

Summary: PREV decrements by one the circuit index (CIDX), except under two conditons: If the circuit is empty (i.e. if CIDX has a value of 0), then SZCHK reports this condition and passes control to MAIN. If decrementing 'CIDX' would make it less than 1, 'CIDX' is set to the Size of the circuit, effectively "wrapping" to the last circuit element.

Input: None.

Output: None.

Errors: Bad Argument Type will occur if 'CIDX' is undefined or contains a non-real value, or if 'CIRC' contains a non-list or a damaged list. Undefined Name will occur if 'CIRC' is undefined. If Type returns a string that produces a nonexistent name when combined with "SHOW", that name will be left on the stack.

Notes: PREV uses SZCHK, Size, and Show. Put PREV into the ▪EE directory.

Print The Current Circuit:

PRINT (282849)

```
« SZCHK 1 SF CIDX 1
Size FOR I I 'CIDX'
STO Show PR1 DROP CR
NEXT 1 CF 'CIDX' STO
»
```

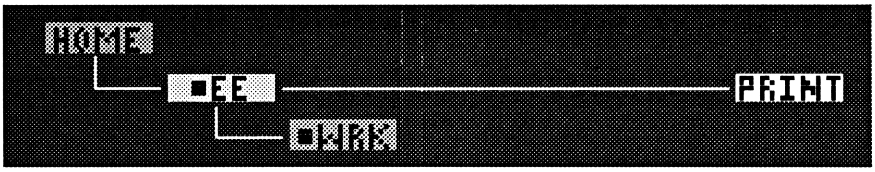
Summary: PRINT prints a sequential listing of the current circuit (all "n" elements) on the infrared printer. For each element, PRINT uses the same format for the printer as Show uses for the display (but the format depends on the circuit type; the one shown here is for a NODE-type circuit):

```
#1: <from> → <type> → <to>
▶ <value>

#2: <from> → <type> → <to>
▶ <value>

#3: <from> → <type> → <to>
▶ <value>
.
.
.
#<n>: <from> → <type> → <to>
▶ <value>
```

If the circuit is empty, nothing will be printed, but the display will show `0 Element Circuit`.



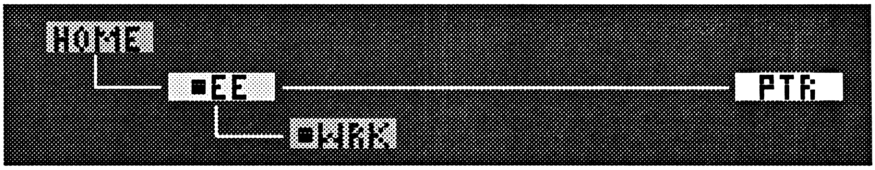
Inputs: None.

Outputs: The printed listing described on the previous page.

Errors: If Type returns a string that produces a nonexistent name when combined with "SHOW", execution will halt, and that name will be left on the stack. Undefined Name will occur if CIRC is undefined. Bad Argument Type will occur if CIRC contains a non-list or a damaged list.

Notes: PRINT is an *optional* routine. You do *not* need it in order to use any of the three kinds of circuit analysis *unless* you want to be able to make paper print-outs of the elelments and configuration(s) of the circuit(s) you analyze. If you do use PRINT, put it into the ■EE directory.

PRINT uses SZCHK, Size, and Show.



Calculate The Ladder Network Power Transfer Ratio:

PTR (41035)

```
« CTR ABS SQ ZL RE
INZ RE / * »
```

Summary: PTR calculates the value of the real Power Transfer Ratio (the Power Gain) for a LADR-type circuit.

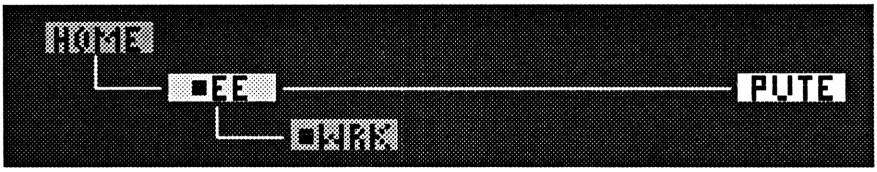
Inputs: None – data is taken from ZL and ZA.

Outputs: Level 1 – a real number – the real Power Transfer Ratio.

Errors: Infinite result will occur if either 'ZA(2,1)*ZL+ZA(2,2)' or the *real portion* of 'ZA(1,1)*ZL+ZA(1,2)/ZA(2,1)*ZL+ZA(2,2)' is zero.

Notes: PTR is necessary only for Ladder Network Transfer analysis. If you need it, put PTR into the ▪EE directory.

PTR uses CTR and INZ.



Put A Circuit Element Into The Circuit:

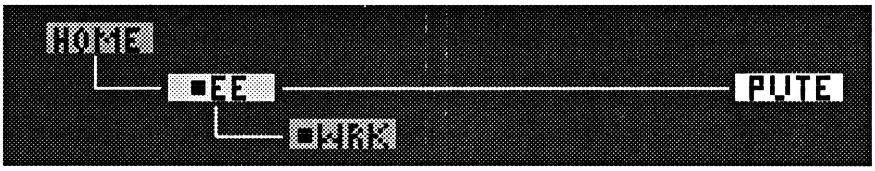
PUTE (526943)

```
« 2 FS? 'SZCHK' IFT
"PUT" Type + STR→
CIRC IF 2 FC? THEN 0
+ Size 1 + 'CIDX'
STO END CIDX 1 + ROT
PUT 'CIRC' STO EDIT
»
```

Summary: PUTE puts the circuit element on the stack into the current circuit at the indexed location. Either PUTNODE, PUTMESH, or PUTLADR is called, depending on the circuit's Type.

PUTE also determines whether it has been called by ADD (flag 2 clear) or by CHNG (flag 2 set): if it's called from ADD, then a new element is added to the end of the list and CIDX is updated to point to it; if it's called from CHNG, then SZCHK is called to determine if the circuit is empty, and CIDX is left unmodified. In either case, control is passed to EDIT upon completion.

Inputs: The necessary inputs depend on the circuit type, as follows:



For a MESH-analysis circuit:

Level 3 – an integer – the forward mesh.
Level 2 – an integer – the reverse mesh.
Level 1 – a real or complex number – the element value.

For a NODE-analysis circuit:

Level 3 – an integer – the "from" node.
Level 2 – an integer – the "to" node.
Level 1 – a real or complex number – the element value.

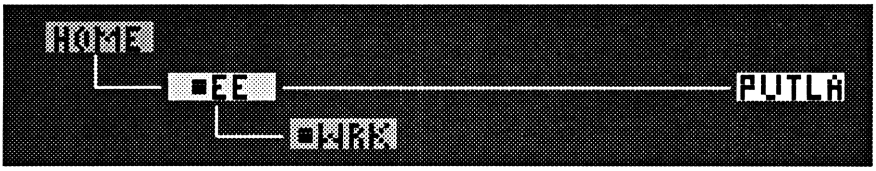
For a LADR-analysis circuit:

Level 1 – a real or complex number – the element value.

Outputs: None.

Errors: If the circuit is empty when PUTE is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: PUTE uses SZCHK, Type, Size, and EDIT. Place PUTE into the ■EE directory.



Prepare To "Put" Into a LADR-Type Circuit:

PUTLADR (15138)

```
« 2 →LIST »
```

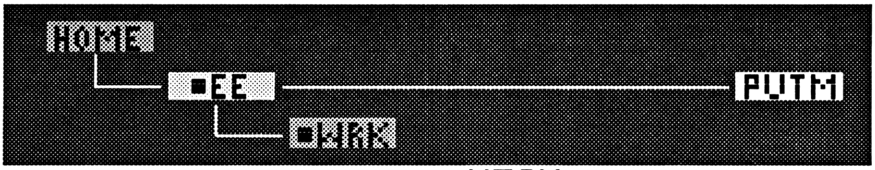
Summary: PUTLADR puts the bottom two stack values into a list, preparing for storage into the current circuit.

Inputs: Level 2 – a real or a complex number – the element value.
Level 1 – a string – the element type.

Outputs: Level 1 – a list object – the list of two input values.

Errors: Too Few Arguments will occur if the stack does not contain the Levels specified (at least two).

Notes: PUTLADR is necessary *only* for doing Ladder Network Transfer analysis. If you need it, put it into the ▪EE directory.



Prepare To "Put" Into a MESH-Type Circuit:

PUTMESH (15118)

```
« PUTNODE »
```

Summary: PUTMESH calls PUTNODE, which "cleans" (i.e. takes the absolute values of the integer portions of) the forward and reverse mesh numbers. Then it puts the bottom four stack values into a list, preparing for storage into the current circuit.

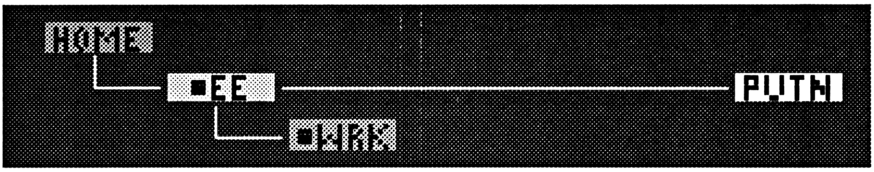
Inputs: Level 4 – an integer – the forward mesh specifier.
Level 3 – an integer – the reverse mesh specifier.
Level 2 – a real or a complex number – the element value.
Level 1 – a string – the element type.

Outputs: Level 1 – a list object – the four "cleaned" input values.

Errors: Too Few Arguments will occur if the stack does not contain the Levels specified (at least four).

Notes: PUTMESH is necessary *only* for doing Mesh-Current circuit analysis. If you need it, put it into the ■EE directory.

PUTMESH uses PUTNODE.



Prepare To "Put" Into A NODE-Type Circuit:

PUTNODE (32166)

```
« 3 AI 4 AI 4 →LIST
»
```

Summary: PUTNODE "cleans" (i.e. takes the absolute values of the integer portions of) the "from" and the "to" values. Then it puts the bottom four stack values into a list, preparing for storage into the current circuit.

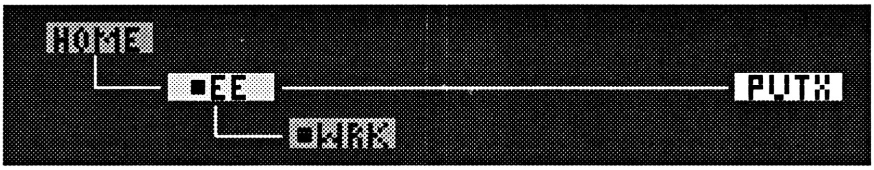
Inputs: Level 4 – an integer – the "from" node specifier.
Level 3 – an integer – the "to" node specifier.
Level 2 – a real or a complex number – the element value.
Level 1 – a string – the element type.

Outputs: Level 1 – a list object – the four "cleaned" input values.

Errors: Too Few Arguments will occur if the stack does not contain the Levels specified (at least four).

Notes: PUTNODE is *necessary for both* Node-Voltage *and* Mesh-Current analyses – in the ■EE directory, if you need it.

PUTNODE uses AI.



Put An Element Value Into An Array:

PUTX (283229)

```
« → A V « SWAP FOR I
2 →LIST A SWAP DUP2
GET V →NUM I SIGN *
+ PUT -2 STEP » »
```

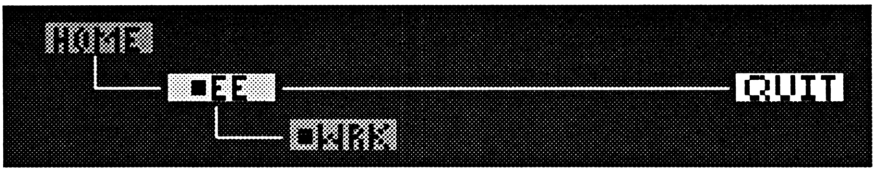
Summary: PUTX combines a circuit-element value with array elements whose indices appear on the stack. The value is *added* to the array elements whose loop index (I) is positive and *subtracted* from elements whose index is negative.

Inputs: Levels 5 through (2n+4) – all 2n of them are integers – the row/column indices of the array.
Level 3 and 4 – two integers giving the range for the loop index, from low to high, respectively.
Level 2 – the array name.
Level 1 – the circuit element value.

Outputs: None.

Errors: Bad Argument Value will occur if a loop index is beyond the available indices of the array. Undefined Name will occur if the array name is undefined.

Notes: Put PUTX into the ■EE directory.



Leave The EEA Application:

QUIT (17143)

```
« HOME 23 MENU »
```

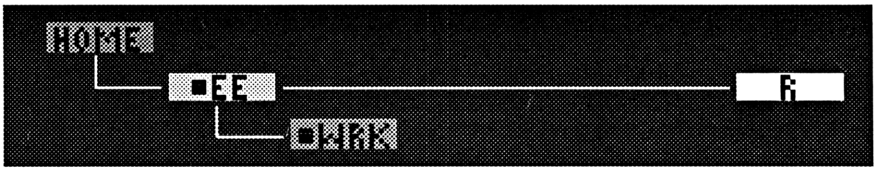
Summary: QUIT leaves the Circuit Analysis Tools by moving to the HOME directory and activating the USER menu.

Inputs: None.

Outputs: None.

Errors: None.

Notes: Put QUIT in the ■EE directory.



Add A Resistance Element To The Circuit List:

R (7802)

```
« "R" PUTE »
```

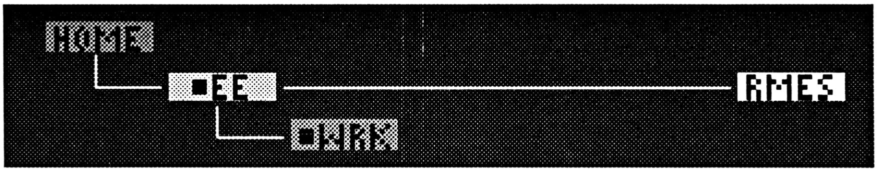
Summary: R invokes PUTE (PUT Element) in order to add a resistance-type element ("R") to the current circuit ('CIRC').

Inputs: Level 3 – an integer – the "from" node or forward mesh.
Level 2 – an integer – the "to" node or reverse mesh.
Level 1 – a real number – the resistance value (R).

Outputs: None.

Errors: If the circuit is empty when R is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: R is *necessary for both* Node-Voltage *and* Mesh-Current circuit analyses. If you need it, put R into the ▪EE directory. R uses PUTE.



Store A Resistance Into The Impedance Array:

RMESH (9517)

```
« ZMESH »
```

Summary: RMESH stores a resistance as an impedance (calculated by Z = R)) into the impedance array, 'ZA'.

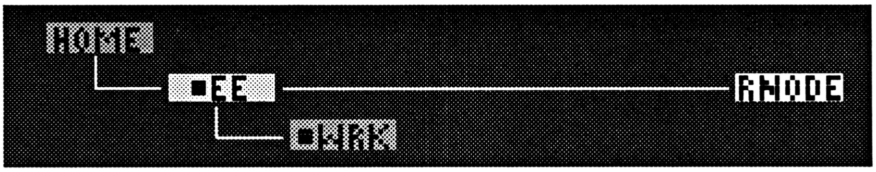
Inputs: Level 3 – an integer – the forward loop.
Level 2 – an integer – the reverse loop.
Level 1 – a real number – the resistance value (R).

Outputs: None.

Errors: Infinite Result will occur if the resistance value is zero and flag 59 (infinite result action) is set.

Notes: RMESH is necessary *only* for Mesh-Current circuit analysis. If you need it, put it into the ▪EE directory.

RMESH uses ZMESH.



Store A Resistance Into The Admittance Array:

RNODE (9347)

```
« ZNODE »
```

Summary: RNODE stores a resistance as an admittance (calculated by Y = 1/R) into the admittance array, 'ZA'.

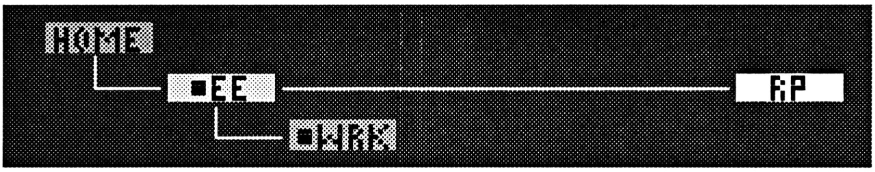
Inputs: Level 3 – an integer – the "from" node.
Level 2 – an integer – the "to" node.
Level 1 – a real number – the resistance value (R).

Outputs: None.

Errors: Infinite Result will occur if the resistance value is zero and flag 59 (infinite result action) is set.

Notes: RNODE is necessary *only* for Node-Voltage circuit analysis. If you need it, put it into the ■EE directory.

RNODE uses ZNODE.



Add A Parallel-Resistance Element To The Circuit List:

RP (10222)

```
« "RP" PUTE »
```

Summary: RP invokes PUTE (PUT Element) to add a parallel-resistance element ("RP") to the current circuit ('CIRC').

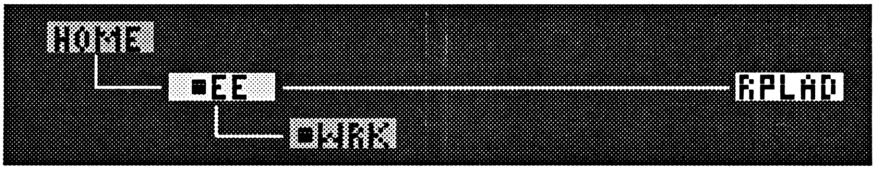
Inputs: Level 1 – a real number – the parallel resistance value (RP).

Outputs: None.

Errors: If the circuit is empty when PUTE is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: RP is necessary *only* for Ladder Network Transfer analysis. If you need it, put RP into the ▪EE directory.

RP uses PUTE.



Create A Parallel Resistance Transfer Matrix:

RPLADR (66741)

```
« INV →NUM 1 0 ROT 1
{ 2 2 } →ARRY »
```

Summary: RPLADR create a parallel resistance transfer matrix of this form:

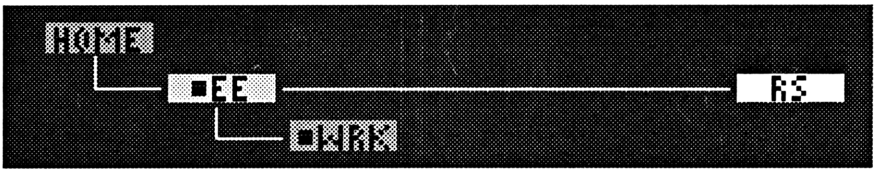
$$\begin{bmatrix} 1 & 0 \\ \frac{1}{R} & 1 \end{bmatrix}$$

Inputs: Level 1 – a real number – the resistance value.

Outputs: Level 1 – a two-by-two array – the parallel resistance transfer matrix.

Errors: Bad Argument Type will occur if the input resistance is not a real or complex number.

Notes: RPLADR is necessary only for Ladder Network Transfer analysis. If you need it, put it into the ▪EE directory.



Add A Series-Resistance Element To The Circuit List:

RS (10285)

```
« "RS" PUTE »
```

Summary: RS invokes PUTE (PUT Element) to add a series-resistance element ("RS") to the current circuit ('CIRC').

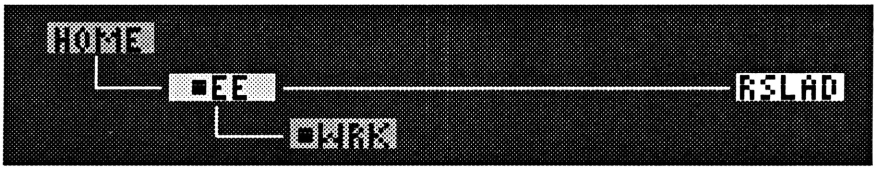
Inputs: Level 1 – a real number – the series resistance value (RS).

Outputs: None.

Errors: If the circuit is empty when RS is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: RS is necessary *only* for Ladder Network Transfer analysis. If you need it, put RS into the ■EE directory.

RS uses PUTE.



Create A Series Resistance Transfer Matrix:

RSLADR (57990)

```
« →NUM 1 SWAP 0 1 {
2 2 } →ARRY »
```

Summary: RSLADR creates a series resistance transfer matrix of this form:

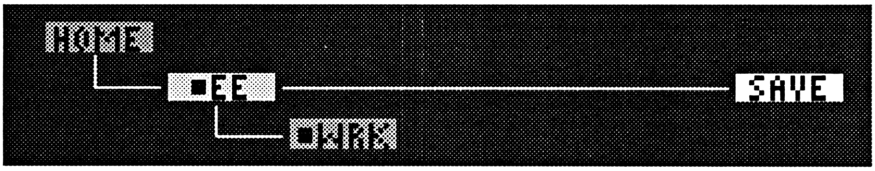
$$\begin{bmatrix} 1 & R \\ 0 & 1 \end{bmatrix}$$

Inputs: Level 1 – a real number – the resistance value.

Outputs: Level 1 – a two-by-two array – the series resistance transfer matrix.

Errors: Bad Argument Type will occur if the input resistance is not reducible to a real or complex number.

Notes: RSLADR is necessary only for Ladder Network Transfer analysis. If you need it, put it into the ▫EE directory.



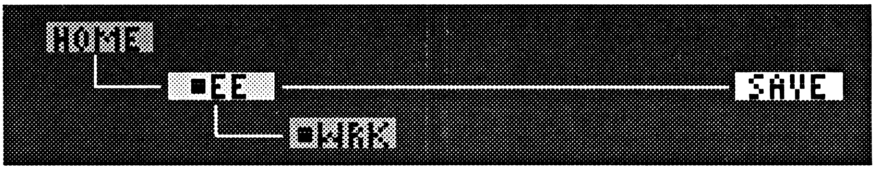
Save The Current Circuit Under A Specified Name:

SAVE (2906031)

```
« IF DUP TYPE 6 ==
THEN ▫CIRC VARS OVER
IF POS THEN DUP →STR
" Exists■OVERWRITE"
+ YES? NOT « DROP
MAIN » IFT END "«"
▫WRK CIRC →STR +
" ▫WRK 'CIRC' STO "
+ CIDX →STR +
" 'CIDX' STO MAIN»"
+ STR→ SWAP ▫CIRC
STO MAIN ELSE
"SAVE Error■Not A Name"
1 DISP ERRBP END »
```

(The ■'s here are NEWLINE characters.)

Summary: Given a name in Level 1 of the stack, SAVE stores the current circuit into that name in the ▫CIRC directory. SAVE first tests the Level-1 object to see if it's really a name. If not, a message is displayed and the program terminates. If it *is* a name – but one that already exists in



the ■CIRC directory – another prompt appears, inquiring as to whether the current contents of that name should be overwritten. If so, then the current circuit's data is overwritten into the name. If not, control is passed back to the MAIN menu, and no storage takes place.

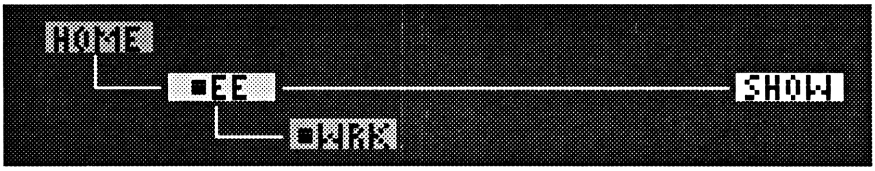
Inputs: Level 1 – the name into which the current circuit is to be stored.
Keyboard – A "yes" (either [Y] or [ENTER]) or "no" (any other) keystroke to confirm overwriting of the current circuit with the one specified.

Outputs: None.

Errors: Too Few Arguments will occur if no object is found on Level 1 of the stack. SAVE Error Not A Name will occur if the Level 1 object is not a name. Garbage will be left in the stack if [ATTN] is pressed in response to the (Y or N) prompt.

Notes: SAVE is an *optional* routine. You do *not* need it in order to perform any of the three kinds of circuit analysis *unless* you wish to name and save different circuits in the HP-28S (which is certainly very handy). If you do use SAVE, put it into the ■EE directory.

SAVE uses YES?, MAIN, and ERRBP.



Display The Current Circuit Element:

Show (31460)

```
« Type "SHOW" + STR→
»
```

Summary: Show displays the current circuit element by invoking the appropriate SHOW program (either NODESHOW, MESHSHOW or LADRSHOW).

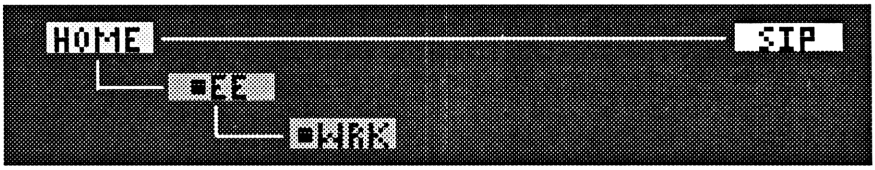
Inputs: None – data is taken from the CIRC list and from CIDX.

Outputs: A display giving the current element's information in an appropriate format (this example is for a NODE circuit):

```
#<n>: <from> → <type> → <to>
▶ <value>
```

Errors: If Type returns a string that produces a nonexistent name when combined with "SHOW", execution will halt, and that name will be left on the stack. Undefined Name will occur if CIRC is undefined. Bad Argument Type will occur if CIRC contains a non-list or a damaged list.

Notes: Show uses Type. Put Show into the ▪EE directory.



Convert The Integer Portion Of A Real Number To A String:

SIP (73219)

```
« IP →NUM RCLF SWAP
STD →STR SWAP STOF »
```

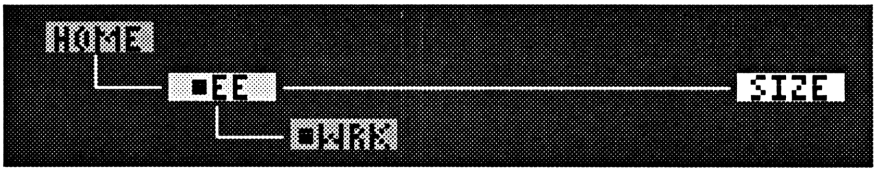
Summary: SIP (String Integer Part) is useful in generating displays of integers. It evaluates the object at stack Level 1 into a real number, whose integer portion is then converted to a string.

Inputs: Level 1 – any object that will return a real number when evaluated by →NUM.

Outputs: Level 1 – a character string (of numerals).

Errors: Bad Argument Type will occur if the input object does not evaluate to a real number. Undefined Name will occur if the input object is an undefined name.

Notes: Since SIP is generally useful, put it in the HOME directory (for use with this book, the ▪EE directory is also OK).



Get The Current Circuit Size:

Size (19582)

```
« CIRC SIZE 1 - »
```

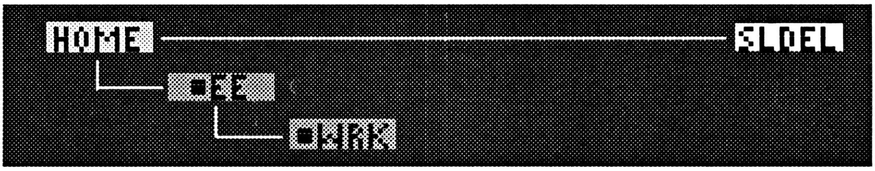
Summary: Size returns the number of elements in the current circuit by getting the SIZE of the circuit list (CIRC) and subtracting 1 (for the header element – a list containing the type string and the frequency). Size will return 0 if CIRC is undefined.

Input: None.

Output: Level 1 – a real number – the number of circuit elements.

Errors: Bad Argument Type will occur if CIRC contains a real number, complex number or binary integer.

Notes: Put Size into the ■EE directory.



Delete An Element Of A String Or List:

SLDEL (138435)

```
« → N M « DUP 1 N 1
- SUB SWAP M 1 +
OVER SIZE SUB + » »
```

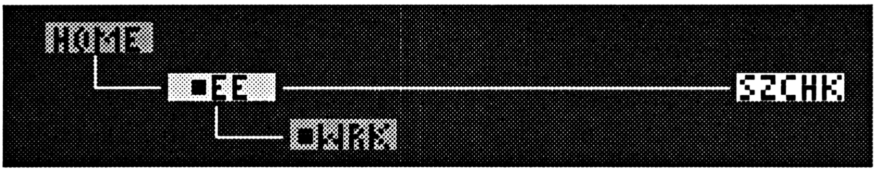
Summary: SLDEL deletes consecutive elements from a string or list, from a "starting position" (less than 1 defaults to 1) to an "ending position" (greater than the object's SIZE defaults to that SIZE). If the starting position exceeds the ending position, no deletions occur.

Inputs: Level 3 – the string or list;
Level 2 – a real number (the starting index);
Level 1 – a real number (the ending index).

Outputs: Level 1 – the "edited" string or list.

Errors: Bad Argument Type will occur when wrong input types are given.

Notes: Since SLDEL is generally quite useful, put it in the HOME directory (for this book, the ▪EE directory is also OK).



Determine If The Current Circuit Is Empty:

SZCHK (127384)

```
« Size NOT « CLLCD
ERRBP SZDSP PAUSE
ABORT » IFT »
```

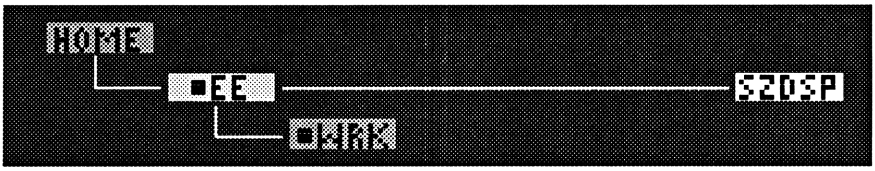
Summary: SZCHK performs a test to see if the circuit list is empty. If so, this fact is displayed with an error beep, and the program aborts.

Inputs: None.

Outputs: None.

Errors: Bad Argument Type will occur if CIRC contains a real number, complex number or binary integer.

Notes: SZCHK uses Size, SZDSP, ERRBP, and PAUSE. Put SZCHK into the ▪EE directory.



Display The Current Circuit Size:

SZDSP (81475)

```
« Size SIP
" Element Circuit" +
2 DISP »
```

Summary: SZDSP displays the number of elements in the current circuit as follows:

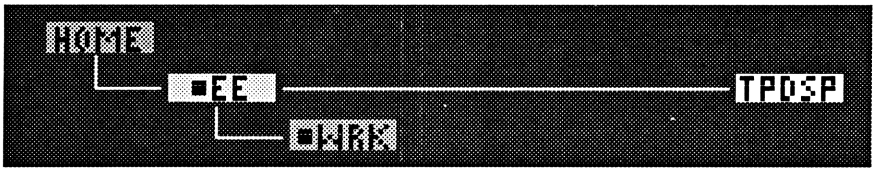
<n>Element Circuit

Inputs: None – data is taken from CIRC.

Outputs: The above display format.

Errors: Bad Argument Type will occur if CIRC contains a real number, complex number or binary integer.

Notes: SZDSP uses Size and SIP. Put SZDSP into the ■EE directory.



Display Current Circuit Type:

TPDSP (46404)

```
« Type " Analysis" +
1 DISP »
```

Summary: TPDSP displays the current circuit's type on line 1 of the display, in the following format:

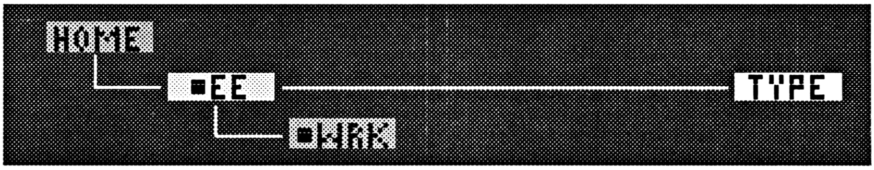
<type> Analysis

Inputs: None – data is taken from CIRC.

Outputs: The display shown above.

Errors: Undefined Name will occur if CIRC is undefined. Bad Argument Type will occur if CIRC contains a non-list or a damaged list.

Notes: TPDSP uses Type. Put TPDSP into the ▪EE directory.



Get The Current Circuit Type:

Type (25106)

```
« CIRC 1 GET 1 GET »
```

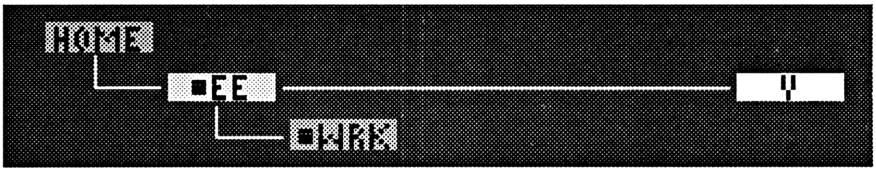
Summary: Type gets the type string from the current circuit. It assumes that the circuit list (CIRC) has been properly initialized.

Inputs: None – data is taken from CIRC.

Outputs: Level 1 – a string denoting the circuit type: "NODE", "MESH", or "LADR".

Errors: Undefined Name will occur if CIRC is undefined. Bad Argument Type will occur if CIRC contains a non-list or a damaged list.

Notes: Put Type into the ■EE directory.



Add A Voltage Element To The Circuit List:

V (7874)

```
« "V" PUTE »
```

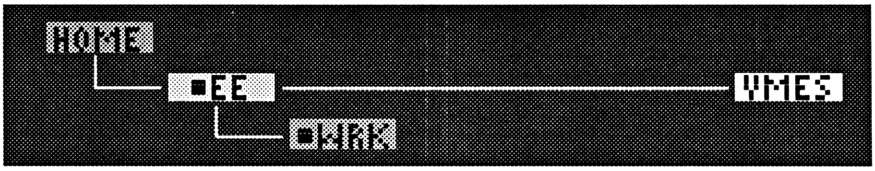
Summary: V invokes PUTE (PUT Element) in order to add a voltage-type element ("V") to the current circuit ('CIRC').

Inputs: Level 3 – an integer – the forward mesh.
Level 2 – an integer – the reverse mesh.
Level 1 – a real or complex number – the voltage value (V).

Outputs: None.

Errors: If the circuit is empty when V is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: V is necessary *only* for Mesh-Current circuit analysis. If you need it, put V into the ▪EE directory. V uses PUTE.



Store A Voltage Into The Voltage Array:

VMESH (14466)

```
« NEG INODE »
```

Summary: VMESH stores a voltage into the voltage array.

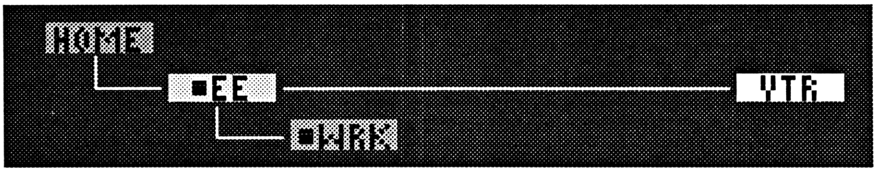
Inputs: Level 3 – an integer – the forward loop.
Level 2 – an integer – the reverse loop.
Level 1 – a real or complex number – the voltage value (V).

Outputs: None.

Errors: Infinite Result will occur if the voltage value is zero and flag 59 (infinite result action) is set.

Notes: VMESH is required *only* for Mesh-Current analysis. If you need it, put it into the ▪EE directory.

VMESH uses INODE.



Calculate The Ladder Network Voltage Transfer Ratio:

VTR (15004)

```
« FTA NEG ZL * »
```

Summary: VTR calculates the value of the Voltage Transfer Ratio for a "LADR"-type circuit.

Inputs: None – data is taken from ZL and ZA.

Outputs: Level 1 – a real or complex number – the Voltage Transfer Ratio.

Errors: Infinite Result will occur if 'ZA(1,1)*ZL+ZA(1,2)' is (0,0) and flag 59 (infinite result action) is set.

Notes: VTR is necessary only for Ladder Network Transfer analysis. If you need it, put VTR into the ▫EE directory.

VTR uses FTA.

Ask A Yes-Or-No Question:

YES? (165216)

```
« →STR CLLCD
"?■(Y or N)" + 1
DISP { "Y" "ENTER" }
GETK POS CLMF »
```

Note: The ■ you see here should be keyed in as a NEWLINE character (press ■NEWLINE).

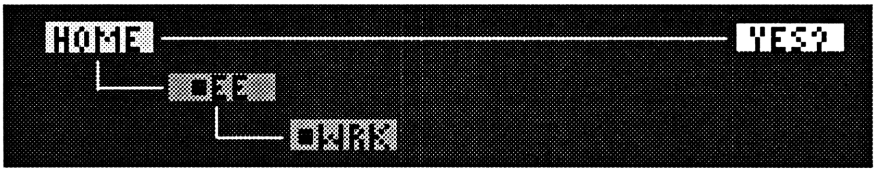
Summary: YES? builds a yes-or-no question from the object in Level 1. This object is converted to a string and displayed as follows:

```
<input object>?
(Y or N)
```

The program will then await (indefinitely) a one-key response, for which it will accept any keystroke except ATTN. Any keystroke besides Y or ENTER will be taken as a NO response and will return a 0 (FALSE). But a 1 or 2 will be returned for Y or ENTER, responses, respectively, which are both taken as YES (TRUE).

This allows the following kind of construction:

```
IF "Are You Done" YES?
THEN Finish
ELSE DoMore
END
```

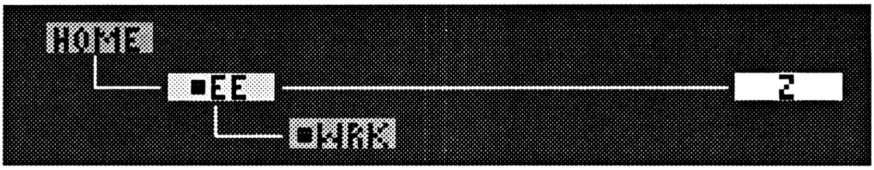


Inputs: Level 1 – the object with which to prompt; Keyboard – a single keystroke other than ATTN.

Outputs: Level 1 – a real number (0, 1 or 2 , depending on the responding input keystroke).

Errors: Too Few Arguments will occur if no prompt object is on the stack. Garbage will be left in the stack if ATTN is pressed.

Notes: YES? uses GETK. Since YES? is generally useful, put it in the HOME directory (for this book, the ■EE directory is also OK).



Add An Impedance Element To The Circuit List:

Z (7946)

```
« "Z" PUTE »
```

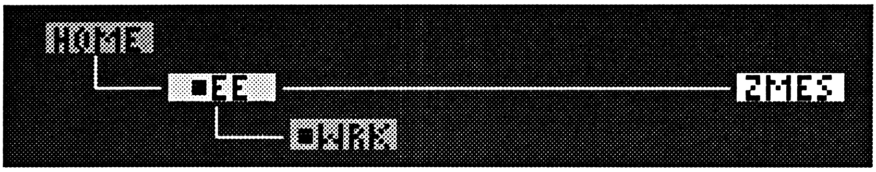
Summary: Z invokes PUTE (PUT Element) in order to add an impedance-type element ("Z") to the current circuit ('CIRC').

Inputs: Level 3 – an integer – the"from" node or forward mesh.
Level 2 – an integer – the"to" node or reverse mesh.
Level 1 – a real or complex number – the impedance value (Z).

Outputs: None.

Errors: If the circuit is empty when Z is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: Z is *necessary for both* Node-Voltage *and* Mesh-Current circuit analyses. If you need it, put Z into the ▪EE directory. Z uses PUTE.



Store An Impedence Into The Impedance Array:

ZMESH (14741)

```
« INV ZNODE »
```

Summary: ZNODE stores an impedance value into the impedance array, 'ZA'.

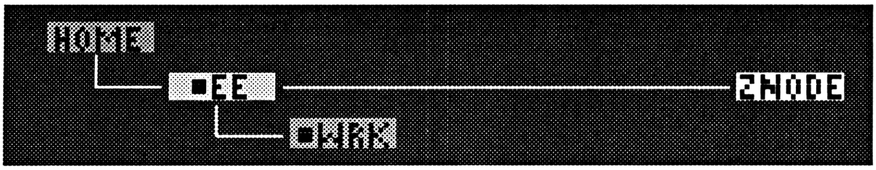
Inputs: Level 3 – an integer – the forward loop.
Level 2 – an integer – the reverse loop.
Level 1 – a real or complex number – the impedance value (Z).

Outputs: None.

Errors: Infinite Result will occur if the impedance value is zero and flag 59 (infinite result action) is set.

Notes: ZMESH is necessary *only* for Mesh-Current circuit analysis. If you need it, put it into the ▪EE directory.

MESH uses ZNODE.



Store An Impedence Into The Admittance Array:

ZNODE (438823)

```
« INV → V « IF DUP2
* NOT THEN + DUP 3
ELSE DUP2 SWAP DUP2
DUP ROT DUP -3 END 3
'ZA' V PUTX » »
```

Summary: ZNODE stores an impedance value as an admittance calculated as Y = 1/Z) into the admittance array, 'ZA'.

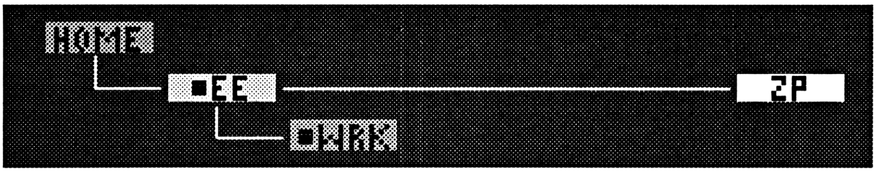
Inputs: Level 3 – an integer – the "from" node.
Level 2 – an integer – the "to" node.
Level 1 – a real or complex number – the impedance value (Z).

Outputs: None.

Errors: Infinite Result will occur if the impedance value is zero and flag 59 (infinite result action) is set.

Notes: ZNODE is *necessary for both* Node-Voltage *and* Mesh-Current circuit analyses. If you need it, put it into the ■EE directory.

ZNODE uses PUTX.



Add A Parallel-Impedance Element To The Circuit List:

ZP (10374)

```
« "ZP" PUTE »
```

Summary: ZP invokes PUTE (PUT Element) to add a parallel-impedance element ("ZP") to the current circuit ('CIRC').

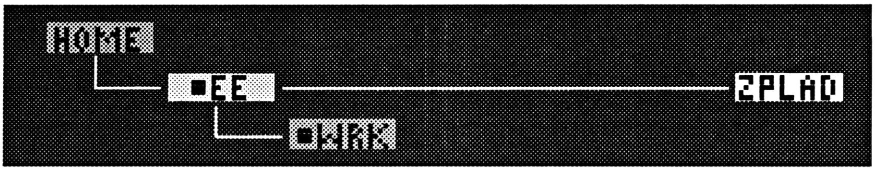
Inputs: Level 1 – a real or complex number – the parallel impedance value (ZP).

Outputs: None.

Errors: If the circuit is empty when ZP is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: ZP is necessary *only* for Ladder Network Transfer analysis. If you need it, put ZP into the ■EE directory.

ZP uses PUTE.



Create A Parallel Impedance Transfer Matrix:

ZPLADR (12010)

```
« RPLADR »
```

Summary: ZPLADR creates a parallel impedance transfer matrix of this form:

$$\begin{bmatrix} 1 & 0 \\ \frac{1}{Z} & 1 \end{bmatrix}$$

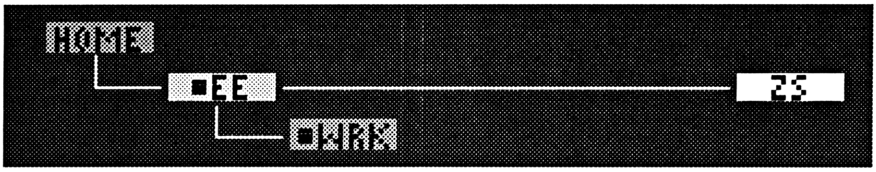
Inputs: Level 1 – a real or complex number – the impedance value.

Outputs: Level 1 – a two-by-two array – the parallel impedance transfer matrix.

Errors: Bad Argument Type will occur if the input impedance is not a real or complex number.

Notes: ZPLADR is necessary only for Ladder Network Transfer analysis. If you need it, put it into the ■EE directory.

ZPLADR uses RPLADR.



Add A Series-Impedance Element To The Circuit List:

ZS (10437)

```
« "ZS" PUTE »
```

Summary: ZS invokes PUTE (PUT Element) to add a series-impedance element ("ZS") to the current circuit ('CIRC').

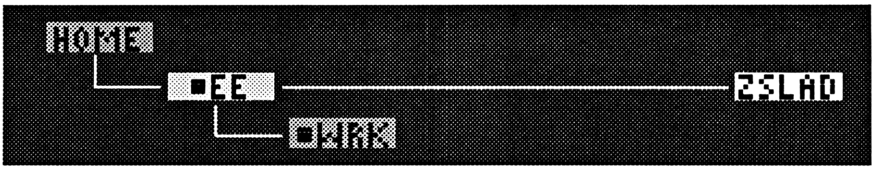
Inputs: Level 1 – a real or complex number – the series impedance value (ZS).

Outputs: None.

Errors: If the circuit is empty when ZS is called from CHNG, SZCHK will report "0 Element Circuit," and control will be passed to MAIN. Bad Argument Type will occur if CIDX is undefined when PUTE is called from CHNG. Undefined Name will occur if CIRC is not defined.

Notes: ZS is necessary *only* for Ladder Network Transfer analysis. If you need it, put ZS into the ■EE directory.

ZS uses PUTE.



Create A Series Impedance Transfer Matrix:

ZSLADR (12061)

```
« RSLADR »
```

Summary: ZSLADR creates a series impedance transfer matrix of this form:

$$\begin{bmatrix} 1 & Z \\ 0 & 1 \end{bmatrix}$$

Inputs: Level 1 – a real or complex number – the impedance value.

Outputs: Level 1 – a two-by-two array – the series impedance transfer matrix.

Errors: Bad Argument Type will occur if the input impedance is not a real or complex number.

Notes: ZSLADR is necessary only for Ladder Network Transfer analysis. If you need it, put it into the ■EE directory.

ZSLADR uses RSLADR.

Notes (Yours)

CHAPTER 6

CIRCUIT ANALYSIS TOOLS: Examples And Discussion

This chapter presents examples and some more in-depth discussions of the ideas and techniques available to you when you use the Circuit Analysis Tools. When using the tools discussed in this chapter, you must have keyed in the necessary objects from the previous chapter already (for the three kinds of circuit analysis, there are three corresponding checklists on pages 91-93).

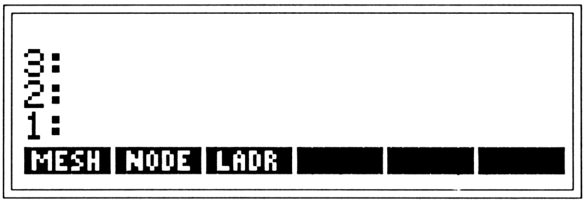
It is not necessary to key in *all* the objects in the previous chapter – just the ones required for the kind of circuit analysis you want to do. **However,** whether or not you're using all three kinds of analysis, **it behooves you to read straight through this chapter** and look at all the examples, because each will teach you something *in general* about the ways in which all of the analyses work.

The Circuit Editor

Before you get into the actual examples of any of the three kinds of circuit analysis, you need to know how to build an accurate description of a circuit for your HP-28S. The circuit editor provides you with one consistent, menu-driven interface for building any of the three different types of circuits.

The Initialization Menu

To begin building a circuit, you type [E][E][A] (for Electrical Engineering Analysis) and press [ENTER] to see the following Initialization Menu:



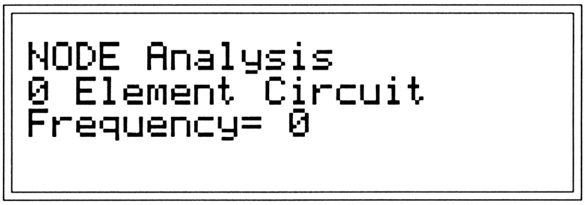
MESH, **NODE** and **LADR** are the choices you have for building three different types of circuits (for Mesh-Current Analysis, Node-Voltage Analysis, and Ladder Network Transfer Analysis, respectively).

The three different types of circuits are incompatible. To perform different kinds of analyses on a given circuit, you must enter the circuit once for each kind of analysis.

Right at this point, then, is where you decide what kind of analysis you want to do on the circuit you're going to build. Then by pressing the appropriate key on the above menu, you'll initialize a new circuit to that type of analysis.

The Circuit Information Display

Press **NODE** to see a Circuit Information display appear briefly:



```
NODE Analysis
0 Element Circuit
Frequency= 0
```

If there had already been a circuit (of any type) currently defined (i.e. if any objects named `'CIRC'` and `'CIDX'` had already existed in the `'▫WRK'` directory), then the first thing you would have seen after invoking `EEA` would have been a Circuit Information display similar to the one above – and it would have been reporting the type, size and frequency of that already-existing circuit. But in this first instance, because there was *no* such pre-existing circuit, you were presented with the Initialization Menu first (on the previous page), to let you choose the type.

So, what does this first Circuit Information display tell you?

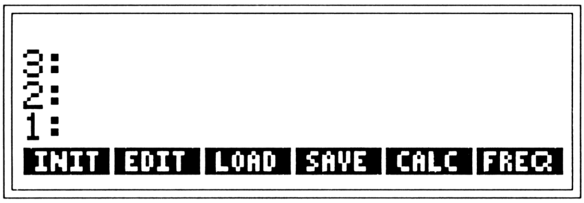
- It tells you that this circuit will be a `NODE`-analysis type circuit;

- `0 Element Circuit` tells you that the circuit has `0` components – and this number will change as you add elements to the circuit.

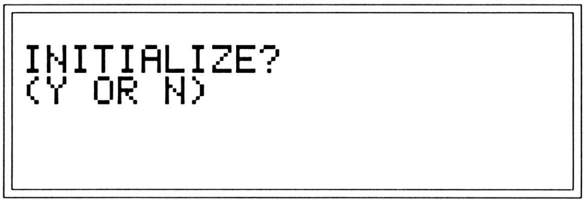
- `Frequency= 0` indicates that you haven't yet specified the sinusoidal steady-state frequency for this circuit.

The MAIN Menu

A Circuit Information display lasts only briefly, so now you'll see the MAIN menu. Look at each of its items (and keep in mind that it has a second "page" of selections, too – which you reach with the NEXT key):



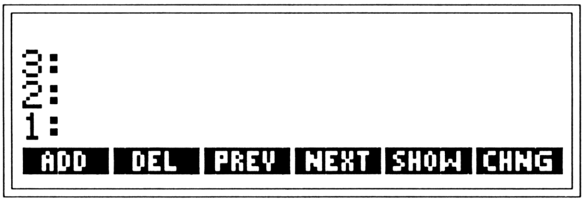
INIT allows you to re-initialize the current circuit. Here is where you can change your mind and "re-zero" (erase all the elements) or even change the type of the current circuit (yes – there is now a "current circuit" – though it has no elements). So if you press **INIT**, you'll see this display:



```
INITIALIZE?
(Y OR N)
```

The question being asked here is, "Do you really want to destroy the information in the current circuit by re-initializing it?" If you press either Y or ENTER, that will be considered a "yes" response, and you'll go back to the Initialization Menu (see page 203). *Any other keystroke* (except ON/ATTN, which ungracefully halts the program) *will be taken as a "no" response.* In that case, the Circuit Information display (see the previous page) will be displayed again (with the same information as before), and then you'll return to the MAIN Menu.

EDIT from the MAIN Menu presents you with yet another menu, the EDIT Menu, which (naturally) contains all your options for editing the current circuit:

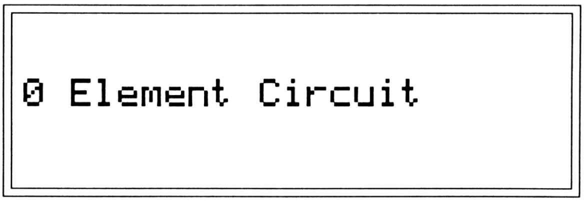


```
3:
2:
1:
```

This menu also has a second page (and again, use the (NEXT) key to see and use it). Here are the items on the EDIT menu:

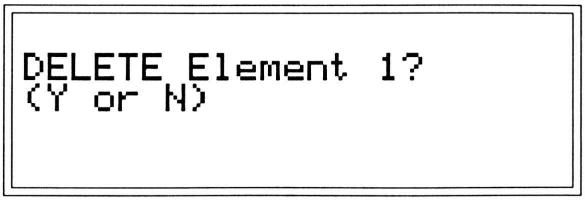
ADD produces a menu of circuit elements specific to the circuit type you are editing. Since the ADD menu is circuit-specific, its uses will be discussed and demonstrated in the different circuit-analysis examples.

DEL deletes the current circuit element from the circuit if it exists. If the circuit is empty (the only time that **DEL** might fail) you will see:



```
0 Element Circuit
```

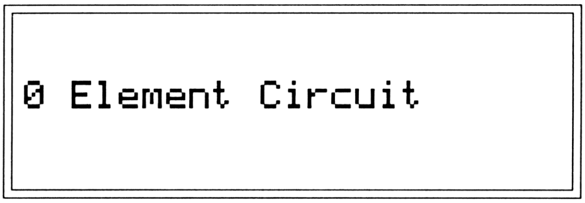
Otherwise, you will be asked:



```
DELETE Element 1?
(Y or N)
```

where the element number is the *current* element's number (i.e., it's not necessarily the **1** shown above). Pressing [Y] or [ENTER] will delete the element. Any other keystroke (except [ON]/[ATTN] will abort the delete. You will always be left in the EDIT menu.

PREV
NEXT
SHOW are three related commands. **SHOW** displays the current element in a circuit-dependent format (and will therefore be discussed in more detail in the specific circuit analysis examples). **NEXT** moves to the next element before displaying it (making it the new current element). Similarly, **PREV** moves to the previous element. Both **PREV** and **NEXT** will "wrap around" the ends of the circuit – "starting over" again if you try to move beyond either end of the circuit. All three will display the error message



```
0 Element Circuit
```

if the circuit has no elements.

CHNG allows you to change the current element. This is another circuit-dependent routine and will be discussed in the circuit-specific sections.

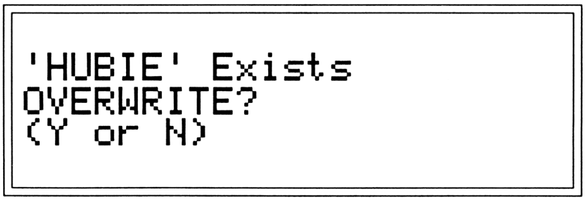
GOTO (on the next page of the menu) allows you to go to a specific circuit element. To use it, you key in the number of the element to which you want to go and press **GOTO**. If the input number exceeds the size of the circuit, the circuit size will be used. If the input number is less than 1, 1 will be used instead. To display the new current element, press **SHOW**.

PRINT is an optional routine (it's only useful if you have a printer) that prints the circuit list, from beginning to end, in the format of SHOW.

EXIT leaves the EDIT menu, redisplays the Circuit Information display and shows you the MAIN menu.

LOAD

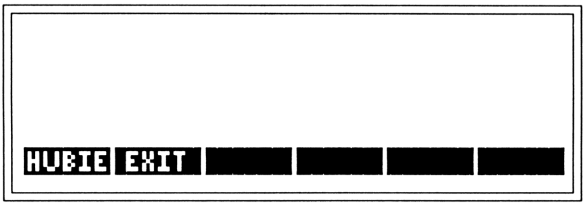
SAVE Back now in the MAIN menu are these two *optional* routines, to SAVE the current circuit (under a name of your choosing) in an isolated directory (▫CIRC), where it cannot easily be destroyed; then to recall (LOAD) that circuit for later calculations. Thus the commands 'HUBIE' **SAVE** will store the current circuit in the ▫CIRC directory in the name 'HUBIE'. If that name already exists in the ▫CIRC directory, you'll see:



```
'HUBIE' Exists
OVERWRITE?
(Y or N)
```

As always, pressing [Y] or [ENTER] says "Yes," thus overwriting the contents of 'HUBIE'; any other keystroke except [ON]/[ATTN] says "No", which will abort the SAVE. Either way, you'll return to the MAIN menu via the Circuit Information display.

LOAD loads a saved circuit. If the current circuit isn't empty, you'll be asked if it's OK to lose whatever you have in the current circuit by overwriting it with a saved circuit. A "yes" response produces a menu of saved circuits:



Selecting any of the named circuits (in this example, HUBIE) will LOAD that circuit and return you to the MAIN menu. Note that there is no "PURGE-like" routine to easily eliminate unnecessary circuits from the °CIRC menu. To delete 'HUBIE', press LOAD, to move to the °CIRC menu, then ' HUBIE and PURGE to delete. The CUSTOM menu won't change, but if you EXIT back to the MAIN menu and press LOAD again, you'll verify that the deleted circuit is indeed gone.

EXIT is an extra item on the LOAD menu which allows you to abandon the LOAD menu and go back into the MAIN menu *without altering the current circuit.*

CALC is the button you press to do the actual "circuit-crunching." The results are different for each circuit type, so you'll hear all about CALC later, as you do specific examples.

FREQ next in the MAIN menu, lets you store a sinusoidal, steady-state frequency for the circuit. To use it, key in the frequency (in Hertz) and press FREQ. The Circuit Information display appears, verifying that this frequency has been stored.

QUIT This final item in the MAIN menu is on the second "page." QUIT allows you to leave the circuit editor (by moving to the HOME directory and turning on the USER menu). None of your circuit information will be altered – it will be there the next time you invoke EEA.

Examples Of Circuit Analysis

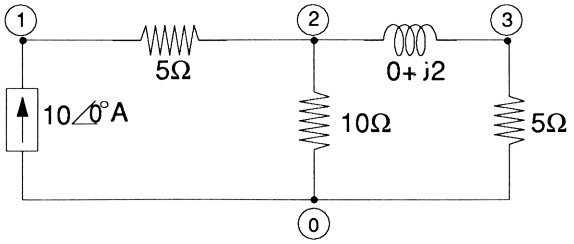
Now you're going to see the Circuit Editor "in action." *Read through all of these examples* – even if they don't involve the kind of circuit analysis you want to do. Every example is meant to demonstrate something.

Node-Voltage Network Analysis

Node-voltage analysis allows you to enter a circuit composed of resistances (R), capacitances (C), inductances (L), general impedances (Z) and independent (sinusoidal, AC) current sources (I) – in any number and combination. You may *not* use voltage sources.

The result of the analysis is an *array of voltages,* as measured between the *reference node* (node 0) and each other *node* (node 1, 2 , etc.) For the purposes of this book, *a node is defined to be the junction between any two or more circuit elements.* Any node can be designated the reference node (0), and the numbering of the other nodes can be rearranged, but there *must* be a reference node (this is the node that is defined to be at a "zero" or "ground" voltage level – the standard against which all other voltages are measured). Also no node number may be skipped. That is, in a four-node network, you must number the nodes 0, 1, 2, 3. You may *not* number the nodes 0, 2, 3, 4 – nor 1, 3, 4, 5 – the program won't work.

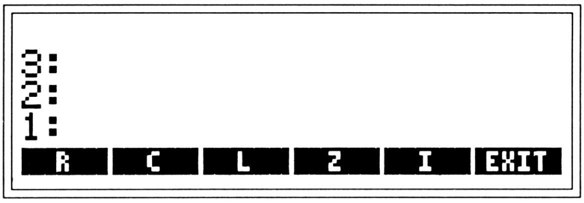
Look at this first example:



To calculate the node voltages in this circuit, do the following:

Type EEA [ENTER]. If there is no current circuit, select NODE from the Initialization menu. If there is a current circuit, save it if you like, and select INIT from the MAIN menu, respond with [Y] to the initialization query, and select NODE from the initialization menu.

You should be back to the MAIN menu now, so press EDIT. Now, you want to add elements to the circuit, so press ADD:



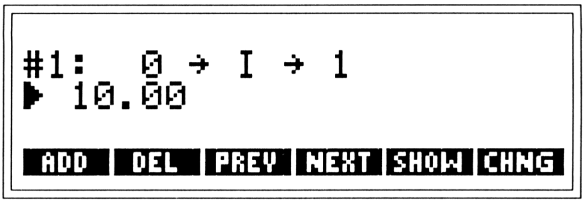
This ADD menu presents you with your options for adding elements to the circuit you're building. To add an element, you put information on the stack in the following order:

<"from" node>
<"to" node>
<element value>

Then you select the element type from the menu. For example, here's what you do to add the current source between nodes 0 and 1:

0 1 10 I (do it now).

The element menu goes away and you're back in the MAIN menu. Select SHOW now to see what you've done (and set a 2 FIX display mode so that your displays will look like these examples): 2 FIX SHOW. Here's the result:



```
#1:  0 → I → 1
▶ 10.00
 ADD   DEL   PREV  NEXT  SHOW  CHNG
```

This says, "Element number 1 is a current source *from* 0 *to* node 1 whose value is 10 (that's 10 + j0)." This exactly corresponds to what you input. So now you input the other elements in the same way:

ADD 1 2 5 R
("The next element is a resistor from node 1 to node 2 whose value is 5.")
0 3 ADD 5 R
("The next element is a resistor from node 0 to node 3 whose value is 5.")
3 2 (0,2) ADD Z
("The next element is an impedance from node 3 to node 2 whose value is 0+j2)."
ADD 0 2 10 R
("The next element is a resistor from node 0 to node 2 whose value is 10.")

Some things to notice about what you just did:

1. Notice the use of j here, instead of **i**. It's difficult to know which to use here: Electrical engineering convention usually prefers j, but the HP-28S uses the symbolic constant, i. However, since the machine's requirements need not concern you, these diagrams and discussion will use the engineers' j.

2. As you know from keying in the objects necessary for MESH- and/or NODE- type analyses, there are objects named C, I, L, R, V, and Z, all stored in the °EE directory. Be careful not to use these or any other °EE names for your own circuit variables when doing circuit reductions or other "side calculations." Suggestion: put subscripts on your circuit elements: R1, ZA, etc.

3. You don't need to press ADD before keying in the element information. Since ADD merely brings up the menu of elements, it can be pressed at any time before the element name is needed. This is generally true of the various menus in these tools: Changing menus does not alter the contents of the stack in any way.

4. Anytime you temporarily use an HP-28S system menu (e.g. COMPLX, you can always get back to your Circuit Tool's current custom menu (e. g. EEA, EET, ADD, LOAD) by pressing CUSTOM.

5. The inductance between nodes 2 and 3 is given in the problem as an impedance and not as an inductance (you can tell because inductance in henrys is a real quantity, but the impedance of an inductance is imaginary). Therefore, it must be input as a general impedance rather than as an inductance.

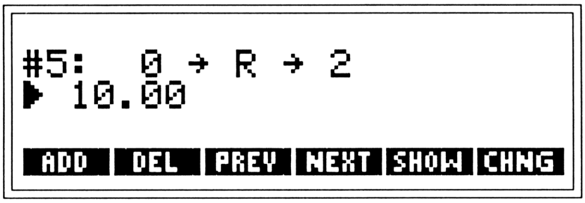
6. None of the impedances of the circuit elements depends on the circuit frequency. Only inductors and capacitors are frequency-dependent, so a frequency need not be input to solve this problem.

7. The complex number used for the inductance must be in the HP-28S's rectangular format. If you prefer to "think" and "key in" in polar form or any of the symbolic forms made available by the Circuit Reduction Tools, you must then use those tools to convert to the HP-28S's recatngular format before selecting the element type.

8. The current source is assumed to cause current flow *from* the "from" node (0 in this case) *to* the "to" node (1 in this case). Either changing the sign on the current source's value (-10.00) or reversing the "from" and "to" nodes would denote current flowing in the other direction.

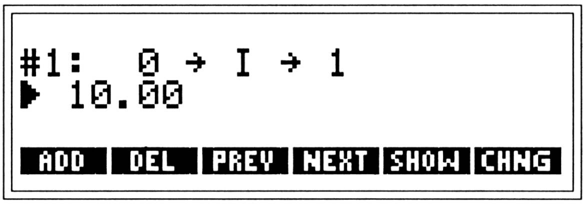
9. You could have keyed in these elements in any order; the numbering of the elements here is for identification purposes only (true only for Node-Voltage and Mesh-Current analyses).

Now that you've finished loading the circuit, you should scan through it to see if you've made any mistakes: Press **SHOW**.


```
#5:   0 → R → 2
▶ 10.00
```

As you can see, this is element #5, the fifth and last element you entered.

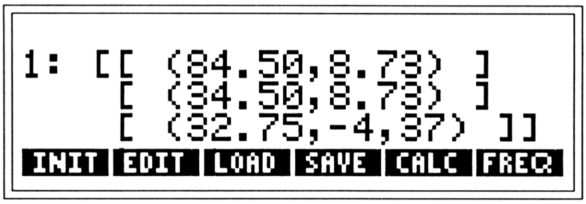
Press **NEXT**.


```
#1:   0 → I → 1
▶ 10.00
```

This is element #1 again – the first element you entered. Thus **NEXT** "wrapped around" from the last element in the circuit back to the first.

Take a little time right now and practice using **PREV**, **NEXT**, **SHOW** and **GOTO** to move around the circuit, just to get a feel for them – and to check the accuracy of this circuit description you've just built for your HP-28S.

Now, to calculate the node voltages, you select **EXIT** to go out to the MAIN menu, then simply **CALC**.



```
1: [[ (84.50,8.73) ]
    [ (34.50,8.73) ]
    [ (32.75,-4,37) ]]
```

(This display assumes multi-line display mode is set.)

What you have here is an *array of node voltages.* Of course, it's easy enough to read off the answers right now: "V_1 (that is, the voltage *difference* between node 1 and node 0, V_1-V_0, where V_0 is defined to be zero) is 84.50+j8.73. And V_2 is 34.50+j8.73, etc.

But what if you'd prefer to see these in polar format? No problem – but to make this resulting array more convenient to use, give it a name: `'A' STO`.

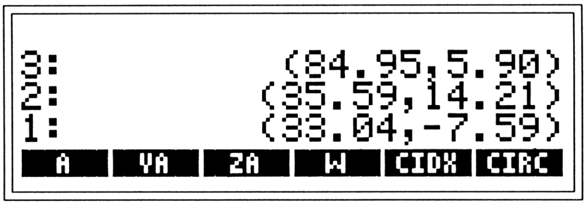
Now, to find the polar-formatted voltages V_1, V_2 and V_3:

`A 1 GET R→P` Result: `(84.95,5.90)`
So V_1 = 84.95∠5.90° volts.

`A 2 GET R→P` Result: `(35.59,14.21)`
So V_2 = 35.59∠14.21° volts.

`A 3 GET R→P` Result: `(33.04,-7.59)`
So V_3 = 33.04∠-7.59° volts.

At this point, press USER to see what's in the current directory (the °WRK directory):


```
3:        (84.95,5.90)
2:       (35.59,14.21)
1:       (33.04,-7.59)
```

A contains the node voltage solution array that you just stored;

VA contains the current array – the array of independent current sources (its name is shared by the node and mesh analysis programs – hence its rather inappropriate name here);

ZA contains the array of impedances, built from the circuit description;

W is 'W', the radian frequency of the circuit;

CIDX contains the *index* to the circuit list – the number of the current element;

CIRC contains the circuit list – the circuit description itself.

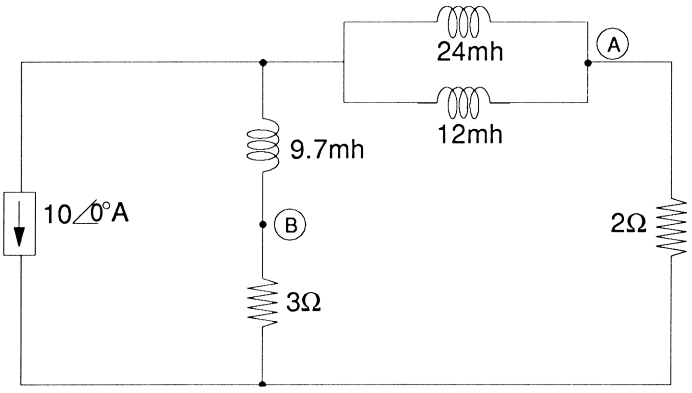
Actually, VA and ZA were used to calculate the solution, so you can also *manually* calculate that solution array (what you since named 'A'):

```
VA ZA /
```

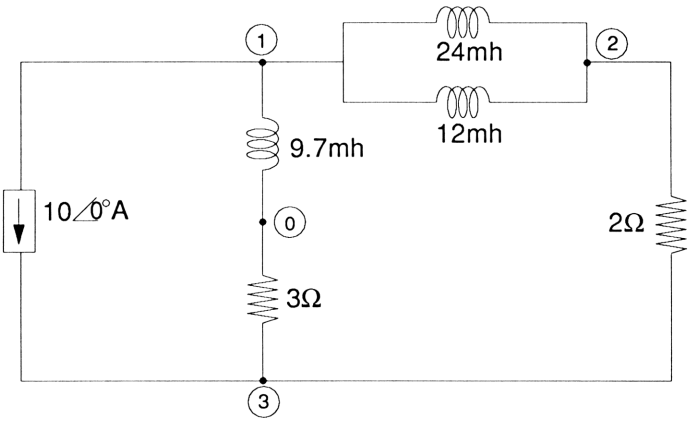
This is actually quicker than returning to the MAIN menu and using CALC (■ CUSTOM CALC), because CALC actually reloads these arrays from the circuit description before performing the array division.

Try another example....

For the following circuit, find the voltage between nodes A and B (i.e. find V_A-V_B) at a frequency of 66 Hz.:



First you must properly number the nodes, making either A or B the reference. Thus:



Next, build the description:

```
EEA INIT (Y) NODE EDIT

0 1 ADD 9.7E-3 L
1 2 ADD 24E-3 L
1 2 ADD 12E-3 L
2 3 ADD 2 R
0 3 ADD 3 R
1 3 ADD 10 I
```

Now go back to the MAIN menu, specify the frequency and grind away!

```
EXIT 66 FREQ CALC ....
```

The result is another solution array – a 3x1 array of rectangular-complex node voltages (in this order): V_1 ($=V_1-V_0$), V_2 ($=V_2-V_0$),V_3 ($=V_3-V_0$).

And to see these in more familiar polar format: `'A' STO 2 FIX`

`A 1 GET R→P` <u>Result</u>: `(17.55,-86.82)`
So $V_1 = 17.55\angle -86.82°$

`A 2 GET R→P` <u>Result</u>: `(2.15,34.26)`
So $V_2 = 2.15\angle 34.26°$

`A 3 GET R→P` <u>Result</u>: `(13.09,3.18)`
So $V_3 = 13.09\angle 3.18°$

Now, as you will recall, you defined V_{AB} to be V_{20} (i.e. just V_2), so $V_{AB} = V_2 = 2.15\angle 34.26°$.

Try rearranging the node numbers to see what you get – e.g. change node 1 to 0, node 3 to 1, and node 0 to 3:

```
INIT (Y) NODE EDIT
0 1 ADD 10 I          (this could also be 1 0 ADD -10 I)
0 2 ADD 12E-3 L
0 2 ADD 24E-3 L
0 3 ADD 9.7E-3 L
1 2 ADD 2 R
1 3 ADD 3 R
EXIT 66 FREQ CALC ....
```

Then to see these in more familiar polar format: `'A' STO` and

`A 1 GET R→P` Result: `(21.89,56.46)`
So $V_1 = 21.89\angle 56.46°$

`A 2 GET R→P` Result: `(18.74,87.55)`
So $V_2 = 18.74\angle 87.55°$

`A 3 GET R→P` Result: `(17.55,93.18)`
So $V_3 = 17.55\angle 93.18°$

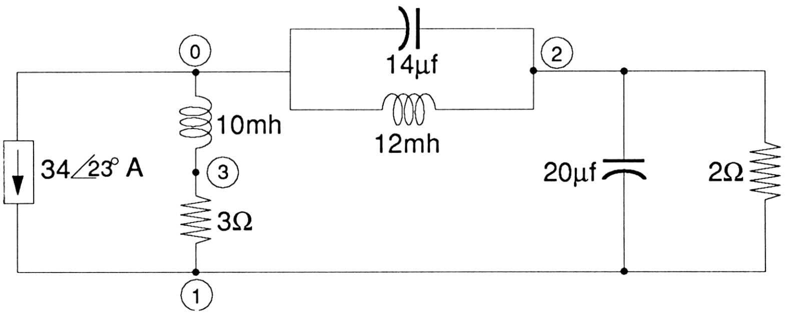
Notice that V_3 in this second configuration is of equal magnitude but of opposite direction (i.e. 180° off) from V_1 in the first configuration – as you would expect.

To calculate V_{AB} using this arrangement, since $V_{AB} = V_2 - V_3$, do this:

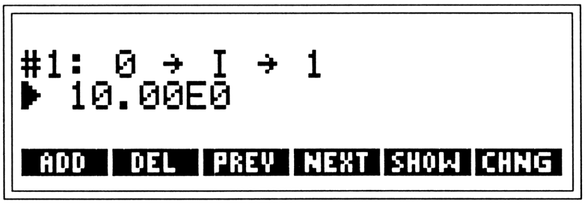
`'A(2,1)-A(3,1)' EVAL R→P` Result: `(2.15,34.26)`

So $V_{AB} = V_2 - V_3 = 2.15\angle 34.26°$ – same as before (as it should be)!

As a final example and a demonstration of how to use **CHNG**, rearrange the circuit so that it looks like this:

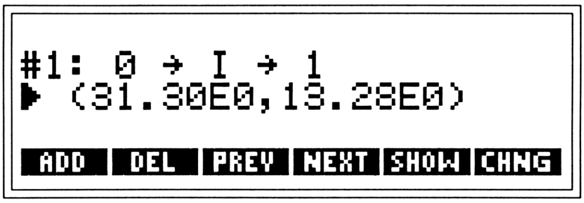


Like so: **EDIT** 1 **GOTO** 3 ENG **SHOW**



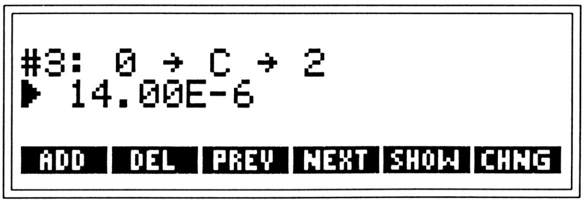
```
#1:  0 → I → 1
▶ 10.00E0
ADD | DEL | PREV | NEXT | SHOW | CHNG
```

Then: **CHNG** DROP (34,23) P→R **I** **SHOW**



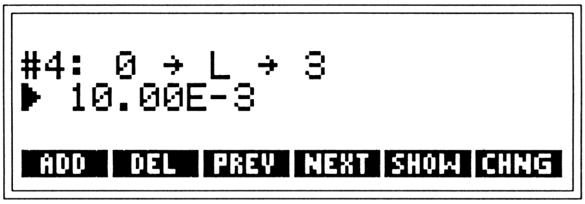
```
#1:  0 → I → 1
▶ (31.30E0,13.28E0)
ADD | DEL | PREV | NEXT | SHOW | CHNG
```

Then: **NEXT** **NEXT** **CHNG** DROP 14E-6 **C** **SHOW**



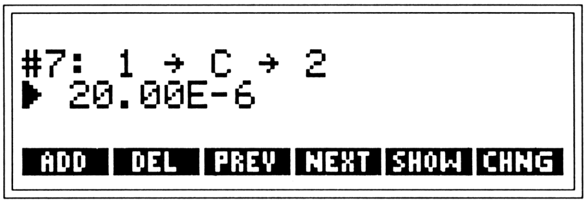
```
#3:  0 → C → 2
▶ 14.00E-6
ADD | DEL | PREV | NEXT | SHOW | CHNG
```

NEXT CHNG DROP 10E-3 L SHOW



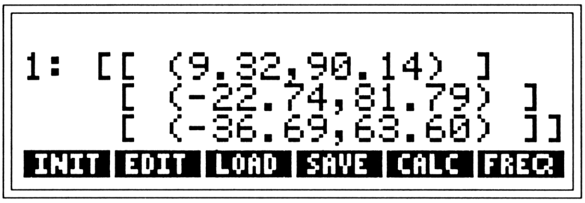
```
#4: 0 → L → 3
▶ 10.00E-3
ADD DEL PREV NEXT SHOW CHNG
```

ADD 1 2 20E-6 C SHOW



```
#7: 1 → C → 2
▶ 20.00E-6
ADD DEL PREV NEXT SHOW CHNG
```

EXIT CALC 2 FIX



```
1: [[ (9.32,90.14) ]
    [ (-22.74,81.79) ]
    [ (-36.69,63.60) ]]
INIT EDIT LOAD SAVE CALC FREQ
```

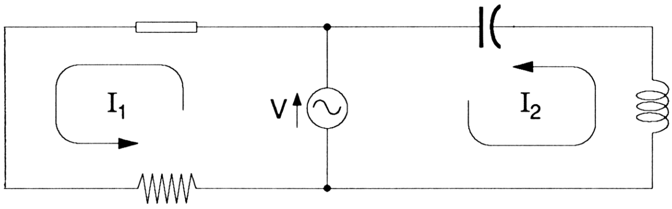
Things to note:

1. CHNG pulls the information out of the current circuit element in the order that you input it. Thus, Level 3 has the "from" node, Level 2 the "to" node, and Level 1 the element value, ready for you to change if you need/want to. CHNG also presents you with the element menu so that you can alter the element type as well. If you abort the CHNG operation by using EXIT, the element information will be left on the stack, and the original contents of the element will be retained.

2. ADD always adds a new element to the *end* of the circuit list, no matter which element is the current element. The ADD'ed element then becomes the current element.

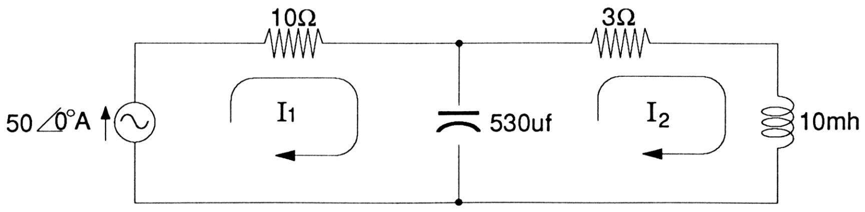
Mesh-Current Network Analysis

Mesh-current analysis allows you to enter a circuit composed of resistances (R), capacitances (C), inductances (L), general impedances (Z) and independent (sinusoidal, AC) voltage sources (V) – in any number and combination. You may *not* use current sources. The circuit is defined and configured with loop currents in the following general scheme (the circuit must consist of at least one current loop containing at least one voltage source and one passive impedance):



The result of the analysis is an array of these defined loop current values. For the purposes of the tools in this book, when you select and identify loop currents, you should be sure that *at least one and no more than two currents pass through each element – and when two currents pass through an element, they flow in opposite directions.* Each element of the mesh is considered to receive a forward current from one current loop and a reverse current from another. If there is no such reverse current, then it is considered to be coming from a non-existent loop "0." Also no loop number may be skipped. That is, in a four-loop network, you must number the loops 1, 2, 3, 4. You may *not* number them 1, 2, 3, 5, for example – the program won't work.

Look at this first example:



The two loop currents, I_1 and I_2, are defined as shown. Notice that both I_1 and I_2 flow through the capacitor, but in opposite directions.

To analyze this example circuit and solve for the two currents:

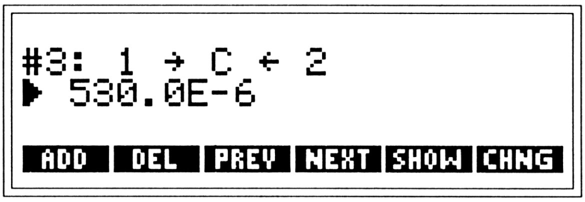
```
3 ENG EEA
```

Then, if there is no current circuit, select **MESH** from the initialization menu. If there is a current circuit, save it if you like, and select **INIT** from the MAIN menu, respond with [Y] to the initialization query, then select **MESH** from the Initialization menu.

At the MAIN menu now: **EDIT** **ADD** `1 0 50` **V**
In other words, "add a 50∠0°-volt source to current loop 1, with no reverse current through it."

Next element: **ADD** `1 0 10` **R**
In other words, "add a 10Ω resistor to current loop 1, with no reverse current through it."

Next element: **ADD** `1 2 530E-6` **C**
In other words, "add a 530-μf capacitor to current loop 1, with a reverse current through it from current loop 2." So in this case, the element is shared between loops 1 and 2 with the currents opposing. Press **SHOW** now to see how this is represented:



```
#3: 1 → C ← 2
▶ 530.0E-6

 ADD   DEL   PREV  NEXT  SHOW  CHNG
```

This element is the last in loop 1. Continue now by keying in the elements of loop 2 (although the order in which you key in elements is immaterial). Of course, since the shared capacitance has already been entered, you shouldn't enter it again.

Next element: **ADD** `2 0 3` **R**
In other words, "add a 3Ω resistor to current loop 2, with no reverse current through it."

Last element: **ADD** `2 0 10E-3` **L**
In other words, "add a 10-mh inductor to current loop 2, with no reverse current through it."

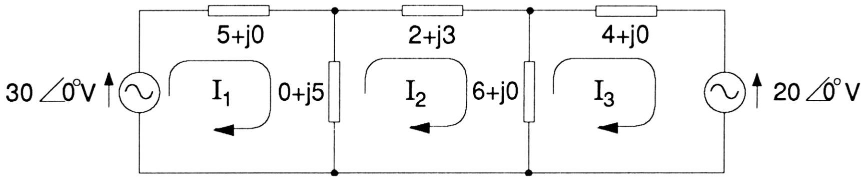
Now **EXIT** and solve this at 60 Hz: 60 **FREQ** **CALC**

You'll get a 2x1 array of the loop currents. To make them easier to read, extract them from the array format and convert the values to polar format:

`ARRY→ DROP R→P` Result: `(4.468E0,-60.75E0)`
So I_2 = 4.468∠-60.75°

`SWAP R→P` Result: `(2.896E0,6.873E0)`
So I_1 = 2.896∠6.873°

As a second example, find the current in loop 2 in the following circuit:



2 FIX INIT Y MESH EDIT
1 0 30 ADD V 1 0 5 ADD Z
1 2 (0,5) ADD Z 2 0 (2,3) ADD Z
2 3 6 ADD Z 3 0 4 ADD Z
3 0 -20 ADD V EXIT

Since there are no frequency-dependent elements, you don't need to store a frequency, so CALC, then 2 GET R→P
Result: (1.73,40.13) So I_2 = 1.73∠40.13°

Now, what if you were to delete the 20V source and change the 0+j5 impedance to 1+j5? How would this affect the current through loop 2?

Find out: EDIT 3 GOTO (the third element is the Z = (0,5).)

Then: CHNG 1 + (add 1 to the impedance. Note that the forward and reverse current loop numbers are in stack Levels 3 and 2 respectively, so everything's all set to reënter as the new Z value) Z

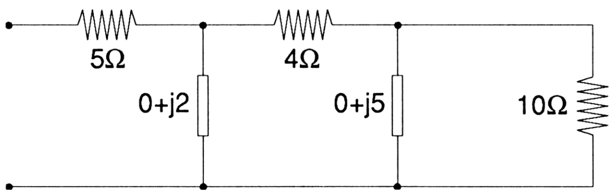
Then: 100 GOTO (you may not remember how many elements there are, but since you know you want to delete the last one in the list, just give a ridiculously large element number; the default for such an error is the last element.) DEL Y EXIT

CALC 2 GET R→P
Result: (2.28,2.85) So I_2 = 2.28∠2.85°

Ladder Network Transfer Analysis

Ladder network transfer analysis is a circuit reduction method by which you treat the branches of the circuit as rungs as a ladder and then reduce the network to a single impedance by step-wise combination of adjacent elements. As well as this input impedance, you can then calculate various output/input ratios (e.g. current, voltage, power).

The program provided here allows for parallel and series elements which may be resistors, capacitors, inductors or general impedances. For example, reduce the following circuit to its equivalent impedance:

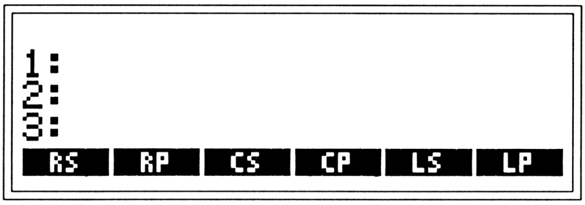


Type EEA ENTER. If there is no current circuit, select **LADR** from the Initialization menu. If there is a current circuit, save it if you like, and select **INIT** from the MAIN menu, respond with [Y] to the initialization query, and select **LADR** from the Initialization menu. You should be back to the MAIN menu then, so press **EDIT** and you're ready to begin building this ladder network.

Note: In a LADR-type circuit, *the order in which you enter the elements is significant.* The *lefthand end* of the ladder is considered to be the *input end,* where a "driving source" would be connected; the *righthand end* is therefore considered the *output end,* and so *the last element on the far right is considered the load impedance* (called Z_L in the programs). *You must enter this element first, then continue to work back from right to left (i.e. from output end to input end), keying in the elements as you encounter them in traversing the ladder in that direction.*

Thus, begin this example with the 10 ohm resistance: ADD 10 RP

In a ladder network, you consider the elements in the vertical branches to be added in parallel, and the elements in the horizontal branches to be added in series. Therefore, each element type has both a series and a parallel entry in the ADD menu (as you've probably noticed by now):



(press NEXT to see the of this menu.)

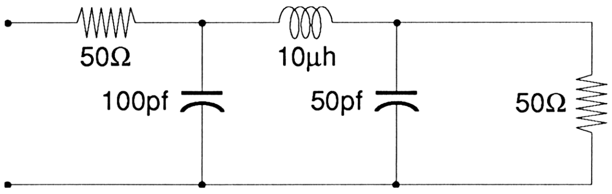
Continue now to build this circuit: ADD (0,5) ZP ADD 4 RS ADD (0,2) ZP ADD 5 RS EXIT

Since none of the elements in this circuit are frequency-dependent, you don't need to input the circuit's frequency. Just CALC....Now you have the CALC menu, which gives you options to calculate any of six different transfer functions for the circuit. Try each of them:

3 FIX INZ
Result: (5.333,1.667) Input Impedance
FTA
Result: (3.559E-4,-0.019) Forward Transfer Admittance
PTR
Result: 0.021 Power Transfer Ratio (*real* power gain)
CTR
Result: (0.033,-0.100) Current Transfer Ratio
VTR
Result: (-0.004,0.189) Voltage Transfer Ratio
FTZ
Result: (-0.333,1.000) Forward Transfer Impedance

Now, store that circuit: EXIT 'LADR1' SAVE

...and try another example:



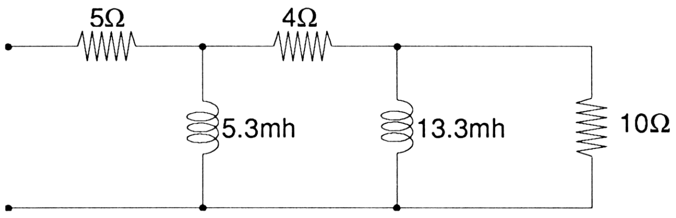
INIT (Y) LADR EDIT
50 ADD RP 50E-12 ADD CP 10E-6 ADD LS
100E-12 ADD CP 50 ADD RS EXIT

This circuit contains capacitances and inductances, so you must specify a frequency – say, 1 MHz.: 1E6 FREQ. Then: CALC....

3 ENG INZ
Result: (104.1E0,62.80E0) Input Impedance
FTA
Result: (-7.103E-3,4.769E-3) Forward Transfer Admittance
PTR
Result: 519.5E-3 Power Transfer Ratio (*real* power gain)
CTR
Result: (-1.039E0,50.29E-3) Current Transfer Ratio
VTR
Result: (355.1E-3,-238.5E-3) Voltage Transfer Ratio
FTZ
Result: (51.93E0,-2.514E0) Forward Transfer Impedance

Go back to the first circuit now and make the following changes, then calculate the equivalent impedance of the resulting circuit:

- Change the 0+j2 impedance to a 5.3 mH inductance.
- Change the 0+j5 impedance to a 13.3 mH inductance.
- Set the circuit frequency to 60 Hz.



Here's what you do:

```
LOAD (Y) LADR1 EDIT
2 GOTO
CHNG DROP 13.3E-3 LP
NEXT NEXT
CHNG DROP 5.3E-3 LP
EXIT

60 FREQ CALC
```

INZ Result: (5.332E0, 1.666E0) The new input impedance.

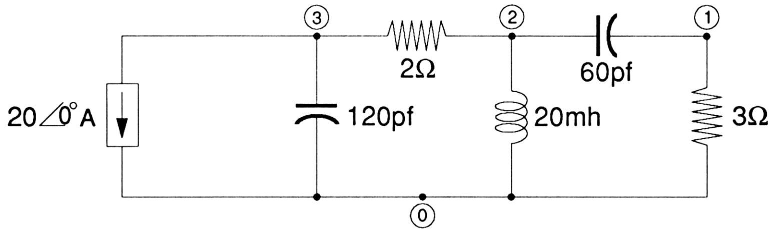
Appendix A

Expanding On The Tools

This appendix gives some more details on "slick and efficient" use of the two sets of Circuit Tools. It also suggests ways in which you can actually add to their capabilities with some programming of your own.

Plotting Frequency Sweeps

Often you may want to know how various parameters in a circuit depend on the frequency of the sinusoidal driving sources. Look at this example:



Using The Circuit Reduction Tools

If you *symbolically* reduce the above circuit using the Circuit Reduction tools (the EET menu), what you'll have is essentially *an equation for an impedance as a function of (radian) frequency:*

`2 ENG EET 'w' PURGE` (by purging the object called 'w', you prevent the HP-28S from associating any specific value with it.)

`MODE SRES TOOLS` (set symbolic results mode – if it's not set already – so that the HP-28S won't generate an error by trying to evaluate the now-undefined name, 'w'.)

```
60E-12 C→Z 3 R→Z ZADDS
20E-3 L→Z ZADDP 2 R→Z ZADDS
120E-12 C→Z ZADDP
```

(The current source doesn't contribute to the circuit's impedance so you leave it out when reducing the impedances to one equivalent.)

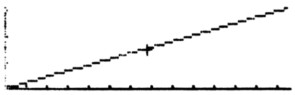
Now it's easy to generate a plot of the resulting algebraic expression, but since this expression is for an impedance that will evaluate to a *complex* number, some modifications must be made before it can be plotted.

First, the result must be transformed into a real number, since you can plot only real parts of a complex number (e.g. magnitude or angle, real portion or imaginary portion). Some useful functions for this include RE (real portion), IM (imaginary portion), ABS (magnitude), and ARG (the angle in the current angular mode). So, suppose you want to plot the *magnitude* as a function of the frequency. You would use ABS (perform this now on the expression sitting on the stack: ABS).

Secondly, you'll probably want to plot the magnitude versus the *cyclic* (Hz.) frequency, F, rather than the radian frequency, ω. So you simply *store a conversion expression* into 'w': 'F' F→w 'w' STO

Then: ■PLOT STEQ

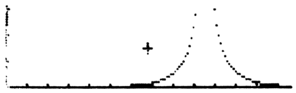
(0,0) PMIN	(set lower limits for the x- and y- axes)
(1000,125) PMAX	(set upper limits for the x- and y- axes)
'F' INDEP	(F is the independent variable, not ω)
DRAW	



And you can also get a *semi-log plot* of this same information (where the x-axis is the common log of the actual frequency) ranging from 0 to, say, 10 Mhz:

```
'F' ALOG F->w 'w' STO (7,10000)
```

■PLOT PMAX DRAW



Using The Node And Mesh Analysis Tools

In order to vary the frequency for plotting the results of either the MESH or NODE analyses, you must store the frequency and perform CALC for each point you plot. A first attempt at a routine to help you do this might look like this:

```
« F FREQ CALC »
```

As in plotting with the Circuit Reduction Tools (see the previous pages), F is the frequency (the independent variable) which FREQ transforms and stores for use by CALC, which would then perform tha calculation. Unfortunately, FREQ (as written on page 120) first calls the program MAIN which effectively aborts the calculation. So rewrite the program:

```
« F FRQ CALC »
```

where FRQ is

```
« → F « CIRC DUP 1
GET 2 F PUT 1 SWAP
PUT 'CIRC' STO » »
```

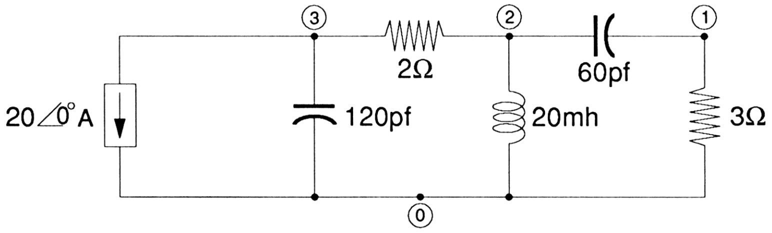
(This is the FREQ program without the call to MAIN. To type this in quickly, you can just `'FREQ' RCL`, then edit out the MAIN call, and `'FRQ' STO`. Put this into the °EE directory if you plan to use it a lot.)

Now, this *still* won't work as a plot, since PLOT can use only *real* values to plot (not arrays of complex numbers). So you must pick a mesh current/node voltage from the returned array and transform it into a real value (using RE, IM, ABS, ARG, or another transformation):

```
« F FRQ CALC 3 GET
ABS »
```

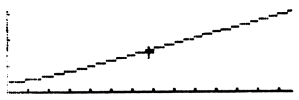
This program returns the magnitude of the third mesh current or node voltage, given a frequency, F. Now *that* is a PLOTtable function.

So here's the sample circuit again. Generate the plot of the voltage at node 3, as a function of the frequency:



```
EEA INIT (Y) NODE EDIT
3 0 ADD 20 I
3 2 ADD 2 R
0 3 ADD 120E-12 C
2 0 ADD 20E-3 L
1 0 ADD 3 R
2 1 ADD 60E-12 C

■(PLOT) (0,0) PMIN (120,305) PMAX
'F' INDEP
« F FRQ CALC 3 GET ABS » STEQ
DRAW
```



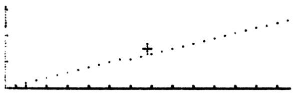
As you can see, because CALC must be called for each point, the plot takes several minutes. The process can be sped up greatly, with the loss of some detail, by entering a larger resolution factor (say, 10) with the RES command of the PLOT menu.

Using The Ladder Analysis Tools

Plotting with the ladder analysis tools is much like plotting with the mesh and node analysis tools; you need to use FRQ instead of FREQ to change the frequency. The main difference is that the plotting program for ladder analysis *must* clear flag 3.

For example, to plot the magnitude of the input impedance of the circuit on the previous page – as a function of frequency:

```
EEA INIT Y LADR EDIT
3 ADD RP 60E-12 ADD CS
20E-3 ADD LP 2 ADD RS
120E-12 ADD CP
■PLOT « F FRQ 3 CF INZ ABS » STEQ
(0,0) PMIN
(500,75) PMAX
5 NEXT RES ■PREV DRAW
```



Note that flag 3 is cleared before the call to the calculation routine (INZ in this case). Flag 3 is an optimizing feature of the ladder analysis tools. When it's set, the circuit list is not "re-crunched" before the calculation, thus speeding up multiple calculations on the same circuit. Clearing flag 3 therefore forces a "re-crunching" of the circuit – necessary because you want to evaluate it each time at a different frequency.

You may want to add 3 CF to your FRQ program – it has no effect on node or mesh calculations.

A Quick Note On Logarithmic Plots

As was mentioned briefly in the discussion of plotting with the Circuit Reduction Tools (page 232), log or semi-log plots are quite easy:

To get a semi-log plot of the function `'SQ(X)'`, where the *y-axis* is logarithmic (base 10), you would transform the function to:

`'LOG(SQ(X))'` or `« X SQ LOG »`

To get a semi-log plot where the *x-axis* is logarithmic, you would either:

- Store `'ALOG(Y)'` in `'X'` (this may mean that you 'll also need to give the commands `'Y'` **INDEP** in the plotting menu to tell the plotter that `Y` is the new independent variable).

- Rewrite the function so that it is a function of `'ALOG(X)'`:

  `'SQ(ALOG(X))'` or `<< X ALOG SQ >>`

Either way, you will need to remember that the logarithmic axis must be *rescaled* (by using `PMIN` and `PMAX`) to reflect that it is now scaled in *factors of 10* rather than the actual x- or y- values.

Log-Log plots are merely combinations of the two methods:

`'LOG(SQ(ALOG(X)))'` or `« X ALOG SQ LOG »`

For odd logarithmic bases, replace `ALOG(X)` (which is equivalent to `X^10`) with `X^N` where `N` is the base you want. Replace `LOG(X)` with something like `LOGN`: `« → N X 'LOG(X)/LOG(N)' »`

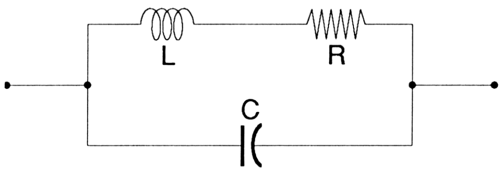
Programming With The Circuit Reduction Tools

The Circuit Reduction Tools were designed to work in the same fashion as the built-in HP-28S commands: You key in values and invoke the function; the results being left on the stack. So you can program with them just as you can with the built-in commands.

For example, the results of node and mesh analyses are arrays of complex numbers in rectangular form, but voltages and currents are commonly represented in polar form. Here's a program that will take the resultant array and convert it to individual complex numbers in degree-polar form (the relative order of the numbers is preserved):

```
VIOUT:      « → A « A SIZE 1 GET
            1 SWAP FOR I A I GET
            →° NEXT » »
```

Something else that you might want to do is to program a certain circuit configuration. For example, here's a useful little program if you have the following simple circuit pattern within which you want to vary only the element values:



```
LRC60:      « → L R C « 60 F→ω
            'ω' STO L L→Z R R→Z
            ZADDS C C→Z ZADDP →i
            » »
```

The name, `LRC60`, tells you that you must key in the L, R and C values in that order and that they will be combined at 60 Hz. The result will be returned in `'Re+i*Im'` format. Note that if you were to omit `→i`, this program it could be used to generate algebraic objects.

Adding Elements To The Analysis Programs

If there are new circuit element types that you want to be able to process with the circuit analysis tools, there are ways to add to the possible element types of each tool. Be careful though: While adding certain element types to the circuit editor is relatively easy, others will require changing the calculation routines so that they correctly process your new element type.

This discussion can only be considered as an introduction to the subject, since a more detailed description would require more space than is reasonable here. Also, it is not this book's intent to teach the subject of mathematical circuit analysis, so most of the descriptions will of necessity be brief and assume that you already understand the mathematics of the matter.

In most (if not all) cases, the objects defined here should be stored in the °EE directory.

Node And Mesh Analysis

The mesh and node analyses are almost identical; in either case, the circuit element list is interpreted and the elements are put into their respective arrays in very similar manners.

For **node analysis**, the matrix equation looks like this:

$$\mathbf{Y}^{\times}\mathbf{V} = \mathbf{I}$$

Here **Y** is the admittance matrix, **V** is the node-voltage vector and **I** is the node current vector.

For **mesh analysis**, the matrix equation is:

$$\mathbf{Z}^{x}\mathbf{I} = \mathbf{V}$$

Here **Z** is the impedance matrix, **V** is the node-voltage vector and **I** is the mesh-current vector.

To add elements to the ADD menu of each of the analyses, it is a simple matter of adding the element name to the appropriate element list (see pages 157 and 150, respectively, for full descriptions of these lists):

(Node Elements:) `NODEELS { R C L Z I EXIT }`
(Mesh Elements:) `MESHELS { R C L Z V EXIT }`

You should note that the node analysis routines only supports passive elements and *current* sources while the mesh analysis routines only support passive elements and *voltage* sources. Since pressing the appropriate menu key from the ADD menu should perform the function of adding that element to the circuit list, the element name should actually call a program.

Consider the addition of a general admittance element to the mesh analysis program. The name `Y` should be added to `MESHELS` and the following program should be stored into that name, `'Y'`:

```
« "Y" PUTE »
```

This is true for both active and passive element types. The character string is the name of the element. `PUTE` (put element) puts the new element in the circuit list in the same way for the particular circuit type as it does for any other elemant. For mesh elements, the stack must contain the forward current loop number, the reverse current loop number, and the element value, in that order.

Then the element must have a corresponding conversion routine. For *passive* elements, the routine must convert the element value into an *impedance*. For *active* elements, the conversion is to a *voltage*.

Thus, for Y, the following conversion routine is necessary:

```
YMESH:      « INV ZMESH »
```

After converting to impedance (**Z** = 1/**Y**), the routine calls ZMESH, which places the impedance into the impedance array. The process is identical for node analysis, so

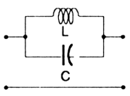
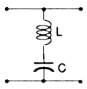
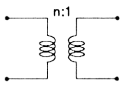
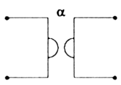
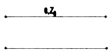
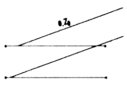
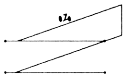
```
YNODE:      « INV ZNODE »
```

Notice that the name of the conversion/storage routine *must* be the element name combined with the analysis type (Y + MESH = YMESH). For active elements, the process is identical except that the element storage routines are named INODE and VMESH.

Ladder Analysis

The Ladder Network Transfer Analysis program in this book uses a calculating technique called a chain-parameter matrix, adapted from the HP-41 Circuit Analysis Pac. The technique defines a parameter matrix for each element type, and the parameter matrices for all circuit elements are multiplied (in the order of the elements' occurrences in the circuit) to produce an overall *equivalent parameter matrix.*

The parameter matrices for the circuit elements allowed in this book's programs are given in Chapter 5 (pages 109, 111, 142, 144, 175, 177, 197, and 199). However, there are many other possibilities. Here are some others that are also listed in the HP-41 Circuit Analysis Pac:

Element Type	Schematic	Chain-Parameter Matrix
Series LC		$\begin{bmatrix} 1 & \dfrac{i\omega L}{1-\omega^2LC} \\ 0 & 1 \end{bmatrix}$
Parallel LC		$\begin{bmatrix} 1 & 0 \\ \dfrac{i\omega C}{1-\omega^2LC} & 1 \end{bmatrix}$
Transformer (n:1)		$\begin{bmatrix} n & 0 \\ 0 & 1/n \end{bmatrix}$
Gyrator (α)		$\begin{bmatrix} 0 & \alpha \\ 1/\alpha & 0 \end{bmatrix}$
Transmission Line (θ, Z_0)		$\begin{bmatrix} \cos\theta & Z_0 i\sin\theta \\ \dfrac{i\sin\theta}{Z_0} & \cos\theta \end{bmatrix}$
Open Stub (θ, Z_0)		$\begin{bmatrix} 1 & 0 \\ \dfrac{i\tan\theta}{Z_0} & 1 \end{bmatrix}$
Shorted Stub (θ, Z_0)		$\begin{bmatrix} 1 & 0 \\ \dfrac{-i\cot\theta}{Z_0} & 1 \end{bmatrix}$

To create a new element type, say the shorted stub, first add the new element name to the LADRELS list:

```
LADRELS:   { RS RP CS CP LS LP ZS
           ZP SSTUB EXIT }
```

As with mesh/node analysis, the new element name must name the program that stores the element value:

```
SSTUB:     « "SSTUB" PUTE »
```

Note that SSTUB requires two input values – Z_0 and θ – but PUTE expects only one value on the stack for ladder analysis. The easiest way around this is to combine the two inputs into a single data object, such as a list or – better yet – a complex number. Thus, to use the new SSTUB program, press (, then key in the Z_0 value, press ,, key in the θ value and finally, invoke SSTUB.

Again, just as with mesh/node analysis, there must be a conversion/storage routine called SSTUBLADR. This routine will take the input value and create the corresponding array. Note that the routine assumes that θ is in degrees.

```
SSTUBLADR: « RCLF DEG SWAP C→R
           TAN * i * INV →NUM
           SWAP STOF 1 0 ROT 1
           { 2 2 } →ARRY »
```

Note: The ladder calculation engine (LADRENGN) will not allow the load impedance (that's the *first element* you key in – on the far righthand end of the ladder) to be anything other than a parallel or series capacitance, inductance, resistance or general impedance.

A Short Note On Units

All of the circuit analysis routines assume standard (or at least consistent) units. If you commonly use some other units, you can use the processes described on the previous two pages to define new element types in your preferred units.

For example, in node analysis, to be able to key in capacitances in, say, picofarads (pf), you could do the following:

1. Add `PICOC` to the `NODEELS` list;
2. Create the object `PICOC`: `« "PICOC" PUTE »`;
3. Create the object `PICOCNODE`: `« 1E-12 * CNODE »`.

Notice how the third step takes advantage of the fact that `CNODE` does most of what's necessary; all you need to do is to scale the capacitance value.

Appendix B

Some HP-28S Basics

Just in case you need a refresher, here are a few sample programs together with the steps needed to key in, name, edit, store and use them.

"How Do I Load A Program?"

Consider the following program:

```
« → a b « a 'F' STO*
b F + →NUM "The answer
is " SWAP →STR + CLLCD
1 DISP » »
```

You *might* key it in this way:

[«] ■ [→] [LC] [A] [SPACE] [B] [«] [A] ['] [LC] [F] ['] ■ [STORE] `STO*`
[LC] [B] [SPACE] [LC] [F] [+] ■ [→NUM] ■ ["] [T] [LC] [H] [E] [SPACE] [A] [N] [S] [W] [E] [R] [SPACE]
[I] [S] [SPACE] ■ ["] ■ [SWAP] ■ [STRING] `→STR` [+] ■ [CONTRL] [NEXT] `CLLCD`
[1] `DISP` [ENTER]

You *might* key it in that way – or you might not – because there are many ways to do it. Take a look now at some of the details here:

[«] ■ [→] [LC] [A] [SPACE] [B] The [«] keystroke signals the beginning of a program; it will always be the first key you press when entering a program. It also turns on the alpha cursor (▮), so that certain typing aids help to make loading the program easier. For example, you don't need to type spaces in between the keystrokes [«] and [■][→] – it's done automatically. This saves you many keystrokes, because *spaces must be keyed in exactly as they appear in a program listing.* Notice that pressing any menu key (such as **STO**) will also automatically put a space before and after the command name it types.

However, *no spaces* are needed around either the « or ' characters, *because they are delimiters* (like ", {, [, (, #, SPACE and NEWLINE), used by the HP-28S to delimit objects. The HP-28S puts spaces around the « simply to improve readability.

Notice that *case is also significant.* The a and b *must* be lowercase, so you press [LC] before keying them in.

"How Do I Name It?"

Once you've keyed a program into the HP-28S so that it's on the stack, you'll need to give it a name by which you can call it and use it.

To give a program a name, the program must be on Level 1 of the stack. Then you need to put a unique and fairly descriptive name on the stack at Level 1 – pushing the program to Level 2. You should put single quotation marks (') around this name to prevent the HP-28S from trying to evaluate it when you [ENTER] it.

So, for example, to name the above program FRED (assuming the correct and complete program is now sitting at Level 1 of the stack), press

[']\[F]\[R]\[E]\[D]\[STO]

This procedure will overwrite any object named FRED in the current directory, so you should take care that the name is unique!

"How Do I Change It?"

Now suppose you want to change the stored program (you keyed it in wrongly or you want to enhance it). You could recall the program (put it on the stack), edit it (with EDIT) and re-store it, or you can VISIT it, which accomplishes all three things in one.

Type ' F R E D ■ VISIT. You can now move around the program with the cursor keys. Typing anything new will *overwrite* unless you're in insert mode (INS), in which case what you typed is inserted. For example, add a NEWLINE to the end of the "The answer is " string. To do it, press ▼ ▼ ▼ ■ ▶ ◀. The cursor is over the quotation mark. Press ← to delete the space and INS ■ NEWLINE to insert the newline. While VISITing, NEWLINE characters actually cause a newline break in the program line.

Press ENTER now to accept the changes – or ON/ATTN will abort the edit without changing the program. Then recall the program (press ' F R E D ■ RCL). Notice that the newline character is replaced by a ■. This recalled program is a *copy* of the FRED program; changes to this copy won't affect the original unless you re-store it (' F R E D STO). Press DROP to remove the program from the stack.

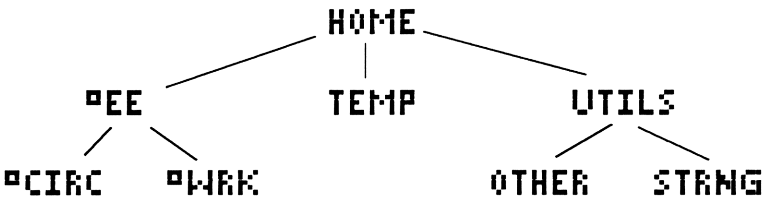
"How Do I Use It?"

Once you've loaded your program, all you need to do to use it is call it by name: F R E D ENTER. If the stack was empty before you started, you'll get an error (Too Few Arguments) because the program needs input. The moral here is that you must always know the requirements of your program before you run it. In this case, the program needs two real numbers on the stack and another real number named 'F'. So start again: 0 ' F STO 1 ENTER 2 ENTER F R E D ENTER.

The answer is 2. You should also notice that the name FRED appears on a menu key when you press USER. All things that you, the *user,* create are stored in *user* memory and showed to you by the *USER* menu.

"Where Do I Put It?"

For convenience, and organization, you can divide your USER menu (user memory) into *directories* – named areas partitioned off from the rest of user memory. The main directory is HOME, but you can create other sub-directories. A typical diagram of directories might be:



The directory you are in is the *current directory.* The directory containing your current directory is its *parent.* All directories sharing the same parent are called *siblings;* all the subdirectories of a directory are its *daughters.* If you were in directory UTILS above, the parent directory would be HOME; the sibling directories, °EE and TEMP; the daughter directories, STRING and OTHER. To put FRED into UTILS, you'd press H O M E ENTER U T I L S ENTER to move there before storing FRED.

Here's why this matters: Typing the name of an object will evaluate that object only if it can be found either in the current directory or its parent (or grandparent or great-grandparent, etc.). *If your current directory is STRNG in the above diagram, you could successfully evaluate (run) your program, FRED, only if it were stored in either STRNG, UTILS or HOME. Were it anywhere else, you could not name it and find it.* This means that since HOME is every directory's ultimate parent, *an object "living at HOME" can be found and evaluated from any directory.*

Index

PO Box 1928 **SOLVE and INTEGRATE** *Corvallis, OR 97339*

For the HP-28S:

______ Please send me a list of available HP-28S software.

______ I'd like to be a Users' Library member. Annual membership includes a catalog, $20 credit toward the purchase of Library software and a quarterly newsletter.

______ Enclosed is $25 (US and Canada)

______ Enclosed is $40 (All other countries)

______ I have some software that I think others could use. Please send me a program submittal package.

For the HP-41,HP-71B, HP-75:

______ Please send me the current software catalog.

______ Enclosed is $10 (US and Canada)

______ Enclosed is $15 (All other countries)

______ I'd like to be a Users' Library member. Annual membership includes a catalog, $20 credit toward the purchase of Library software and a quarterly newsletter.

______ Enclosed is $25 (US and Canada)

______ Enclosed is $40 (All other countries)

______ I have some software that I think others could use. Please send me a program submittal package.

For the Series 80:

______ Please send me the current software catalog.

______ Enclosed is $5 (US and Canada)

______ Enclosed is $10 (All other countries)

______ I'd like to be a Users' Library member. Annual membership includes a catalog, $25 credit toward the purchase of Library software and a quarterly newsletter.

______ Enclosed is $25 (US and Canada)

______ Enclosed is $40 (All other countries)

______ I have some software that I think others could use. Please send me a program submittal package.

(Prices are subject to change without notice.)

OVER ☞

General information

_____ Please send me information about custom PROGRAMMING. Solve and Integrate will translate any existing Library software or write new software to your specifications.)

_____ Please send me information about custom BARCODE. (Solve and Integrate will print HP-41 barcode for your HP-41 program.)

_____ I would like to know/like you to know: ________________

____

____

____

____

____

____

- -

Please make your check payable to **Solve and Integrate** *OR*

VISA or MasterCard # ________________________ Exp. ______

Your signature ________________________ Phone (   ) ________

Name ____

Address ____

City ________________________ State ______ Zip ________

Return form to: **Solve and Integrate**
P.O. Box 1928
Corvallis, OR 97339
(503) 754-1207

Reader Comments

We here at Grapevine love to hear feedback about our publications. It helps us write books tailored to our readers' needs. If you have any specific comments or advice for our authors after reading this book, we'd appreciate hearing from you!

Which of our books do you have?

Comments, Advice and Suggestions:

May we use your comments as testimonials?

Your Name: Profession:

City, State:

How long have you had your HP calculator?

Please send Grapevine Catalogues to the following persons:

Name ______________________________

Address ______________________________

City ______________ State ________ Zip ________

Name ______________________________

Address ______________________________

City ______________ State ________ Zip________

Solve
and
Integrate

HP-28S Software Power Tools Electrical Circuits

This cover flap is handy for several different things:

--Tuck it just inside the front cover when you store this book on a shelf. That way, you can see the title on the spine.

--Fold it inside the back cover--out of your way--when you're using the book.

--Use it as a bookmark when you take a break from your reading!

GRAPEVINE

HP-28S Software Power Tools

Electrical Circuits

Engineers and soon-to-be-engineers! Here are the software tools `you've been waiting for! Now you can use your HP-28S, with its complex-matrix-crunching power, to analyze your steady-state AC circuits.

You can do either mesh or nodal analysis, construct general networks and ladders, and use a host of small utility routines to do side calculations as you wish. You'll build clear, friendly, easy-to-edit descriptions of your circuits, which may have any of the following elements in series or in parallel: Resistors, capacitors, inductors, independent voltage sources, independent current sources. Then you can vary the frequency of your sources and plot results, either on the display or the infrared printer. And you can name and store your circuits for later use, too!

No matter how little experience you have with the HP-28S, you'll never be "snowed under" while trying to get this program up and running. Every routine is explained, every piece of the program is documented, and a little diagram of the HP-28S' memory is always handy, ready to show you "where" you are and what you're doing. The best news yet is this: There is an appendix of additional routines and pointers to allow you to customize the core program to your own preferences. This book teaches you about the machine even as it guides you through this easy and powerful software!

ISBN 0-931011-25-6

0 12841 00025 5

HP Part # 92220X

FROM THE PRESS AT
GRAPEVINE PUBLICATIONS, INC.
P.O. Box 118 • Corvallis, Oregon 97339-0118 • U.S.A. • (503) 754-0583