XLI - THE PCA SCRIPT LANGUAGE

Introduction

The XLI interpreter is built into PCA / ARGUS. The interpreter can handle astrological interpretations as well as other text or graphics outputs and special calculations. A number of modules and interpretations are commercially available while other are free. Some of them can be downloaded from the Electric Ephemeris website www.electric-ephemeris.com.

You may also program XLI modules yourself. You can do some simple stuff without being a programmer, but you can also write very sophisticated modules or interpretations with a log of graphic details, specialized calculations, tables, ephemerides, etc.

You can add as many modules as you like and they can be branched and chained into any structure, and you may add menus to navigate around.

This documentation will give you the information necessary to write your own modules and provide a complete XLI function reference and examples.

You may look into any modules provided with your PCA program, or you may download extra modules and look into them to figure out how they are programmed.

Using and writing interpretations (entry level)

The XLI Toolkit offers an extensive and utmost flexible mecha-nism for writing computer interpretations. You may define even very complex rules and texts. Even though this manual provides the information necessary, it is not a beginners job to write a complex--rule interpretation.

Luckily, the toolkit also provides an entry-level solution, with ready-made interpretation skeletons and a built-in fool-proof text-editor. You may use this to write your own full blown inter-pretation, or you may use it as a note-organizer for basic astrologi-cal combination rules.

You may have objections about computer interpretations. We would certainly agree that the computer cannot "understand" the person behind the chart, nor have any psychological insight. But computers are very good at organising facts and freeing you of the systematical (and boring) work. So you may for example use the interpretation skeletons to insert your experiences of certain combinations when you meet them, and let the program scan all new charts to remind you, when the same combination reappears. As a beginner, you may put your notes from books and lectures into the skeletons in the same way.

If you wish to do more complex things, you must refer to the next part of this manual, describing the XLI programming language.

INSTALLATION OF NEW MODULES IN ARGUS:

If the module is not prepared for automatic installation, you must use the setup menu in ARGUS to anually create a menu item and connect it to any XLI file on the harddisk.

If the module is prepared for automatic installaion, the setup menu has a button "Install" which lets you insert a diskette in drive A: and it will copy the files, create the menu item and establish the connection automatically.

To prepare the installation diskette, you must place a text file called TXT.DEF on the installation diskette. Here follows an example of a TXT.DEF file:

```
Transit-Interpretation
TYDT
200
TT1.TXT
TT2.TXT
TT3.TXT
TT4.TXT
TT5.TXT
TT6.TXT
TT7.TXT
TT8.TXT
TT9.TXT
TT10.TXT
```

The first line is the text which will be put on the menu item, which should be the name of the interpretation.

The second line is the name of the subdirectory under ARGUS, in which the interpretation files should be placed.

The third line is a number, telling how many kilobytes the installaion will need, so that the installation program can give a warning if the harddisk is too filled up to hold the files.

The fourth line onwards are the names of the text files. The first of these must be the entry file, i.e. the first in the call chain.

Up to ver 3.1 release 17, all texts must fit on one diskette. From release 17 multi-disk intallation is supported:

Multi-Disk installation (from relase 17)

In TXT.DEF, put an '&' before the first filename on next disk to produce a "next disk please" prompt.

If in the above example TT1-TT5 should be on disk 1 and TT6-TT10 on disk 2, change the TXT.DEF:

```
Transit Interpretation
TYDT
200
TT1.TXT
TT2.TXT
TT3.TXT
TT4.TXT
TT5.TXT
```

&TT6.TXT
TT7.TXT
TT8.TXT
TT9.TXT
TT10.TXT

XLI Programming

THE CONCEPT OF XLI MODULES

The PCA astrology program has a built-in option to interpret external add-on files. These files may be interpretation texts, choice menus, position printouts, aspect tables, educational programs, and even completely separate programs as for example text editors.

These add-on modules which we in this book will refer to as XLI-modules may be written by anyone, just using the basic DOS editor that came with the computer. How to do this, is what this chapter is about.

Writing your own modules or interpretations will need something between no programming skills at all up to expert level depen-ding on the task. So you can choose your own level and proceed to higher levels if you like.

ABOUT THIS CHAPTER

You will be guided through the XLI mysteries in stages revea-ling the different features of the language starting at the very basic level and leading towards more sophisticated use.

At each level you will find reference to ready made tested examples that you can try out for yourself.

XLI FILES

The XLI-modules are text files (ASCII-Files). Files are named groups of data, that can be stored on your disc and printed to your printer. A text file as opposed to a program file or binary file can be read on the screen or printed to paper and still make some kind of sense to the reader.

The file system needs that all files have names of 1-11 letters or numerals. The first 1-8 positions is the filename, and the last three are called the extension. The extension is normally used to tell something about the kind and purpose of the file. The XLI-files all have the extension XLI or TXT. The filename and extension are separated by a fullstop e.g. TRANSIT.XLI. The .TXT extension is intended for interpretation texts, while the XLI extension should be used for functional modules doing other things.

The XLI files are made of a mixture of text and coding (program instructions). The text is to be printed (on conditions) or may be just explanatory comments. The coding is instruction to the PCA program to do certain tasks, for example look into charts and check positions and aspects, change dates, add numbers, store ratings or whatever. The text and coding is separated following certain rules, which are described in this manual.

CREATING OR EDITING XLI-FILES

CHOOSING AN EDITOR

To be able to create or change a textfile you will need a text editor program. There are a lot of such programs available. ARGUS itself has a very simple editor program. You may also use Windows Notepad. For bigger files you need a more capable editor.

If you cannot find a useful editor amongst your existing soft-ware you may use an ordinary wordprocessor. The problem about this is, that wordprocessors are somewhat too advanced for this job, and you must deliberately ask it to remove all its inter-nal instruction, underlining, different fonts etc from its files. Nearly any wordpro-cessor has instructions to create ASCII files which does exacly this. But there may still be concern about automatic linefeeds, page shifts and the like.

Finally, there are a lot of public domain or shareware editor programs available, which are very low priced or free. These may be obtained either from the special PD/SW dealers for a low handling fee or if you have a modem, you may download them from the numerous BBS (Bulletin boards) around.

RUNNING THE XLI MODULES FROM PCA

The PCA manual describes 4 ways of accessing external modules. Though these methods can be used quite interchangeably, we would recommend:

The XLI-Menu: Just click into File | XLI-Menu and navigate to find the module to run.

A Macro: Press C and enter XOfile,ext. "file" means the first part of the filename, "ext" the extension, which is normally XLI. Note that file and extension are separated by a comma instead of the usual fullstop. Instead a fullstop must be inserted after the extension. For example to call the module EPHGEN.XLI you should enter XOEPHGEN,XLI.

An F-key: Use the makro key definition in the files pulldown menu, and enter a makro string into a free F-key definition field. The macro must be same format as for a single macro. Remember to save the F-key definitions by clicking Preferences | Save-to-disk.

NOTE: A running XLI module may be interrupted by ESC returning to the PCA main menu. This will normally do no harm, but some modules will change the input menu, and by pressing ESC, you will not give them a chance to reset it to its original values.

XLI FILE BASICS

Having found a suitable editor program, you may begin to look at the XLI-files provided. You'll find some of them quite complex, and others quite easy.

An XLI-module or interpretation may be just one single file, but often several files are chained together. If chained, The XLI interpreter will read the complete chain. Chaining files are done at the end of each file by stating the name of the next file. If no name is stated, the file will be considered the last, and the XLI interpreter will put control back to the user and the main menu.

THE XLI PARAGRAPH

An XLI module is made up of one or more (chained) files, and each file is made up of one or more paragraphs. The paragraph is the basic building block of the XLI module.

Each paragraph will have some program instructions (Coding). These instructions can do a lot of things. For interpretations, they are used to calculate or find out whether the following text applies to the given chart, for example if the Moon is square to Mars. For modules they may setup menus, change characteristics of the PCA main program, control printer, print graphs or figures, control the XLI interpreter etc etc.

After the coding, there may be some text for printout. This text may be of two kinds: Commentary text or heading, and ordinary text.

The following example is a very short file just printing a text if the Sun is in Leo (in the latest Radix chart):

```
$                                       $-line
1 PSI 5 =                               coding

*Sun in Leo                             Heading/comment (optional)

Dignity, good self-esteem, etc...       Text (optio-nal)
$                                       Next $-line
```

$-LINE: A paragraph must always start with a $-line. This is just a line consisting of one $-sign.

CODING: Next comes the coding part. This may be one or more lines starting just after the $-line.

The coding simply means:

1	(Sun=1)
PSI	(planet in sign)
5	(Leo=5)
=	("is").

You may find, that the instructions seem "reversed order". The idea is quite simple though and will be explained later. For the moment, you don't need to bother.

HEADING: The Heading part must start with an asterisk. There may be just one or several lines. The headings are printed only if the module is called with `OI` (Alternative interpretation). The heading part may be omitted.

TEXT: The text part comes last. It can be any text, and it may be as many lines as you wish. It is possible to insert templa-tes for printing variable numbers or to a limited degree varible text strings, for example to put the person's name into the text. It is also possible to produce bar graphs.

COMMENTS

It is often very useful to put comments into your XLI modules. Comments are notes for the programmer and for others, who like to look into your files. If you after some time get back and want to add or change anyting, you will find, that you do not remember any more, what your coding is doing, so you will have to analyze each code. By inserting comments, you have a way of reminding yourself, what each part of your program is doing. The modules provided with this toolkit have a lot of comments.

Comments have no influence on the module itself. It will work the same with or without comments. You may find a slight reduction in speed if you put a lot of comments, because the XLI interpreter will need to identify and discard all the comment parts. Normally this will be no problem, and we strong-ly recommend commenting your codes.

To insert a comment, start with a semicolon and put the comment after this. The interpreter will ignore the semicolon and the remainder of that line.

If you need several lines of comments, each comment line must start with a semicolon.

The comments will normally be placed in the coding part of the paragraph to explain the workings of the coding. You may place the comments on the same line as the codes or in a line of its own. Look into the coding examples and see how it is done.

If you wish to place comments in the text or header parts of the paragraph, the semicolon must be at the very start of the line. That means, that you may not mix printable text and comments on the same line.

CODING YOUR FILES

The XLI interpreter uses a unique coding language. It uses a notation somewhat like FORTH or the old HP-pocket calculators. Technically it is called "Stack-orientated". To program, you write a mixture of numbers and codes on one or more lines as you wish. You will find, that breaking your coding into many short lines is easier to read and understand. This also leaves room for adding comments.

To illustrate, the coding below calculates the distribution of the fire element in a chart using a point system:

```
$
14 PSI 0 1 5 9 IN 3 MUL      ;Asc in fire, 3 points
1 PSI 0 1 5 9 IN 3 MUL ADD   ;Sun in fire, 3 points
2 PSI 0 1 5 9 IN 2 MUL ADD   ;Moon in fire, 2 points
5 PHS 0 1 10 IN              ;Mars in 1. or 10. house
1 PHS 0 1 10 IN OR           ;Sun in 1. or 10. house
3 MUL ADD                    ;3 points
3 10 FOR                     ;count planets Mercury to Pluto
1 CNT PSI 0 1 5 9 IN ADD     ;if in fire, add 1 point each
NEXT                         ;end of count loop
11 STO 0                     ;store result in cell 11
```

Don't worry too much about the codes here. Their meaning will be explained later.

The codes are here arranged so that each line works as a kind of "functional subunit". The comments are very useful as a reminder of what you are doing and will make it easier to read.

You may put up to 1000 characters into the coding field in-cluding comments. If you need more coding, you must spread it on more paragraphs. If a paragraph has no text fields, just coding, the interpreter will just continue on the next para-graph.

The stack orientation means, that you must adapt to the first--in-first-out way of thinking. Adding 2 and 2 will look like this:

```
2 2 ADD
```

Note, that the numbers come first, then the instruction what to do with the numbers. So ADD will happily add whatever it finds on the stack, in this case the two numbers 2 and 2. These two numbers are removed, and the result, 4 is put back on the stack.

This way of doing things is actually not much different from the PCA main menu. First you enter your data, then you press `R`, `P` or whatever, which will use, whatever data they find in the input menu. You are not asked for data AFTER you press `R` (for Radix).

WRITING TEXT LINES

The coding must always be terminated by one blank line. From then off and till the next $-line you may insert any quantity of text, limited only by disc space. We would recommend, however, that you keep your text in small paragraphs which are more manageable. The lines must, however, be no longer than 80 characters,

fitting the IBM screen. It is possible to merge a limited number of variable strings and numbers into the text, if you wish to print planetary positions etc from the program.

LINE CONCATENATION

If you for some reason wish to write lines longer than 80 characters, e.g. to printer or file, you may do so by line concatenation. If a line ends with a backslash (\), the normal carriage return and linefeed will be omitted, so that the following line will just continue at the end of the current (not printing the backslash of course).

You'll find an example of line concatenation in the EXPOS utility. To export the nativity name and all the positions, a line of more than 80 characters is needed. So the name printout ends with a backslash, then comes the line with the positions. In the printout the two will be merged into one line.

THE STACK

It was earlier mentioned, that XLI is stack-orientated, and that this means that you must put values first. Stack operation is beautifully simple and very powerful. However, because we normally write calculations in a different style (alge-braic notation), it takes time getting used to it.

Here is an example of an ordinary calculation task:

```
(5+8)*(8-3)
```

The result should be 65.

The parenthesis are necessary to assure, that the calculations are done in the correct order. In some systems, some of the parenthesis may be omitted, if there exists a definition that gives priority to certain operations (*/+-) over others. Parenthesis and operator hierarchy is in fact very akward and complicated, but we are used to looking at problems this way.

The same problem coded in XLI would look like this:

```
5 8 ADD 8 3 SUB MUL
```

This reflects the way you actually would solve the problem: add 5 and 8, keep the result while subtracting 8 and 3, then multiply the two results. No parentheses are needed, calcula-tions are shown in the same order that they must be executed.

If the parentheses in the above example were moved like this:

```
5+(8*8)-3
```

The result should now be 66.

In XLI notation it would look like this:

```
5 8 8 MUL ADD 3 SUB
```

The instructions rather than the parenthesis are rearranged to reflect the changed execution order.

This brings us back to the stack. To do the above calculation, the system must have some means of keeping the intermediate results. Here five is kept while multiplying 8*8, before the addition is possible. This is done using the stack.

In stack orientated languages like XLI, you must first put the numbers on the stack, then add, multiply etc.

To illustrate the matter, we will look at the above example once more, showing the stack content for each operation:

```
----------------------------------------
CODING  5   8   8 MUL ADD   3 SUB
----------------------------------------
                8
            8   8  64       3
STACK   5   5   5   5  69  69  66
----------------------------------------
```

Each time you put a number in the coding, it will be put on top of the stack. The MUL and ADD functions use the numbers on the stack and place the result there as well.

So you do not really need to worry about what to do with the result. Just leave it on the stack till you need it next time.

You cannot overload or empty the stack, it is actually cir-cular. It is your own responsibility, that the stack holds the numbers you need. On the other hand, you do not need to cleanup waste, when you are finished.

The size of the stack is 64. So you may not put more than 64 numbers on the stack before the first numbers will be over-written. This will rarely be any problem.

The XLI interpreter has a facility called the DEBUGGER. With the debugger switched on, you may check your coding step by step and check the contents of the stack. The debugger prints a bit differently than in the example shown above:

```
DEBUG

5       5
8       8       5
8       8       8       5
MUL     64      5
ADD     69
3       3       69
SUB     66
```

We will use this format for the examples in the remainder of this chapter to conform to the DEBUG printout style.

MEMORY CELL ARRAY

This is an alternative to storing numbers on the stack. Because the stack is limited in size and is constantly changing, it will often be difficult to remember, how "deep down" the stack, your numbers are placed.

There are 3000 memory cells in XLI numbered 0-2999. There you may save numbers and calculation results for later use.

The two codes STO and RCL are used to store and recall numbers from the memory array.

There are three main uses for the memory cell array:

1) store results, for example planetary positions etc.

2) create tables, for example the rulers of the 12 signs:

```
 5  1 STO     ;Mars    (5)  rules Aries       (cell 1)
 4  2 STO     ;Venus   (4)  rules Taurus      (cell 2)
 3  3 STO     ;Mercury (3)  rules Gemini      (cell 3)
 2  4 STO     ;Moon    (2)  rules Cancer      (cell 4)
 1  5 STO     ;Sun     (1)  rules Leo         (cell 5)
 3  6 STO     ;Mercury (3)  rules Virgo       (cell 6)
 4  7 STO     ;Venus   (4)  rules Libra       (cell 7)
10  8 STO     ;Pluto   (10) rules Scorpio     (cell 8)
 6  9 STO     ;Jupiter (6)  rules Sagittarius (cell 9)
 7 10 STO     ;Saturn  (7)  rules Capricorn   (cell 10)
 8 11 STO     ;Uranus  (8)  rules Aquarius    (cell 11)
 9 12 STO     ;Neptune (9)  rules Pisces      (cell 12)
```

3) store parameter constants.

You may program a printout of aspects etc, using certain orb limits and a certain set of planets.

You may of course insert the orb limits etc. directly into the coding where needed. But if you store them in memory cells at the very start of the module, and recall them when needed in your coding, it is very easy to make modifications, because you do not need to search for the insertion place(s), you need just to change one number at the module start.

STO and RCL are used intensively throughout the utility modules in this toolkit.

INTEGER NUMBERS

The mathematics in XLI works on integers only. This means, that you cannot handle fractional numbers. The calculation:

```
5 2 DIV      (divide 5 by 2)
```

will produce 2, not 2.5. This is important to remember.

The integers may range from -32768 to +32767. These limits may seem strange if you do not know about binary numbers. But don't worry, it will rarely be any problem. If you try to add 1 to 32767, you will actually get the result -32768. So when the limit is reached the numbers start happily over from the other end. Of course this result is wrong and you should normally prevent your coding from producing numbers of that size.

CALCULATING MIDPOINTS AND ANTISCIONS

(THE INTANG UNIT)

The way integer overflow is handled as described above is actually exploited in PCA. Zodiacal positions are normally 0-360 degrees, and exceeding 360 degrees you will just start over from 0.

The intang unit is a way of holding angles using the full integer range, so that overflow in both directions are auto-matically handled. Such angles may be multiplied (harmonics) added and subtracted, never worrying about the problem of exceeding the circle.

A number of XLI functions use the intang unit, and of course there are also functions to turn them back into a readable format. One intang unit equals approximately 20 seconds of arc (19.77) which is better than the accuracy of PCA.

Here is an example showing how to calculate midpoints using intang units, thus avoiding the problem of finding the correct one of the two midpoints (the one on the shortest arc):

Say that you wish to find the Mars Saturn midpoint:

```
5 PPOS            ;Mars intang position
7 PPOS            ;Saturn intang position
2 DUP SUB 2 DIV   ;half of shortest arc, signed
ADD               ;correct midpoint obtained
```

Do not worry too much about the details, just note that no limit tests are needed.

Next example shows a midpoint aspect test: Check if a third planet (Sun) is placed within a certain orb on the midpoint axis (of Mars/Saturn):

```
5 PPOS          ;Mars intang units
7 PPOS          ;Saturn intang units
ADD             ;Sum
2 DIV           ;Half-sum (midpoint axis)
1 PPOS SUB      ;Angle between Sun and above axis (orb)
2 MUL 2 DIV     ;Trick to include both 180 and 0 degrees
ABS             ;Orb must be positive
ITOM            ;convert intang to minutes of arc
60 >            ;compare and check if less than 1 degree
```

If you want also 90 degree aspects to the midpoint axis, just change the line: `2 MUL 2 DIV` to `4 MUL 4 DIV`.

If you have ever tried to program these calculations using floating point (decimal) arithmetics, you will admit, that the concept of intang units simplifies the matter.

Antiscions (mirror points) are very easy to calculate. To calculate the mirror of Mars, just code:

```
5 PPOS CHS     ;change sign
ITOM           ;if needed, convert to minutes
```

You may also mirror planets in other axis. Say that you want Mars mirrored in the nodal axis:

```
5 PPOS         ;Mars
11 PPOS SUB    ;change origin to Node
CHS            ;mirror
11 PPOS ADD    ;change origin back to 0 Aries
ITOM           ;if needed, convert to minutes
```

The ITOM converts the intang units to a positive angle in minutes. In some cases you may want a positive or negative value of 0 to +-10800 (+-180 degrees). In that case, just use ITOMS instead of ITOM.

For example if you want to calculate an aspect angle, subtrac-ting two planets, for example Sun-Mars:

```
1 PPOS 5 PPOS SUB     ;Mars-Sun angle (intang units)
ITOMS ABS             ;0-10800 positive
```

So the aspect angle will be the shortest angle between the two planets, and positive.

Also if you calculate signed orbs or declinations and latitu-des, they are +- values rather than 0-360.

WRITING SIMPLE INTERPRETATIONS

As we have seen, the XLI paragraph consists of four parts: $-line, coding, heading and text.

When writing interpretations, these parts have the following meaning:

1. $-line paragraph delimiter
2. coding astrological rule
3. heading astrological explanation
4. text interpretation text

The heading is not mandatory, but it provides a means of allowing the end user to choose if he wishes an astrological explanation put into the interpretation text or just want the plain text printed.

Here's an example of a very simple interpretation paragraph:

```
$
8 PHS 2 =

*Uranus in the 2. house

Ups and downs in fortune, unsettled state, gains through discoveries, mechanism,
banks, railways, music and through electricity.
$
```

The code line is the way to tell the PCA interpreter, that it should inspect the latest calculated chart to see, if Uranus was in the 2. house:

```
8 PHS          ;fetch the actual house position of Uranus
2              ;stack the desired house position
=              ;compare the two numbers (positions)
```

If the two numbers are equal, i.e. if `8 PHS` equals `2`, the result will be `1`. In XLI `1` means "true" and `0` means "false". If the result is true the text part will be printed, if false it won't.

The heading is the way to tell the reader (not the computer), which rules have been used. This will be printed together with the text part, if the interpretation was called with the `OI` option from the PCA main menu (rather than just I).

You may leave out the heading or just put the explanation into the text itself, the interpretation will function anyway, but in that case the end user will not be able to have it printed optionally. The heading may be any number of lines and is terminated by a blank line.

The text part may be any number of lines and is terminated by the $-line starting the next paragraph.

CODING EXAMPLES FOR INTERPRETATIONS:

If you call the EZZ toolkit editor (see the chapter on using and writing interpretations) and look into the interpretation skeletons, you'll find a lot of useful examples of coding basic astrological rules. When you choose a rule for editing, you'll see a window at the top of the screen showing the coding for that rule.

Here is a complete list of the codes referencing the stored chart(s):

Simple codes:

PSI	planet in sign
PHS	planet in house
RX	planet retrograde
HSI	house in sign
HRU	house ruler
APOW	aspect power
ANUM	aspect number
AORB	aspect orb
PDEG	planet position (degrees)
BDPR	print birth data
NPR	print name

Advanced codes

AZP	aspect test primitive
PPOS	planet position (intang unit)
HPOS	house position (intang unit)
PV	planet speed (intang unit)
HV	house speed (intang unit)
PLA	calculate single planet (intang unit)
XPLA	retrieve additional planetary information
HOUSE	calculate single house (intang unit)
JD	get julian date
DDIF	find difference between two dates
BHUS	house position (universal)
KUN	get 3 closest Kündig sections

Examples using the advanced codes are given later.

Now let us look at some examples of common coding tasks:

```
2 PSI 10 =     ;Moon (2) in Capricorn (10)
6 PHS 11 =     ;Jupiter (6) in 11.th house
3 RX           ;Mercury (3) retrograde
7 RX           ;Saturn (7) retrograde
6 HSI 8 =      ;6.th house cusp in Scorpio (8)
1 HSI 12 =     ;Ascendant (1) in Pisces (12)
14 PSI 12 =    ;Ascendant (14) in Pisces (12)
```

The last two examples are actually testing the same thing. The ascendant is house no. 1 and "planet" number 14. Six of the houses have "planet" numbers, so they can be tested using the planet rule codes.

For a list of the numbers of planets, houses, aspect numbers etc, see the function reference section, where these numbers are listed in tabular form.

Note that the results above are all YES or NO: Retrograde or not retrograde, equal to or not equal to etc. YES (`1`) or NO (`0`) results are used when a text must be printed or not printed. The code `2 PSI` produces a result

between 1 and 12. To turn it into a YES or NO you must test this result againts a fixed number using = (equals) > (greater than) < (less than) or IN (member of a selection).

More examples:

```
4 HRU 6 =      ;Jupiter rules 4.th house
4 HSI RUL 6 =         ;Same result as above
3 PSI RUL 8 =         ;Uranus dispositor for Mercury
2 6 ANUM 3 =          ;Moon square Jupiter
2 6 ANUM 4 =          ;Moon trine Jupiter
1 7 ANUM 1 =          ;Sun conjunct Saturn
1 7 AORB ABS 10 < NOT ;Sun Saturn aspect orb 1 degree
```

The aspect examples do not care about the order of the aspec-ting planets, so you may write 2 6 ANUM or 6 2 ANUM as you like.

Note that AORB calculate aspect orbs to one tenth of a degree, so 1 degree=10, 2 degrees =20 etc. AORB is signed, so you must change it to a positive value (using ABS) before the test.

For these aspect tests to work, they must be within the orb limits given in the PCA orb installation and main menu, i.e. they should be included when you press A from the main menu to print the aspects.

```
14 5 APOW 8 < ;ASC in close aspect to Mars
```

The APOW (aspect power) ranges an aspect from 0 (just at the orb limit) to 10 (exact). So in this example it must be nine or ten (8 is less than the power) The orb limit is the one, you entered in the PCA menus. Actually, it is similar to the thickness of the aspect lines in the drawn chartwheel.

```
8 PDEG 154 =  ;Uranus between 4-00 and 4-59 in Virgo
```

This may be used for degree astrology. You may also do some arithmetic and find sign, decanate etc from the PDEG code. It works only in whole degrees, though. Later we will describe the code PPOS, which is more accurate working in intang units.

```
NPR BDPR      ;Print the natives name and birthdata
```

NPR prints the name (1 line used), and BDPR the data using 5 lines.

ARITHMETICS

The above examples are all very basic single astrological testing rules. The XLI toolkit provides a number of math functions, so that you may combine rules, count points, planets etc etc..

COMPARING

>	greater than
<	less than
=	equal
<>	not equal
NOT	not

In the above examples we have often used the = (equals) sign to test for certain house numbers or signs. Another example showed the use of < (less than) to see if a power was above a certain value.

To know if a text paragraph must be printed or not, you need an end result of YES of NO, not a value. Having found the house number of Jupiter, you will need 12 texts, one for each house, and each text paragraph must have the Jupiter house tested if it is 1, 2, 3... etc.

You may ask why >= (greater than or equal) and <= (less than or equal) functions are missing in the above list. Correct, they are missing. But you may construct them from the others. For example to test if Jupiters house position is greater than or equal to 7:

```
6 PHS 7 > NOT         ;Jupiters house not less than 7
6 PHS 6 <             ;Jupiters house greater than 6
6 PHS 7 < 6 PHS 7 = OR       ;Jupiters house greater than or equal to 7
```

The code NOT changes YES (1) to NO (0) or vice versa.

COMBINING RULES

Say that you wish to test if either the Sun or the Moon is in the tenth house:

```
1 PHS 10 =     ;Sun (1) in house 10
2 PHS 10 =     ;Moon (2) in house 10
OR             ;either ?
```

The OR code combines the two conditions (YES/NO results) already done. As usual, you must have the two results first before you use the OR code.

You could also have asked if the Sun and Moon are BOTH in the 10. house:

```
1 PHS 10 =     ;Sun (1) in house 10
2 PHS 10 =     ;Moon (2) in house 10
AND            ;both ?
```

The codes OR and AND combines two rules to "either" and "both".

You may expand the number of conditions by adding lines. Say that you want to test if Mars is in a mutable sign:

```
5 PSI  3 =     ;Mars (5) in Gemini      ( 3)
5 PSI  6 =     ;Mars (5) in Virgo       ( 6)
OR             ;either
5 PSI  9 =     ;Mars (5) in Sagittarius ( 9)
OR             ;either
5 PSI 12 =     ;Mars (5) in Pisces      (12)
OR             ;either
```

Note the seemingly uneven placement of OR. The first OR tests "either" in Gemini or Virgo, the next "either" tests the first result or Sagittarius, and the last test the second result or Pisces. This must be done, because OR can only test two condi-tions at a time. You may use other setups, which will produce the same result:

Version 2

```
5 PSI  3 =     ;Mars (5) in Gemini      ( 3)
5 PSI  6 =     ;Mars (5) in Virgo       ( 6)
OR             ;either
5 PSI  9 =     ;Mars (5) in Sagittarius ( 9)
```

```
5 PSI 12 =     ;Mars (5) in Pisces      (12)
OR             ;either
OR             ;either
```

Version 3

```
5 PSI  3 =     ;Mars (5) in Gemini      ( 3)
5 PSI  6 =     ;Mars (5) in Virgo       ( 6)
5 PSI  9 =     ;Mars (5) in Sagittarius ( 9)
5 PSI 12 =     ;Mars (5) in Pisces      (12)
OR             ;either Sgr or Psc
OR             ;either above or Vir
OR             ;either above or Gemini
```

Note that in the last version you put 4 conditions on the stack and do the combining afterwards. Only the last `OR` will reach down to the bottom and fetch the first (Gemini) condition.

Remember also, that using a lot of `OR`'s will expand the pro-bability of a final YES, and using a lot of `AND`'s, your final result will most often be NO.

COMBINING "AND" AND "OR"

Say that you wish to test if both Sun and Moon is in water signs:

```
1 PSI 4 =      ;Sun in Cancer
1 PSI 8 =             ;Sun in Scorpio
1 PSI 12 =     ;Sun in Pisces
OR OR          ;either of above = Sun in water
2 PSI 4 =             ;Moon in Cancer
2 PSI 8 =             ;Moon in Scorpio
2 PSI 12 =     ;Moon in Pisces
OR OR          ;either of above = Moon in water
AND            ;both Sun in water and Moon in water
```

TESTING ELEMENTS ETC.

Testing for water signs, mutable signs etc are a bit tedious, because you must test each single sign. There are a few tricks you may use:

```
2 PSI 0 4 8 12 IN     ;Moon in water
```

The code `IN` will check your Moonsign against a set of possibi-lities in one go. The result of the `IN` is always YES or NO, so you do not need the equals sign here. There are a few things to remember using `IN`:

1) You must ALWAYS start with a zero before the values to test against. The zero works as a delimiter so that `IN` can know how many values back in the row to use. So you cannot use `IN` to test equal to zero. Luckily signs, houses and planets are numbered 1 upwards.

2) The values after the `0` must be in the range `1-16`.

The other trick testing elements is purely mathematical:

```
4 PSI 4 MOD 1 =       ;Venus in fire
4 PSI 4 MOD 2 =       ;Venus in earth
4 PSI 4 MOD 3 =       ;Venus in air
4 PSI 4 MOD 0 =       ;Venus in water
```

The MOD (modulus) function calculates the remainder of the division with 4, which will be 0,1,2 or 3 then starting over. In fact this will usually be the simplest and most efficient way, but you may find it less obvious to perceive.

COMPARING TWO CHARTS

PCA have two sets of stored positions for charts, one for radix, and one for all other charts.

If you want the positions of both for comparison, you will need a means of choosing which of the two memory blocks to access:

Planet 1-18: Latest calculated chart (any kind)
Planet 21-38: Latest calculated radix chart

House 1-12: Latest calculated chart (any kind)
House 21-32: Latest calculated radix chart

As you can see, you just add 20 to the planet number to get the radix. So if latest calculation was for example a progressed chart, the progressed Moon will have number 2, and the radix Moon will have number 22.

The following examples show which codes this will work for assuming a progressed chart as the latest calculation:

```
34 PSI 5 =           ;Radix ASC (34) in Leo (5)
14 PSI 5 =           ;Progressed ASC (14) in Leo (5)
29 PHS 8 =           ;Radix Neptune in 8. radix house
9 PHS 8 =            ;Progressed Neptune in 8. progressed house
6 HSI 4 =            ;6th progressed house cusp in Cancer
26 HSI 4 =           ;6th radix house cusp in Cancer
11 PDEG 316 =        ;Progressed Moon's node between 16.00 and 16.59 Aqr
27 RX                ;Radix Saturn retrograde
7 RX                 ;Progressed Saturn retrograde
23 HRU 5 =           ;Mars rules radix 3rd house
3 HRU 5 =            ;Mars rules progressed 3rd house
2 33 ANUM 2 =        ;Progressed Moon opp. radix MC
2 13 ANUM 2 =        ;Progressed Moon opp. Progressed MC
1 5 AORB ABS 20 >    ;Progr. Sun aspect Progr. Mars orb<2 deg.
1 25 AORB ABS 20 >   ;Progr. Sun aspect radix Mars orb<2 deg.
21 25 AORB ABS 20 >  ;radix Sun aspect radix Mars orb<2 deg.
1 21 APOW 10 =       ;Progr. Sun aspect radix Sun exact
```

Further this planet numbering will work on the more advanced codes PPOS, HPOS, PV and HV. One example will be given here though. When comparing charts, it is often important to test which PRO-GRESSED house a RADIX planet is in and vice versa. So even if we do not explain it in detail, here are some coding examples for this problem:

```
24 PPOS 2 BHUS 4 =   ;radix Venus in progr. 4th house
4 PPOS 1 BHUS 6 =    ;progr. Venus in radix 6th house
31 PPOS 2 BHUS 1 =   ;radix node in progr. 1st house
```

As you can see, you must use two codes: `PPOS` and `BHUS`. The first number is the planet (using the expanded numbering), and the second number between the two codes is `1` for radix house and `2` for progressed house.

This may of course be used for synastry as well, if the latest calculation was a 2.nd radix (option `T` in the PCA main menu). Then the above lines would mean e.g. "John's Venus in Susan's 4th house", "Susan's Venus in John's 6th house" etc.

COUNTING, LOOPING AND BRANCHING

FOR	start of FOR-loop
CNT	get FOR-loop counter
NEXT	end of FOR-loop
XIF	exit FOR-loop prematurely
ENDIF	End of if-clause
IF	Start of if-clause
ELSE	Alternative part of ef-clause
NFN	set file branch index
WAIT	withhold heading
CONT	skip withhold

You will often need to repeat the same operation a number of times. For example, you may wish to count the number of planets in water signs. This could of course be done the hard way:

```
 1 PSI 0 4 8 12 IN
 2 PSI 0 4 8 12 IN ADD
 3 PSI 0 4 8 12 IN ADD
 4 PSI 0 4 8 12 IN ADD
 5 PSI 0 4 8 12 IN ADD
 6 PSI 0 4 8 12 IN ADD
 7 PSI 0 4 8 12 IN ADD
 8 PSI 0 4 8 12 IN ADD
 9 PSI 0 4 8 12 IN ADD
10 PSI 0 4 8 12 IN ADD
```

For each line coming out TRUE (`1`) the result will be in-cremented. Now if you start putting a null result (on the stack) and add the others, you'll get this slightly changed setup:

```
 0
 1 PSI 0 4 8 12 IN ADD
 2 PSI 0 4 8 12 IN ADD
 3 PSI 0 4 8 12 IN ADD
 4 PSI 0 4 8 12 IN ADD
 5 PSI 0 4 8 12 IN ADD
 6 PSI 0 4 8 12 IN ADD
 7 PSI 0 4 8 12 IN ADD
 8 PSI 0 4 8 12 IN ADD
 9 PSI 0 4 8 12 IN ADD
10 PSI 0 4 8 12 IN ADD
```

Now you have ten lines, with nearly the same coding, exept for the first number which increments one for each line, i.e. counts from 1 to 10.

You may setup a COUNTING LOOP to do this job more elegantly:

```
0
1 10 FOR
1 CNT PSI 0 4 8 12 IN ADD
NEXT
```

The 10 lines are now replaced by just one line using a counter for the planet number. To make the counter count, these two lines were added:

```
1 10 FOR        ;starts the counting loop
NEXT            ;loops back until end of count
```

The line(s) in between will go into action 10 times. The `1 CNT` will use the incrementing count value (`1-10`). The end result will be exactly the same as in the first example, not using a counter. It will not process faster though, actually rather slightly slower, but it saves you a lot of coding.

Counting loops may count up (`1 10 FOR`) or down (`10 1 FOR`), you may also count using negative values (`-7 8 FOR`). You may easily put the `NEXT` code several paragraphs further on, so that a lot of paragraphs will execute inside the loop. It only has to be in the same file. For example:

```
$
1 3 FOR
1 CNT NUMS

This is line #
$
NEXT

$
```

This will print the following:

```
This is line 1
This is line 2
This is line 3
```

The `NEXT` code is here placed in the following paragraph, so that the text will be printed for each count. Don't worry too much about the `NUMS` code. This is used to print the counter and will be discussed later.

You may setup several (up to 10) counting loops inside each other. For example, if you wish to count the aspects from all planets to all planets, you will need two planet counters: The first should count from Sun to Neptune, the next should count from the current first counter to Pluto. If this is not immedi-ately clear, try to setup a table: Sun-Moon, Sun-Mercury,.... Neptune-Pluto , and check the changing planet numbers.

This example will count the squares in the chart:

```
0                   ;no squares
1 9 FOR             ;outer loop
1 CNT 10 FOR        ;inner loop
1 CNT 2 CNT         ;planet 1 and planet 2
ANUM                ;aspect number
3 =                 ;is it a square
NEXT                ;end inner loop
NEXT                ;end outer loop
```

Note, that `1 CNT` is the first counter, `2 CNT` is the second counter. They are numbered in the succession they are created, i.e. the outermost counter will be number 1, and in this example `2 CNT` will be the innermost. When using "nested FOR--loops", that is several loops inside each other, be well aware which counters you are using.

Counting loops are used extensively in the XLI user modules.

IF-ENDIF

Sometimes you must have something calculated or written out, only on a certain condition. Say for instance, that you are setting up a counting loop for the 10 planets and the two angles MC and ASC. So counting from 1-14, you wish to skip the Moon's node (11) and the part of fortune (12). The solution is:

```
1 14 FOR              ;setup complete coung
1 CNT 0 11 12 IN NOT IF     ;check range
....                  ;your code here
ENDIF
NEXT
```

The dotted line should be replaced by your aspect tests or whatever. The range check first looks if the loop counter is equal to `11` or `12`, then reverses this condition to "not equal" to `11` or `12`. If the latter is the case the following lines will be executed, if the counter in fact equals the unwanted values, the lines will be skipped until the `ENDIF` code.

If you have programmed in other langauges, you may look for an ELSE code, so that you may add other lines of code executing for the node and the part of fortune. There is no ELSE code in XLI, you must do a trick to get that effect:

```
1 14 FOR              ;setup complete coung
1 CNT 0 11 12 IN NOT  ;check range
1 DUP IF                    ;IF not 11 or 12
....                  ;your code here
ENDIF NOT IF          ;ELSE, that is if 11 or 12
....                  ;your code here
ENDIF
NEXT
```

There is one warning about this construction: If you produce a result in the first set of lines, the second `IF` construction will not work correctly. That is because the result of the first `IF` test is duplicated for use in the second test. So that result must be held immediate accessible during the first set of lines.

The `ENDIF` doesn't have to be placed in the same paragraph, but must be within the same file.

You may have several `IF`-`ENDIF` constructions within each other (nested):

```
IF
IF
IF
ENDIF
ENDIF
ENDIF
```

There must be exactly the same number of `ENDIF`'s as `IF`'s. The first `IF` corresponds to the last `ENDIF` etc.

WARNING: Often you will mix `FOR-NEXT` loops and `IF-ENDIF` con-structions. But never make them overlap:

Correct:	WRONG!	Correct:	WRONG!
FOR	FOR	IF	IF
IF	IF	FOR	FOR
ENDIF	NEXT	NEXT	ENDIF
NEXT	ENDIF	ENDIF	NEXT

XIF

This special code will exit a `FOR-NEXT` loop prematurely on a certain condition. An example of this is given in the section of creating menus, and in the example of listing graphic transits.

FURTHER ARITHMETIC

MATH:

BOO	integer to boolean
ABS	absolute value
CHS	change sign
ADD	add
SUB	subtract
MUL	multiply
MULT	multiply by fraction
DIV	divide
MOD	modulus
DIVR	divide with remainder
MAX	take maximum of two values
MIN	take minimum of two values

STACK:

DUP	duplicate value on stack
FETCH	fetch value on stack
ZZO	set zero offset on stack
GET	get number on stack (fixed offset)
PUT	put number to stack (fixed offset)
XY	exchange numbers on stack
INC	increment stack pointer
DEC	decrement stack pointer

As soon as you wish to do something more than just simple tests and counts, you will need to perform calculations. We will only comment a few of the math and stack functions here. If you need something, look at the list and refer to the FUNCTION REFEREN-CE for a specification how to use any of these functions if you think it will serve your needs.

What we will do here is to mention some common problems and how to solve them.

CHANGING INTANG UNITS TO DEGREES MINUTES AND SIGN

A number of the advanced astrological functions produce results in intang units. We will show how to print such values in a readable format:

```
3 PPOS          ;Mercury position (intang)
ITOM            ;convert to minutes of arc
1800 DIVR       ;convert to sign and minutes
1 ADD           ;renumber signs to start with 1
XY              ;exchange sign and remaining minutes
60 DIVR         ;convert minutes to degrees and minutes
NUMS            ;prepare printout

Position is degrees: ##  Minutes: ##  Sign no.: ##
```

The `NUMS` will print the total 3 results produced by the calculation in the text field shown. For each template (`##`) the last result on the stack will be removed and printed. Therefore the results must be calculated in reverse order (the first number to print must be calculated last).

Now to the arithmetics: The DIVR produces two results: the remainder and the quotient of the division. Say, that you have a position of 9-28 Leo. This is 7768 minutes total. So we will repeat the calculation with this example showing the stack:

```
3 PPOS          23569
ITOM            7768
1800 DIVR       4       568
1 ADD           5       568
XY              568     5
60 DIVR         9       28      5
```

The result(s) are thus `9 28 5` which can then be printed using the `NUMS` code.

As you can see from for example `568 60 DIVR`, that the remainder (`28`) is calculated first, then the quotient (`9`) is placed on top. The sign number remains untouched beneath the following calculations and results.

It may be necessary for you to draw a sketch of the stack as shown above to be sure, that you get it right. You may test your coding on screen by adding the DEBUG code, then calling your test module from PCA. DEBUG will then display you the actions on the stack step by step as shown above.

MULT - SCALING YOUR VALUES:

The integer arithmetic in XLI have certain serious limitations.

Say for instance, that you wish to change minutes of arc back to intang. No function in the XLI provides this facility. What you really need is to divide your minutes with 21600 and multiply by 65536.

But dividing by 21600 will produce 0 instead of a decimal number between 0 and 1. And if you try to multiplying with 65536, you will get a serious overflow problem, because the integers do not range further than 32767. You may reduce the fraction, e.g. using a quarter circle instead: multiply by 16384 and dividing by 5400.

The `MULT` code lets you multiply and divide in one go, keeping the highest accuracy possible:

```
16384 5400 MULT
```

You cannot write `65536 21600 MULT`, because the number `65536` does not exist in the XLI, the allowable range is `-32768` to `+32767`.

You will often need this code if you plot graphs. The XLI can turn the system into graphics mode and draw instead of print. Positioning the drawing pen will often need scaling.

MAX AND MIN

Say that you calculate element strength, and wish the result to be within certain limits, for example -99 to 99, so that the value -99 means -99 or less, and the value 99 means that the value is 99 or more. After having calculated your points, the following coding will do just that:

```
-99 MAX 99 MIN
```

Another use may be to find the strongest of a number of ran-gings. Say that you have calculated points for the four ele-ments and put them into the storage cells 1-4.

One question may then be: what is the strongest element power. The answer is produced by the following coding:

```
1 RCL 2 RCL MAX 3 RCL MAX 4 RCL MAX
```

Another question may be: Which element is strongest?

```
1               ;assume fire
1 4 FOR         ;setup counting loop
1 CNT RCL       ;get current element
2 DUP RCL       ;get strongest found till yet
< IF            ;if the current is stronger
DEC 1 CNT       ;exchange with current count
ENDIF           ;end condition
NEXT            ;end loop
```

The result will be `1-4` (fire-water). If the strongest point score is shared by more elements, the first of these is chosen by the routine.

Note the code `DEC`. This will dispose the current result. So the construction `DEC 1 CNT` removes the till yet strongest element number and inserts the current instead.

POWERCONTROLLING PCA

MEGET	get value from main menu
MEPUT	put value to main menu
NPUT	put name to main menu
PSTAT	change printer status or suppress all output
FUNX	call single PCA main menu option
CML	execute defined macro (destructive)
CALL	call defined macro (non-destructive)

One of the strong features of the XLI interpreter is, that PCA may be operated by a file instead of interactively by you.

Even without XLI you have the macros, which can execute complex tasks from one keypress. A macro can also call the XLI interpreter.

But the XLI interpreter can also in turn set up its own macros and run them. You are not allowed though to let the XLI use a macro to call XLI once more, that will not work.

Ultimately you could have another program running, which created a PCA XLI file, started PCA automatically to calculate planets, ephemerides, charts, chartwheels etc, storing the results on disk, and finally returning to the calling program, which could then pick up the files and retreive the results.

A typical use of the macro call in XLI is found in the PSETUP module. Here you choose a printer type from a menu, and according to your choice, a macro is executed, which calls the installation menu, "presses keys" and inserts the correct values for that printer including the printer initial string.

There are two codes calling commandlines. The `CALL` is normally the one to go for. It will execute immediately. Further it will work properly, even if the interpreter was itself called with a macro. For example, it is possible to enter a macro from the keyboard, which amongst other things calls the PSETUP module. So you could automatically change printer or printer port (using the `PORTX` module) by entering these calls into a macro.

The other call is `CML`. This is not as useful as `CALL`. It will in fact replace an existing macro with a new one, and it will not execute until after the XLI module has finished. So say you inserted first a `CML` and then a `CALL` in your XLI file. The `CALL` would (surprise) execute first, leaving `CML` sleeping and undisturbed. When the XLI module finish, the `CML` will then continue. But remember, that `CML` will destroy any previous macro.

The calls are coded like this:

```
$
CALL

RAV
$
```

Note that the macro itself is put into the text field of the paragraph, not in the coding. It has to be the first line in the text field. Normally, you should not put real printable text into such a paragraph.

CONTROLLING OUTPUT

Running complex jobs, you may not always need everything printed on the screen. For example, you may need the positions for a radix and a progressed chart for further calculations and display. This could for example be done like this:

```
$
CALL

R1T.P1R.
$
```

This will run the radix using the current menu data, then get todays date from the internal clock (`1T.`), run the progressed and finally restore the radix time (`1R.`). But say, that you have some display or print running in your XLI module, that you would not like to be disturbed by the printout of those two charts. The answer is simple:

```
$
CALL

(R)1T.(P)1R.
$
```

Just put parenthesis around the calculations, you do not wish to have displayed or printed.

There exist a more powerful XLI code: PSTAT, which may turn on or off the screen or printer, and reset the printer status to its previous state, whatever it was. See the function reference section for a precise description.

MANIPULATING MENU INPUT DATA

You may directly access the input data in the PCA main menu. This is done by the MEGET and MEPUT codes. Every bit of birth data in the input menu has a number, for example:

```
0 MEGET;fetch day
1 MEGET;fetch month
2 MEGET;fetch year
3 MEGET;fetch BC flag
```

A complete list is given in the function reference. You may then have the XLI input its own data, run charts etc. Say for instance, that you would like to do the progressed chart for 5 years onwards from the current:

```
$
0 19 FOR        ;save the current menu data
1 CNT MEGET     ;get an item
1 CNT STO       ;store in memory cell
NEXT
CALL            ;call (invisibly) radix, then enter current
;               ;time.

(R)1T.
$
2 MEGET;current year
1 DUP 4 ADD     ;end year
FOR             ;count the five years
1 CNT MEPUT     ;insert the year
CALL            ;show progressed chart

P
$
NEXT            ;loop back
0 19 FOR        ;restore the original data
1 CNT RCL       ;fetch from memory cell
1 CNT MEPUT     ;put back to menu
NEXT

$$$
```

Here we have used another method of saving and restoring the original menu data. Of course, in this case you could as well just have called a macro of 1R, which would have resto-red the data of the last radix. But in other cases, for examp-le, if you wish to run other radixes in between, you must use this more safe method. Even this method may not work, however, if you interrupt the XLI prematurely pressing ESC and returns to human control before the module has finished.

Name insertion into the input menu can be done using a macro call:

```
$
CALL

0BEVERLY.
```

```
$
```

to insert the name BEVERLY, but if the name is manipulated using the string handling routines the `NPUT` code can be used for inserting a name from the string array. See the section of string handling and refer to the function reference for the use of `NPUT`.

The sex field is accessed through `7 MEPUT`/`7 MEGET`.

The house system cannot be accessed through `MEGET` and `MEPUT`. You must use the macro method.

Printer status may partially be manipulated using the `PSTAT` code.

GRAPHIC TRANSITS

NDATE	convert graphic transits sector to date
GTR	call user defined graphic transit
RTA	reset transit aspect list
NTA	get next transit aspect

If you like the graphic transit facility, you may setup other versions with special features using the `GTR` code. You'll find a description of this in the function reference.

Any kind of graphic transits will save the aspects found. These may then be retreived, analysed, displayed or interpreted by an XLI module. The aspects can be retreived one by one in the succession, they appear. For each aspect, the start and end time (approximately), planet numbers and aspect type is avai-lable.

This example will print out the aspects:

```
$
2 CARY

,Sun,Moo,Mer,Ven,Mar,Jup,Sat,Ura,Nep,Plu,Nod,Ptf,MC,ASC
,Cnj,Opp,Sqr,Tri,Sex,ssq,ses,qqx,ssx,
$
1

No    From         To          Aspect:
-------------------------------------------------------------------------------
$
RTA                 ;reset aspect retrieve counter
1 10000 FOR         ;loop all aspects
NTA                 ;get next aspect
1 DUP NOT XIF       ;exit if last one
4 FETCH 20 MOD      ;get radix planet and adjust planet number
5 FETCH             ;get aspect number
15 ADD              ;adjust to aspect name index above
5 FETCH             ;get transit planet (number no change)
5 FETCH NDATE       ;convert end sector to date
7 FETCH NDATE       ;convert start sector to date
1 CNT               ;fetch overall counter
NUMS                ;display all values

####  ## ## ####   ## ## ####  @@@ @@@ @@@
$
```

```
NEXT
1

-------------------------------------------------------------------------------
$$$
```

The first lines are defining the planet and aspect names for printout purposes.

The header and footer lines are always printed, so the coding for this is just `1` (YES).

The `1 10000 FOR` is not counting 10000, it is just to be sure to have enough looping power. The XIF will exit at the last aspect.

The `RTA` resets the aspect pointer, so that the next one retrie-ved will be the first.

The `NTA` will get one aspect off the queue. Now a lot of `FETCH` codes are used. That is because the results must appear in the correct succession for printout, and some of them must be adjusted.

Here is a breakdown of the result stack to show, what happens:

```
NTA             sect1 sect2 radx tran asp
4 FETCH         tran sect1 sect2 radx asp
20 MOD          tran20 sect1 sect2 radx asp
5 FETCH         asp tran20 sect1 sect2 radx
15 ADD          asp15 tran20 sect1 sect2 radx
5 FETCH         radx asp15 tran20 sect1 sect2
5 FETCH         sect2 radx asp15 tran20 sect1
NDATE           d2 m2 y2 radx asp15 tran20 sect1
7 FETCH         sect1 d2 m2 y2 radx asp15 tran20
NDATE           d1 m1 y1 d2 m2 y2 radx asp15 tran20
1 CNT           cnt d1 m1 y1 d2 m2 y2 radx asp15 tran20
```

The names for the result values are quite abbreviated. If you are in doubt what they mean, look at the explanation of the coding above, and if needed refer to the definitions in the function reference part.

This works also with the collective transits, so you may use the above to setup a collective transit list.

Aspects, that repeat during the period will be listed several times. The transit interpretation skeleton has a mechanism for avoiding repetition. The method is quite complex, so don't try to analyze the code, unless you need a real challenge.

SETTING UP YOUR OWN MENUS

MENU	setup screen menu
OPT	wait for keypress from defined set
WKEY	wait for keypress
XIF	exit loop

The ability to setup choice menus means, that you could expand the PCA into a really huge program with a limitless number of choices. The toolkit itself adds the two menus: The user-module menu (`XS`, file TTUS.XLI) and the interpretation brancher menu (`OI`, file TTFLET.TXT).

You may continue this menu-branching as much as you wish, and all the branching will be programmable using the macros.

NOTE: There is one limitation: You cannot use the CALL code to make XLI call other XLI modules. This would overload the XLI interpreter. But you may use the CML code, which executes the macro only AFTER the current module has finished.

Here is a very simple example of a menu choosing between two further XLI modules, A and B:

```
$
MENU

    A  select module A

    B  select module B

    X  back to PCA main menu

$
0 88 66 65 OPT NFN

$$$
A.XLI
B.XLI
```

The MENU code temporarily turns off any printers and puts the text field to the screen. You may write anything in the text part, it works only as a reminder of the possible keypresses.

The OPT code turns on the printer again (if it was turned off) and waits for keypresses. Only the keypresses defined in the list before the OPT code are accepted. The 0 value in the start of the line MUST be present, and just works as a delimiter telling where the list starts. The following values until the OPT code are the numbers of the valid keys. A has number 65, B has number 66 and X has number 88 (the ASCII numbers of the key symbols).

The branching itself is obtained by the NFN code. The result of OPT is 1,2,3,... etc depending on the key codes' position in the list. NFN will preset the branch file pointer to 1st, 2nd, 3rd... filename after the $$$ line.

Note that there is no filename for choice X. This should have been inserted after B.XLI, but is left blank. When XLI finds a blank filename to branch to, it returns to the main menu.

You do not necessarily need file branching to select different things. You could also have checked the OPT result with IF-ENDIF constructions. The PSETUP.XLI module uses this technique. You'll find another example in the GRAD.XLI, where you opt for degrees and minutes on the chartwheel. After the MENU part this coding manages the choice:

```
$
0 66 65 OPT 1 DUP           ;duplicate choice result
1 = IF 1 3 CONFX ENDIF             ;If A pressed, set degrees ON
2 = IF 0 3 CONFX ENDIF             ;If B pressed, set degrees OFF

$$$
```

The option result is duplicated for two tests, result=1 or result=2. If you need to expand the number of tests, you must add one (1 DUP) duplication before each test line except the last, to keep one copy of the result available.

NOTE: This is a very primitive menu developed for DOS. With Argus, you can use MENUX to create more interesting menus with buttons and mouse clicks.

DISPLAYING CHOICES IN THE MENU

The `MENU` mechanism is not elaborate enough to let you enter values or strings, but you may enter choices and have them displayed. You'll find one example in the interpretation brancher (choice `OI` from the PCA main menu): If you press the E key (EDITOR), you will have the word EDITING displayed at the top. This is done by rewriting the complete menu:

First a loop is constructed, which will display the menu a number of times, each time waiting for either of the possible keypresses. Some of these keypresses will exit the loop, while others may change the menu display. This is done by inserting variable fields into the menu itself, where numbers or strings can be displayed using the `NUMS` code.

A very basic example of this technique is the PORTX.XLI module, which changes the printer and plotter port redirection:

```
; 0 1 CONFX : printer port = lpt1: (default)
; 1 1 CONFX : printer port = lpt2:
; 2 1 CONFX : printer port = com1:
; 3 1 CONFX : printer port = com2:

; 0 2 CONFX : plotter port = lpt1:
; 1 2 CONFX : plotter port = lpt2:
; 2 2 CONFX : plotter port = com1: (default)
; 3 2 CONFX : plotter port = com2:

$
1 CARY ;define the strings to display in the menu

LPT1,LPT2,COM1,COM2
$
0 1 STO;store the default port for printer
2 2 STO;store the default port for plotter

$
1 10000 FOR    ;setup "endless" loop
NUMS           ;display strings
MENU           ;start menu
2 RCL          ;string for plotter port assignment
1 RCL          ;string for printer port assignment
```

```
        A    Printer port: @@@@@@@@

        B    Plotter port: @@@@@@@@

        C    Insert
```

```
$
0 67 66 65 OPT;get option from keyboard
1 DUP 3 = XIF  ;exit loop if key 3 pressed
1 DUP 1 = IF   ;if A pressed...
1 RCL 1 ADD    ;increment printer port number
4 MOD          ;after value 3 start over from 0
1 STO          ;store back into memory cell
DEC            ;remove intermediate result
ENDIF          ;end A-choice handling
1 DUP 2 = IF   ;if B pressed...
2 RCL 1 ADD    ;increment plotter port number
4 MOD          ;after value 3 start over from 0
2 STO          ;store back into memory cell
DEC            ;remove intermediate result
ENDIF          ;end B-choice handling
NEXT           ;loop back

$
1 RCL 1 CONFX  ;choice C, loop ended, set printer port
2 RCL 2 CONFX  ;and plotter port

$$$
```

The `@@@@@@@@` fields in the menu will display one of the strings defined by the `CARY` code in the start, depending on the content of memory cells 1 and 2. To get them displayed, these two values are put on the stack after the `MENU` code by `2 RCL 1 RCL`.

Each time `A` or `B` is pressed either of the two values is incremented (modulo 4), and the menu redisplayed with the new state.

Pressing `C` will activate the `XIF` code that exits the loop. Then the current chosen port values are inserted.

This is another example using the same technique to input a number. The `MENU` mechanism is not really suited for that job, but as you'll see, even if it is clumsy, it works:

```
$
0 0 STO              ;reset input number to 0

$
1 10000 FOR          ;setup endless loop
NUMS                 ;prepare number display
MENU                 ;setup as menu
0 RCL                ;current number to display

ENTER NUMBER: #########
$
 0 13 57 56 55       ;accept ENTER and digit keys
54 53 52 51 50
49 48 OPT            ;get key
1 SUB                ;set range to 0-9 (and 10 for ENTER)
1 DUP 10 = XIF       ;exit if ENTER pressed
0 RCL 3276 > IF             ;avoid number overflow
0 RCL 10 MUL         ;shift one digit left
```

```
ADD                 ;add new digit
0 STO ENDIF         ;store new value back
NEXT                ;end loop

$
0 RCL               ;display end result
NUMS


RESULT: #########
$$$
```

You could prune this a bit by adding a test for the backspace key to remove one digit, the minus key to change sign (in which case you should expand the range test to avoide negative overflow) etc. It should in theory also be possible to change the coding to input strings using the string array and string handling functions.

Setting up menus with frames and things will make your XLI module look much more professional, even if it does not produce smart mouse-driven pop-up and pull-down menus in colours. The examples and modules in this toolkit are kept very basic, to make them easier to understand.

PRINTING VARIABLE STRINGS AND NUMBERS

NUMS display numbers and strings

We have often in the foregoing seen the `NUMS` code used to print values or names of planets, aspects etc. inserted in the text field.

The text field of the paragraph will normally be fixed, that is, you decide exactly, how it is going to appear, and one printout of the same paragraph will look identical to the next. Used for ordinary interpretation, the text fields are varied from chart to chart only by their succession.

The exception is the variable insertion. A variable is a number or bit of text (a string) which may change depending on for instance the chart. You may for instance have printout of the planetary positions. The planet names will be fixed, but the degrees, minutes and sign name will vary.

```
Jupiter ## ## @@@
Saturn  ## ## @@@
```

The letters will appear as they are, but the `##` and `@@@` fields are TEMPLATES, that will change according to the current results on the stack.

Templates are of two kinds: numeric templates: `####` and string (text) templates: `@@@@`. You enter a template as a continuous row of `#`- or `@`- characters.

You may think of the template as a window or "hole" in the screen or paper, where you can see the current value of changing numbers or text strings.

To use the templates, you must place the code `NUMS` somewhere in the coding for that paragraph. The coding itself must provide the numbers or strings you wish to display. If the coding produces the result `3`, then a `####` template in the text will display the number `3`, and alternatively `@@@@` will display string number 3. Text strings have numbers, which will be explained later.

You may display up till 64 values/strings in one paragraph. Your coding must provide the numbers you need in the reverse succession. You must calculate as many numbers, as you put templates into the text field.

A very simple example:

```
$
2 MEGET          ;fetch PCA main menu year
1 MEGET          ;month
0 MEGET          ;and date

Day: ##  Month: ##  Year: ####
$
```

If the menu date is 23th sep 1944, this paragraph would display:

```
Day: 23  Month:  9  Year: 1944
```

The size of the templates must be big enough to hold all the digits possible. If there are less digits, blanks will be inserted (september is displayed blank-9), if there are more, you will only see the last digits.

We will repeat the same example with the string template used for the month, so that month will appear by name, not by number:

```
$
1 CARY

,jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec
$
2 MEGET          ;fetch PCA main menu year
1 MEGET          ;month
0 MEGET          ;and date

Day: ##  Month: @@@  Year: ####
$
```

The first paragraph defines a numbered row (ARRAY) of text strings, so that we can use the usual month numbers. The comma in the start means, that there is no month number zero.

You may place the three templates in the above example close together:

```
Date: ##@@@####
```

producing something like `23sep1944`.

Of course you cannot do the same with the numbered month version, because that will result in just one long template displaying one number only.

EPHEMERIS GENERATOR

(A complete example of printing planetary positions)

The following example is the ephemeris generator included in the XLI toolkit, here with comments:

The first line defines the sign abbreviations. To be able to output date, also the month names are defined. The signs are defined one-off, so that Aries get number `0`, Taurus number `1` etc. The months will get the numbers `12-23`. So to get the index for signs, subtract one, and the index for a month, add 11.

```
$
```

```
2 CARY

ar,ta,ge,cn,le,vi,li,sc,sg,ca,aq,ps
Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Okt,Nov,Dec
$
```

The next codes enter time and zone into the menu, so that you will get the positions for Noon-GMT. You may of course change this to get the positions for local midnight or whatever.

```
$
12 4 MEPUT            ;hours=12
0 5 MEPUT             ;minutes=0
0 6 MEPUT             ;seconds=0
0 8 MEPUT             ;zone hours =0
0 9 MEPUT             ;zone minutes =0
2 MEGET NUMS          ;put the menu year into the header below

-------------------------------------------------------------------------
EPHEMERIS of planetary positions for year #### at NOON
-------------------------------------------------------------------------
$
```

Set up a counting loop for the 12 months of the year. Print the month name in the month header below:

```
$
1 12 FOR 1 CNT 11 ADD NUMS

@@@      Sun    Moon   Merc   Venu   Mars   Jupi   Satu   Uran   Nept   Plut
-------------------------------------------------------------------------
$
```

A second counting loop is now set up for the max 31 days of each month.

Now the month and day is inserted into the menu using the `MEPUT` code, then use `DNORM` to normalize the date. This means, that for example the 31th of april, which is nonexisting will be changed to the 1st of may.

```
$
1 31 FOR
1 CNT 1 MEPUT
2 CNT 0 STO 0 MEPUT
DNORM
$
```

Now here comes the trick to obtain the unequal number of days in each month: Check if `DNORM` changed the month number, in which case a month overflow is detected and the `IF-ENDIF` will skip calculations and printout for that date.

```
$
0 RCL 0 MEGET = IF

$
```

Now a third counting loop calculates the planetary positions. They are processed in reverse order, because the last result will be printed first.

```
$
10 1 FOR
3 CNT PLA

$
```

The `PLA` code produces 2 results, longitude and speed. To discard the speed, `XY DEC` swaps the two results and removes the topmost. Then the longitude is converted to fit the printout.

There is a trick in the `DEC XY INC`: `DEC` seems to discard the degrees, then `XY` swaps sign and minutes, but the following `INC` restores the minutes again. The `DEC-INC` construction allows `XY` to do the swapping one level down the result stack.

```
$
XY DEC          ;discard speed
ITOM            ;convert intang units to minutes
1800 DIVR       ;convert to sign and minutes
XY 60 DIVR      ;convert to degrees, minutes and sign
DEC XY INC      ;rearrange to degrees sign and minutes
NEXT            ;do all 10 planets

$
```

Finally before printing one line the day and month is fetched, still in reverse order. The printout line have 32 templates total: Day, month and then degrees, sign and minutes for each of the 10 planets. The template line is organized to fit the month header done earlier.

```
$
1 MEGET 0 MEGET
NUMS

## ## | ##@## ##@## ##@## ##@## ##@## ##@## ##@## ##@## ##@## ##@##
$
```

Here ends first the invalid date condition, then the month loop. The `1` (meaning "yes") is put there to force a printout of the dotted line terminating the month printout.

The last `NEXT` terminates the month count.

```
$
ENDIF NEXT 1

-------------------------------------------------------------------------------
$
NEXT

$$$
```

You may change this structure to your personal needs: a Moonphase ephemeris, Hindu positions, local noon positions etc. You may add Julian date weekday number or whatever you like.

Weekday calculation:

To calculate the weekday, you may use the following coding. It will take the menu date, calculate the difference from the 1st of jan 1950 (a sunday) and modulate by 7 to get the weekday. To allow for days before 1950 (negative difference), the `7 ADD 7 MOD` is used to assure a positive result. This routine will work for a time span of approx +- 88 years. Outside this timespan you will get wrong results caused by integer overflow.

A tip if you wish to create a routine for a greater timespan: use `DDIF` on a fixed century, and then use a correction table for other centuries.

```
$
1 CARY

Sun,Mon,Tue,Wed,Thu,Fri,Sat
$
2 MEGET 1 MEGET 0 MEGET      ;get menu date
1950 1 1                           ;compare to fixed date
DDIF                         ;find difference in days
7 MOD                        ;modulate to 0-6
7 ADD 7 MOD                  ;compensate for negative
2 MEGET 1 MEGET 0 MEGET      ;print date
NUMS

## ## #### @@@@@@@
$$$
```

STRING MANIPULATION

CARY	define string array
NAME	get current name into string array
PLACE	get current place into string array
STDEF	define one string in string array
STCAT	concatenate strings
STCMP	compare strings
STCUT	cut slice of string
STLEN	return string length

The XLI toolkit has some ways of handling strings, even if it's not a full blown string handling system. You cannot write your own wordprocessor using XLI!

There is room for only a total of 255 characters. So the number of strings you may create depends on their length. Each string uses one extra character for household. That means, that you may create for example one long string of 255 characters, or 25 strings of 9 characters each (25*(9+1)=250). Defining names for all planets, signs, aspects etc. soon will bring you to the limit. A way around this problem may be to redefine strings on the go as needed.

The definition of strings uses the code `CARY`. The strings them-selves are entered in the text field of the paragraph. You write `1 CARY` if you have one line of string definitions, `2 CARY` if you have 2 lines etc. You decide for yourself how many lines you wish to use. You could just as well have defined the months like this:

```
$
5 CARY

,jan,feb
mar,apr,may
jun,jul,aug
```

```
sep,oct,nov
dec

$
```

The above defines strings number `0-12`. String zero ist just empty. All earlier defined strings are erased by `CARY`.

If you wish to define or redefine for example string `12`, use the code `STDEF` istead of `CARY`:

```
$
12 STDEF

dez
$
```

This will redefine string `12` from `dec` to `dez`.

The `CARY` code is used in many of the utility modules in the toolkit.

There are a number of other string handling: you may cut, join or compare strings. This is used in the TXT2NAME module, where direction flags (`NSEW`) and sex (`MF`) are tested. You'll find more information in the function reference part.

GRAPHICS

GRON	switch to graphics mode
GROFF	switch back to text mode
GRCOL	set graphics colour
PENUP	lift pen
DRAW	draw line
DFAT	draw fat line
DRSYM	draw defined symbol

Graphics look pretty, and a lot of modern software puts its heavy sales arguments on graphic features.

To understand this chapter, you must know, that output may be produced in two quite different modes: text or graphic.

Text output is using a very limited number of letters, digits and other symbols: the character set, which has 256 members. This allows for compact data and fast screen, printer and file handling. An ordinary book may be stored in text mode in say 400 KB of memory. If it was to be stored in graphics in laser print quality, the 400 KB would not even hold a single page.

When using graphics on the computer screen, it must first be switched to graphics mode. In this mode, you are not guaranteed the ability to print text, all letters must be drawn (which takes time and computer resources).

The PCA graphics (normally used for just the chartwheel) have a number of limitations:

1) You may not print text, only draw lines. It is possible though, to draw a very limited set of letters and digits, so that you may label different parts of your graphs.

2) There is a maximum of 12000 single strokes available for each graph. The symbol Cancer uses 60 single strokes, whereas the digit 1 uses just 3.

In an XLI module printout, graphics and text mode may not be mixed, you can have some lines of text, then some "lines" of graphics etc.

Three utility modules in the toolkit use graphics: The hindu square chart, the biorythms and the graphic ephemeris. Also there is a fourth module just drawing the symbols available.

The most interesting module is the graphic ephemeris. It has a number of other interesting XLI features, so we will go through it in detail:

THE GRAPHEPH MODULE:

```
;XLI Graphic Ephemeris utility module
```

First the month names are set up as strings. All text drawn in graphics mode must be defined as text strings:

```
$
1 CARY

,Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
$
```

Saving the menu values have been discussed earlier. To calculate planets, we need only to change the date and time. So only the first 8 positions in the menu are saved. The memory cells used for this are cell 100-107.

Writing modules, you should keep a good record of the use of memory cells, so you do not overwrite important information, and so that you can easily find the information, you need.

```
$
0 7 FOR         ;save menu values
1 CNT MEGET
1 CNT 100 ADD STO
NEXT

$
```

Next part inserts time. You could also instead have done a `CALL` with a macro of 212.

```
$
12 4 MEPUT      ;init time to noon
0 5 MEPUT
0 6 MEPUT

$


;SET USER CONSTANTS
;------------------
```

If you later on wish to change the appearance of the graph, it is a very good idea to have the values saved in memory cells like this instead of putting the values directly into all the different places, where you need them. Here they can be easily found and changed. You could even later expand the module with a menu giving you different choices:

```
$
1024 10 STO     ;set frame width (max 1024)
```

```
600  11 STO    ;set frame height (max 974)
1 12 STO       ;fist planet to display (1=Sun)
11 13 STO      ;last planet to display (11=Node)
2 15 STO       ;horizontal grid option 0,1 or 2
1 16 STO       ;vertical grid option 0 or 1

$
```

Now use `GRON` to change the output to graphics mode. The `3 GRCOL` chooses the colour for the following draw. Colour `3` is the graph foreground chosen in the PCA colour installation menu.

```
$
GRON
3 GRCOL ;draw frame

$
```

The next part will draw the outer frame using the stored width and height values. The coordinates are negative (`CHS`) or positive, because the origin is in the middle of the screen or print area.

The `DFAT` code draws not just one line, but a series of parallel lines. `3 2 DFAT` means: draw 3 lines with a mutual distance of 2 dot units.

```
$
10 RCL CHS 11 RCL CHS 10 RCL CHS 11 RCL 3 2 DFAT
10 RCL CHS 11 RCL 10 RCL 11 RCL 3 2 DFAT
10 RCL 11 RCL 10 RCL 11 RCL CHS 3 2 DFAT
10 RCL 11 RCL CHS 10 RCL CHS 11 RCL CHS 3 2 DFAT

$
```

The next three paragraphs draw the grid. The colours are the ones chosen for angles and houses in the PCA installation menu.

```
$
15 RCL 0 < IF
4 GRCOL        ;draw horz. 5 degree lines
11 RCL 15 DIV 0 STO   ;heigth of one degree
-2 2 FOR                    ;5 lines total
10 RCL CHS            ;horizontal start (x1)
1 CNT 5 MUL 0 RCL MUL ;vertical start (y1)
2 DUP CHS             ;horizontal end (x2=-x1)
2 DUP                 ;vertical end (y2=y1)
1 2 DFAT              ;draw one line
NEXT                  ;repeat for all 5 lines
ENDIF

$
15 RCL 1 < IF
5 GRCOL        ;draw horz. 1 degree lines
-14 14 FOR            ;29 lines total
1 CNT ABS 5 MOD 0 =   ;is it a 5-degree line
NOT IF                ;draw if not
10 RCL CHS            ;horizontal start (x1)
1 CNT 0 RCL MUL             ;vertical start (y1)
2 DUP CHS             ;horizontal end (x2=-x1)
```

```
2 DUP                ;vertical end (y2=y1)
1 2 DFAT             ;draw
ENDIF
NEXT
ENDIF

$
16 RCL 0 < IF
4 GRCOL        ;draw vert. month lines
10 RCL 6 DIV 0 STO   ;width of one month
-5 5 FOR                    ;11 lines total
1 CNT 0 RCL MUL             ;horizontal start (x1)
11 RCL CHS           ;vertical start (y1)
2 DUP                ;horizontal end (x2=x1)
2 DUP CHS            ;vertical end (y2=-y1)
1 2 DFAT             ;draw
NEXT
ENDIF

$
```

Now the names of the months defined earlier with the `CARY` code are drawn. To do this, the code `DRTXT` is used.

```
$
1 12 FOR             ;Draw month names
1 CNT                ;string number
0                    ;rotation=0=vertical
50                   ;size
1 CNT 7 SUB          ;x-position -6 to 5
0 RCL MUL            ;use stored month line scaling
0 RCL 4 DIV ADD             ;move a little right
11 RCL 50 ADD        ;place text 50 units above frame
DRTXT                ;draw month name
NEXT

$
```

Next comes the planet movement curves. Included are the 10 planets and the Moon's node, if you did not change the constants in cell 12 and 13.

```
; DRAW PLANETARY MOVEMENTS
;-------------------------
$
10 RCL 30 DIV 0 STO  ;x-step=width/30
12 RCL 13 RCL FOR    ;count planets
1 CNT 2 = NOT IF            ;exclude Moon (too fast)

$
```

This sets the colour of the planet as defined in the colour installation menu. Planet colours start at no. `17`.

```
$
1 CNT 17 ADD GRCOL   ;set planet colour
```

```
$
```

Counting the year goes in steps of two days. The counter moves from 0 to 180 in steps of one, so this must be multiplied by two and have add one to get the date. This result is entered into the input menu using `MEPUT` even if you may have a silly date like 302nd of january, the `DNORM` function will convert this to the correct day and month.

```
$
0 180 FOR             ;date loop, 2 days step
102 RCL 2 MEPUT       ;get year from saved menu values
2 CNT 2 MUL           ;get date step
1 ADD                 ;add to the 1st of january
0 MEPUT        ;insert day in menu
1 1 MEPUT             ;insert month=1
DNORM                 ;normalize the date

$
```

The `ZZO` code used below is not really necessary here, but shown as an example. If you have a lot of calculations going, it may be difficult to remember how deep down the result stack, a given value resides. `ZZO` puts a marker letting you use `GET` and `PUT` to access results placed AFTER the `ZZO` code. `0 GET` will get the first result, `1 GET` the second etc.

```
$
ZZO                   ;set origin in result stack

$
```

The PLA code calculates single planets using the menu date and time. It calculates both longitude and speed, but here you need only the longitude. To remove the speed, the two results are swapped, and the DEC code cancels the (now topmost) speed. A bit of arithmetics converts the longitude from intang to minutes and sign.

```
$
1 CNT PLA             ;calculate planet using menu values
XY DEC                ;get longitude (discard speed)
ITOM                  ;convert to minutes of arc
1800 DIVR             ;convert to sign & minutes

$
```

Now the result stack holds: SIGN MINUTES

The graph superimposes all the signs (30 degree system), so to draw the movement, only the date (x-value) and position in sign (minutes) are used. The sign number itself is used only to check if the planet moves past a sign limit.

```
$
2 CNT 90 SUB          ;calculate horizontal position
0 RCL 3 MULT          ;horizontal position
4 STO                 ;save x
0 GET 900 SUB         ;vertical position=minutes of arc
11 RCL 900 MULT             ;scaled  to +-frame heigth (cell 11)
5 STO                 ;save y
1 GET 6 STO           ;save sign
```

```
$
```

Each time a position is calculated it is saved in cell 4 (x), cell 5 (y) and cell 6 (sign). After the line drawing, these values are moved to cell 1-3 as "old" values. For each step you will then be able to draw a line from the "old" x,y to the "new" x,y. The only exception is the first step, where no old values yet exist. Therefore the first line below excludes drawing in that case.

Also excluded are sign shifts, which must be drawn differently.

```
$
2 CNT IF                    ;if not first position
6 RCL 3 RCL = IF            ;if same sign as last position
4 RCL 5 RCL          ;start x,y
1 RCL 2 RCL          ;end x,y
1 2 DFAT             ;draw line of movement
ENDIF

$
```

If sign shift is detected, the line of planetary movement must be split in two: The first part, which moves from "old" x,y to the sign limit (the grid frame), and the second part moving from the other sign limit to the "new" x,y. The calculation is split in two parts: One direct and one retrograde.

The calculation needs some insight into triangle geometry. A further complication is, that using integer arithmetic, it is necessary to use the `MULT` code to be able to scale the values properly. `MULT` combines a multiplication and a division avoiding round-off and overflow errors.

```
$
6 RCL 3 RCL SUB             ;if sign changed to next
12 MOD 1 = IF        ;calculate when
5 RCL 2 RCL SUB             ;y difference (minutes only)
11 RCL 2 MUL ADD 7 STO      ;real y diffence (including sign)
11 RCL 2 RCL SUB     ;remaining y movement in old sign
1 DUP                ;use twice
4 RCL 7 RCL MULT     ;scale new x proportionally
XY                   ;remaining y movement again
1 RCL 7 RCL MULT SUB ;scale old x and subtract
1 RCL ADD            ;add to old x
7 STO                ;crossing 0 degrees at this x
1 RCL 2 RCL
7 RCL 11 RCL 1 2 DFAT;first part drawn
7 RCL 11 RCL CHS
4 RCL 5 RCL 1 2 DFAT ;second part drawn
ENDIF

$
6 RCL 3 RCL SUB             ;if sign changed to prev
12 MOD 11 = IF       ;calculate crossing point
5 RCL 2 RCL SUB             ;y difference (minutes only)
11 RCL 2 MUL SUB 7 STO      ;real y diffence (including sign)
11 RCL CHS 2 RCL SUB ;remaining movement in old sign
1 DUP                ;use twice
4 RCL 7 RCL MULT     ;scale new x proportionally
XY                   ;remaining y again
1 RCL 7 RCL MULT SUB ;scale old x and subtract
1 RCL ADD            ;add to old x
```

```
7 STO                 ;crossing at this x
1 RCL 2 RCL
7 RCL 11 RCL CHS
1 2 DFAT              ;first part drawn
7 RCL 11 RCL
4 RCL 5 RCL 1 2 DFAT  ;second part drawn
ENDIF

$
```

Now after drawing a line bit, the x,y and sign information is moved from "new" (cell 4,5,6) to "old" (cell 1,2,3), and the daycount and planet count loops continue. Note the `ENDIF` placement between the two `NEXT`'s. You must always keep these constructions either entirely overlapping or not at all. Partial overlap e.g. `FOR IF NEXT ENDIF` will produce chaos, and should be avoided.

```
$
ENDIF
4 RCL 1 STO           ;keep last x
5 RCL 2 STO           ;and y
6 RCL 3 STO           ;and sign
NEXT ENDIF NEXT

$
```

Finally, the saved menu values are put back to the input menu, so that it will look the same as before the call, provided, that you did not interrupt the printout pressing ESC.

```
$
0 7 FOR               ;restore menu values
1 CNT 100 ADD RCL
1 CNT MEPUT
NEXT
WKEY
GROFF

$$$
```

This concludes the GRAPHEPH module.

CHANGING PCA CONFIGURATION

PCA has a number of installable settings. You can change these using the systemvariables edit facility in the preferences menu. However they may also be changed using the SYSTR function.

Further there are some special functions to change system settings.

PTMP: Placing this code in the start of your module will assure, that your changes will be reset when the module finishes. Without PTMP the values will change for the rest of your Argus session.

CONFX: There are a couple of these settings available, which decide a number of program options. Some of them are values others are flags, which can be only 1 (true) or 0 (false). A full list of the system variables including the CONFX values are given at the end of this document.

SYMIX is used by the SYMBOL to change the glyphs for Uranus and Pluto

AYA sets the zodiac origin offset. Normally this is 0, which means that the zodiac starts at the vernal equinox as used by western astrologers. Setting an offset means, that you may start the zodiac anywhere, on a fixed star as used by hindu astrologers and siderial astrologers, or you may set it to the Moon's north node to obtain a draco-chart. AYA will work only as long as you are running the XLI module. You may however circumvent this by setting 128 0 CONFX which will keep the offset active. If you then go to the installation menu and press U (update), your PCA will be customi-zed to run siderial astrology. You will realize this from the chart position printouts, which will have the offset printed under the birth data. The modules HINDU and SID use this code. The offset must be entered for 1900, and is auto-adjusted for the current year taken from the PCA input menu.

COL customizes the colours and can actually do no more than you can do using the colour installation menu. It is the only way though, that you can change the colours from XLI, because you cannot create a macro manipulating the colour installation, which uses cursor keys mostly. The modules COLSETUP and BIO use COL.

Further information on these codes are given in the function reference.

RUNNING ANOTHER PROGRAM FROM WITHIN PCA

EXEC execute external program
AUTO.XLI execute PCA job from external program

Calling external programs without having to stop PCA means, that on retum, you will find PCA in the state you left it with the same data and jobs, you were running before the call. For example, if you just need to lookup something on your harddisk, edit an XLI file for testing or send a fax. This however, could be easier done, if you have a multitasking or taskswitching operating system.

More interesting is, that positions, aspects or whatever calculated by PCA may be transferred to the program you call, and values calculated by the external program can be transferred back to PCA.

The AUTO.XLI is not a code but an XLI file that, if present, will execute as soon as PCA starts without showing the menu first. The AUTO.XLI may be coded also to quit PCA automatically when done, by putting a macro at the end consisting of a Q (quit). Ultimately, any number of astrology programs (with similar facilities), databases, statistics programs, communication programs sending and receiving letters and faxes may cooperate in one big session.

EXECUTING an external program

You may execute any .COM or .EXE program from an XLI file. Here are some examples of using EXEC:

```
$
EXEC

\DOS\EDIT.COM
$$$
```

The coding field should just have the code EXEC. The first line of the text field must hold the complete path and program name including .EXE or .COM.

CALLING PCA FROM ANOTHER PROGRAM TO CALCULATE POSITIONS

You may also call PCA from other programs to do some dedicated task. To do this, you can add the special job to the commandline, for example:

PCA SPECIAL.XLI Q

PCA will try to find a file called SPECIAL.XLI which should be the XLI module doing your special task. Of course you could choose any name for this. The Q is a macro, which just asks PCA to terminate when finished.

MISCELLANEOUS TIPS AND TRICKS

FEED	conditional page eject
GRAF	display bar graph
INFIL	read 1 line of data file

Avoiding page breaks

Often, when you print tables etc. you will want some number of lines to appear all on the same page. To avoid a page break, if the printer happen to be at the end of a page, use the code `FEED`. For example `12 FEED` will assure, that the following 12 lines will stay unseparated. If less than 12 lines remain on the current page, the formfeed will perform immediately, so that all 12 lines are moved to the next page. Note also the new codes XFEED and CFEED with additional features

Printing Bar graphs

You may print horizontal bar graphs to visualise the size of numbers. This works in simple text mode using the IBM characters, for example a block character (220). An example of this is given in the ELGRAF.XLI which follows here:

```
ELGRAF printing bar graphs for tripli- and quadruplicities
1 STO  ild
2 STO  jord
3 STO  luft
4 STO  vand
5 STO
6 STO
7 STO

$
PTMP
1 1 NTOS 49 -1 SYSTR            ;Bar graph
NPR 1                           ;Print name

Antal planeter inkl ASC:
$
1 OEM

               0  1  2  3  4  5  6  7  8  9 10 11
--------------------+---+---+---+---+---+---+---+---+---+---+---+---+
$
0 OEM
```

```
0 1 7 FOR 1 CNT STO NEXT     ;Rest STO vars
1 14 FOR                     ;Count planets
1 CNT 11 > 1 CNT 0 14 IN OR IF          ;and ASC
1 CNT PSI                    ;Signs
1 SUB 4 MOD 1 ADD            ;Elements
1 DUP RCL 1 ADD XY STO
1 CNT PSI
1 SUB 3 MOD 5 ADD            ;Quadruplities
1 DUP RCL 1 ADD XY STO
ENDIF
NEXT
7 1 FOR 1 CNT RCL 4 MUL NEXT 182 GRAF

  Element Fire          #
  Element Earth         #
  Element Air           #
  Element Water         #
$
1 OEM

--------------------+---+---+---+---+---+---+---+---+---+---+---+---+
$
0 OEM 182 GRAF

  Cardinal signs        #
  Fixed signs           #
  Movable signs         #
$
1 OEM

--------------------+---+---+---+---+---+---+---+---+---+---+---+---+
$
0 OEM

$$$
```

File input/output

File output for example for data export is done setting up a print file. You can then run one or any number of paragraphs with text or with inserted templates to print out number and strings. Putting a backslash at the end of a line allows you to put bits of lines together to longer lines.

To get input from a file, you use the code `INFIL`. `INFIL` reads a line of a textfile. It is possible just to read one line as is, or you may read special bits, numbers or strings. A detailed explanation is given in the function reference part.

The following example will read a line of the file XXFILE.TXT into string `0`. It could then be printed, inserted as a name in the namefile, compared to other strings etc.

```
$
1 INFIL

XXFILE.TXT
@
$
```

To print a complete textfile on screen or printer (no lines being longer than 80 characters), the following code will do:

```
$
1 32000 FOR          ;setup maxnumber of lines
1 INFIL -1 = XIF            ;break at end of file

XXFILE.TXT
@
$
0 NUMS

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@-@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@-
@@@@@@@@@@@@@@@@@
$
NEXT

$$$
```

To extract 6 numbers from an input line:

```
$
1 INFIL

XXFILE.TXT
# # # # # #
$
```

The `@` and `#` templates can be mixed with search characters to setup quite complicated input tasks. For example in the NAME2TXT module the following `INFIL` is used:

```
$
1 INFIL


NAMEFILE.TXT
"@"###, @1,#####, @2,##, @3,##, @4, @5,
$
```

The above input spec string means:

`"`	look for the first quote character
`@"`	read the following characters until but not including the next quote character into string 0
`###`	read three numbers separated by any character(s)
`,`	look for next comma
	read any number of (leading) spaces (if any)
`@1,`	read the following characters until but not including the next comma into string 1
`#####`	read five numbers separated by any character(s)
`,`	look for next comma
	read any number of (leading) spaces (if any)
`@2,`	read the following characters until but not including the next comma into string 2
`##`	read two numbers separated by any character(s)
`,`	look for next comma
	read any number of (leading) spaces (if any)
`@3,`	read the following characters until but not

```
       including the next comma into string 3
##     read two numbers separated by any character(s)
,      look for next comma
       read any number of (leading) spaces (if any)
@4,    read the following characters until but not
       including the next comma into string 4
       read any number of (leading) spaces (if any)
@5,    read the following characters until but not
       including the next comma into string 5
,      look for final comma
```

You may find a number of other uses and combinations if you go through the function reference on INFIL.

There is a similar code for outputting to file called UTFIL. See the function reference.

LOW LEVEL ASPECT CALCULATION (AZP)

The code `AZP` is a primitive aspect routine which needs more programming from your side. The advantage is, that it is inde-pendant of the PCA orb settings and that it is completely flexible. Below is an example of its use:

```
2 PPOS 8 PPOS SUB AZP 4 = XY ITOMS ABS 60 > AND   ;Moo Tri Ura
```

Here follows a breakdown of the above coding:

```
2 PPOS 8 PPOS SUB     ;calculate Moon-Uranus angle (intang)
AZP                   ;calculate aspect number and orb
4 =                   ;check that it is a trine
XY                    ;fetch the orb (intang)
ITOMS                 ;convert intang to minutes of arc
ABS                   ;remove orb sign
60 >                  ;check within one degree
AND                   ;combine aspect number and orb checks
```

The `XY` fetches the orb one level down the stack, and leaves the first condition (the trine test) just below. When the orb testing is done (`60 >`), you will have the result of the orb test topmost, and the trine test below. The `AND` will combine the two and the result will be `1` (true) if both the two conditions were met.

The `AZP` function produces two results, the aspect number (placed topmost on the stack) and the actual orb (next). The actual orb is signed, so you may care about the order in which you subtract the two planets, so you can use the orb sign to find out if the aspect is applying or separating (if you know which one is the fastest).

A more advanced coding example is given here to calculate the orb speed. Here the actual planet speeds are used, so you do not need to care about which planet is the fastest, and whether they are retrograde etc.:

```
$
3 PPOS 4 PPOS SUB AZP        ;aspect no. orb
XY 1 DUP 12 MUL                     ;12 * orb
3 PV 4 PV SUB                ;orbspeed (signed)
1 DUP NOT IF                 ;check div by zero
DEC DEC 999 1 ENDIF          ;if so replace by 99
```

```
DIV                          ;orbspeed (months)
XY ITOMS                     ;orb (signed minutes)
NUMS                         ;print results


Venus aspect Mercury

        orb:    ######
        speed:  ######
        aspect: ######
$$$
```

The orb will be a signed value. The sign does not tell if the aspect is applying or separating, but if the angle is less than or bigger than the precise aspect.

The orbspeed will be a number of months, if the chart is a pro-gressed one. That is the reason for multiplying by `12` in line 2. This number will also have a sign which in fact tells if the aspect is applying or separatiing. So -7 means, that in a progressed chart, the aspect was exact 7 months ago. This calculation assumes, that the planets are moving at constant speed. Planets around their station will not fit this ideal case. To get the exact time of perfecting the aspect, you will need to do repeated planetary calculations to adjust for speed nonlinearity.

NOTE Argus has an improved version of AZP called AZE. Please refer to the function reference

THE XLI DEBUGGER

Debugging programs may be a very hard job. You think, that you have analyzed your problem well, coded it correctly, and still even simple tasks may seem to behave completely differently from the expected. Your first thought may be: faulty toolkit, faulty PCA, faulty computer/harddisk, virus attack or whatever. The point is, that you cannot exclude any explanation until you really find out, what's actually going on. Very often, when you finally find the reason, what you first thought impossible, suddenly will appear obvious.

To find out what's going on, you need a magnifying glass to follow the action of your code step by step. This is what the debugger does. The execution of XLI modules may run in two modes: normal mode and debug mode.

In debug mode, you'll have to press a key for each code, so it will run very slowly, even if you hold the key down for repeat. But the advantage is, that for each code executed, you will see the upper-most numbers on the result stack. Positions where nothing has been calculated yet will show zeroes. You may also find old stuff from calculations and codes earlier in the module, that you did not remove, but just ignored.

Activating the debugger:

The debugger (or debug mode) may be activated at any point in your module. Just put the code `DEBUG`, where you want the debugging to start. When the debugger has started, you may single step pressing any key except ESC, which is used for leaving debug mode. So when you're done, just press ESC, and your module will continue normally. If you keep pressing ESC, you will get the option to leave the module and go back to the PCA main menu.

If you put `DEBUG` inside a loop, pressing ESC will leave debug mode only until it loops back to `DEBUG` next time. You may then by repeatedly pressing ESC follow the result at just one point in the loop, not having to tediously step through all the single codes inside the loop. To get out of the loop may be a bit difficult. You must either double-press ESC very quickly, or hold down ESC long enough to make the return-to-main option appear.

If you put DEBUG inside an IF-ENDIF construction, it will only be activated if the IF statement is true.

There is another way of starting the debugger without changing your code. Press CTRL-backspace while the module is running. The drawback by this method is that debugging will start at some quite random position.

In practice:

You may experience a number of problems writing XLI modules or interpretations. The debugger does not solve all of these, but will enlighten most coding errors.

Find the paragraph, which produces false output, or which fails to produce the expected output, insert the code DEBUG as the first code in the paragraph and run the module. When the program goes into debug mode, press (any) key a number of times letting the debugger single step. Check the intermediate results and compare them to what you expected.

If the XLI cannot find the file you've written, read the error message carefully and see if there are any characters in the filename quoted by the errormessage, that should not be there. If the filename looked for is the first, in which case you should check your installation menu (interpretations) or input (OXS call). It could also be part of the chain, in which case you should check the file that calls it as the next. This may not always be the file you expect, the interpreter may have processed a number of outputless files before the error arrives. Check that the filename is complete and intact, and is followed by at least one line (blank or not).

There is also a code called XLOG which instead of letting your single step your module writes the whole lot to disk. Please note, that this may slow down program execution somwhat.

ENCRYPTING YOUR TEXTS AND MODULES

Interpretations and modules for PCA are quite vulnerable to piracy copying. Real programs (EXE and COM files) may be copy protected, but interpreted textfiles like XLI-files cannot. Even if the main PCA program is not copyprotected either, it has the user name unerasable inserted.

To add a minimum of safety from illegal copying of your modules, you may use the program SCRAM to encrypt your interpretations and modules, and at the same time prevent, that they are used with more than one users program. The encryption uses the PCA serial number as encryptation key.

To encrypt a module, use the following DOS-command:

```
SCRAM filename serialnumber (RETURN)

for example SCRAM TT1.TXT 13260
```

Of course TT1.TXT must be present. SCRAM will produce a new file TT1.SCM. This will run with PCA, but the .SCM extension means, that PCA will decrypt it checking it against its serialnumber. If the serialnumber does not match, only rubbish will be output.

If your interpretation or module consist of several files, you must encrypt all of them. You

You do not need to change the file extensions to .SCM, neither in the installation menu or in the file chains themselves. If PCA cannot find a file, it will automatically try to look for a file with the same name, but with the extension .SCM. As soon as an SCM file is found, PCA automatically assumes, that the remainder of the file chain is .SCM files.

The SCRAM program is available for developers on request from Electric Ephemeris.

Complete XLI Function reference

The following complete alphabetical function code reference explains the details of each function.

For each function a stack specification is given. The topmost figures taken off the stack are called X,Y,Z,P,Q,R etc.

For example:

`X Y DIVR → X mod Y  X/Y`

means that X and Y are removed from the stack and that first X mod Y, then X/Y are put back on the stack. So after the operation X/Y will be on top of the stack.

`X IF →`

means that X is removed, and nothing is put back.

`X STO → X`

means that X is used, but remains unchanged on the stack.

`ENDIF →`

means that the code does nothing with the stack.

FUNCTION REFERENCE

Complete list of functions:

Functions marked with * are windows version only.

Data:	**MEGET**	fetch value from PCA input menu
	MEPUT	insert value into PCA input menu
	NFI	fetch namefile entry no. n
	NPNT	read/write namefilepointer
	NPUT	insert name into PCA input menu
	BDPR	write birth data
	NPR	write name
	DDIF	find distance between to dates
	NDATE	konvert transit-timeslice to date
	JD	find julian date
	INFIL	read one line from textfile
	UTFIL	*print to file, equivalent to INFIL
	MOVCH	*move chart data around
	SWDAT	*swaps input data sets
	CNTRY	*get country name from current data
	AREA	*get timezone area code from current data
	MENAM	*insert name in string n
	BDSEL	*select input dataset
Astrology:	**PSI**	planet in sign
	PHS	planet in house

	RX	planet retrograde
	HSI	house in sign
	HRU	house ruler
	APOW	aspect power
	ANUM	aspect number
	AORB	aspect orb
	AZP	aspect test primitive
	PDEG	planet position (degrees)
	PPOS	planet position (intang units)
	HPOS	house position (intang units)
	PV	planet speed (intang units)
	HV	house speed (intang units)
	PLA	calculate one planet (intang units)
	XPLA	find auxiliary planet information
	HOUSE	calculate one house (intang units)
	BHUS	house position (universal)
	KUN	fetch the 3 closest Kündig sections
	GTR	call userdefined graphic transit
	RTA	reset transit aspect list
	NTA	fetch next transit aspect
	RECH	*fetch latest horoskope type
	ZODOF	*Change zodiac origin to x
	AZE	*aspect primitive
	PNAME	*get system string index for planet name
	SNAME	*get system string index for sign name
	REF	*switch interpretation reference mode on/off
	ANAME	*get system string index for aspect name
Mathematics:	>	bigger than
	<	less than
	=	equal
	<>	not equal
	IN	group membership test
	AND	bitwise AND
	OR	bitwise OR
	CPL	bitwise complement
	NOT	logical negation
	BOO	intger to boolean
	ABS	absolute value
	CHS	change sign
	ADD	add
	SUB	subtract
	MUL	multiply
	MULT	multiply by fraction
	DIV	divide
	MOD	modulus
	DIVR	divide with remainder
	MAX	find maksimum of two values
	MIN	find minimum of two values
	ITOM	intang units to minutes of arc (unsigned)
	FPON	*turn floating point mode on
	FPOFF	*turn floating point mode off
	FTOI	*convert floating point value to integer
	XOR	*exclusive or
	ITOF	*convert integer value to floating point
	PTR	*convert coordinates from polar to rectangular

	SPRED	*spread conjuncted positions
	ITOMS	intang enheder to mintes of arc (signed)
String handling:	**CARY**	define string variables
	NAME	fetch current name into the streng array
	PLACE	fetch current city into the string array
	STDEF	define one string in the string array
	STCAT	concatenate strings
	STCMP	compare strings
	STCUT	cutout string
	STLEN	find length of string
	NTOS	*convert number to string
	STPOS	*find substring in string
	ANTOI	*convert number field of comma-separated line to integer
Stack control:	**DUP**	duplicate value in stack
	FETCH	fetch value in stack
	ZZO	place origin on stack
	GET	fetch number from stack
	PUT	put number into stack
	XY	exchanges numbers on top of stack
	INC	add one to the stack pointer
	DEC	subtract 1 from stack pointer
Memory:	**STO**	save in memory cell
	RCL	fetch from memory cell
Flow-control:	**ENDIF**	end of if-construction
	IF	start of if-construction
	FOR	start of FOR-loop
	CNT	get FOR-loop counter
	NEXT	end of FOR-loop
	XIF	quit FOR-loop conditionally
	NFN	set filebranch index
	WAIT	withhold heading
	CONT	release withheld heading
	PSTAT	change printer status or suppress all print
	EXEC	execute an external program
	DELAY	wait for n milliseconds
	ELSE	*conditioned branch
	PROC	*define subroutine
	RETN	*return from subroutine
	SUBR	*call subroutine
	DEBUG	enter debug mode
Interactive:	**MENU**	create menu
	OPT	wait for defined keypresses
	WKEY	wait any keypress
	MENUX	*windows type dialog box
	KEY	*simulate a special keypress
Layout:	**FEED**	conditional formfeed
	GRAF	show bar graph
	NUMS	print numbers or strings
	ZMODE	*text wordwrap mode
	MONS	*set monospaced font mode

	OEM	*set OEM font on/off
	FONTS	*define fonts
	FONT	*select font definition
	XFEED	*conditional formfeed
	CFEED	*conditional formfeed
	TXCOL	*change text color
	TAB	*tabulate
	PAD	*set string templates padding mode
	CENT	*set centered text justification
Graphics:	**GRON**	enter graphics mode
	GROFF	leave graphics mode
	GRCOL	set graphics colour
	PENUP	lift the pen
	DRAW	draw line
	DFAT	draw fat line
	DRSYM	draw defined symbol
	DRTXT	draw text string
	PENC	*set pen color
	PENW	*set pen width
	BRCOL	*set fill color
	POLY	*draw polygon
	RSIZE	*set graphics scaling
	ORGIN	*set graphics origin
	BMP	*import and diplay bitmap (BMP file)
	GMODE	*change currently open graphic window mode
	SCALE	*define graphic frame
	WHEEL	*draw chartwheel
Macro:	**CML**	execute defined macro (destruktive)
	CALL	call defined macro (non-destruktive)
	FUNX	call single PCA function
	MACn	*call single key macro
	MAC.	*Clear window
	MACOF	*Suppress all output
	MACON	*restore output
	macro 8	*set orbis
	Macro 7	*set harmonic number:
Configure PCA:	**COL**	insert colour definition
	CHROT	set chartwheel rotation
	CONFX	set special configuration value
	SYMIX	create symbol choice table
	PXL	setup user defined printer translation table
	AYA	set ayanamsha value (minutes of arc)
	PTMP	*backup system settings to restore at module exit
	TIDY	*restore system settings before module exit
	SYSTR	*Set or read system string
	SYSAV	*save selected part of system configuration to disk
Security;	***	There is a small subset of XLI commands that adds the ability to improve protection of texts or XLI programs. If you need such features, contact us.
Miscellaneous:	**SYN**	initialise synastry
	NAX	auxiliary namefile access

PPATH *get Argus program path
XPATH *get current XLI module path

<

Stack: X Y < → Y < X

"Less than" checks if Y is less than X. If so it places a 1 on the stack, else a 0.

< >

Stack: X Y < > → Y < > X

"Differs from" checks if Y is different from X. If so it places a 1 on the stack, else a 0.

=

Stack: X Y = → Y = X

"Equals" checks if Y is equal to X. If so it places a 1 on the stack, else a 0.

>

Stack: X Y > → Y>X

"Greater than" checks if Y is greater than X. If so it places a 1 on the stack, else a 0.

ABS (absolute)

Stack: X ABS → abs(X)

Absolute value means changing to positive number, if X is negative.

ADD

Stack: X Y ADD → X+Y

Adds two numbers

ANAME

Converts an aspect number to a negative system string number for use with NUMS. ANAME is the equivalent of writing: 618 ADD CHS

The CFG lines holding the planet, sign and aspect names are ASCII and do not hold symbols. To be able to display either abbreviations or symbols depending on the program setting, the NUMS mechanism is intercepted, so that accessing these strings will be translated to symbols. For example -542 NUMS should normally print "Moon" in the @@@@ template, but if the Argus is set for symbols it will write the Moon symbol instead. Because it is difficult to memorise and calculate the cfg indexes each time you want to display a planet or sign, the codes PNAME, ANAME and SNAME can be used to fetch the index. So 1 ANAME NUMS will display a conjunction symbol or abbreviation, 3 PNAME NUMS a Mercury etc.

see also PNAME, SNAME

AND

Stack: X Y AND → X AND Y

Checks if two conditions are both true. This means that:

1 1 AND → 1
1 0 AND → 0
0 1 AND → 0
0 0 AND → 0

1 represents "true" and 0 "false".

Technical note: if AND, OR is used on other numbers than 1 and 0 the result will be a "bitwise" comparison. To be able to analyse the result, you will have to convert the numbers to binary, i.e. each 16 ones and zeroes, and compare each set of bits separately. This may be used creatively if you are experienced in this field. If not, better be sure, that you use the function on zeroes and ones only.

ANTOI

p n ANTOI -> (integer)

Conversion of digit group p in stringarray[n] to integer The result of the conversion will be modulus +-32768

This code is used to retrieve integer values from a commaseparated data string, which could look something like
the string defined below. The result in the example below will be 3980.

$
1 STDEF

274,993,3287,11281,2,435,882,3980,9999,10039,4,0,34,.8123
$
8 1 ANTOI NUMS

value 8= #######

$$$

ANUM (aspect number)

Stack: X Y ANUM → (number)

This code tells the kind of aspect between X and Y.

No aspect	0
Conjunction	1
Opposition	2
Square	3
Trine	4
Sextile	5
Semisquare	6
Sesquisquare	7
Inconjunct	8
Semisextile	9

X and Y are the two planets aspecting. For the planet numbers allowed, see PSI.

AORB

Stack: X Y AORB → (orb)

This code tells the orb of the aspect between X and Y. The result is in tenths of a degree. The orb value may be positive or negative:

If the angle measured from X to Y is less than the ideal aspect angle, the orb is positive, if it is more than the ideal angle, it is negative.

For example if:

Angle from Sun to Mars 89 degrees : 1 5 AORB → 10
Angle from Mars to Sun 271 degrees: 5 1 AORB → -10

The result 10 means one degree (ten tenths). It may help to imagine a decimal point.

Even if the aspect is out of the defined orb, this function will still return a nonzero value. This value will apply to the closest aspect. For instance if the angle between X and Y is 131 degrees, you will get the result 40 because it is 4 degrees from a sesquisquare. This works even if you cancelled out sesquisquares by defining their orb to zero in the installation menu.

X and Y are the two planets aspecting. For the planet numbers allowed, see PSI.

APOW (aspect power)

Stack: X Y APOW → (power)

The aspect power is a number between 1 and 10 dependant on the actual orb of the aspect. If the orb is zero (exact aspect), the power will be 10, if it is just on the orb limit the power will be zero. So the power will depend on the maximum orb you defined in the menu and in the aspect orb installation.

X and Y are the two planets aspecting. For the planet numbers allowed, see PSI.

AREA

n AREA

Loads the timezone area code into string array
n.

AYA (Ayanamsha)

Stack: X AYA →

Some astrologers use the siderial zodiac instead of the tropical one used by most western astrologers. Hindu astrology refers always to the siderial zodiac which rely on the fixed star constellations rather than on the equinoxes. There are however disputes, which fixed stars should define the siderial zodiac starting point.

The AYA code lets you define an "ayanamsha", the difference between the two zodiacs, and automatically subtract this from all calculations. This will apply as long as you are working from the XLI interpreter. As soon as you return to the main program, PCA will revert to the tropical zodiac. If you want the main program also to use your defined ayanamsha, you may use the CONFX code, see this. Position printouts with ayanamsha, will have the ayanamsha angle written under the date, time, etc.

To enter your favourite ayanamsha, you must know its size at the 1.st of january 1900, measured in minutes of arc. For example to insert LAHIRI ayanamsha, code:

```
$
1348 AYA

$
```

The module SID.XLI uses the AYA code to let you insert some common used ayanamshas. At the same time it uses CONFX as mentioned above to prevent PCA to automatically revert to tropical zodiac.

AZP (aspect primitive)

Stack: X AZP → (orb) (aspect no 1-9)

This is an aspect routine giving just raw data without using the orb limits inserted in the main and installation menus. Also, the values are intang units. The function produces both the aspect number and the actual orb.

The X is the angle between the two planets measured in intang units. This may be calculated using the PPOS code to get the planets positions and then subtracting them to get the angle. For example:

1 PPOS 4 PPOS SUB AZP

Aspects numbers returned are:

0 - 20 degrees:	Conjunction
20 - 40 degrees:	Semi-sextile
40 - 50 degrees:	Semi-square
50 - 70 degrees:	Sextile
70 - 110 degrees:	Square
110 - 130 degrees:	Trine
130 - 140 degrees:	Sesquisquare
140 - 160 degrees:	Inconjunct (Quincunx)
160 - 180 degrees:	Opposition

The orb is a signed number giving the distance from the ideal angle. The sign is calculated the same way as in the AORB code (see this), but the orb is given in intang units.

AZE

v AZE -> (orb((aspectno 0-maxaspect)

This is the modern Argus version of AZP (see AZP) including all valid aspects. The aspect has to be within the current orb definition for that aspect, i.e. if aspect scheme R is selected all aspects are checked against the aspect orb in scheme R before they are accepted.

BDPR (birth data print)

Stack: BDPR →

Print date, time, longitude and latitude on four lines. This code is useful together with the code NPR in the start of an interpretation to print birth details.

BDSEL

n BDSEL

Select input menu data card: 0=radix 1=current 2=Present. So for example, to make a radix chart for the time now, you could code 2 BDSEL MACR to make the radix calculation take the input data from the present data menu with the horary data.

Index 0: Currentdata
- 1: Radixdata
- 2: Currentdata
- 3: Temprdata
- 4: Tempcdata
- 5: Presentdata

This does not affect the actual card selection in the user interface.

Index 3 and 4 (temprdata and tempcdata) lets you make the selectedpointer point to the data saved by PTMP. This means, if you use PTMP and then change the radixdata and/or currentdata, you can get access to the original data using 3 BDSEL or 4 BDSEL. You can then use SWDAT to swap these data around. Be aware though, that this could change the original data you tried to save with PTMP.

BHUS

Stack: X Y BHUS → (housenumber 1-12)

This code will find the house position of any point using the houses of either the latest radix or the latest non-radix.

X: the position to test in intang units

Y: 1 for latest radix houses
2 for latest non-radix houses

For example having calculated radix and solar return, to find the position of the radix Moon in the Solar return houses, use this coding:

22 PPOS 2 BHUS

The planet numbers allowed are listed under PSI.

BMP

x0 y0 x1 y1 BMP (Draw bitmap)

You can load a bitmap from a BMP file into your graphic. If X0,X0 is different from X1,Y1 then the bitmap will be streched to fit the rectangle, whose upper left corner is x0,y0 and whose lower right corner is x1,y1. If these two points are equal, the bitmap will not be streched but appear in the size, which matches the output device bit resolution. This normally looks better, but will be big on a screen and very small on high resolution printers.

If x0>x1 the bitmap will be mirrored left-right and if y0>y1 then the bitmap will be mirrored upside down.

When stretching, the bitmap suffers quite much and the quality is not very good. This seems to be a Delphi 1 problem, and the same drawing looks better in Delphi 3.

BOO (Boolean)

Stack: X BOO → (1 or 0)

Checks that a number is nonzero. This code is used, when you wish to combine numbers with AND and OR, and these numbers may be more than just 0 or zero. Having calculated for example two sets of points, then to check, that they are both nonzero, you would code

```
X BOO Y BOO AND
```

Just writing X Y AND will do a bitwise AND and this will probably not be what you are looking for.

BRCOL (fill color)

000 BRCOL changes brush color to black, 080 BRCOL changes the brush to green etc. The brush is the colour used for filling polygons.

CALL

Stack: CALL →

Execute a specified macro. The macro must be given as the first line in the text field. CALL reads a line in the text field and executes it. The example below will insert the current date and time and calculate a radix chart.

```
$
CALL

1T.R
$$$
```

See also CML

CARY

Stack: X CARY →

Reads lines of strings into the internal string array. For explanation of the string array see the chapter on this.

The strings must be separated by commas and will read into array index 0, 1, 2 etc.

For example:

```
$
2 CARY

ari,tau,gem,cnc,leo,vir,lib,sco,sgr,cap,aqr,psc
Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
$
```

This will read the sign and month names into the string array. The sign names will get the indexes 0-11 and the months 12-23.

Note that there is no comma at the end of the line, the line shift will act as a comma. If you put a comma in front of the first line, the signs will get indexes 1-12 and the monthes 13-24.

Remember, that the string array may hold no more that 255 characters total. This includes the separating commas.

CARY is used for mass loading of the string array. To load a single entry, use the code STDEF.

WARNING: CARY will whipe all earlier definitions of the string array, e.g. NAME.

CENT

Use this code to center headings. 1 CENT will make the following text adjust centered. 0 CENT will turn this off.

CFEED

n m CFEED (conditional formfeed)

This code is used to make room between different parts of a printed text. If less than n lines remain on the page, a simple formfeed is executed. If more than n lines remain, this means that a too big gap will appear, and instead of a formfeed m blank lines will be outputted. (But not beyond the end of the page. However it would be unreasonable if m was bigger than n).

see also FEED, XFEED

CHROT (chart rotation)

Stack: X CHROT →

The PCA chartwheel presentation have a couple of rotational variants, you may set with this code. The possible values for X are:

0: ascendant horizontal (default)
1: MC vertical
2: radix ascendant horizontal
3: radix MC vertical
4: aries to the right
5: aries to the left

This setting may be saved using the U option in the installation menu.

The meaning of 2 and 3 is that the signs will have the same orientation as in the previous calculated radix chart, so it will be easy to realise how much progressed angles have moved.

CHS

Stack: X CHS → -X

Change sign means changing to positive number if X is negative and vice versa.

CML (commandline/Queue)

Stack: CML → 0

This code will setup a macro, which will execute as soon as the XLI module has finished. It will overwrite any queue pending. This code may be used for example in an AUTO.XLI module, which will force PCA to do specialized tasks as soon as it starts, before it leaves control to you showing the main menu.

The macro must be given in the text field. CML reads a line in the text field and executes it. The example below will call a radix and aspects using the menu values.

$

```
CML

RA
$
```

A more flexible macro handler is the code CALL (see this).

CNT (count)

Stack: X CNT → (counter value)

This code is used inside a FOR NEXT counting loop to get the current value of its counter.

If only one loop is started, X must be one. If more than one loop is running inside each other, X must be 1 for the outermost loop (the first one started), 2 for the next and so on. Up till 10 loops may be running simultaneously. For an example, see code FOR.

CNTRY

n CNTRY

Loads the country name from the "current" data card into string array n.

COL (colour)

Stack: X Y Z COL →

Set the colour for one colour item. This code allows to customise the individual PCA colours. It has the same effect as using the colour setup in the installation menu.

X is the device number (1-2). Use 1 for screen and 2 for printer.

Y is the colour item number (0-39). Each item, that you may assign a separate colour has an item number. For a list of these numbers, see GRCOL.

Z is the colour number (0-999), where each digit represents the quantity of red, green and blue respectively. So bright red has number 900, bright green 090 and blue number 009.

You must end your colour definition with 0 0 0 COL to activate the defined colours.

If you just need a temporary colour setup for this module, see PTMP how to do this.

!!Remember, that the EGA/VGA screens cannot show more than 16 colours at one time. If you try to define more than that, the last ones will be random.

CONFX (configuration extras)

Stack: X Y CONFX → X

PCA has a number of "hidden extras". This is mainly possibilities wished for by certain users, which are implemented without having to make the full "new version features" with installation menu option, further manual pages and more confusion for newcomers. For the experimentor however it may be delightful finding new crazy features in his program.

Some of them are quite useful however, for example to redirect the printer output to LPT2: or the plotter output to a parallel port.

The settings may be run just once and then saved permanently using the U option in the installation menu.

```
  1 0 CONFX : point (part of fortune)=chiron
  2 0 CONFX : midpoint trees aspect type and orb suppressed
  4 0 CONFX : Huber ap using true equation
  8 0 CONFX : Kündig sections in english version
 16 0 CONFX : Huber ap Jan Romander (slightly different speed)
 32 0 CONFX : Silver reed plotter instead of Sharp
 64 0 CONFX : page numbers suppress
128 0 CONFX : ayanamsha allowed in main program

  0 1 CONFX : printer port = lpt1: (default)
  1 1 CONFX : printer port = lpt2:
  2 1 CONFX : printer port = com1:
  3 1 CONFX : printer port = com2:

  0 2 CONFX : plotter port = lpt1:
  1 2 CONFX : plotter port = lpt2:
  2 2 CONFX : plotter port = com1: (default)
  3 2 CONFX : plotter port = com2:

  0 3 CONFX : no degrees and minutes on chartwheel
  1 3 CONFX : degrees and minutes on chartwheel

  N 4 CONFX : kundig section fraction used: N could be 10, 20, 40 etc.
  0 5 CONFX : latin sign names
  1 5 CONFX : national sign names

  0 6 CONFX : EPSON printer with n/180 inch line spacing
  1 6 CONFX : EPSON printer with n/216 inch line spacing
```

Notes:

!!It was the intention to include Chiron in PCA ver 2.2.8, but the we could not obtain the accuracy, we wanted. The Chiron setting here is VERY inaccurate. Even in this century, it may be several degrees off, so don't use it for serious work.

The true equation for Huber AP uses the Koch house equations rather than the interpolation which is used by the Huber school. Using the true equation seems more "mathematical correct", and makes the movement of the AP more smooth.

The english version of PCA does normally not have the Kündig section option (XK from the main menu), but may get it using the CONFX code.

Page number suppressing may be obtained by setting the formfeed system in the installation menu to "none", but this will turn off the pageing mechanism completely. The setting here is used if you just wish to avoid the numbering.

Ayanamsha is the difference between the tropical and siderial zodiac. The hindu astrologers use the siderial zodiac, which may be set using the AYA code. This will normally be turned off automatically when returning to the PCA main menu. You must set the CONFX also to allow keeping the ayanamsha for the main program as well. Then all printouts and chartwheels will print with siderial positions.

Kündig sections are normally one tenth of the arc between the rulers of the MC/IC. This is default and obained by using 0 4 CONFX or 10 4 CONFX. You may however set it to 20 or 80 or really any value.

The CONFX code returns the old value. This allows you to test, for example just getting the value of CONFX 0 without changing it could be done like this:

```
0 0 CONFX 1 DUP 0 CONFX DEC
```

!!If you want to set or reset one bit in CONFX 0 the following codes will do just this:

```
0 0 CONFX 128 OR 0 CONFX DEC ;sæt bit 7 i CONFX 0
0 0 CONFX  64 CPL AND 0 CONFX DEC   ;sæt bit 6 i CONFX 0
```

CPL

Stack: X CPL → not(X)

Bitwise complement. This is a highly technical code changing all zero bits to ones and vice versa. To analyse the result, you will have to convert X to binary, i.e. to 16 ones or zeroes.

DDIF (Date difference)

Stack: X Y Z P Q R DDIF → (difference)

Calculates the distance in days between two dates:

X = date1 - years
Y = date1 - months
Z = date1 - days
P = date2 - years
Q = date2 - months
R = date2 - days

The difference is date1-date2, for example:

1950 3 6 1950 4 6 DDIF → -31 (days)

The difference may be up to +- 32767 days. If the distance is greater, you will get the value 32767. This code is used in the biorythms module to find the age in days.

DEBUG

Stack: DEBUG →

This code is used by the XLI programmer to check the effect of his coding. If you are in doubt whether a bit of coding is doing what you intend, you may temporarily insert this code just before the coding to examine.

As soon as the interpreter reaches DEBUG it switches to single step mode. This means, that it will execute just one instruction code at a time, then waiting for (any) keypress to go on with the next.

In single step mode, the screen will for each instruction display the instruction name and the topmost values on the stack. So you may follow step by step what your coding is doing.

To leave single step mode, just press ESC, and your module will execute on at normal speed.

DEC (decrement stack pointer)

Stack: X Y DEC → X

Decrement code will remove the upper number on the stack. This is useful if some superfluous stuff must be disposed of. For example:

X Y Z STO → X Y
X Y Z STO DEC → X

The first line will store Y in memory cell Z, but Y will still be left back on the stack. If you do not need Y any more but the X beneath, the second line shows how DEC will dispose of Y.

DELAY

Stack: X DELAY →

Pause for approximately X milliseconds. This may be useful for demos or messages and displays which should just be viewed for a short while.

Note that the milliseconds measure is not exact, but depends on your computer. 486's seem to have the double speed, so for instance 2000 DELAY causes a pause of one second instead of two.

DFAT (Draw fat line)

Stack: X1 Y1 X2 Y2 N D DFAT →

Draw a multiple line (like the ones used for exact aspects) from X1,Y1 to X2,Y2, consisting of N parallel lines D units apart. D should be only a few units to simulate one thick line.

DIV

Stack: X Y DIV → X/Y

Divides X by Y. If Y is zero, you will get an error message.

DIVR (divide with remainder)

Stack: X Y DIVR → X mod Y X/Y

This is a combined division and modulus function. It is equivalent to X Y 2 DUP 2 DUP MOD 3 FETCH 3 FETCH DIV.

If Y is zero, you will get an error message.

DNORM

Stack: DNORM →

This code will compensate for out-of-range changes in the menu value, and at the same time set the time and place values to calculate planets etc. using code HOUSE and PLA etc.

It is possible to add a value to day, month, year, hours, minutes etc. and if they get out of range, e.g. hours exceeds 24, the complete set of figures will be rearranged. You may add or subtract to days, years, hours, minutes or seconds, but not months. Note also, that you may not exceed the 32767 integer range limit.

You may also use this to check if an entered day is valid:

```
$
0 MEGET 1 MEGET 2 MEGET DNORM
2 MEGET =
XY 1 MEGET = AND
XY 0 MEGET = AND

Date is valid
$
NOT

Date is invalid
$
```

DRAW

Stack: X Y DRAW →

Move the pen or draw a line to position X,Y. If it is the first DRAW instruction at all or the first after a PENUP instruction the pen will move to X,Y without drawing, else it will draw a line to X,Y from where the last DRAW instruction left the pen.

Example: To draw two rectangles you may code:

```
$
GRON
```

```
3 GRCOL
-800 -600 DRAW
-200 -600 DRAW
-200  600 DRAW
-800  600 DRAW
-800 -600 DRAW
PENUP
 800 -600 DRAW
 200 -600 DRAW
 200  600 DRAW
 800  600 DRAW
 800 -600 DRAW
PENUP
WKEY
GROFF

$
```

DRSYM (Draw symbol)

Stack: Z R S X Y DRSYM →

Draw a symbol from the PCA symbol table:

Z is the symbol number
R is the rotation
S is the size
X and Y is the coordinates where the symbol center should be

The following code will print all the symbols available:

```
$
GRON
3 GRCOL
0 9 FOR 0 9 FOR
1 CNT 2 CNT 10 MUL ADD
0
100
1 CNT 150 MUL 750 SUB
2 CNT 150 MUL 750 SUB
DRSYM
NEXT NEXT
WKEY
GROFF

$$$
```

The module **DRSYMS** will print all the available symbols with their numbers.

DRTXT (Draw text string)

Stack: Z R S X Y DRTXT →

Draw string array index Z:

Z is string number
R is the rotation
S is the size
X and Y is the coordinates where the first symbol center should be

---

DRWTT

M C Z T X Y S DRWTT ;draw text using tt-font

This code inserts text into a graphic. You can also use DRTXT to do that, but in that case you have only the plotted symbols whose shapes are defined in PCA.CFG. This code uses the current font, but text cannot be rotated.

M is the mode. The mode defines how the text is justified.

0 1 2 3: Horizontally right adjusted to y-axis
4 5 6 7: Horizontally centered around y-axis
8 9 10 11: Horizontally left adjusted to y-axis
12 13 14 15: Horizontally window centered
0 4 8 12: Writing above x axis
1 5 9 13 Writing on X axis (vertically centered)
2 6 10 14: Writing below x axis
3 7 11 15 Writing at bottom of screen

C is the color using the 000-888 RGB color system of Argus.

Z is the font size, not points but the Argus graphic units, so that 2047 is the full window heigth;

T is the font style, 0=normal 1=bold, 2=italic, 4=underline. The figures may be added to create compound values, e.g. 1+2=3 means bold-italic.

X and Y define the text placement in the graphic coordinate system, i.e. 0,0 is window center, and +-1024 are the screen edges.M C Z T X Y S DRWTT ;draw text using tt-font

---

DUP

Stack: X Y DUP → X (duplicate)

Duplicates one of the numbers of the stack. To duplicate the uppermost, use 1 DUP, to duplicate the next use 2 DUP etc.

Examples:

X Y Z 1 DUP → X Y Z Z
X Y Z 2 DUP → X Y Z Y
X Y Z 3 DUP → X Y Z X
X Y 2 DUP 2 DUP → X Y X Y

ELSE

The ELSE code can now be used in IF..ENDIF constructions.

ENDIF

Stack: ENDIF →

Description: Ends the current IF condition. You must always have a matching number of IF's and ENDIF's. See code IF for further explanation.

EXEC

Stack: EXEC →

Call external program. This may be any EXE or COM file. Note however, that it must be small enough to fit in memory together with PCA. PCA uses approximately 300 kb, so if your system leaves 500 kb for programs, you will have 200 kb left for your external (user) program.

You may use this facility for example for calling the DOS editor to edit you files. The following small file with the name TTUZ.XLI, will do exactly this (assuming you are using MS-DOS ver 5 placed in a subdirectory called \DOS):

```
$
EXEC

\DOS\EDIT.COM
$$$
```

To call the editor from the PCA main menu, you just press XZ.

If you are writing programs yourself in PASCAL or C you may write programs, which you can call from the PCA menu, and with which you may exchange information. A small interface in your coding will make read/write access the memory cells and the string array in the XLI interpreter small interface in your coding. There are two examples given with this toolkit, which you may expand to your needs: XUSER.C and XUSER.PAS written in Turbo C and Turbo Pascal. The XUSER.XLI will provide the calls. An XUSER.EXE file is also provided, which is the PASCAL version compiled.

The parameter transfer works like this: When calling the external program, you must put @-sign after the EXE filename:

```
$
EXEC
XUSER.EXE @
$
```

This will make the XLI-interpreter write two memory addresses on the command line when calling the external program. These pointers may be retrieved and converted from the ASCII representation to the languages internal notation.

The actual commandline which the XLI interpreter !!creates for the call may look like this:

```
XUSER 23891   16542   23891   22562
```

The four numbers are:

Segment of the memory cell array start
Offset of the memory cell array start
Segment of the string array start
Offset of the string array start

The handling of these pointers and examples of read and write of the registers are given in the XUSER.C and XUSER.PAS source files.

The memory cell array is signed integers represented by two bytes each, first the low order, then the high order.

The string array is in fact one string of bytes starting with an unused length byte. Then comes the actual (sub)- strings, separated by a TAB (chr 9) character.

If you need to run bigger programs than can fit in together with PCA, you may put an amphersand in front of the file path. This will make PCA swap to disk, leaving nearly all available memory to the program called. In this case, the parameter exchange does not work, because the adresses of the number and string array is no longer valid. If you need to transfer data, you must save and retrieve them using the disk.

Turbopower Software execswap routines are used for the swapping.

FEED (formfeed)

Stack: X FEED →

This code will perform a page shift (formfeed) on printer outputs if less than X lines remains on the current page.

If you are going to print a table or anything else, which you do not want the risk of having cut anywhere, use this code to assure that the following X lines are not cut by a page break. X must be a number between 1 and the maximum number of written lines on the page as set in the PCA printer installation menu.

FETCH

Stack: X Y FETCH → X (fetched)

Reorders the stack by fetching the value Y steps down the stack and putting it on the top. Examples:

X Y Z 2 FETCH → X Z Y
X Y Z 3 FETCH → Y Z X

1 FETCH has no meaning, as it will just fetch the uppermost, and put it at the same place.

FONT

Select font number defined with FONTS. 3 FONT select font 3, 13 FONT selects font 13 and so on. The font is set for the text screen. If a graphics screen is selected, it is set both for the text screen and for the graphics screen where it can be used by the DRWTT command.

FONTS

This command lets you define up till 15 fonts to use in your interpretation. A font is defined by its size, style, color and name. If you just want to use colors, you can define a number of fonts with the same name, style and size and give them different colors. As soon as they are defined, you can switch between them in you text. Here is an example of defining 3 fonts. You must put the number of fonts you want before the FONTS code and provide the same number of lines in the text below holding the definitions:

3 FONTS

9 0 0 Arial
9 1 0 Arial
10 0 800 Argus

This defines font 0 as Arial with size 9 points normal style and color black.

Font 1 is defined as Arial size 9, Bold and color black.

Font 2 is defined as Argus symbol font size 10 normal style and color red.

When defined, you can choose font 0-7 inside the text using the characters number 184-191. If you have defined more than 8, fonts 8-15 can only be selected using the code FONT, which works for a whole paragraph.

You may leave out the font name. In that case the default font typeface is used as setup in the Argus preferences.

FOR

Stack: X Y FOR →

This is a looping instruction, so that you may have your code execution repeated a number of times. It works together with the code NEXT.

The loop is running a number of times determined by X and Y. A counter starts at X and ends at Y. The counting may go either up or down so both 1 8 FOR and 8 1 FOR are valid. The loop will always run at least 1 time, so for example 2 2 FOR will run 1 time, 2 3 FOR two times etc.

Example: 0 10 FOR 0 1 CNT STO NEXT

will fill the value zero into memory cells 0-10 inclusive.

You may have several loops inside each other. For example, this coding will calculate the number of planetary aspects between two charts:

$

```
0                           ;start value
1 10 FOR                    ;radix2 planet count
    20 30 FOR               ;radix planet count
        1 CNT 2 CNT ANUM    ;test aspect number
        BOO ADD                   ;increment if not zero
    NEXT                    ;end radix count
NEXT                        ;end radix2 count
```

FPOFF (turn floating point mode off)FPON (turn floating point mode on)

see also FPON

FPON (turn floating point mode on)

Floating point mode means, that the stack will operate on a floating point value basis instead of the usual integer.

The floating point stack is not the same as the integer, and it will contain different values. To operate the floating point stack, you also need to use special floating point operators. So MUL and ADD does not work. There is a whole set of fp-codes available. These are mainly taken over from the PCM (cosmic mirror) interpreter and used together with this.

If you insert a floating point constant in your coding, that is a number with a decimal point, it will be put on the floating point stack instead of the integer stack irrelevant of the current stat of FPON FPOFF.

The whole issue of using floating point in ARGUS is still not developed into a really useful feature. Please do not use it relying on that it will stay unchanged in the future.

see also FPOFF

FUNX

Stack: X FUNX $\rightarrow$

Call a single PCA output menu function, that is, those who normally needs pressing a letter key, e.g. Radix, Aspects etc.

X must be the ascii-value for the letter you would have pressed in the menu, e.g. 65 for A (aspects) 80 for P (progressed) etc. Consult an ascii table to find the numbers you need.

To call the extra functions: XP, XN, etc. use lower case, that is p (112), n (110) etc.

This code is for single function calls only. It will usually be a more flexible solution to use the CALL function instead, which is more flexible and will execute a complete macro.

FTOI (convert floating point value to integer)

This pops a value off the floating point stack, converts it to integer and puts it on the integer stack.GET

see also ITOF

GET

Stack: X GET → (value at ZZO marker + X)

Gets the value X positions from the ZZO marker on the stack. You must at some point have set the ZZO marker before you use GET and PUT.

Examples:

ZZO 3 7 2 0 GET → 3 7 2 3
ZZO 3 7 2 1 GET → 3 7 2 7
ZZO 3 7 2 2 GET → 3 7 2 2

GMODE

n GMODE (change currently open graphic window mode)

mode 1 normal mode (show while drawing)
mode 2 hidden mode (draw on buffer)

Mode 2 means, that screen update is overwriting, not merging, so that a movie-like effect is obtained.

GRAF

Stack: X Y Z .... N GRAF

This code is used to produce horizontal bar graphs. The bars will have lenghts showing the size of numbers, e.g. element strengths, planetary power or whatever. The numbers to show graphically must be pushed on the stack (in reverse order as usual).

The graphs will appear in the text area starting where the character # is found. So where you want a bar to start, place a # and leave the remainder of the line blank, because it will be overwritten by the graph anyway. The bar lines can be mixed with lines, rulers etc., any text to explain or enhance the appearance.

There must be the same number of lines with a #-character as you want bars, and there must be the same number of numbers pushed on the stack (X Y Z..)

The numbers to show must be limited to positive numbers maximum 80, else you will get a range check error message. If the graph starts in a column other than 1 (as in the example below) the remainder of the line will be less than 80, and the number entered should be limited adequately.

After the numbers to show, you must stack the ASCII code of the character, with which you want to build the bar, typically a block graphics character (e.g. 220)

Example: Print a bar graph of the numbers in register (RCL) 20-23:

```
$
23 RCL 22 RCL 21 RCL 20 RCL 220 GRAF

----------------------------------------
```

```
Cell. 20 #
Cell. 21 #
Cell. 22 #
Cell. 23 #
---------------------------------------
$
```

GRCOL (Graphics colour)

Stack: X GRCOL →

Set graphics colour. This sets the colour used for any following graphics output. X is a number between 0 and 39. It represents the colours chosen for the different items in the PCA colour installation menu. X=0 will thus set the colour chosen for the text background, X=7 to the colour chosen for Taurus etc. The following is a complete list of colours:

0 : Text background
1 : Text foreground
2 : Graph background
3 : Graph main line
4 : Graph angles
5 : Graph houses
6 : Aries
7 : Taurus
8 : Gemini
9 : Cancer
10 : Leo
11 : Virgo
12 : Libra
13 : Scorpio
14 : Sagittarius
15 : Capricorn
16 : Aquarius
17 : Pisces
18 : Sun
19 : Moon
20 : Mercury
21 : Venus
22 : Mars
23 : Jupiter
24 : Saturn
25 : Uranus
26 : Neptune
27 : Pluto
28 : Node
29 : Conjunction
30 : Opposition
31 : Square
32 : Trine
33 : Sextile
34 : Semisquare
35 : Sesquisquare
36 : Inconjunct
37 : Semisextile
38..39 : unused

So you will not be able to set the colour to specifically "red" or "pink" or any absolute colour, you must select from the colour set already chosen in the colour installation menu. If however, you need a special colour setup, you may temmporarily create this suing the COL code. Note also, that the colours may differ on the screen and printer due to the installation colours setup.

GROFF (Graphics off)

Stack: GROFF →

Turn off graphics mode. For screen output this will clear the screen, and go back to text mode. For printer output the accumulated graphics will be printed.

!! When XLI returns to PCA, a GROFF is automatically executed. Therefore, do not state GROFF unless you wish to continue XLI in text mode. Automatc GROFF has the advantage, that the program waits for you to press key without whiping the screen.

GRON (graphics on)

Stack: GRON →

This code switches the PCA output to graphics. The PCA has two output modes which works both for screen and printer: textmode and graphics mode. In text mode you may write to ASCII files and work freely with any kind of text. In graphic mode you may draw lines and symbols, but you cannot write text normally. If you need to print numbers or letters, you will have only a limited number of symbols, and you will have to DRAW them, that is, for each symbol you must specify placement, rotation and size. The symbols available are very simple stroked ones with no fill. That means, that on coarse screens they may turn into unreadable blots, and on highresolution printers they will look very light and thin.

The screen will clear and change to graphics mode.

The printer will seem to do nothing. When printing graphics, the computer gathers graphic information until the GROFF instruction is met. First then will the graphics be printed. The maximum of graphic information, that the program can hold is 12000 single strokes. A line is one stroke, a symbol may consist of several strokes. If this limit is passed, only the first 12000 strokes will print, the remainder will be disposed of.

The graphics coordinate system has 0,0 on the middle of the screen (or paper) and X and Y may have values between + and - 1024.

GTR (Graphic transits)

Stack: X Y GTR →

You may use this code to create a customized version of the graphic transits feature. The parameters you may change are the following:

1: Orb
2: Time slice
3: Calendar scale layout
3: Which transit and radix positions to include

4: Where to put horizontal lines

X is the time slice, calculated as 365 / X days.
Y is the orb in minutes of arc.

The other parameters are read from the first 3 lines in the text field:

!!1.st line (choice of planets).

Planets are assigned letters, so that:

		Radix	Transit
Sun		A	a
Moon		B	b
Mercury	C	c	
Venus		D	d
Mars		E	e
Jupiter		F	f
Saturn		G	g
Uranus		H	h
Neptune		I	i
Pluto		J	j
Node		K	k
Part of fortune		L	
MC		M	
ASC		N	
11. house		O	
12. house		P	
2.house		Q	
3. house		R	

Note that you may not use a transit position of higher than the Moon's node.

The planets setup are in groups of five characters, for example:

```
1MNfk
```

This specifies the aspects to consider.

MN means: count from radix MC to radix ASC
fk means: count from transit Jupiter to transit Moon's node

The succession will be: First all aspects to radix MC, then all aspects to radix ASC. The last to letters are the "inner loop" or the fastest counting series. If you want the transits in the outer loop, that is, first all aspects from transit Jupiter, then from transit Saturn etc. you must put fk before MN:

```
2fkMN
```

Note the number before the block, that was 1 in the first example and 2 here. If you want the planets in the inner loop to be quoted first, write 2, else write 1. For example:

```
1fkMN   MC tri Jupiter ......
2fkMN   Jupiter tri MC ......
```

Normally you will want the transit first, so put 2 if you write the transits first and 1 if you write the radixes first.

You may put several blocks of 5, so you may have several sets of planet counts after each other in the same printout. For example if you want the transits Sun-Mars, but not Moon, you must setup first a loop for the Sun, then for Mercury to Mars:

```
1AKaa1AKce
```

You may put a line between the blocks by inserting an 'L':

```
1AKaaL1AKce
```

This will print first transit Sun to radix planets, then a line, and then transit Mercury-Mars to radix planets.

The planet setup line must be no more than 80 characters total.

!!2. Layout string:

The layout string has two purposes:

1: Determine the time span which is the number of characters times the time slice defined by the GTR instruction. For example if Y is 60 the time slice will be 365/60 = approx 6 days. If the string length is 60, you will have exactly one year of transits.

2: Design the ruler, i.e. the characters to appear, between the aspect markers (the shaded characters).

Below an example of a year transit ruler:

| . . . . | . . . . | . . . . | . . . . | . . . . | . . . . | . . . . | . . . . | . . . . | . . . . | . . . . | . . . . |

and a month transit ruler

. . . . | . . . . . | . . . . . | . . . . . | . . . . . | . . . . . | .

!!3. The date marker line:

The third line determines where to put the date indicators in the header line. It may contain the following characters:

Y: Write the year at this position and increment year count
y: Just increment year count.
M: Write the month at this position and increment month count
m: Just increment month count.
D: Write the day at this position and increment day count
d: Just increment day count.
>: Start all the aspect rulers just after this position

Note: Y,M and D uses the starting year, and does not automatically take their position into account. Therefore the increment is necessary for possible further printouts. If you are not printing all increment steps, you must place lowercase letters in between to get the count correct.

The counts will just continue, so do not rely on correct calendar adjustment. However, if a month exceeds 12 its count will start over from 1, and the new year will be printed at its place (without updating the year count). See the XY option in PCA to view the effect.

Below is shown the example of a month transit date marker line. Only every 5 dates are shown.

```
M     Y         >D  d  d  d  D  d  d  d  d  D  d  d  d  d  D  d  d  d  d  D  d  d  d  d  D  d  d  d  d  D
```

Study the TTRANSIT.XLI for detailed examples of alternative printouts.

HITS

T A1 A2 B1 B2 P1 P2 HITS

This code is used for calculating aspects which has culminated between two instances in time. It therefore needs four charts: A1 and B1 which are the two aspecting charts at time t1 and A2 and B2 which are the same two charts aspecting at time t2. If A1 and B1 are the same chart it will mean I-aspects and the redundants are automatically omitted. Also sign shifts, house shifts, and stations may be checked.

T is the type of events:

1: aspects
2: sign shifts
4: house shifts
8: stations

If you want combinations, these figures can be added into a compound: eg. sign shifts and stations will make T=10.

A1 A2 B1 B2 are the chart numbers. Refer to the chart index number shown under MOVCH.

P1 is the planet set for chart A, (the promissors), P2 is the set for chart B (the significators). Planet sets are set up like this: P Q R ... S N. The last number is the count, e.g. 1 2 3 3 means Sun, Moon, Mercury totalling 3. 0 6 7 8 9 10 6 means Chiron and Jupiter through Pluto totalling six.

The returned hits are placed in the memory (STO) cells in position 100 onwards in sets of 10, so the first hit starts in cell 100, next hit in 110, next in 120 etc. Cell 100 shows the total of hits. Each hit has the following format:

offset meaning

0 Number of hits left
1 Type (1=aspect, 2=signshift etc)
2 Day
3 Month
4 Year
5 A Planet no
6 Aspect number/leaving sign no/leaving house no/dir1
7 B Planet no./entering sign no/entering house no/dir2

"dir" means direction 1 for direct, -1 for retrograde. So if the second hit is: Mercury is turning retrograde, cell 115 will be 3 (Mercury), 116 will be 1 (was direct) and 117 will be -1 (but changed to retrograde).

Note: This code was introduced to produce a general output facility for aspect timing. It was abandoned again, because it performs very slowly, so it cannot compete with software having dedicated aspect schedule output. Anyway, the code is still available for experimental use.

The position indices used are:

Name	pla. no	index	used by
Current	0.. 18	0	current (earlier radixchart)
Radix	20.. 38	1	radix (earlier current chart)
Current	40.. 58	2	current (earlier auxchart1)
Aux1	60.. 78	3	xli only (earlier auxchart2)
Aux2	80.. 98	4	xli only (earlier presentchart)
Present	100..118	5	present

HOUSE

Stack: X → HOUSE

Calculate one house cusp using the menu values.

Note, that if you change the menu values using the MEPUT code, you must also use DNORM to prepare the data correctly for HOUSE.

HPOS (house position)

Stack: X HPOS → (intang position of X)

This code gives the position of house X in intang units. You may convert this value to minutes of arc using the ITOM code.

The house numbers allowed are 1-12.

HRU (house ruler)

Stack: X HRU → (ruler)

Find ruler of house no X in latest chart or latest radix chart. For a listing of the house numbers allowed, see HSI. The ruler numbers are like RUL.

HSI (house in sign)

Stack: X HSI → (Sign occupied by house cusp X)

Find the sign on a house cusp, either in the latest chart or in the latest run radix chart. The numbers allowed are: IF

	Latest chart	Latest radix chart
1. House	1	21
2. House	2	22
3. House	3	23
4. House	4	24
5. House	5	25
6. House	6	26
7. House	7	27
8. House	8	28

9. House	9	29
10. House	10	30
11. House	11	31
12. House	12	32

HV

Stack: X HV → (speed of house X)

This code gives the speed of house X measured in intang units per day/year etc, that is, per the natural unit for the chart type in question. To convert to minutes of arc, use the ITOM code. In radix, transit and return charts, where the houses move very fast, you must add 360 degrees to get the speed correctly.

The house numbers allowed are listed under HSI.

IF

Stack: A IF →

means that if A is zero, following code will stay inactive, until a matching ENDIF is met. If A is anything other than zero, the following code will work.

You must always have a matching number of IF's and ENDIF's. They may be "nested", for example IF.. IF... ENDIF... ENDIF. The "inner" IF and ENDIF are the matching pair.

You may well put an IF code in one paragraph, and the matching ENDIF in a later one. This will make the paragraphs inbetween active or inactive depending on the IF test.

IN (in group, membership)

Stack: X 0 P Q .. R IN → (1 or 0 depending on X)

Checks if X has one of the values P.....R. If so the result will be 1 (true) else 0 (false). The zero between X and the list is mandatory. The numbers in the list are strictly limited to be within 1-15 inclusive. The code is mainly used for testing if for instance the sign occupied by a planet belongs to a certain group of signs, e.g. water signs, barren signs or whatever.

The above is equivalent to:

X P =
X Q = OR
....
X R = OR

INC (increment stack pointer)

Stack: X INC → X ?

This code restores the number last removed from the stack. To use this code correctly needs some stack knowledge. For example the code IF will use and remove the number on the stack, but INC may get it back again:

X Y IF → X
X Y IF INC → X Y

This may be useful if you need the IF condition for further purposes inside the IF-ENDIF construction.

INFIL (input line of data from file)

Stack: X INFIL → (data, data, data) result code

Description: Reads one line of string and or numeric data from a text file. This could be numeric data created by an external program for graphic or tabular presentation, which PCA could output through its screen or printer drivers. Or it could be birth data imported from a database or from another file format.

If X=1: Read one line. If the file is not open, it will be opened.

If X=0: close the file (if it is open).

The file name must be given as the first line in the text field. You may have only one file open at a time. If the file is open and you state a different filename, the existing file will be used and the new name ignored.

The file is read sequentially.

A format string must be given as the second line in the text field. This will specify the line reading sequence. The data may be mixed numeric and string data. String data is put into the string array, and numeric data is pushed on the stack. Numeric data may not exceed 63 items, which is the size of the stack, and string data may not exceed 255 characters total.

If you need mass data handling, you must create a loop reading one line at the time and use the data before the next line is read.

The result code is the last number pushed on the stack. It may have the following values:

-2 : Data read incomplete, data missing in line.
-1 : End of file reached
 0 : File closed
 1 : Successful read;

The line is read from left to right using a line read pointer.

The format string may consist of the following elements:

SPACE(s) : Skip any number of contiguous spaces in the data line, so that the line read pointer points to the first non-space.

#: read one number from the data line and put it on the stack. If the line read pointer does'nt point to a digit, it will be moved forward until it does, or the data line is exhausted. Negative numbers may be read as well. If the number exceeds 32767 or -32768 it will be set to these maximum (minimum) values.

@x read string of characters ending with character (x) from data line and place it in string array 0. X may be any character (!!except @ or number digits). If the string exceeds the available space in the string array, it will be truncated. !!X itself is not read.

@nnx read string of characters ending with character (x) from data line and place it in string array nn. X may be any character (except @). nn may be any number between 0 and 99.

ITOF (convert integer value to floating point)

This pops a value off the integer stack, converts it to a floating point value and puts it on the floating point stack.

see also FTOI

JD (julian date number)

Stack: JD → (JD 10000nds) (JD ones) (JD fraction)

This will take the menu values for date, time and zone and convert them to julian date number. If you change the menu values using the MEPUT code, you must also use DNORM to prepare the data correctly for JD.

The function returns three numbers even if julian day is just one figure. This is because the XLI interpreter handles short integers only, which would overflow trying to hold a JD.

The last part, (the "JD fraction") is the time of day expressed as a fraction.

To print the Julian date and time, you may use the following code:

```
JD
NUMS

JD: #### ####.####
```

This will work for positive Julian dates only, so do not move back before JD 0.

KEY

n KEY (simulate a keypress)

Normally the XLI interpreter is not meant to operate the user interface. Rather user interface is operating the interpreter. Otherwise a power struggle could arise.

There are however exceptions to this rule. The KEY code will simulate the user pressing a key with the ASCII value n. So 13 KEY will simulate pressing the ENTER key and therefore toggle output window focus. 32 KEY will simulate the SPACE bar and therefore toggle the data input dialog box on and off. Other codes are doing other specialized jobs like:

```
255   : startxli;
254   : TerminatePCA;
253   : Window | fullsize
252   : Window | tile
251   : Window | cascade
```

250 : Window | new
249 : Window | close

KUN (Kündig sections)

Stack: KUN → D M Y HH MM SS D M Y HH MM SS D M Y HH MM SS

For users of the Kündig rectification method, this code offers the possibility of some automatizing. From the data in the main menu, the three closest Kündig sections are calculated and the results put on the stack in chronological order. Each calculation includes Day, month, year, hours, minutes and seconds. The timezone, longitude and latitude are considered constant, as given in the menu.

What is meant by the "closest" section is: The second section will always be the closest to the given time, the two other are the closest on EACH side. So there will always be at least one section before and one section after the given time.

MACn

This is a complete series of codes calling calculations and charts. MACR calls the radix, MACT calls transit, MACW calls the bi-wheel etc. It is a replacement for the more clumsy CALL and then having a macro line. For long macros and for composite macros CALL is still the prefered method, while for a single command, MAC.. is more convenient

Special MAC commands: Equivalent macro character

MAC. Clear window .
MACOF Suppress all output (
MACON restore output)

MAX

Stack: X Y MAX → (X or Y whichever is the largest)

Chooses the largest of two numbers and disposes of the other.

MEGET (Menu Get)

Stack: X MEGET → (menu value)

The PCA input menu consists of a number of integers representing the inputted data. With this code and the corresponding MEPUT, you may manipulate data in the input menu. The menu values are:

0 MEGET → Date, Day
1 MEGET → Date, Month
2 MEGET → Date, Year
3 MEGET → Date, 0=AD 1=BC
4 MEGET → Time, hours
5 MEGET → Time, minutes

6 MEGET → Time, seconds
7 MEGET → Sex, 0=male 1=female
8 MEGET → Timezone, hours
9 MEGET → Timezone, minutes
10 MEGET → (always zero)
11 MEGET → Timezone, east=0 west=1
12 MEGET → Geo latitude, degrees
13 MEGET → Geo latitude, minutes
14 MEGET → (not used)
15 MEGET → Geo latitude, north=0 south=1
16 MEGET → Geo longitude, degrees
17 MEGET → Geo longitude, minutes
18 MEGET → (not used)
19 MEGET → Geo longitude, east=0 west=1

The menuvalues marked (not used) are still set to something when you enter data. For example if you are entering data for harmonic and type: !!23 0 21 -, then menuvalues !!26-27 will hold the extra (not used) numbers 21 1. The minus produces the 1 even if harmonic may not be "north/south" or whatever. It is just, that the mechanism is equivalent. So here you have a secret and hidden way of passing a number from the main menu to the XLI mechanism.

The menuvalues 3,11,15,19,23 and 27 are set to either one or zero when entering data. Menuvalue 7 is set to 0 or 1 by toggling the change-sex key (/).

The shown parameter values work on the "current" data-input card. You may also access the "radix" and "present" data input values by adding 20 or 40: 20-39 will access the radix data and 40-59 will access the present data.

MENAM

n MENAM (insert name in string n)

This is similar to the NAME code, but the name is taken from the current input menu, while NAME takes it from the current saved chart. So if the user enters new data, NAME will not fetch the new name until a chart is calculated, while MENAM will. MENAM is equivalent to MEGET. If n is negative, the name is moved from the string n back to the input menu.

MENU

Stack: MENU → 1

This code in combination with the code OPT lets you setup choice menus. First you set up a paragraph having just the code MENU in it code part. This will temporarily switch off any printer output and print the text part, which will be printed as it is.

The menu text, as in fact any menu, could be thought of nothing but a personal note, reminding you what the following keypressed will do.

The following paragraph must hold the code OPT to stop the program waiting for your keypresses and if needed switch the printer on again.

It is possible to insert stored values and text strings into the menu using the NUMS code (see this). If you do this, you should put MENU first, then the NUMS code.

An example of this is the toolkit version of TTFLET.TXT, which puts the word "EDITING" into the menu, if this choice is made.

MENUX (XLI defined dialog box)

This code sets up a dialog box with buttons, edit fields, labels, checkboxes and radiobuttons as required. When the user closes the box, the return value is pushed on the stack, and any other values or entered strings are transferred to where they were taken from. Pressing ESC (or clicking a cancel button if present) will leave old values as they were.

You may edit strings in the XLI string list, but you cannot edit numbers in the STO array.

MENUX takes a large number of arguments to define its controls. The format is:

kind left top height width key valueindex reserved reserved reserved
kind left top height width key valueindex reserved reserved reserved
kind left top height width key valueindex reserved reserved reserved
kind left top height width key valueindex reserved reserved reserved
.....
n width height caption
MENUX

The first n lines define one control each.

The last line before MENUX define the number of controls and the size and caption of the dialog box itself. The dialog box will always appear on the screen center.

The arguments for each control are:

kind: 255 Edit box
    254 Label
    253 Checkbox
    252 Radiobutton
    240..251 reserved, not used
    0..239 Button

Buttons kind-numbers must be different to distinguish which button has been pressed. This number will be the one which is pushed on the stack when the menu box is closed. If the kind number is nonzero, clicking that button will cause the dialogbox to close.

left, top, height, width: Placements and dimensions of the control relative to the dialog box measured in pixels.

key: Button: if set to 1 this means, that pressing the ENTER key will will be the equivalent to clicking this button. If set to 2 this means, that pressing the ESC key will have same effect as clicking this button. If more than one button on the dialog box have their key=1 or more than one button have key=2, pressing ENTER or ESC will not know which button to activate, so this should be avoided.

Radiobutton or checkbox: STO cell number from where to get or put the button status: nonzero=on, zero=off.

Other controls: no effect.

valueindex: This is the XLI string number from which a text is taken to use as:

    Button: Caption

Editbox: Default input text
Label: Label caption
Checkbox: Checkbox caption
Radiobutton: Radiobutton caption

reserved: not used, could be set to zero. In future PCA version, other arguments may be needed, so these three position are reserved for this, so that it will not be necessary to change the format and cause backwards incompatibility.

The last definition line holding information about the dialog box itself has these arguments:

n: The number of controls, i.e. the number of lines above to use. Note, that there may be a restriction on the size of n accoring to the size of the XLI stack. Each line holds 10 values, so the number of controls allowed is (stacksize-4)/10. The stacksize is currently set to 1024 allowing for 100 controls.

width,height: Width and height of dialog box. It may take a bit of experimenting to get this right and place the controls to look right.

caption: The index of the XLI string list to use as text on the dialog box's caption bar.

Functionality:

When the menu definition is set up, and the XLI meets the MENUX code, it will create the dialog box according to the above definitions. The user can then press buttons, edit editfields etc as needed.

To close the box, the user must click a button, which has kind neither zero or 2 Or by pressing ENTER if any button has kind<>0 and <>2 and key=1.

If the box is closed with a button of kind<>2 (not an ESC-button), the values in any editbox will be transferred back to the string list item it was taken from (defined by the ix argument). If checkboxes or radiobuttons are present, the result value (1 or 0) will be transferred back to the STO cell, from which their initial value was taken (defined by the "key" argument).

To escape the box without making any changes, it can be closed using the closebox on the caption bar, by pressing ALT-F4 or by pressing a key whose kind<>1 or by pressing ESC and a key has kind=2 and key=2. Such a key should be labeled "CANCEL"

If any character in the range 128..191 is used for labels, buttons etc. the ARGUS font is used, so it is possible to insert planet symbols etc.

menubox test example 1;

```
$
1 CARY  ;Sun is written with & and letters, to show that ALT-U can be used

,S&un,,,ƒ,„,…,†,‡,ˆ,‰,Š,Cancel
$
  3  10  50  50 20 0  1 0 0 0   ;Button (1 and 2 are reserved for OK and CANCEL)
  4  10  75  50 20 0  2 0 0 0   ;Button
  5  10 100  50 20 0  3 0 0 0   ;Button
  6  10 125  50 20 0  4 0 0 0   ;Button
  7  10 150  50 20 0  5 0 0 0   ;Button
  8  10 175  50 20 0  6 0 0 0   ;Button
  9  10 200  50 20 0  7 0 0 0   ;Button
 10  10 225  50 20 0  8 0 0 0   ;Button
 11  10 250  50 20 0  9 0 0 0   ;Button
 12  10 275  50 20 0 10 0 0 0   ;Button
```

```
  2  10 310  70 30 2 11 0 0 0   ;CANCEL-button
11 350 350 0 MENUX 2 SUB
NUMS

ModalResult= #######
$$$


;menubox test example 2;

$
1 11 STO
0 12 STO
0 13 STO
1 14 STO
2 CARY

TEST XLI-DIALOG BOX,STO 11,STO 12,STO 13,STO 14
String 6,String 6 to edit,OK,CANCEL
$
253  10  25 150 20 11 1 0 0 0 ;checkbox
253  10  50 150 20 12 2 0 0 0 ;checkbox
252 170  25 150 20 13 3 0 0 0 ;radiobutton
252 170  50 150 20 14 4 0 0 0 ;radiobutton
254  10  75 150 20 0  5 0 0 0 ;label
255 100  75 200 20 0  6 0 0 0 ;editbox
  1  50 125 100 30 1  7 0 0 0 ;OK-button
  2 200 125 100 30 2  8 0 0 0 ;CANCEL-button
8 350 200 0 MENUX
14 RCL 13 RCL 12 RCL 11 RCL
6
NUMS

String 6 after edit: @@@@@@@@@@@@@@@@@@@@@@@@@@@@

STO values:                    11                    12                    13                    14
         ####    ####    ####    ####

ModalResult= #######
$$$
```

MEPUT (Menu Put)

Stack: X Y MEPUT →

The value X is forced into menuvalue number Y. Y must be between 0 and 19.

See code MEGET for an explanation of the menuvalues and index.

If you change date and time, you should use the code DNORM as well to normalise the date if out of range, and to make the system accept it as the "current time" for a number of functions.

MIN

Stack: X Y MIN → (X or Y whichever is the smallest)

Chooses the smallest of two numbers and discards the other.

MOD

Stack: X Y MOD → (X+Y) mod Y

The modulus returns the remainder, when dividing X with Y. For negative numbers less than Y, it works non-standard, and will still return a positive number. If Y is negative, you will get an error message.

MONS monospaced

1 MONS changes output to monospaced, meaning that all characters are forced into position, even if they are proportional. This may cause wide proportional characters to overlap.

0 MONS changes back to using the fonts own character spacing.

MOVCH move chart data around

When calculating charts, all chart information is saved in a data structure. There are 5 such structures:

Name	pla. no	index	used by
Current	0.. 18	0	current
Radix	20.. 38	1	radix
Current	40.. 58	2	current
Aux1	60.. 78	3	xli only
Aux2	80.. 98	4	xli only
Present	100..118	5	present

So when any chart is calculated, its data is saved in struture current, except the horary clock, which uses its own structure (present). Calculating a radix, will be saved in both Current and Radix. Later calculations of chartwheels, aspects, midpoints etc are using these data.

With MOVCH you can move data around. For instance 1 3 MOVCH will move the data from Current into Aux2. The following example demonstrates this by swapping the radix and the current data, so that calling the bi-wheel will have radix positions in the outer wheel and current positions in the inner:

```
0 3 MOVCH      ;move current to aux1
1 0 MOVCH      ;move radix to current
3 1 MOVCH      ;move aux1 (former current) to radix
MACW           ;call macro bi-wheel
```

The two aux charts are for temporary storage. However, you may access the positions etc using expanded indexes. It was always possible to use e.g. 5 PPOS to get current Mars, and 25 PPOS to get radix Mars. But now you can get Aux1 Mars writing 45 PPOS, Present Mars writing 85 PPOS etc. This applies to all the codes, where you normally would add 20 to get the radix data. The chart type list shows the planet numbers to use. The value Index is used for the HITS code.

MUL

Stack: X Y MUL → X*Y

Multiplies two numbers.

MULT multiply by fraction

Stack X Y Z MULT → X * (Y/Z)

This arithmetic routine will multiply X and Y as longints to avoid overflow, and then divide by Z. The result will be converted back to short integer which may or may not produce an overflow, but this is the responsibility of the programmer.

The routine is useful for scaling coordinates.

NAME

Stack: X NAME →

Insert the current name into string array number X.

NAX (extended namefile)

Stack: NAX →

This is a highly specialized code used in connection with the namefile extension produced by the CQN program. This program is not part of the ordinary PCA package and must be ordered separately if needed.

The CQN program converts a comma-and-quote file holding names and birth data into a valid PCA namefile. But CQN also creates an extra namefile: NAMEX.DAT which holds additional information taken from the same comma-and-quote file, e.g. addresses etc to print directly on interpretations etc. to mass process chart mailing.

To access the data in NAMEX.DAT, use the code NAX which inserts the NAMEX data into the string array. Then you may print it out using NUMS and inserts rows of @@@@@'s into the text field.

NAMEFILE.DAT and NAMEX.DAT are not connected in other ways, that they are created with the same number of names in the same succession. If you change NAMEFILE.DAT, it will no longer match NAMEX.DAT. If NAX finds a name mismatch, the string array will be left empty.

NDATE

Stack: X NDATE → (year) (month) (day)

This is a code specialised for interpretation of graphic transits. When using the code NTA, you will get start and end time for an aspect. These times are given as sector numbers. This code translates the sector number (X) to day, month and year.

Example:

```
$
NTA XY NDATE 4 FETCH NDATE NUMS

From: ## ## ####   to: ## ## ####
$
```

The dates will be approximate, if the time sectors are more than 1 day. The yearly transits use 6 day sectors (intervals), whereas the monthly transits use half-day intervals.

NEXT

Stack: NEXT →

This code ends a FOR loop. There must always be a corresponding number of FOR's and NEXT's.

NFI (Namefile index)

Stack: X NFI → Y

Move data from namefile entry no. X to the input menu.

The result (Y) should normally be zero unless X points beyond the last entry or the namefile entry X has a blank name. In that case Y will be the size of the namefile. The namefile pointer itself will not be moved.

The function may be used to find the size of the namefile:

```
1000 NFI
```

You may also use the function to selectively fetch a specified entry or series of entries.

If you insert a series of events into the namefile and terminate with a blank name, you may let the program run a series of progressions for these entries and stop when the blank is met:

```
$
0 NPNT 1 DUP NPNT DEC 1000 FOR      ;count from current entry
1 CNT NFI                    ;get entry
XIF                          ;exit loop if blank
CALL                         ;call progressed chart

P
$
NEXT
```

$$$

NFN (next file name)

Stack: X NFN

File branching. If a module consist of more than one file, which is the case in most interpretations, you normally put the name of the next file in the chain after the three $$$-signs at the end of the file. But you may put more than one filename, using one line for each, and use the NFN code to select which file to branch to. 1 NFN (default) will branch to the first file, 2 NFN to the filename on the second line after the $$$ signs etc. 0 NFN will return to the main program, and the same will for instance 5 NFN if there are less than five filenames in the list.

The NFN code may appear in any paragraph in the file, the value will be remembered when the $$$-signs are met.

NOT

Stack: X NOT → not X

Checking if a condition is false. For example to check if X is "greater than or equal to 5" is the same as "not less than 5", and you may code like this:

```
X 5 > NOT
```

NPNT

Stack: X NPNT → Y

This code changes the namefile pointer and hands back the former pointer position. If the pointer points to entry 13 and you write:

```
0 NPNT
```

the result will be 13 and the namefile pointer will be changed to entry 0.

If you just wish to know the pointer position without changing it, you must write:

```
0 NPNT 1 DUP NPNT DEC
```

You cannot set the pointer outside the size of the namefile. A negative number will put the pointer to zero and if you put a number greater than the namefile size, the pointer will be put at the end of the file.

Note, that putting the pointer to entry zero works only until the main menu returns. At this point, entry zero will be loaded into the input menu, and the pointer will move automatically to entry 1.

NPR (birth name)

Stack: NPR →

Print current name using one line.

NPUT (Name Put)

Stack: X NPUT →

Inserts a name into the PCA input menu taken from string array number X.

NTA

Stack: NTA → S E T R A

where S = startsector
E = endsector
T = transitplanet
R = radixplanet
A = aspect number

This is a code specialised for interpretation of graphic transits. It will fetch the next aspect in the list of transit aspects. The aspects are those calculated in the latest graphic transit calculation. If the list is exhausted (no more "next" aspects) S will be 0. This also applies if no graphic transits have been calculated yet, since PCA was started.

The sector numbers S and E will be a number between 1 and 61. If you have setup your own graphic transits using the code GTR the range may be different, starting with 1 and ending with the number of timeslices defined.

The transit planet (T) will be a number 1-11 (Sun-Node).

The radix planet (R) will be a number 20-38.

The aspect number (A) will be a number 1-9.

For explanation of planet numbers, see code PSI.
For explanation of aspect numbers, see code ANUM.

NTOS

n i NTOS

Convert number n to string and place it in string array no i.

NUMS

Stack: X Y Z .... NUMS → X Y Z .....

This code lets you insert variable numbers and strings into the text field. A paragraph, whose coding part contains the NUMS code will always be written, it does not matter if the top of stack is 1 or 0.

The text field should contain "templates" that is fields of #######'s (for numbers) or @@@@@@@'s for text, which will be replaced by the numbers or text when printed.

The number of arguments (X Y Z .....) depends on the number of values or strings you wish to print, and must match the number of templates in the text field. They must be put on the stack in reverse order, that is the first value to display must be put on the stack as the last.

An example: The following coding will display the values in memory cells one and two:

```
$
2 RCL 1 RCL

Value in memory cell 1: #######
Value in memory cell 2: #######
$
```

To print strings, these must be defined in the string array. The matching number put on the stack must be the number of the string in the string array. For example to print the sun sign in the latest chart, try this:

```
$
1 CARY

,ari,tau,gem,cnc,leo,vir,lib,sco,sgr,cap,aqr,psc
$
1 PSI NUMS

The sun sign is: @@@
$
```

If the number on the stack is negative, the string will be looked up in the system strings. For example -502 NUMS will display the word "jan" if you put a @@@ template in the text part. Planet, sign and aspect names which have the system string numbers 540..., 570... and 618... will be displayed either as strings or with symbols dependant on the setting of CONFIX 5 (0=symbols nonzero=abbreviations).

Strings and numbers may be mixed freely in the text part. The number templates may not exceed 20 characters, the text templates may be any length, provided that the complete line does not exceed the 80 characters allowed. Numbers will be written in the right side of the template and strings in the left side.

Text and number templates may touch each other for example: ##@@## to make the compact planetary position representation: 28ta51 for 28 51 Tau.

If you wish two templates of equal type to touch, for example to merge two strings, you must use the line continuation mark "\" and break the line. For example the julian day example, where a long number is mixed of two short ones:

```
$
JD
NUMS

JD: ####\
####.####
```

$

This will print e.g. `2449054.9102`.

The NUMS code is extremely useful for exporting data!

Note, that the NUMS code not only works with output but also with lines used for arguments to other functions including: STDEF, CARY, CALL, CML, EXEC and FONTS

OEM

1 OEM will make Argus try to emulate the DOS character set, which means that if your module is using accented or other national characters, they will be translated from the DOS set to fit windows ANSI standard. Also the frame characters will be emulated, so that it is possible to draw grids etc using the IBM/DOS framing characters (only the single line ones). To switch it off, use 0 OEM.

PLEASE NOTE: The MONO setting will override the OEM, so do not use both. OEM is itself monospaced.

ORGIN

x y ORGIN (displace graphics origin)

You can move the origin, so that e.g. 0,0 is the lower left corner, or if for example you are drawing a group of things at a special location, then it may be an advantage to replace the origin. This means, that you can move a whole group of graphics elements just by changing the origin, rather than changing all the x an y's in the drawing. X and Y used in the ORGIN command are always relative to the original origin in the center, so it is independant of what previous ORGIN commands were doing. The X and Y scaling is the units defined in the latest SCALE, or if no SCALE command has been issued, the units are +- 1024.

OPT

Stack: 0 X Y...Z OPT → (key pressed)

As described for the MENU code, you must use OPT to wait for keypresses (and to switch the printer on again, if it was switched of by MENU).

Together with OPT, you define wich keypressed to wait for. You MUST (as shown) always start with a zero, then X, Y etc will be the different keys you may press. The key numbers are 65 for A, 66 for B etc. which in fact are the ASCII codes for the symbol pressed. OPT is not case sensitive, so e.g. 65 will work for both capital and lowercase "A".

You may put no more than 31 possible keypresses.

You may use the options in the command-queue, OPT will look for commands there, before it checks the keyboard.

The ESC key will always get you on from the MENU/OPT without doing anything.

The result of OPT is a number between 1 and N, where N is the number of possible keypresses you have defined. For example:

```
0 88 65 66 OPT
```

will produce 1 if key "A" is pressed, 2 if key "B" and 3 if key "X" (88) is pressed.

The result may be used for example with code NFN to determine which file to branch to, or it may be stored and tested with IF to decide which instructions to do.

OR

Stack: X Y OR → X OR Y

Checks if either of two conditions are true. This means that:

1 1 OR -> 1
1 0 OR -> 1
0 1 OR -> 1
0 0 OR -> 0

1 represents "true" and 0 "false".

Technical note: if AND, OR is used on other numbers than 1 and 0 the result will be a "bitwise" comparison. To be able to analyse the result, you will have to convert the numbers to binary, i.e. each 16 ones and zeroes, and compare each set of bits separately. This may be used creatively if you are experienced in this field. If not, better be sure, that you use the function on zeroes and ones only.

PAD

n PAD

When printing strings using the NUMS code and a @@@@@@@ template, the string will normally be shorter than the template. If the interpreter is in padding mode, the remaining part of the template will be filled with spaces so that lines remain the same length independant of the string length. In non padding mode the template will be truncated to fit the string. 1 PAD (default) will put XLI in padding mode and 0 PAD will put it in truncate mode. Note, that the code SYN also puts XLI in truncate mode.

Example: @@@@@@@@@@@ and @@@@@@@@@@ are friends

0 truncate mode: Tom and Jennifer are friends
1 padding mode: Tom and Jennifer are friends

2 tab mode: inserts tabs instead of spaces. ARGUS uses tabs to produce e semi-monospaced print.

PDEG (planet degrees)

Stack: X PDEG → (degrees 0-360)

This code finds a planet's position in degrees. So if the Moon for instance is in 3-51 Sagittarius, the coding:

```
2 PDEG
```

will produce the value 243. The minutes of arc are just removed, not rounded up, so any position between 3-00 and 3-59 Sgr will result in the value 243.

This code may be used for "Degree astrology" for instance "Sabian symbols", or to find decanates, for example:

```
4 PDEG 30 MOD 10 DIV
```

will produce following results for Venus in:

First decanate (any sign): 0
Second decanate (any sign): 1
Third decanate (any sign): 2

The planet numbers allowed are listed under PSI.

PENC

c PENC

000 PENC makes the graphic pen black, 888 PENC makes it white, 800 PENC makes it red etc.

The colour numbers in ARGUS are 000 to 888, so use only digits 0-8, e.g. 699 is not allowed. This is different from DOS PCA, and is due to the compatibility with windows colors.

PENUP

Stack: PENUP →

Graphics lift pen. Think of the line drawing capability as moving a pen, that may be either lifted over the paper, thus not drawing or be down touching the paper as it moves, thus drawing a line. PENUP will lift the pen, so that the next following DRAW will just move the pen to the new position without drawing.

PENW

w PENW

Set penwidth to w in user coordinates. So 2048 penw will make following draw commands draw lines filling the whole plotting area.

PHS (planet in house)

Stack: X PHS → (house position of planet X)

Finds the house position of planet X. The house position is in the planets own chart, not crossreferenced to latest radix (as with the ordinary PCA chart printouts). The planet numbers are listed under PSI. The result is a house number between 1 and 12.

PLA (planet)

Stack: X PLA → (speed) (position)

This code calculates a single planetary position and speed using the information in the PCA input menu. X must be in the range 1-12. The values for position and speed are intang units.

Note, that if you change the menu values using the MEPUT code, you must also use DNORM to prepare the data correctly for PLA.

PLIN

PLIN totln curln (get current printline info)

This code returns information about the current text printing cursor is located on the printed page.

totln is the max number of lines on the page curln is the line currently being printed

If the printer is not active, PLIN returns -1.

PLACE

Stack: X PLACE →

Insert the current city name (if available) into array number X.

PNAME

Converts an planet number to a negative system string number for use with NUMS. PNAME is the equivalent of writing: 540 ADD CHS

See also ANAME, SNAME

POLY

Draw polygon using the current pen and brush colours. To draw a polygon, you must first provide the coordinates for the corners. This is done the same way as drawing, that is e.g.

```
$
GRON
8 PENC
80 BRCOL
PENUP
-200 -200 DRAW
-200 200 DRAW
200 200 DRAW
200 -200 DRAW
POLY

$$$
```

So instead of terminating a shape with PENUP, you use POLY. POLY will automatically close the shape, if the last point is not the same as the starting point. Coordinates have their origin in the center, and you can use values between -1024 and +1024.

PPATH

n PPATH (get Argus program path)

Loads the program path into stringarray[n]

see also XPATH

PPOS (planets position)

Stack: X PPOS → (intang position of X)

This code gives the position of planet X in intang units. You may convert this value to minutes of arc using the ITOM code.

The planet numbers allowed are listed under PSI.

PROC

n PROC (define a subroutine number n)

You may use subroutines in XLI. They are identified by a single number (n) of your own choice (but not by a name). To define a subroutine, you must place it somewhere in your code BEFORE it is used. This is because the XLI interpreter must do one scan (dummy run) first, which does nothing but make note of the address.

It is perfectly legal to place the subroutine in another file than where it is going to be called from. This will take a bit more time for the call, but this will only be noticable in intensive looping with a lot of calls.

Each subroutine must be defined with an individual n, else your calls will be ambiguous.

After the PROC must follow the actual code of the subroutine. This can be several paragraphs, or even several chained files, and must be terminated by the RETN code. Without the RETN code, your XLI program will just scan forever, never execute.

PSI (planet in sign)

Stack: X PSI → (sign number of planet X)

Get the sign occupied by planet X in the latest calculated (any) chart or latest calculated Radix chart.

Planet numbers are:

	Latest chart	Latest Radix chart
Sun	1	21
Moon	2	22
Mercury	3	23

Venus	4	24
Mars	5	25
Jupiter	6	26
Saturn	7	27
Uranus	8	28
Neptune	9	29
Pluto	10	30
Node	11	31
Part of fortune	12	32
MC	13	33
ASC	14	34
11. house	15	35
12. house	16	36
2.house	17	37
3. house	18	38

The result is a sign number (1-12)

PSTAT

Stack: X PSTAT →

This code controls the printer and screen output:

0 PSTAT: turns the printer output off
1 PSTAT: turns the printer output on
2 PSTAT: saves the current output status
3 PSTAT: returns to the former output status
4 PSTAT: turns all printer and screen output off
5 PSTAT: turns output on again
6 PSTAT: turn screen output off
7 PSTAT: turn screen output on
8 PSTAT: push current io mask on stack
n 9 PSTAT: change iomask to n

If you for instance wish to temporarily turn off the printer output (if on) use 2 PSTAT 0 PSTAT.

To turn it on again (if it was turned on before) use 3 PSTAT.

There may be a couple of situations, where you wish to turn off the screen output temporarily:

Say that you wish to use the aspects from a specially defined graphic transit without showing the full calculation on screen. So use MEPUT to insert the data, 4 PSTAT to turn off output, GTR with the setup needed to call the graphic transits and then 5 PSTAT to restore the normal output. Now you will have the aspects available for the codes RTA and NTA.

The iomask is a filter, which determines where the output will go (calculations, interpretations, chartwheels etc).

Currently, only 3 channels are defined:

channel 0 Screen
channel 1 Printer
channel 6 Text File

If more than one channel is open at any time, the output will go to all open channel. For example when the printer is activated, the output will go to both screen and printer which means that channel 0 and 1 is open. With 9 PSTAT you could change the output to go to printer only by switching off channel 0.

When a calculation is run "invisibly" by putting it in a macro parenthesized e.g. (ra)v this means, that during radix and aspects all channels are closed and no output is written, even if the calculations for them is going on behind the curtains.

The channels are each represented by one bit in the iomask. So currently n could have the following values:

binary	decimal	result
0000000000000000	0	no output
0000000000000001	1	screen only
0000000000000010	2	printer only
0000000000000011	3	screen and printer
0000000001000000	64	textfile
0000000001000001	65	textfile and screen
0000000001000010	66	textfile and printer
0000000001000011	67	textfile, screen and printer

printing to a text file:

First the XLI code 65 9 PSTAT should be executed. From then on all text printing goes to screen and textfile (not to printer). A file called ARGUS.PRN will get all the textoutput added. When the code 1 9 PSTAT is executed, the textfile output is switched off. Beware of calling 65 9 PSTAT again before the ARGUS.PRN is copied or moved, because switching on the text output will whipe the file.

PTMP (parameter change temporarily)

Stack: PTMP →

If you change the PCA settings using XLI, you may assure, that the changes are temporary just for the XLI module, and will be set back to the old values, as soon as the module has finished. This may be a special colour setup, an ayanamsha or whatever else you can do with the XLI configuration codes.

PTMP will also restore the menudata. I for instance you have calculated a series of charts by changing the current data using MEPUT it will automatically be restored to the values it had when code PTMP was executed. So normally, you would place PTMP at the very start of your code. The restore will happen at the time, when all macros and XLI files are exhausted. If you wish to restore earlier, you should use the code TIDY.

The system variables are saved in a file called SYSVARS.$$$.

see also TIDY

PTR

Polar to rectangular conversion of coordinates. If you need to draw circles or other shapes, where you would normally need to calculate sines and cosines, this code will do the conversion for you.

The polar coordinates are

V: the angle measured in intang from right going counterclock

R: Radius which is 0-1024

So: V R PTR -> X Y

Example: draw a circle radius 700 in steps of 10 degrees, which is approx 1820 intangs:

```
$
GRON
0 34 FOR
1 CNT 1820 MUL 700 PTR DRAW
NEXT
PENUP

$$$
```

PUT

Stack: X Y PUT →

Writes one value into the stack, overwriting the value already there. The position is offset from the ZZO marker, which MUST have been set before at some point. If the ZZO marker is not set, you risk to crash the program or produce unreliable results.

Examples:

```
ZZO 3 7 2 11 0 PUT → 11  7  2
ZZO 3 7 2 11 1 PUT →  3 11  2
ZZO 3 7 2 11 2 PUT →  3  7 11
```

PV

Stack: X PV → (speed of planet X)

This code gives the speed of planet X measured in intang units per day/year etc, that is, per the natural unit for the chart type in question. It may be negative, if the planet is retrograde. To convert to minutes of arc, use the ITOM code.

For house cusps and the part of fortune in radix and transit charts which runs very fast, you must add 360 degrees to get the speed correctly.

This code may be used to calculate the orbspeed of an aspect:

```
X PPOS Y PPOS SUB AZP XY 12 MUL
X PV   Y PV   SUB DIV
```

This coding will result in the number of months an aspect is from its exact value, negative is separating positive if applying. Note however the following problems:

1. If X and Y runs equal speeds, you will get a division by zero error.

2. If one of the planets is a radix, you should take only the speed of the moving planet.

3. If one of the "planets" is a transit angle or a solar or lunar return, the speed will be more than one revolution a day, and the formula will fail.

The planet numbers allowed are listed under PSI.

PXL (printer character set translation)

Stack: PXLATE →

You may define a custom character set translation of some of the graphic characters (128-208). Unfortunately the ASCII standard defines only the first 127 characters. The extended character set (128-255) is different on different printers. Of course the IBM character set is widely supported, but you may have a printer, which cannot print this set. The problem is mainly the national characters.

To setup a custom translation, use the code PXLATE, and insert a translation string as the first line in the text field. It may be up to 80 characters long, and will define which characters should be printed. The first position in the string represents chr(128), the next chr(129) and so on. The characters you put in the string are simply the characters you wish to have printed.

Example is given in the module ECMA.XLI

RCL (recall)

Stack: X RCL → (value from memory cell X)

Fetch a value from memory cell number X. The memory cells are not initialised, so you have to put something there using code STO before it will have any meaning to read it back using RCL.

RECH (recent chart)

Stack: RECH → (chart type)

This code will tell which type the "latest calculated chart" is, e.g. Radix, Progressed, Solar arc or whatever. It may be useful to print a heading or to branch to a certain interpretation.

The TT0.TXT delivered with PCA uses this to determine which interpretation to choose, namely either a normal radix interpretation or a transit one.

The chart type numbers are the ASCII value for the key you press to calculate it. For the ones called with the X (extra) prefix for example month transits (XM) use lower case value (m=109). Here is the complete list:

 82 R Radix
 80 P Progressed
109 m Month graphic transit
121 y Year graphic transit
 84 T Synastry (Transit as second chart)
 76 L Lunar return
 83 S Solar return
 69 E Tertiary 1

70 F Tertiary 2
66 B Solar arc direction
71 G Composite
68 D Day chart

REF

1 REF switches referenced interpretation on, 0 REF swtiches it off.

RETN (return from subroutine)

This code terminates your subroutine. When the subroutine is scanned, this code means: "end of scan", while when it is actually later executed, this code means "return to calling point".

REVNO -> Argus revision number

RND

Stack: X RND → random(X)

Produces a random number between 0 (including) and X (not including).

RSIZE

n RSIZE (size up or down)

You may make the following graphics draw smaller or bigger using this code. n is the percent magnification. For instance, the chartwheel will normally print in a +- 1024 square. To make it print half size, you use 50 RSIZE first. If you only want to scale one graphic differently, do not forget to put it back to 100 RSIZE when that graphic is drawn.

RTA

Stack: RTA →

This is a code specialised for interpretation of graphic transits. It will reset the aspect counter to the first in the sorted order. See also the code NTA.

RUL

Stack: X RUL → (ruler)

Find the ruler of sign number X. The rulers are the "modern" ones including Uranus, Neptune and Pluto. So for example: 8 RUL will produce 10 (Pluto) which is the ruler of Scorpio (sign 8).

RX

Stack: X RX → (1 or 0)

If planet X is retrograde the result is 1 (true), if direct, it is zero (false). The planet numbers allowed are listed under PSI.

SCALE

x y SCALE (define graphic frame)

Normally graphics work within a coordinatesystem of x and y between +-1024. This can be changed using SCALE, which sets alternative user coordinates. With SCALE, x and y can also be different, so one is not any more confined to using a square graphic area. The GRON and window resize handler will then adapt the device plotting area to make the graphic as big as possible within the window allowed.

The x and y units are still equal so that e.g. drawing a square of 100 * 100 will actually look like a square on both screen and printer.

The SCALE code should be executed before GRON to have any effect. GRON will do the actual rescaling and clear the graphic frame.

Printed pages:

Note, that to make the graphic frame fit as intended on a printed page, you may have to change margins and scaling. To understand how this works, you could think of SCALE trying to fit its frame as big as possible within another outer frame.

This outer frame is made from the left and right, top and bottom margins of the printed page. The margins are defined in the preferences page layout menu of ARGUS.

Then the graphic frame is scaled down according to the wheelsize defined in the page layout preference.

Finally the graphics margin, which is also defined in the page layout is put around the graphics frame push surrounding text to a distance from the graphics.

To get rid of these restrictions, you may change the page layout from within XLI before calling SCALE. There is no single code for this, but here is the procedure:

```
$
PTMP
0   1 NTOS  1 -1 SYSTR          ;set left margin to 0
1   1 NTOS  6 -1 SYSTR          ;set right margin to 1
100 1 NTOS 15 -1 SYSTR            ;set plot scale to 100 %
0   1 NTOS 18 -1 SYSTR           ;set vertical margin to 0 %

$
```

The PTMP assures, that the Argus page layout change is local to the current XLI session.

SNAME

Converts a sign number to a negative system string number for use with NUMS. SNAME is the equivalent of writing: 570 ADD CHS

see also ANAME, PNAME

SNO (serial number)

Stack: SNO → (number)

Get the serial number of the PCA program running.

SPRED

x y z SPRED

Spread planets like in chartwhel. The positions must be located in intang units in the STO array from x to y. So there will be y-x+1 planet positions to spread. z is the minimum angle allowed also in intang.

STCAT (string concatenate)

Stack: X Y Z STCAT →

Join to strings into one. X, Y and Z are string array indexes for 1st string, 2nd string and destination respectively.

So for example:

```
$
1 CARY

Julian,+,Claire,
$
0 1 3 STCAT      ;join Julian and '+' and place as string 3
3 2 3 STCAT      ;join Julian+ and Claire and put back to 3
3 NUMS

@@@@@@@@@@@@@@@@@@@@@@@@
$$$
```

The above should print Julian+Claire

STCMP (string compare)

Stack: X Y STCMP → (0 or 1)

Compare to strings in the string array. The result of the compare will be:

0: Strings are equal
1: Strings are different

The compare is not case sensitive:

```
$
1 CARY

Ähnlich,+,ähnlich,
$
0 2 STCMP NUMS

######
$$$
```

This will produce a zero, because string 0 and string 2 are regarded as equal, even if one starts with a capital, and the other with a lowercase letter.

STCUT (string cutout)

Stack: X Y P Q STCUT →

Cut out a substring from a string in the string array.

X: Source string index
Y: Destination index
P: Cutout start at character number
Q: Cutout end at character number

The following example prints 9 different cuts of the same word:

```
$
1 CARY

Macedonia,
$
1 9 FOR;loop 9 times
0             ;index of string to cut
1             ;index where to put the result
1 CNT         ;start of cut
1 CNT 3 ADD   ;end of cut
STCUT
1 NUMS

@@@@@@@@@@@@@@@@@@
$
NEXT

$$$
```

This will print like this:

Mace

aced
cedo
edon
doni
onia
nia
ia
a

STDEF (string define)

Stack: X STDEF →

Insert a string into the string array number X. The string must be in the first line in the text field. For example to insert the phrase: "IBM the big blue" as string array number 17:

```
$
17 STDEF

IBM the big blue
$
```

This is used to define or replace single string array elements. If you wish to define the complete string array in one go, it is easier to use the CARY code.

STLEN (string length)

Stack: X STLEN → (length of string X)

String length of number X in the string array.

Example:

```
$
1 STDEF

A t l a n t i c
$
1 STLEN NUMS

#########
$$$
```

This will print the number 8 which is the number of characters in the word: Atlantic.

STO

Stack: X Y STO → X

Store X in memory cell number Y. There are 3000 memory cells available (numbered 0-2999), so it is possible to create tables etc. Note that X is not removed, so storing the same value in several cells could look like this:

26 0 STO 1 STO 2 STO 3 ST0 ....

which will store the number 26 in cell 0-3 inclusive.

STPOS

x y STPOS (find substring in string) -> position Look for substring stringarray[x] in string stringarray[y].

If not found, returns 0, else return position no.

SUB

Stack: X Y SUB → X-Y

Subtracts two numbers

SUBR

n SUBR (Call previously scanned subroutine)

This code will call subroutine n (if it was defined). If you call an undefined subroutine (no previous PROC with the same value of n) you will get an error message telling that no subroutine n exists.

Parameters:

Note, that you may use the stack to transfer parameters and results. In the example below, a chartwheel printed by the subroutine is resized with a figure, which is calculated by the calling routine and pushed on the stack. The subroutine takes this value from the stack and resizes appropriately.

Here is a very simple subroutine example:

```
$
1 PROC RSIZE 12 0 WHEEL RETN

$
GRON
0 15 FOR 1 CNT 10 MUL 1 SUBR NEXT
GROFF

$$$
```

SWDAT

x y SWDAT

Swaps the input menu data between cards:

x y

1 0 radix <> actual
1 5 radix <> horary
1 6 radix <> selected
0 0 current <> horary
0 6 current <> selected
5 6 horary <> selected

NOTE: The swap includes the fields: Name, Place (city), Country and area code. Note that the selected card may have been forced by Argus to point to ACTUAL since the XLI was called. Swapping 0<>0, 1<>1 etc has no effect. X and y must be different, but the order is irrelevant, so 0 2 SWDAT and 2 0 SWDAT will have the same effect.

NOTE: index 3 and 4 are temporary copies of index 1 (radix) and 2 (current). If the code PTMP has been used the XLI will restore the original values upon return, meaning that the data in index 3 and 4 will be used to restore the input menu. Accessing them here means, that it is actually possible to change even the restore values, even if this is normally not recommended. They should only be read.

The full list of possible indices is:

Index 0: Currentdata (earlier radixdata)
1: Radixdata (earlier currentdata)
2: Currentdata (earlier presentdata)
3: Temprdata (earlier selecteddata)
4: Tempcdata (new)
5: Presentdata (new)
6: Selecteddata (new)

SYMIX (symbol index)

Stack: SYMIX →

Change the astrological symbols assignment. The symbol tables have a few alternatives, which you may insert instead of the original ones. The original ones are different for different language versions of PCA.

The assignment is determined by a string, which you should apply as the first line in the text field. Only the first 24 symbols may be reassigned, so the string should not exceed 24 characters. An example:

$
SYMIX

ABCDEFGHIZKLMNOPQRSTUVWX
$$$

As you can see the letters follow an alphabetical order apart from, that J ist replaced by Z. This means, that the Pluto symbol is replaced by a german version.

The positions in the string represent the symbol to replace, the letters the replacement.

The replacement symbols are:

A..J = Sun to Pluto (Danish symbols)
K..V = Ari to Psc
W = Node
X = Arrow
Y = German Uranus
Z = German Pluto
[= English Pluto

The SYMBOL.XLI file uses the SYMIX code to insert your favourite version.

SYN

Stack: SYN →

This is a highly specialised code for synastry output, of the type, where you merge the two person's names into the text.

SYN have two effects:

1) It inserts the current name and the radix name into position 1-4 in the string array. If any strings are already defined, their indexes will be incremented by 5. If radix person is Louise, and Radix2 person is Alexander, the string array will be:

String 0: empty
String 1: Louise
String 2: Alexander
String 3: Louise's
String 4: Alexander's
String 5: (former string 0) etc.

2) It turns on a non-padding mode which will change the way NUMS inserts strings into the text. Normally, when a row of @@@@@@@@@@@ is met NUMS will insert a string and if there are more @'s than the length of the string, it will be padded with spaces. This assures, that all lines will keep their lengths. SYN will turn the padding off, so that the text will smooth without empty spaces after the names. However, you must put enough @@@@'s to hold the longest name you will accept, which may be up to 20. This will make most lines holding names truncate quite much, so better use say 10, and tell users not to insert longer names.

SYSAV

n l SYSAV (Save selected part of system strings to PCA.CFG)

Save l lines from line n onwards to PCA.CFG. This means, that XLI can make a change and update the disk, for example if you have an XLI module to make the user install options.

SYSTR

S C SYSTR

In PCA for DOS this code was able to access a few of the system strings. In Argus, SYSTR can access all the PCA.CFG strings. The lines are numbered from zero onwards.

S is the line number in the configuration file.

C is the destination index in the string array. If C is negative, it means write configuration string, if it is positive or zero it means read. So to read CFG line 1123 into string array no 34, the code is 1123 34 SYSTR. To write it back, the code is 1123 -34 SYSTR.

Serial name, number, checksum and features string cannot be written back.

There are a few special cases for S, to save a bit of 2.4.9 compatibility:

0: read name of namefile
1: read header
3: read serial name
4: read current macro

TAB

n TAB (tabulate)

Tabulate to column n on screen and printer.

TABL

n TABL

Print a horizontal line in text from current position till position n using the tabline color defined in PCA.CFG confix[6]. If n=0 it will draw till right margin.

TIDY

This code tests if there has been a PTMP data save code during the current XLI session. If so, the values of the system variables and the 'current' birthdata will get restored to the value they had when the PTMP code was met. The test for PTMP is checking for the filename SYSVARS.$$$, which is then erased.

TXCOL

c TXCOL (change text color)

Change the current text colour for screen (and printer if it is currently printing). C is an 3-digit RGB value between 000 and 888.

UTFIL

X UTFIL print to file, equivalent to INFIL

X=n write n lines to file
X=0 close file
X=-1 open file for appending

if X>0 then the two first lines in the text part are used:

line 1=filename
line 2=print line

if the file is already open, the filename is ignored and writing will go to the current writefile. When finished, the file should be closed by coding 0 UTFIL.

IF the code NUMS is included in the paragraph, variables can be output the same way as normally written to screen or printer. In that case you just include templates in the print line where you want the numbers or strings to be printed:

The result is: ### and the current string is @@@.

Se the XLI manual how to use templates.

To append text to an existing file, you must first open it using -1 UTFIL having the first textline specify the filename. Succeding n UTFIL (n>0) will append to this file as normal.

WAIT

Stack: WAIT →

Say you are writing an interpretation part with a heading and a number of paragraphs. The conditions for these paragraphs may be of such a kind, that in some cases none of them are true. It could for example be a heading called "Special features" treating different aspect patterns and other combinations. If a chart has none of these patterns, you may get the heading without any text. The problem is, that at the time, the heading must be printed, you do not know if it should be printed at all!

Of course you may make all the coding twice, one to check all conditions just to see if the heading should be printed, and then next to check for each paragraph text. But this is tedious and clumsy.

The solution is to collect all these paragraphs in one file. The first paragraph of this file must hold the heading and the code WAIT:

```
$
WAIT

SPECIAL FEATURES:
----------------
$
```

After this all the other paragraphs belonging to this heading should follow.

Now the heading in the WAIT paragraph will be withheld, and not printed until a valid paragraph is met. First then the heading is printed, and then the paragraph.

The pending heading will be discarded when the file end is met. That is why you may put just the paragraphs belonging to this heading in that file.

You may have only one paragraph with the WAIT code, and this MUST be the first paragraph in the file.

WHEEL

n b WHEEL (draw chartwheel mixed with other graphics)

Even if you can use MACV, MACW etc to make a chartwheel, these commands will start by clearing the graphics frame before drawing. The WHEEL code will just call the wheel without disturbing other graphics.

n=0 current chart
n=1 radix chart
n=2 current chart
n=3 aux 1 chart
n=4 aux 2 chart
n=5 present chart
n=12 double wheel with radix as outer and current as inner wheel
n=25 double wheel with current as outer and present as inner wheel

If n is two digits, the first is the outer and the second is the inner

b=0 houseless wheel
b=1 normal wheel
b=2 houseless wheel without data written below
b=3 normal wheel without data written below

WKEY

Stack: WKEY →

Print the usual "Please press key" prompt, and wait for any keypress before continuing.

XFEED

n XFEED

Global FEED replacement. A single paragraph can be set to be guaranteed the n first contiunuous lines without pagebreak by setting n FEED. If you use n XFEED this will be automatically valid for all following paragraphs until the end of the XLI-session. However, even without executing XFEED, any XLI session has a default of 2 guaranteed unbroken lines (orphan resistance) until you change the this default.

XIF (eXit if)

Stack: X XIF →

This code will exit a FOR loop immediately, if X is "true" (not zero), so that it jumps directly to the first instruction after NEXT without doing the instructions inbetween.

This may be used, if you do not know on beforehand how many times the loop should count, but that it should be dependant on a condition.

For example, to test the number of aspects in a graphic transit list, you will only know when the list is exhausted (NTA produces a 0), that you should stop counting.

```
$
RTA             ;reset transit list
0 10000 FOR     ;start large margin count
1 CNT           ;get count number
NTA             ;get aspect details
NOT XIF;exit if last (not) aspect
```

```
NEXT
NUMS

There were ##### transit aspects found!
$$$
```

XPATH

n XPATH (get current XLI module path)

Loads the XLI path into stringarray[n]

see also PPATH

XPLA (extra planetary information)

Stack: X XPLA → (info)

This code retrieves further coordinates from a previous PLA calculation.

X may have the values:

0: heliocentric longitude, intang unit
1: heliocentric latitude, intang unit
2: heliocentric longitude velocity, intang unit
3: geocentric lat, intang unit
4: geocentric distance value AU*1000
5: geocentric right ascension, intang unit
6: geocentric declination, intang unit

So for example to print the declination for the Moon in degrees and minutes, you may code:

```
2 PLA 6 XPLA ITOMS 60 DIVR XY ABS XY NUMS

### ###
```

Note: Do not try to retrieve these figures for the part of fortune, this will not work.

XOR

x y XOR -> x xor y (exclusive or)

XPOS

i b p XPOS (enhanced)

p is the planer number (0-12) or cusp index (13-18)

b is the chart type:

0 current
1 radix

2 current
3 auxchart1
4 auxchart2
5 present
6 auxchart3

i is the information type:

0: helio longitude
1: helio latitude
2: helio long velocity
3: geo lat
4: geo distance value AU*1000
5: geo right ascension
6: geo declination
7: ayanamsha (intang)
8: harmonic*10
9: armc (intang)

XY

Stack: X Y XY → Y X

Swaps the two uppermost numbers on the stack.

ZMODE text fold mode

1 ZMODE will make long lines fold following the window width. So use this code in the start of your interpretation, if you want it to be auto-adjusting the paragraph width. The lines will not be expanded to obtain an adjusted right margin.

0 ZMODE will make long lines unfolded, so you will see only the first part of them.

ZODOF

x ZODOF (Change zodiac origin to x}

This will change the angular offset for planetary and house logitudes in the current chart. For instance to calculate a draco chart, just do an ordinary chart and then call 11 PPOS ZODOF. The offset is calculated from the original zodiak origin, so if call ZODOF more than once with different origin angles, they will all be relative to tropical zero Aries.

ZZO

Stack: ZZO → (no action)

Marks an origin on the stack. The stack will normally be accessed "dynamically" from the top. You may however wish to access it from the bottom, which needs an origin. ZZO will mark the current top of stack, so that the following numbers put on the stack will get position 0, 1, 2 etc. These may then be accessed using the codes GET and PUT.

This does not affect the ordinary action of the stack, you may still use all the other codes as before.

APPENDIX 1 - Tables

MACROS

Macro 8 is a macro for changing the master orb limit and the orb scheme

8R set orbscheme R
8P set orbscheme P
8S set orbscheme S
8C set orbscheme C
8X set orbscheme none
8ddmm. set master orb to ddmm

Macro 7 will set the harmonic number:

7iii,fff.

The comma separates the integer and fractional part of the harmonic number and the point terminates the argument as ususal. So e.g. the macro 712,6. will change the harmonic number to 12.6. To put it back to 1, the macro is just 71.

ITEM NUMBERING

ITEM	current	radix	GTR radix	GTR transit	colour
Chiron	0	20	@	`	
Sun	1	21	A	a	18
Moon	2	22	B	b	19
Mercury	3	23	C	c	20
Venus	4	24	D	d	21
Mars	5	25	E	e	22
Jupiter	6	26	F	f	23
Saturn	7	27	G	g	24
Uranus	8	28	H	h	25
Neptune	9	29	I	i	26
Pluto	10	30	J	j	27
Node	11	31	K	k	28
Part.fort	12	32	L	l	
ASC	14	34	N	n	
2. house	17	37	Q	q	
3. house	18	38	R	r	
4. house					
5.house					
6.hous					
7.house					
8.house					
9.house					
MC	13	33	M	m	
11.house	15	35	O	o	
12.house	16	36	P	p	

Conjunction	29
Opposition	30
Square	31
Trine	32
Sextile	33
Semisquare	34
Sesquisquare	35
Inconjunct	36
Semisextile	37

Only the indices for Current and Radix positions are shown above. It is also possible to access the positions of
horary and the auxchart1 and 2 etc (used as temporary storage):

1: Positions numbers: *)
0..18: Current
20..38: Radix
40..58: Current (equal to 0..18, earlier Auxchart 1)
60..78: Auxchart 1 (earlier auxhchart 2)
80..98: Auxchart 2 (earlier Horary chart)
100..118: Horary chart

*) Position numbers are used in the followin codes:

PSI PHS RX HSI
HRU APOW ANUM AORB
PDEG PPOS HPOS PV
HV

PCA.CFG System variables:

Line:

0: File heading, not used
1: Language number:
2: Page header for printouts
3: Page footer for printouts
5: Left margin for printouts (% of page width)
6: Right margin for printouts (% of page width)
7: Printer font: Name,Pointsize,Style
8: Screen font: Name,Pointsize,Style
9: Symbol font name:
Characters (128-175) will use this font, the rest will use printer or screen font. Symbol size and style will adapt to the rest of the characters.
10: Symbol font mapping table:
This is a row of characters, normally chr(128-175). If an alternative symbol font is used which has the symbols placed differently, you may change the mapping. For example, if Moon, which in ARGUS is no 130 in the new font is no 277, then the 3rd character in the mapping table string must be chr(277). Note, that there is also a mapping table called SYMIX which works on both text printout and on the chartwheels.

11: If present and in the range 50-500 defines the screen lineheight.
Default is 130.

12: If present and in the range 50-500 defines the printer lineheight.
Default is 130.

13: If 1 the screen font size will shrink with window resize.
If anything else it won't.

14: Left,top.width and height of main window stat latest session
15: Plot scale: size of chartwheel on printouts (0-100)
16: Horizontal position of chartwheel on printouts (0-100)
17: Vertical position of chartwheel on printouts (0-100)
18: Vertical margin for printed chartwheel (0=tight, 100=full page)

26: Chart style: 0=Lotus, 1=English, 2=Universal, 3=French
27: Chart orientation:
0 ASC left
1 MC up
2 Radix ASC left
3 Radix MC up
4 Aries right
5 Aries up
6 Aries left
7 Aries down

28: Aspectstyle:
0 To Symbol
1 To center
2 To Degree
3 No aspect

29: Planet size (0-100=
31: SYMIX Symbol mapping table:
This table works both for the plotted symbols on
the chartwheel and the printed fonts on the positions
output. It is used to determine which symbols to use
for Pluto and Uranus. It works the same way as the
font mapping table (system variable 10). You should
first make table 10 match the Truetype font used for
text printouts, then if necessary map the correct Pluto
and Uranus symbol witn this table.

32: Size of degrees and minutes figures on chartwheel.
33: Country default code e.g. F (without area number)
34: Aspect linewidth
35: General (not aspects) chartwheel linewidth (1000ths of wheelsize)

40: Moons node:
0 Mean node
1 True node
: Astrological reference in interpretations (1=yes 0=no)
: Midpoint aspects type
1 180 degr.
2 90 degr.
3 45 degr.
43: Midpoint sort system

0 360 degr
1 180 degr
2 90 degr
3 45 degr
4 22½ degr

44: House system
0 placidus
1 Koch
2 equal
3 Regiomontanus
4 campanus
5 topocentric
6 nat. degree
7 porphyry
8 alcabitius

45: Secondary progressed house system
0 no movement
1 naibod
2 kundig
3 ar solar arc
4 ecl solar arc
5 1 degr
6 true motion 360/year

46: Tertiary progressed house system
0 no movement
1 naibod
2 kundig
3 ar solar arc
4 ecl solar arc
5 1 degr
6 true motion 360/year

47: Method for part of fortune
0 traditional
1 Kontinental

48: Age point system
0 part of fortune
1 huber age point
2 true huber age point
3 logarithmic (Mann)

49: Style for graphic transits/Progressions
0 shades (like DOS version)
1 bars

54: Master orb limit DDMM, eg. 830 means 8 deg 30 min

55: Solar arc system
0 true houses
1 equal movement

56: solar return system
0 tropical
1 siderial

57: Composite chart system
0 all houses are midpoints
1 Robert Hand method

58: Relationship chart system
0 midpoint of lat and long
1 great circle midpoint

59: Day chart method
0 Traditional
1 Kundig

65: Version number (not used)
66: Release number (not used)

67: checksum (control number for serial name etc)
68: serial number
69: serial name
70: Initial data for input menu (comma separated)
date,time,zonename,Latitude,Longitude,Charttype,,ZoneID:
71: Features switches:
d=demodisk
n=network
i=license info on main caption
m=pcm runs any date from demo
72: Initial data for horary input menu
Name,date,time,zonename,Latitude,Longitude,Charttype,,ZoneID:

73: Customer number

79: Aspect types: (comma separated pairs)
List of ARGUS supported aspects. The first figure
in each pair is the angle of the aspect in the format
DDDMM, e.g. 13500 for a sequiquadrate and 5125 for a septile
(51 deg. 25 min). The second figure in each pair is either
0 (do not include) or 1 (include), which reflects the setting
of the checkboxes in the orb menu.

80: Configuration swithces(0-63), inherited from DOS version (comma separated)
0: bit 0=1 Chiron included
bit 1=1 Exclude midpoint aspect type and orb
bit 5=1 Jobdate printed
bit 6=1 No page numbers on printouts
3: 0 No degrees and minutes on chartwheel
1 degrees on chartwheel
2 degrees and minutes on chartwheel
5: 0 use symbols in printouts
1 use latin abbreviations in printouts
2 use national abbreviations in printouts
6: Tabline colour 000-888 or -1 for text color;
7: Reserved for black use of Argus (no menus, auto-quit)
8: bit 0=1 No orbspeeds, aspects printed in 4 columns
bit 1=1 Data not automatically moved to actual input card
bit 2=1 No aspects on bi-wheel
bit 3=1 Short radix houses on bi-wheel
bit 4=1 include bonatti sections
bit 5=1 suppress chartwheel aspect to angles
bit 6=1 include PtFt on UK-chartwheel
9: additional topmargin on printed pages (dotrows)
Other indexes are reserved for future use.

81-84: Aspect orb limits for scheme 1-4
(DDMM eg. 830 for 8 deg 30 mins)
0: orb combine:
0 Minimum orb
1 Mean orb
2 Maximum orb
1-31 orb limits for aspects no 1-32
32-50 orb limits for planet number 0-18
51 orb limit for midpoints

87: Screen colours, comma separated
0 Text background
1 Text foreground
2 Graph background
3 Graph main line
4 Graph angles
5 Graph houses
6 Ari
7 Tau
8 Gem
9 Cnc
10 Leo
11 Vir
12 Lib
13 Sco
14 Sgr
15 Cap
16 Aqr
17 Psc
18 Sun
19 Moon
20 Mercury
21 Venus
22 Mars
23 Jupiter
24 Saturn
25 Uranus
26 Neptune
27 Pluto
28 Node
29 Cnj
30 Opp
31 Sqr
32 Tri
33 Sxt
34 ssq
35 ses
36 qqx
37 ssx

88: Printer colours, comma separated
indexes, see above

92: Date style:
0: european
1: american

93: East/West default
0: East
1: West

94: Country default. If a valid country abbreviation is found here
e.g. 'GBE' choosing 4 for city input will search this country only
while field 5 will work globally as usual.

95: warninglevels. If this is a number (0-65535) bits set will show
warning types to switch OFF.
bit 0 set: turn off data input warnings

bits 1..15 not yet implemented.

100: Filename for automatic load with text editor

101: Translate table for printfile (using the 9 PSTAT feature)
This string is a 255 length translate table. If for example
table position n='%' this means that chr(n) will print as '%'.
If position n=chr(0) this character will not print. If the table
is shorter than 255, chr(255) and downwards to table end will print
normally (untranslated).

110-115: Chart-for.. type names (Male, female, etc.)

121-132: Macro definitions for F1-F12

248-420: Shape tables for graphic symbols
The first four positions in each string tells what
the symbol is.
The following characters are Hex-codes:

For example the shape of 1 is simply 3 points:
1A4F x=1Ah=26 y=4Fh=79 move
C071 x=C0h=64 *) y=71h=113 draw
C08F x=C0h=64 *) y=8Fh=15 **)draw
*) value is bits 0-6, bit 7 means move or draw
**) value is bits 0-6, bit 7 set means end of shape

So drawing is done in a square matrix of 128*128

760-774: Interpretations installed

775-789: XLI-modules installed