

ARGUS BASIC
Programmer's Reference Manual



VIDICODE Datacommunicatie BV
Zoetermeer 1990 - 1997

ARGUS BASIC Programmer's Reference Manual
copyright 1990 -1997 VIDICODE Datacommunicatie BV
Zoetermeer

All rights reserved.

Designed, edited and typeset in The Netherlands.

No part of this publication may be used or reproduced, stored in a database or retrieval system, transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher, except in the cases of brief quotations embodied in critical articles and reviews.

Making copies of any part of this book for any purpose other than your own personal use is a violation of copyright laws.

Further copies of this manual and information on the Argus range of products and services may be obtained from the publishers.

Please address all inquiries to:

EUROPE:

Vidicode Datacommunicatie BV
Blauwroodlaan 140
2718 SK Zoetermeer
Netherlands

Voice: (+31) 79 3617181
Fax: (+31) 79 3618092
Email: VIDICODE@VIDICODE.NL
Website: WWW.VIDICODE.NL

USA:

Vidicode Datacommunication Inc.
1616 Shipyard Blvd. Suite 259
Wilmington, NC 28412

Voice: (+1) 910-452-5600
Fax: (+1) 910-254-3525
Email: SALES@VIDICODE.COM
Website: WWW.VIDICODE.COM

PROGRAMMING THE MODEM	7
1.1 Program structure	7
Program lines & commands	7
Variables & constants	7
Variable names	8
Integers & integer arrays	8
Strings & string arrays	8
Labels	9
Calculations & expressions	10
Hexadecimal numbers	10
Logical operations	10
Events	11
1.2 Writing programs	12
Direct programming	12
Indirect programming	13
1.3 Running programs	14
Running files from the filing system	15
1.4 Program and variable storage	15
1.5 I/O ports	16
Port syntax	17
1.6 The Argus filing system	18
Text files	19
Binary files	20
BASIC programs	20
1.7 Program backup	20
Backing up Program memory	21
Backing up the RAM-, FLASH- or Hard-disk	21
ARGUS BASIC - COMMAND REFERENCE	22
Syntax	22
ABS absolute value	24
ACTIVE set modem active	25
ADC read analogue to digital conversion	27
AND logical AND	28
ANSWER answer a call	29
ASC ASCII value	30
AT AT scan	31
AT ([port]) (SLASH) ON/OFF	32
AT INPUT ([port])	32
AT REPEAT	32
AT [string]	32
AUTO automatic line numbering	34

BAUD	set baud rate	35
BOOT ON/OFF	enable/disable BOOT file	37
BRK	transmit a break	38
BUFFER	reserve buffer space	39
BUTTON	read the S/A button position	42
CALL	jump to machine code subroutine	43
CHR\$	character string	44
CLEAR	clear variables/memory	45
1.	CLEAR	45
2.	CLEAR ALL	45
3.	CLEAR ([port])	45
4.	CLEAR INPUT ([port])	45
5.	CLEAR OUTPUT ([port])	45
6.	CLEAR BUFFER	45
7.	CLEAR PRINTER	45
8.	CLEAR! or CLEAR FILE	45
9.	CLEAR ROM	45
CLOCK\$	read internal clock	47
CONNECT	connect to remote modem	48
COPY	copy files	49
CTS	Clear To Send	50
DATA	define constant data	51
DATE\$	read system date	52
DAY	read/set day of week	53
DCD	Data Carrier Detect	54
DDAY	read/set day in month	55
DELETE or DEL	delete program lines/files	56
DIAL	dial number	58
DIM	reserve space for arrays	59
DIR	list file directory	60
DIR ROM	list ROM directory	62
DIV	integer divide	63
DSR	Data Set Ready	64
DTMF ON/OFF	dial tone multiple frequency	65
[var] = DTMF	read DTMF receiver	66
DTR	Data Terminal Ready	67
DUTCH	select language and screen mode	68
ECHO	enable/disable character echo	69
END	end of program	70
ENGLISH	select language and screen mode	71
EOR	logical Exclusive OR	72
ERL	error line system variable	73
ERN	error number system variable	74
ESCAPE	set Escape character	75
FALSE	Boolean system variable	76
FAX MODEM ON/OFF	enable/disable fax selector	77
FAX ON/OFF	turn FAX on or off	78
FAX RECEIVE	receive fax file	79
FAX SEND ...TO...	send fax file	80
FOR ... (TO) ... (NEXT) ... (STEP)	loop structure	85
FREQ	read frequency measurement	87
GET	get byte (with wait)	88
GET\$	get character (with wait)	89
GOSUB ... RETURN	jump to subroutine	90
GOTO	unconditional jump	92
HANDSHAKE	select/enable handshaking	93
RTS/CTS selected:		93
XON/XOFF selected:		93

Handshaking via LINKed ports	93
HANGUP disconnect a call	95
HOOK hook control	96
HOUR read/set hour of day	97
IF ... THEN ... ELSE condition test	98
ILINE read input line	99
INPUT read string/number	100
IPOINT default input port	102
KEY get byte (without wait)	103
KEY\$ get character (without wait)	104
LCASE convert ASCII to lowercase	105
LCASE\$ convert string to lowercase	106
LED control LEDs	107
LEFT\$ extract left part of a string	108
LEN get length of string	109
LENGTH set or read serial port word length	110
LINEFEED enable/disable Line Feeds	111
LINK connect logical data streams	112
LIST list BASIC program	115
LIST AUTO list program for editing	116
LOAD load BASIC program or data	117
LOAD ROM load BASIC programs from socket 2	118
LTRIM\$ remove leading spaces	119
MATCH find matching files/data	120
MATCH ROM find matching files in socket 2	122
MID\$ extract middle portion of string	123
MIN set/read minute value	124
MNP (4/5) ON/OFF turn error correction on/off	125
MOD integer remainder operator	126
MODEM select standard modem mode	127
MONTH set/read month of year	128
NETWORK Argus LAN network command	125
NEW new program	132
OLD restore program	133
OLINE set/read digital output line	134
ON AT AT scan event handler	136
ON BRK BRK event handler	137
ON BUTTON USER button event handler	138
ON CONNECT connection detect event handler	139
ON DTD DTR loss event handler	140
ON DTR DTR event handler	141
ON ERROR error handler	142
ON ESCAPE Escape event handler	143
ON ... GOSUB/GOTO jump on value	144
ON HANGUP loss of connection event handler	145
ON OFF disable all events	146
ON RESET reset event handler	147
(a) A !BOOT program is found on the disk	147
(b) No program running at moment of reset	147
(c) Program running at moment of reset	148
ON RESET RUN [filename]	148
ON RESET RUN	148
ON RESET ...	148
ON RING RING event handler	150
ON SEQUENCE Esc. sequence event handler	151
ON SLASH AT scan event handler	152
ON TIMEOUT timeout event handler	153
OPOINT default output port	155

ORI	output ring indicator	157
OUTPUT	send data to port	158
PARITY	read or set parity	159
PBOT	program bottom	160
PEEK	read byte from memory	162
PEEK\$	read string from memory	163
PLAY [filename]	playback a recorded file	164
PLAY ON/OFF	turn play on/off	166
PLAY VOLUME [num]	playback attenuation	167
POKE	put byte into memory	168
POKE\$	put string into memory	169
PORT	last event port system variable	170
PRINT	send data to port	171
PRINT	positioning & formatting	171
PRINTER	enable/disable printer port	173
PRINTER	and PRINTER ON	173
PRINTER	OFF	173
PRINTER	LINEFEED ON/OFF	173
PRINTER	INPUT/OUTPUT	173
PROTOCOL	set file transfer protocol	174
PTOP	program top system variable	178
PULSE	set/do pulse dialing	179
PUT	write byte value to port	180
PUT\$	write character to port	181
READ	read data items	182
RECEIVE	receive file or data	183
RECEIVE #[port], [address]	receive data	184
RECEIVE [filename], [type]	receive file	185
RECEIVE ALL	receive disk backup	189
RECEIVE TIMEOUT = [int]	set Sreg66 timeout	190
RECORD [filename]	record a voice file	191
RECORD ON/OFF	turn record mode on/off	193
RECORD TIMEOUT [num]	set record timeout	194
REM	remark	195
RENAME	rename files	196
RENUM	renumber program	197
REPEAT ... UNTIL	loop structure	198
REPORT	report result or error	199
REPORT	ERN	199
REPORT	or REPORT RESULT	199
RESET	generate hardware reset	200
RESET ALL SREG	reset all S-registers to factory settings	201
RESET MODEM 2	reset modem-chip only	202
RESTORE	restore pointer to data command/buffer/RAM-disk	203
RESULT	last status of modem action	204
RETURN	return from procedure/ set return character	206
1.	Return from procedure	206
2.	Set value of RETURN character.	206
RIGHT\$	extract right portion of string	207
RING	ring counter	208
ROM	access RAM/ROM-disk in socket 2	209
RPT\$	repeat string	211
RTRIM\$	remove trailing spaces	212
RTS	ready to send	213
RUN	run a program	214
RUN ROM	run a program from ROM/RAM-disk in socket 2	215
SAVE	save program or data to memory/disk	216
SAVE ROM	save files in RAM-disk in socket 2	217

SCAN	set answering speed/scan	218
SEC	read/set seconds value	219
SEND	send file or data	220
SEND	#[port], [address], [address] send data	221
SEND	[filename], [type], [option] send file	222
SEND ALL	make a backup of disk	227
SEQUENCE	select escape sequence character	228
SGN	get sign of number	229
SOUND (MODEM)	make a sound on speaker or phonenumber	230
SPC\$	repeat spaces	231
SPEAKER ON/OFF	speaker control	232
SPEED	set line modem-speed	233
SPEED BUFFER ON/OFF	speed buffer on/off	234
SREG	set/read S-registers	235
STOP	stop program execution	236
TIME\$	read or set time	237
TRACE	trace program-lines execution	238
TRIM\$	remove spaces	239
UCASE	convert ASCII to uppercase	240
UCASE\$	convert string to uppercase	241
VAL	convert string to number	242
VAL\$	convert number to string	243
VOICE	set modem to voice-mode	244
VOLUME	set speaker volume	246
WAIT ... (SEC/MIN)	wait some time	247
WHILE ... DO ... WEND	loop structure	248
XOFF	define XOFF character	249
XON	define XON character	250
YEAR	read/set year value	251
BASIC Error Messages		252
BASIC Keywords and associated short forms		256
Keyboard & LCD codes		266

This page intentionally left blank

1.1 Program structure

Argus BASIC is part of the Argus Programmable Modem, the Argus Call Recorder and of the Telecommunications Centre Lite (T.C.Lite). A keyboard and LCD display are optional for the Argus Programmable Modem and always available on the Argus Call recorder and the T.C.Lite. A filing system is optional for the Argus Programmable Modem and can be a RAM-disk, a FLASH-disk, a Hard-disk or an Argus LAN (Local Area Network). The T.C.Lite only supports the FLASH-disk.

Argus BASIC is much like any other version of BASIC in terms of both syntax and structure, but there are also many significant differences. This chapter provides an overall description of the modem programming environment and goes on to describe each of the Argus BASIC commands in detail.

Program lines & commands

A BASIC program consists of a series of lines which must start with a 'line number' and may contain up to 255 characters. A line that is entered without a line number will not be treated as part of a program but will be executed immediately. For example, the following command :

```
PRINT "Of Mice and Men - by John Steinbeck"
```

will display the quoted message on your screen as soon as you press [Return]. Alternatively, when you type the line:

```
10 PRINT "Of Mice and Men - by John Steinbeck"
```

it will be treated as line 10 of the current program and stored in the modem's memory until the program is executed. Program lines may be entered in any order but the modem will always list and execute them sequentially from lower line numbers to higher ones (unless control of flow is changed by a branch command such as `GOTO`).

A single program line may contain more than one command with individual commands being separated by a colon. The following line contains three commands:

```
10 X=9 : PRINT "The square of ";X;" is " ; : PRINT X^2
```

The first command defines a 'variable' called `X` and gives it the value 9. The second command outputs a message, including the value of `X`, to the currently selected output port, and finally, another `PRINT` command outputs the value of X^2 . The overall effect of the line is to output the following:

```
The square of 9 is 81
```

As many commands may be placed on a single line as will fit in the maximum line length of 255 characters.

BASIC keywords such as `PRINT` may be entered in upper or lower case but the modem will convert them all to capitals. Also, it is not necessary to leave spaces between commands, as the modem will insert them automatically.

During program entry it is advisable to leave (numerical) gaps in the line numbering, so that additional lines may be inserted later. The convention is to number lines 10, 20, 30 etc., although there is a command for re-numbering programs (`RENUM`).

Variables & constants

Data stored or used in a BASIC program may exist as a 'constant' or as a 'variable'. The following examples are constants:

```
25
```

```
"The quality of mercy is not strained.."
```

Constants are so-called because they have a fixed value that does not change during program execution.

The term *variable* is used to describe a named piece of data whose value can change during program execution. For example:

```
X=25
```

```
Quote$="The quality of mercy is not strained.."
```

Here, *X* and *Quote\$* are variables which can be given different values at different times. After the command:

```
X=43
```

the value of *X* will be 43, not 25 as previously.

There are only two types of constants and variables in modem BASIC which are known as 'integers' and 'strings'. Floating point (fractional) numbers are not allowed.

Variable names

Variable names must start with a letter, may contain letters, numbers and the underline character and may be up to 127 characters long. They may be typed in upper or lower case but the modem will convert them during listing so that only the first character is capitalized. The underline character is generally used to split a variable name into two or more parts to make it more readable, in which case the first letter after each underline will also be capitalized.

The following are examples of valid variable names:

A	X395
Home_Num\$	Start_Pointer
Inp_Line\$	S\$
Name\$	Start_Date\$
Telephone	Temperature

String variable names are suffixed by '\$' as in most versions of BASIC, but, whereas it is normal to use the '%' or '#' symbols to distinguish between integer and floating point variables, this is not necessary on the modem so that any variable not suffixed by '\$' will be treated as an integer.

Integers & integer arrays

Argus BASIC will handle integer numbers and variables in the range -32768 to 32767 (\$0000 - \$FFFF). Arrays (tables) can only have one dimension and the number of elements is limited to 32767. Space for arrays is allocated using the standard form of the DIM command. For example:

```
DIM A(100)
```

reserves space for 100 integers in array *A*. Only 1 dimensional arrays are allowed. Multiple arrays can be defined in a single DIM command:

```
DIM Table1(300), Table2(500)
```

To assign the contents of the tenth element in the array *Table1* above, to the integer variable *R*, you would use the notation:

```
R=Table1(9)
```

Element 0 is the first element.

Integers occupy two bytes of memory.

Strings & string arrays

Strings are sequences of up to 32767 characters enclosed in double quotes.
String variables are assigned values using the equal '=' sign:

```
Postcode$="GL53 7PJ"
```

```
Title$="Introduction to Basic"
```

Arrays of strings can also be defined:

```
DIM Options$(8)
```

```
DIM Lines1$(200), Line2$(50)
```

Strings can be added using the '+' sign, or by just writing the one after the other; thus A\$B\$ is the same as A\$+B\$.

When A\$="ABC" and B\$="DEF" and these two strings are added, the result will be "ABCDEF" as one string.
The minus sign can also be used with strings, but its operation is very different. It will search for the *numerical location* where a given string is found within another string. Thus:

```
x="HELLO" - "LL"
```

will result in x=3.

If the result is 0, the string was not found. E.g.:

```
IF A$ - "Post" GOSUB Zip
```

will execute the procedure Zip only if "Post" is found in A\$.

Normally when you input a string a carriage return will terminate the input. To insert a carriage return between B\$ and C\$ use A\$=B\$ ' C\$; or to add a carriage return use A\$=A\$ ' .

Labels

In addition to line numbers, Argus BASIC allows the use of 'labels' to uniquely identify particular program lines; e.g. the first line of a subroutine.

Label names must be preceded by a '%' symbol and must appear at the start of a line *immediately* following the line number. In all other respects labels obey the same rules as variable names and, as with variables it is advisable to choose meaningful names that indicate the function of the label.

```
300%Count_pulses
```

```
550%Check_password
```

Labels can be used interchangeably with line numbers in commands such as GOSUB and GOTO:

```
GOTO Try_again
```

```
GOSUB Mail
```

```
IF A=1 GOTO Yes ELSE GOTO No
```

The commands LIST and DELETE can also be used with labels:

```
LIST phone
```

```
LIST p1,p2
```

```
DELETE part3,part16
```

To retain compatibility with earlier versions of Argus BASIC, it is permissible to precede the label with '%'. Thus:

```
GOSUB %Mail
```

is equivalent to:

```
GOSUB Mail
```

If you want to use a label with the same name as an existing command, you must always place a % sign in front of the label, e.g.:

```
GOTO %Connect
```

```
GOSUB %Delete
```

Calculations & expressions

Like any version of BASIC, Argus BASIC can be used for calculations; but as a result of the fact that only integer numbers are allowed, all mathematical operations give integer results.

The following mathematical operators are available:

```
+      : addition
-      : subtraction
*      : multiplication
/      : integer division
DIV    : same as /
MOD    : remainder from integer division
ABS    : absolute value
SGN    : sign
```

The following are examples of expressions using integer variables:

```
X=10 * Y
Scs = (HOUR * 60 * 60) + (MIN * 60)
Res = X / 3 + Y
```

Hexadecimal numbers

Numbers can be entered and displayed in decimal or hexadecimal format. Hex numbers must be preceded by the ampersand (&) character:

```
Hex=&3AC0
Address=PTOP+&1000
Print &A84B
```

Converting hexadecimal numbers into decimal format is achieved by preceding the hex value by the tilde (~) character. For instance, to print the decimal equivalent of a hex value or variable:

```
PRINT ~1200      displays 4608

PRINT ~PTOP      displays the value of PTOp in
                  decimal

PRINT ~1000+50    displays 4146 (i.e. 4096 + 50)
```

By adding the command VAL\$, the decimal value can be stored in a text string:

```
G$=VAL$~120
```

assigns the value "78" to the string variable G\$.

Logical operations

Argus BASIC also supports a full range of logical operators which are used to make comparisons between expressions, usually within conditional commands such as `IF...THEN...ELSE`, `REPEAT...UNTIL` etc.

The logical operators are:

<code>AND</code>	: bitwise AND
<code>EOR</code>	: bitwise Exclusive OR
<code>FALSE</code>	: logical 0
<code>NOT</code>	: bitwise negation
<code>OR</code>	: bitwise OR
<code>TRUE</code>	: logical 1
<code>=</code>	: equal to
<code>></code>	: greater than
<code><</code>	: less than
<code><=</code>	: less than or equal to
<code>>=</code>	: greater than or equal to
<code><></code>	: not equal to

Logical operations by the modem are carried out in a 'bitwise' fashion. For example, the command:

```
X = 6 AND 8
```

would set X to 0 as follows:

```
00000110 AND 00001000 = 00000000
```

Similarly:

```
X = 6 OR 8
```

would set X to 12:

```
00000110 OR 00001000 = 00001110
```

Strings can be compared using logical operators:

```
IF A$="Z" THEN ...
```

```
IF A$>B$ THEN ...
```

```
X=A$<=B$
```

Events

Argus BASIC recognizes many events, partly on the hardware side, that makes programming a lot easier. You can cause any part of your program to be triggered into action when these events occur

While the modem is in BASIC mode, the events enabled will always be operative, even if the program is not running. The events are:

```
ON AT
```

```
ON BRK
```

```
ON BUTTON (1/2/3)
ON CONNECT
ON DTD
ON DTR
ON ERROR
ON ESCAPE
ON HANGUP
ON RESET
ON RING
ON SEQUENCE
ON SLASH
ON TIMEOUT
```

So, if you would like your modem to welcome you with a start-up message whenever you start your terminal program, you might include in your program:

```
10 ON DTR GOSUB Welcome
1100%Welcome
1110 Print "Hello there, this is your modem."
1120 RETURN
```

1.2 Writing programs

There are two distinct methods of programming the modem.

The first, which you have used so far, is referred to as *direct programming*; i.e. programs are typed straight into the modem from your computer, using the terminal mode of your communications software. This method is suitable for entering small programs and for testing individual functions; but due to the lack of editing facilities provided by the modem, it is impractical for large programs.

Large programs are best created using a word processor on your computer and then 'downloading' them into the modem when they are ready for testing. This is referred to as *indirect programming*.

Direct programming

Assuming that the modem is now in command mode with the >1 prompt displayed on screen, you can enter BASIC commands or program lines. The modem is now functioning as a computer and is simply using your PC's keyboard and screen for input and output respectively.

Before starting to enter any new program you should use the following commands to ensure that no other programs are loaded and that the new program is stored in the correct place:

```
BUFFER OFF
NEW
```

For reasons of speed, direct programming is best carried out at 19200 baud or 38400 baud. If your terminal does not support 38400 baud, use the next highest speed that is available. If your terminal is currently running at a slower speed, you may reconfigure the modem as follows:

- Enter the command `WAIT FOR AT`, or press the **STOP** button on the front panel. This will put the modem into 'AT' scan mode.
- Change the baud rate in your terminal software to the required speed.
- Go back on-line. If you pressed the **STOP** button, type:

```
AT*B [Return]
```

if you entered the command `WAIT FOR AT`, type:

```
AT
```

- The baud rate serial and data format for port 1 will now be set to match that of your terminal, and the command mode prompt will be shown.

There is only one editing tool available if you are editing on-line. `CTRL-R` will display the last line entered. The cursor will be at the end of the line and you can delete the characters backwards and/or add new characters.

Despite the limited editing facilities that are available when entering programs directly, it is still an ideal way to explore the modem and learn about the various functions that are provided. Most commands or commands when entered without line numbers will be executed immediately, and you will see the results.

Indirect programming

It is simpler and more efficient to use the word processing or text editing facilities on your computer to write larger programs. In this case you will first create an ASCII text file of the program and save it on disk. This file may then be transferred (downloaded) into the modem, using the ASCII file transfer option in your communications software. The sequence of events for creating and loading a program in this way is as follows:

- Use a word processor or text editor to create the ASCII version of the program.
- Save the program in a file on disk.
- Load your communications software and get the command mode prompt from the modem.
- Use the ASCII file transfer option in your communications software (NOT X/Y-MODEM or any other error correcting protocol), to send the program file to the modem.

Note: If you encounter problems with 'echo' during the transfer procedure, you should reset the modem before disabling echo (using the `ECHO OFF` command), and try again.

When programming indirectly, the first few lines of your program file must contain certain commands which prepare the modem to receive the remainder of the program. Specifically the following must be included:

- `CLEAR` or `BUFFER OFF` - to clear that part of the modem's memory which is used for variable storage.
- `PBOT=&3000` - sets the address in the modem's memory at which the program will be stored. `PBOT` is a system variable that stands for Program `BO`Ttom.
- `NEW` - clears program memory and prepares the modem BASIC interpreter to accept a new program.

If you include line numbers in your word-processed file you must keep track of them yourself. For obvious reasons you will not be able to use the modem's `RENUM` command until you have uploaded the program.

If you do not include line numbers in your program text file, you must include the `AUTO` command at the beginning of the file so that line numbers will be generated for you during uploading. One disadvantage of using a word processor to create program files without line numbers is the difficulty of implementing `GOTOS` or `GOSUBS` to line numbers. As an alternative to line numbers in programs where a limited number of subroutine calls or `GOTO` commands are used, you should use labels. Because labels are easier to remember, we recommend that you *only* use labels.

The first few lines of your program should be:

With line numbers:	Without line numbers:
PBOT=&3000	PBOT=&3000
BUFFER OFF	BUFFER OFF
NEW	NEW
10 REM Program start	AUTO
.	REM Program start
.	.
.	.

Prior to downloading you should ensure that you have hardware handshaking (RTS/CTS), enabled within your terminal software to avoid losing characters during the transfer.

Note: the modem uses hardware handshake by default, but software handshaking can also be selected using the `HANDSHAKE` command.

You should also ensure that `ESCAPE` is enabled in order that it can be used to cancel auto line-numbering when downloading is complete. (See `ESCAPE`.)

Remarks in a text file can be best implemented by using the `\` sign instead of the `REM` command. The BASIC interpreter discards the rest of the line behind `\` when it is in auto mode. Example:

```
X=12 \ Start for X
```

1.3 Running programs

The modem has two (2) different operating modes: it can be used either as a *Hayes compatible* modem or as a *Programmable* (or BASIC) modem. It is important that you know exactly how to change from one mode to the other.

To go from modem-mode to BASIC-mode:

- enter `AT*B` [Return]

To go from BASIC to modem-mode:

- enter `MODEM` [Return], or have `MODEM` as a command in a program line.

or

- In case of an Argus with 5 buttons: first set the **S/A** button in the in-position and push the **STOP** button, then set the **S/A** button in the out-position again.

or

- In case of an Argus with keyboard: press `> + >> + >>>` at the same time. This will jump directly to modem-mode.

The effect of entering modem-mode from BASIC is to:

- disable all events but `ON RESET`. You must be careful here, since once `ON RESET` is effective, the modem will execute the command associated with `ON RESET` after every reset or power shutdown.
- leave all variables and buffers as they are.

While still under development, a program will normally be executed using the `RUN` command. Once development is complete, it is recommended that the program be started with the command `ON RESET`, or `ON RESET GOTO <line number>`. This will make your program start whenever you press the **RESET** button, or after a power failure in the same way. No variables or buffers will be cleared when this happens. If you want the variables and buffers to be cleared, make `CLEAR` one of your first program commands.

```
10 ON RESET PRINT "You are in BASIC once again"

20 PRINT "***";
```

```
30 WAIT 300
```

```
40 GOTO 20
```

If the **RESET** button is pressed immediately after this program has been entered (but before it has been run), nothing will happen. After the program has been run once, the modem will return to BASIC mode after you press the **RESET** button, print "You are in BASIC once again", and send asterisks to your screen.

Pressing the **RESET** button will restart the program again from the first line. If an [Esc]ape sequence ([Ctrl-C]) is now entered, the program will stop; but ON **RESET** will still be enabled. Pressing the **START** button prints 'You are in BASIC once again' and the program will show the asterisks again. If you now press the **STOP** button again, you will return to modem mode once more.

As soon as you press the **RESET** button again after that you will return to BASIC once more and the program will run once more, etc.

Running files from the filing system

If you have a filing system (RAM-, FLASH- or Hard-disk or LAN), you can load and save BASIC programs. The command `RUN "Name"` will first load the program 'Name' in system memory on address `PBOT` (default `&3000`) and then run it. Once ON **RESET** `RUN "Name"` has been entered, you will always start the program 'Name', when you push the **RESET** button.

For standard applications, we have also included a special program named 'BOOT'. This means that if there is a program in memory/disk with the name `BOOT`, it will always run when you press **RESET** or after a power shutdown, even if ON **RESET** is active or inactive. The only way to disable `BOOT` is with the command `BOOT OFF`.

1.4 Program and variable storage

The area of memory between `&3000` and `&7FFF` is known as *Program memory*. BASIC programs are normally stored starting at location `&3000` and working upwards, while the variables used by a program are stored in the area just below `&7FFF`.

A system variable `PBOT` is used to hold the current program start address, and can be changed to allow two or more programs to be stored at the same time in Program memory.

The highest location used by a program is held in another system variable called `PTOP`. The size of a program can therefore be calculated by subtracting `PBOT` from `PTOP`, and the amount of space remaining for variable storage may be calculated by subtracting `PTOP` from `&7FFF`.

Two other commands will reduce the amount of space available for variables:

`BUFFER . . .` if the `BUFFER` command has been used to define a 'user buffer', the specified amount of space will be reserved starting from `&7FFF` and working downwards. This means that variables will be stored at a lower location than usual.

`BUFFER OFF` may be used to release previously allocated space.

Note: the use of `BUFFER` will always clear all variables and is therefore recommended to be one of the first commands in any program.

If a program runs out of memory, it is also possible to split up a big program into smaller programs. Depending on the application, a program can be called from within another program with `RUN "Name2"`, without losing all variables!

Variables are stored in alphabetical order in memory so that, for example, if an integer `A` is the first used variable, its value will be stored in locations `&7FFE` and `&7FFF`.

The first letter of a variable name is therefore very important as two 26-entry tables are maintained (one for integers and the other for strings), which contain the start address for variables starting with a specific letter. Once an entry has been created in the table, all variables with the same first character are stored in the same memory area. As only the first defined variable

with a specific start character is directly accessible, a sequential search of memory must be made to find other variables starting with the same letter. For this reason you should try to use as many different start characters as possible when speed of operation is important.

This method of storage results in the creation of up to 52 separate storage areas. If no variables starting with the letter 'T' are used then no space will be reserved for them. The `CLEAR` command which is used to destroy all program variables simply clears the look-up table and sets the variable pointer to `&7FFF`.

Assuming that no buffer memory is allocated, the memory map will be as follows:

Address	Letter/variable	type
<code>&7FFF</code>	integer variables starting with Z	
	integer variables starting with A	
	string variables starting with Z	
	string variables starting with A	
<code>PTOP</code>	end of program	
<code>PBOT</code>	start of program	

The way in which variables are allocated can be demonstrated by entering the following program:

```
PBOT=&3000
NEW
DIM A(6000)
```

After execution of this program, a single variable with 12K allocated has been created in memory. When variable `B` is defined with the command:

```
B=0
```

you might notice that it takes a little while before the prompt returns. This is because the entire area of memory allocated for the integer array `A` must be shifted in order to create space for the integer variable `B`. Once this has been done, further access of `B` will take no noticeable time, a fact that you may prove by now entering the command:

```
B=1
```

1.5 I/O ports

Each I/O port on the modem is identified by a unique port number:

Port	Device
1	Serial port 1 (25 pin D type)
2	Serial port 2 (9 pin D type)
3	Telephone line (modem chip)
4	reserved
5	reserved
6	Keyboard input & LCD display output
7	Parallel Printer port
8	Memory buffer

Port 1 is serial communications port 1 with asynchronous speeds up to 115200 baud. Synchronous operation is also possible.

Port 2 is serial communications port 2 with asynchronous speeds up to 38400 baud.

Port 3 is the telephone line and is only used if a modem connection is active.

All data streams associated with these ports (receive and send for each), are managed by the modem's operating system, using interrupt driven buffers. All BASIC commands related to the receipt of data (`INPUT`, `GET`, etc.) automatically use the input buffers. Commands which serve to transmit characters, such as `PRINT` and `PUT`, use the output buffers. Buffer management is therefore transparent to the user, whose only concern should be the type of handshaking used (see below).

I/O buffers are 256 bytes in length and are filled and emptied automatically as data is transferred. However, overflow of input buffers will occur at any speed if data is removed from the input buffers more slowly than new data is received AND no handshaking protocol is in use. Loss of data due to overflow can be avoided by enabling either software (`XON/XOFF`), or hardware (`RTS/CTS`) handshaking as appropriate.

Under normal conditions, a program attempting to place data into a full output buffer will be suspended until space becomes available. However when a `LINK` has been established between two ports handshaking must again be used to prevent overflow. Full descriptions of the `HANDSHAKE` and `LINK` commands are given under the appropriate headings in the BASIC reference section.

Port 6 is used for the keyboard as an input and the LCD display as an output (if available). At the back of this manual there are tables for both keyboard and LCD codes..

Port 7 is the parallel printer port. The connector is situated inside the case but a slot in the rear panel is provided for a parallel cable. The parallel port can be defined as an input only or output only.

Port 8 is unusual in that it is not a hardware port, but refers to an area of memory that may be used as a data buffer by BASIC programs. The buffer must be created explicitly using the `BUFFER` command before it can be used, but thereafter it may be treated in the same way as any other port, using commands such as `PUT`, `GET`, `PRINT` and `INPUT`. It can be used for input and output in the way a FIFO stack works.

Port syntax

Port numbers in BASIC commands must always be preceded by the hash or pound (#) character. For example, to print a message to port 2 you could use the following:

```
PRINT#2, "Message from port 2"
```

In all cases the word "PORT" can also be used e.g.:

```
PRINT PORT1, "Message from port 1"
```

Similarly, to read a string from port 2:

```
INPUT#2, A$
```

Your program may be easier to understand if you give names to the ports that you use by assigning them to variables:

```
P=3
```

```
PRINT#P, "text"
```

```
INPUT#P, A$
```

For those commands used to set port attributes such as the baud rate, and those used to output data, you may specify more than one port. The following command will print the message "Commencing file transfer..." to both ports 1 and 2 simultaneously:

```
PRINT#1,#2, "Commencing file transfer..."
```

Commas between the port numbers are optional (`PRINT#1#2`).

Listed below are those commands which may include one or more port numbers as parameters:

AT	ESCAPE	LENGTH	PARITY	SBITS
BAUD	GET	LINEFEED	PRINT	SEND
CLEAR	GET\$	LINK	PUT	XOFF
CTS	HANDSHAKE	LIST	PUT\$	XON
DCD	INPUT	MATCH	RECEIVE	
DSR	KEY	ORI	RTS	
DTR	KEY\$	OUTPUT		

For obvious reasons commands used to input data may only specify a single port.

When repeated access is required to one or more ports you may define default values for input and output using the `IPORT` and `OPORT` system variables which are described fully under the relevant BASIC reference headings.

1.6 The Argus filing system

The primary aim of any communications system is the successful transfer of information. However temporary storage of information is very often required to synchronize communications between two different systems. Very different from any other modem, this modem can store data to be sent, or data that has been received temporarily, to allow transfer to a remote or to the local system later. The data may consist of computer programs, text files or other forms of data. Various standards have been adopted over the years to suit different applications, including MNP or V42 error correction and Xmodem or Ymodem file transfer. The mechanism to support some of the common standards have been built into the modem, and have, as far as possible, been made transparent to the programmer. As a result, the task of receiving, storing and sending files and messages is handled almost entirely by the modem.

In order to manage the various types of information that may exist, the modem can have its own filing system. There are 4 types:

1. RAM-disk (up to 1 Mb)
2. FLASH-disk (up to 2 Mb)
3. Hard-disk (up to 3.6 Gb)
4. Argus LAN (network server + station's)

The RAM/FLASH-disk is 'bank-switched' into the memory map from &A000 to &C000. Bank-switching is fully automatic and requires no special action by the programmer.

In the case of an Argus Programmable Modem, there are two 32-pin sockets provided inside the modem, each of which will accept a 128Kb, 256Kb or 512Kb RAM module. The second socket will accept a 128Kb or 512Kb FLASH module, only if the first socket contains a 128Kb RAM module.

In the case of a T.C.Lite (with keyboard and LCD display) there is only one socket which will accept a 128Kb, 512Kb or 2Mb FLASH module.

The Hard-disk and LAN are only available for the Argus Programmable Modem. The network server is an Argus with Hard-disk and a station is an Argus with build-in network facilities.

The filing system is not a standard feature of the modem, but an option that you can order with your modem, or add later, as and when required.

The filing system of the modem recognizes three distinct types of files:

- Text messages
- Binary files (voice also)
- Argus BASIC programs

File storage is handled entirely by the modem's operating system. Files are stored sequentially in RAM/FLASH-disk, the oldest first.

On a Hard-disk, the files are stored in 'directories' which are created and removed automatically by the operating system; the first character of a filename is the directory entry. Only the characters 0-9, @, and A-Z are possible as the first character in a filename!

The following commands can be used to manipulate files:

CLEAR	PLAY/RECORD
COPY	RECEIVE
DELETE	RENAME
DIR	RESTORE
FAX	RUN
LOAD	SAVE
MATCH	SEND

The syntax for manipulating files in general is:

```
COMMAND ![filename]
```

or

```
COMMAND FILE [filename]
```

However in most cases the exclamation mark (!) or the word "FILE" can be left out. Only with MATCH, CLEAR and RESTORE they must be used.

Filenames may be up to 16 characters long and may only contain ASCII characters from 33 to 127 (so they cannot include spaces). Two other characters with special meanings that can be used in filenames are:

- * : the asterisk (star), which is used as a multiple character 'wildcard'.
- ? : the question mark, which is used as a single character 'wildcard'.
- . : the full point (dot), which is used to split filenames into a maximum of three sections called *extensions*. This allows groups of related files to be given the same extension.

In order for files of a similar type to be grouped together within a directory listing, it is logical to use filename extensions. For example, all BASIC program files could be given the extension '.BAS'; text files could be '.TXT'; etc. To display a list of BASIC programs you would then use:

```
DIR "*.BAS"
```

Text files

Straightforward text files are stored in the disk 'as is', i.e. one byte is used for each ASCII character. However, when it comes to transferring text files the modem supports a number of the conventions used by telex-based systems. For instance, the command:

```
RECEIVE#1, "TEST", T
```

prepares the modem to receive a text file via serial port 1 (the T is short for Text or Teletype). First, the modem transmits the message:

```
Please enter your message:
```

to port 1. It then reads any characters received from port 1 and places them in the disk in a file called "TEST". The file is only closed when either:

- a) a single line containing the characters "NNNN" is encountered in the input stream

or:

- b) an ESCAPE is received.

The second method can be disabled if necessary by using the `ESCAPE OFF` command.

When a transfer is complete, the `DIR` command may be used to verify that a new file has been created. `DIR` will also show the type of the file, how many 256-byte blocks it occupies and how many free blocks (or clusters on hard disk) remain.

A text file can be 'read' by sending it to the port to which you are connected. Thus to display the file "TEST" from the previous example you would use:

```
SEND "TEST" , T
```

Text files can be removed from the disk using the `DELETE` command, but they must be read at least once before this can be done.

Binary files

Binary files may contain any type of data and are transferred using the Xmodem or Ymodem file transfer protocol. For example, to download a binary file into the modem using Xmodem, you could use the command:

```
RECEIVE "DATA1" , X
```

Once this has been entered the modem will initiate an Xmodem receive sequence. The required file can now be downloaded by selecting the Xmodem file transmit option in your terminal software. When the transfer is complete the file will be stored in the disk using the name given in the `RECEIVE` command.

To re-transmit the file use:

```
SEND "NAME" , X
```

This will only work if the receiving system supports Xmodem file transfer. Multiple files can be merged by using the wildcard character in the filename specification:

```
SEND "* .DATA" , X
```

However in most cases it is better to use the Ymodem file transfer, because file length is preserved (see `PROTOCOL` command).

BASIC programs

BASIC programs can be stored to and retrieved from disk using the `SAVE` and `LOAD` commands. Disk programs are accessed by their filename:

```
SAVE "TESTPROG1 . PRG"
```

or

```
LOAD "TESTPROG2 . PRG"
```

The file type as shown in a directory listing will be 'B' for BASIC program.

Programs may also be loaded into Program memory and executed with a single `RUN` command:

```
RUN "HOSTSYS . PRG"
```

1.7 Program backup

The importance of making regular backup copies of both programs and data from your computer cannot be stressed too often, and the same golden rule applies when writing programs for the modem.

If you use a word processor to create and edit programs, your normal computer backup procedures can be applied. Backups should be made every time you modify a program.

A different procedure will be required to maintain backups of programs entered directly into the modem. Although the modem memory is battery-backed and does not lose its contents when power is shut off, it is possible that low usage over a

period of time will cause the battery to run down and memory to be corrupted. For this reason you should make copies of programs from the modem into your computer or to diskette, to use as permanent backups.

Backing up Program memory

Currently active programs, i.e. those in Program memory, are stored in a compact, encoded (tokenized) form, and cannot therefore be read without the aid of the `LIST` command. When you use `LIST`, the stored program is decoded and displayed as ASCII text in the form in which it was originally entered. You may take advantage of this fact to obtain backups in ASCII format of programs that have been entered directly, by using the following procedure:

- Load your terminal software and put the modem into Command mode.
- Use your terminal software to open a file for the reception of ASCII text.
- Go back on-line to the modem and use the `LIST` command to list the program.
- When listing is complete, close the file and save it on disk.

This file can now be stored permanently as a backup, or it can be edited and downloaded back into the modem when editing is complete. To avoid loss of data while using `LIST` in this way, make sure that either software or hardware handshaking is selected as appropriate.

Backing up the RAM-, FLASH- or Hard-disk

Even when you have a filing system installed, the need to make a back-up of all your programs and data, is still there. Normally you will save programs, text and binary files in the disk. It is extremely difficult to erase the disk by accident, and the battery backup (in case of RAM-disk) is quite reliable because of the watchdog facility. Nevertheless, we think that it is a wise precaution to backup valuable data now and then onto floppy or the hard disk of your computer.

The easiest way to create a backup is to connect to the modem locally or from a remote terminal and go into BASIC. Then tell the modem:

```
SEND ALL or SEND "*. *.*.*.*",C
```

and use your communications program to download the complete contents of the disk with Xmodem-CRC file transfer.

To restore the complete contents of the disk from your PC, tell your modem to:

```
RECEIVE ALL or RECEIVE "C",C
```

and use your communications program to return the same data you have received earlier (also with Xmodem-CRC file transfer).

Of course you can also back up separate files from the disk by using the `,C` option. In this way you get a backup-part:

```
SEND "*. PRG",C
```

```
SEND "MESS*. SYS",C
```

Separate received backup-parts can be added together in the PC as one big file, which can be used later to restore a disk.

The `,C` option tells the modem that it is a file Copy, meaning that all information such as name, length and date, is included in the transfer.

Argus BASIC - Command Reference

Argus BASIC is a custom communications' variant of the popular BASIC language found on many micro-computers. It is a relatively fast implementation which owes much of its speed to the use of a 68HC11 microprocessor running at 3.68 MHz, and integer-only operation. However, multiple high-speed communication streams can still present problems, and some care may be required when developing more adventurous applications in order to avoid potential timing problems; especially if you are using error correction and/or data compression at the same time, at high baud rates such as 9600 baud. Remember also that the execution speed of a program can depend greatly upon how it is written. In this respect you should take note of the following if speed is critical in your application:

- Avoid using port numbers in commands - use default values by assigning `I PORT` and `O PORT` appropriately where this is possible.
- Minimize the use of brackets in expressions and function calls - `CHR$7` is the same as `CHR$(7)`.
- Use short variable names.
- Distribute the first characters of variable names across the whole alphabet.
- Combine multiple commands into single program lines.
- Omit variable names with the `NEXT` clause of `FOR . . . NEXT` loops.
- Minimize the use of string variables.
- Split long calculations into separate, short sections.
- Avoid using the `READ` and `DATA` commands - they are relatively slow.
- Using line numbers for `GOTOS` is faster than using labels.
- If you do use labels keep them short (e.g. `%OL`, `%2`).

You should realize that the application of these guidelines may result in poorly structured programs that are difficult to read. Some compromise may be required, particularly during the development stage.

Syntax

Each of the command descriptions in this section is followed by a series of *examples*, a *syntax description*, and a list of *associated commands* to be used as a cross-reference.

The following notation is used in the syntax descriptions:

[-] items in square brackets indicate one of the program 'elements' listed below.

[num-var]	: numeric variable
[num]	: a numeric variable or constant
[string-var]	: string variable; e.g. <code>Name\$</code>
[string]	: string variable or constant
[integer]	: an integer value from -32767 to 32768
[integer n1 ... n2]	: an integer value in the range n1 to n2
[port]	: a port number (1,2,3,7 or 8)

[option]	: a single character option code
[type]	: a single character type code
x y	: either x or y
([...])	: the enclosed item is optional
(, ...)	: optional repetition of the previous item

Other items that appear in the syntax of a command, e.g. #, are shown 'as is'.

ABS is used to obtain the absolute value of an integer; i.e. its magnitude.

```
ABS (-10)
```

will return the value 10.

It is commonly used to find the difference between two numbers:

```
diff=ABS (X-Y)
```

Examples:

```
A=ABS (B)
```

```
PRINT ABS (Temp-38)
```

Syntax:

```
[num-var] = ABS [num]
```

See also:

```
SGN
```

This command will simulate any modem connection, after the two modems are connected. It can therefore best be used after the ON CONNECT event. When entering the command, it will set the link between the terminal port and the line port (3). The terminal port must first be set with the system variables IPORT and OPORT. A multiplexed or half duplex connection is also simulated if selected. Exiting the command must be done by ON-events, which are set before the ACTIVE command was given. The following events are possible during ACTIVE.

ON HANGUP	:	Connection is lost
ON TIMEOUT	:	An inactivity timeout has passed
ON SEQUENCE	:	The escape-sequence is detected (+++)
ON DTD:	:	DTR on the port has dropped
ON ESCAPE	:	An escape is detected

Example:

```
10 ON CONNECT GOTO %Conn
20 DIAL "123456"
30 REPORT:END
40%Conn
50 ON HANGUP GOTO %Loss
60 ON TIMEOUT 10 TIMEOUT OFF: GOTO %Hang
70 ON SEQUENCE GOTO %Hang
80 ON DTD GOTO %Hang
90 IPORT = 1: OPORT = 1
100 REPORT
110 ESCAPE OFF
120 ACTIVE
130%Hang
140 HANGUP
150%Loss
160 REPORT
170 ON OFF
180 END
```

Syntax:

```
ACTIVE
```

See also:

ON CONNECT

The modem is capable of performing an analogue measurement of the signal voltage level on the line. A value between 0 and 255 is returned.

If the modem is 'on-hook' the measurement is always zero.

If you take the modem off-hook and your exchange applies a dial tone, or if a carrier is received, you will be able to see the signal voltage on the line.

To test this, enter:

```
10 DIAL "123456" : REM dial non-existent number  
  
20 REPEAT : PRINT ADC :WAIT 10 :UNTIL FALSE  
  
RUN
```

For specialist applications ADC, together with `FREQ` can be extremely useful, to evaluate what is going over the line. Most users however will have no need for these commands.

Examples:

```
PRINT ADC  
  
Millivolts=ADC
```

Syntax:

```
[num-var] = ADC
```

See Also:

```
FREQ
```

AND**logical AND**

AND performs a bitwise logical 'AND' operation on two numeric values; i.e. each bit of the first operand is ANDed with the equivalent bit in the second operand. The result of 12 AND 5 is therefore calculated as follows:

```
operand 1    12    =    00000000 00001100
operand 2     5    =    00000000 00000101

result   4    =    00000000 00000100
```

The most common use of AND is to check whether or not two or more test conditions are true:

```
IF (X = 2) AND (Y=10) THEN ....
```

The first stage in evaluating this IF command is to check if X=2. If it is, then the first operand for AND will be TRUE (1). If Y=10 is also true then the second operand will also be TRUE (1), and the resulting test will be:

```
operand 1     1    =    00000000 00000001
operand 2     1    =    00000000 00000001

result   1    =    00000000 00000001
```

Because both operands are TRUE the result is also TRUE and the IF condition is satisfied.

Examples:

```
SWITCH(1)=RTS AND CTS

IF MONTH=1 AND DDAY=1 THEN PRINT "New Year!"

IF X>0 AND Y<10 AND Z=5 THEN %calc

REPEAT PUT GET : count=count+1 : UNTIL count>80
```

Syntax:

```
[num-var] = [integer] AND [integer]
```

See also:

EOR, OR, NOT

ANSWER is used to make the modem answer a call and is identical to the modem command A (ATA). The modem will take the line and produce an answer tone, and a carrier or a baud rate scan, as set with the various relevant S-registers.

A typical application is when you do not want to make the modem auto-answer, but first want to observe certain conditions e.g.:

```
50 ON RING %test
.
200 %TEST
210 IF DTR#2 ANSWER
220 RETURN
```

On exit the system variable RESULT can be read to determine the status of the modem (CONNECT, NO CARRIER). When DCD#3 is set, a successful modem connection is made. This can also be detected by the ON CONNECT event.

ANSWER can also be used to set the number of rings that is required before the modem will answer an incoming call. E.g.:

```
ANSWER 3
```

A third syntax is ANSWER ON|OFF. ANSWER ON on is identical to the modem command ATSO=1 and ANSWER OFF to ATSO=0.

Examples:

```
ON RING ANSWER
ANSWER = 3
ANSWER OFF
```

Syntax:

```
ANSWER
ANSWER [num]
ANSWER ON|OFF
```

See also:

```
CONNECT, HANGUP, ON RING, ON CONNECT, ON HANGUP
```


ASC	ASCII value
------------	--------------------

ASC returns the ASCII code of its character parameter. For example:

```
ASC"B"
```

will return the value 66.

When the parameter is a string of more than one character, the ASCII code for the first character in the string is returned, so that:

```
ASC"John"
```

will return the value 74 -- the ASCII code for the letter 'J'.

Examples:

```
X=ASC "@"
```

```
X=ASC X$
```

```
PRINT ASC "g"
```

```
Char=ASC GET$
```

Syntax:

```
[num-var] = ASC[string]
```

See also:

```
CHR$
```

AT is used to initiate 'AT' scanning on one or both of the two serial ports. This is the means by which Hayes compatible modems determine the speed and data format being used by the terminal to which they are attached. When operating in this mode, the modem will scan the specified ports for incoming data. The bit pattern produced by the two characters 'A' and 'T' is unique and can be recognized at baud rates from 75 through to 38400 in a variety of data formats. The advantage of this is that the modem need not be pre-configured for baud rate and data format before connection -- it can determine the appropriate settings when the user first enters 'AT'.

Note that the number of stop bits cannot be recognized, so that if your terminal is set to 7, *Even*, 2, the modem will detect the 'AT' and configure the serial port for 7, *Even*, 1 operation. Most computers and terminals will not be affected by the missing stop bit, but if you do encounter problems you should set the number of stop bits explicitly, using the `SBITS` command.

The modem will successfully 'AT'-detect at the following speeds and data formats:

75	150	300	600	1200	2400
4800	9600	14400	19200	38400	57600

7 bits, even parity

7 bits, odd parity

8 bits, even parity

8 bits, odd parity

8 bits, no parity

If no port number is specified, the default input port as defined by `IPORT` will be assumed.

There are four variations of the command:

AT ([port]) (SLASH) ON/OFF

This will enable or disable the state of a port, where it is waiting for an AT to be entered. When AT is entered, the port is re-configured. To respond to the AT entered, you will have to enable and use the ON AT event. When 'SLASH' is added as an option and the ON SLASH event is enabled, this event can also be generated when a / is received, but then the port is not reconfigured.

AT INPUT ([port])

Will input a line and send it to the modem command line interpreter. The result of the execution of the command line is returned in the system variable RESULT. If you want a written output, you can use the command REPORT instead.

AT REPEAT

Will repeat the last command line sent to the modem.

AT [string]

Will send the string to the command line interpreter of the modem. The string itself should not start with AT. So you will have to enter:

```
AT "DT123 "
```

to make the modem dtmf-dial the number 123 from BASIC. Reading the system variable RESULT will return the result of the action taken.

Notes

When the modem is 'AT' scanning, the complete reception of characters inclusive of ESCAPE is disabled for the selected port; i.e. if port 1 is being scanned, an ESCAPE on port 2 will still be recognized, and vice versa. These will be reset to their previous values when the 'AT' scan is complete.

All active background processes including any LINKS, ON TIMEOUT or ON BUTTON functions, remain active during 'AT' scanning.

After a scan, the baud rate, parity and character length can be read by your program using BAUD, LENGTH, etc.

Examples:

```
AT#2 ON
```

```
AT SLASH#1 ON
```

```
AT INPUT
```

```
AT REPEAT
```

```
AT "DT"+Number$
```

Syntax:

```
AT ([port]) (SLASH) ON|OFF
```

```
AT INPUT ([PORT])
```

```
AT REPEAT
```

AT [string]

See also:

BAUD, LENGTH, PARITY, SBITS, INPUT AT,
ON AT, ON SLASH

AUTO**automatic line numbering**

`AUTO` is used during direct programming to enable automatic generation of the next line number each time [Return] is pressed.

The command has two optional parameters which specify the start line number and the increment.

When used without parameters, the start line number and increment are both 10, and the sequence of line numbers generated will be 10, 20, 30, etc.

If a single parameter is used, such as:

```
AUTO 50
```

this is taken as the start line number which is incremented by 10, giving the sequence 50, 60, 70, etc.

When two parameters are given, the first is used as the start line and the second as the line increment. The following example will generate numbers 100, 105, 110, etc.

```
AUTO 100,5
```

The first parameter can be omitted by including a comma before the second:

```
AUTO ,5
```

In this case the line numbers generated will be 10, 15, 20, etc.

If you use `AUTO` while there is already a program in memory, duplicate lines will be overwritten.

To cancel auto line numbering you must enter the appropriate `ESCAPE` character (which is [Ctrl-C] by default), so you should make sure that `ESCAPE` is enabled.

If you forget to enable `ESCAPE`, the `STOP` button will also cancel auto line numbering.

' \ ' added behind `AUTO` line can be used for remarks. These remarks will not be stored as part of the program.

Examples:

```
AUTO 5,5
```

```
AUTO ,5
```

```
AUTO 100
```

Syntax:

```
AUTO ( [integer] ) ( , [integer] )
```

See also:

```
RENUM
```

BAUD is used to set or read the communications speed of the specified serial port. If no port number is specified, the default output port as defined by `OPORT` will be assumed. The second parameter is a code which represents the required baud rate according to the following table:

Code	Baud rate
0	75
1	150
2	300
3	600
4	1200
5	2400
6	4800
7	7200
8	9600
9	14400
10	19200
11	38400
12	57600
13	115200

One or more port numbers may be specified in a single BAUD command, as in the following example, which sets ports 1, 2 and 3 to 2400 baud:

```
BAUD#1#2#3=5
```

The baud rate of Port 1 (the main port) can be set at either a positive or negative number. Setting the baud rate to a negative number will configure the port for synchronous operation. Using a positive number, will configure the port for asynchronous operation

When you set the S-register 51 of the modem to 0, the baud rate of the modem port (port #3) will also influence the modem operation. The modem will only connect to carriers that match the baud rate. However, any AT-command sent to the modem will override this baud rate setting.

It is not very practical to set the baud rate on a port if you intend to effect an AT scan on that port directly afterwards - because the AT scan will reconfigure the port. In this case, either you set the port for a certain baud rate using BAUD; or you carry out an AT scan on the port and read the baud rate that results with BAUD. For example:

```
Print BAUD#2
```

Pressing the **RESET** button does not affect the baud rate, but does re-initialize serial communication and flush the send/receive buffers.

Examples:

```
BAUD 11
```

```
BAUD#1, 12
```

```
BAUD#Port, Rate
```

```
PRINT BAUD
```

Syntax:

```
BAUD ( #[integer 1..3] , ) ( #... , ) [integer 1..15]
```

See also:

AT, LENGTH, PARITY, SBITS

After every reset, the modem looks for a file 'BOOT' in the RAM/ROM/FLASH- or Hard-disk.

In case of a RAM-disk in an Argus Programmable Modem, it first looks in socket 1, with the first filing system and if it cannot find it, it will look in socket 2, if there is a ROM/RAM in socket 2.

If the BOOT file is found, it is loaded in BASIC at PBOT=&3000 and then executed. It overwrites the ON RESET rules.

If the BOOT file is not found, the normal ON RESET rules will apply. With BOOT ON, this feature is enabled.

With BOOT OFF, the search for the BOOT file is not done after a reset, and the normal ON RESET rules will apply.

See also:

ON RESET, ROM

BRK**transmit a break**

BRK is used to make the modem transmit a break on the specified port. When no port is specified OPORT is used. The duration of the break can be controlled with S-register 43, but the default value of 0.50 seconds is usually sufficient. See ON BRK also.

Examples:

```
BRK#3
```

```
ON HANGUP BRK#1
```

Syntax:

```
BRK (# [port])
```

See also:

```
ON BRK
```

As a result of the fact that the various components of a communications system operate at different speeds, it is often necessary to store data temporarily in an area of memory called a *buffer*. The advantage of this is that one device can take data out of the buffer while another puts new data in. If the buffer is sufficiently large, the faster of the two devices will not have to wait for the other. For instance with a computer, when you read a character from a file on disk, a whole block of data from the file - perhaps 512 bytes - is read into an area of memory called a *file buffer*. When you read the next character the processor does not have to wait for the disk drive again, because the data is already in the file buffer.

The `BUFFER` command on the modem is used to create a buffer of the specified size in RAM. This buffer is treated as if it were a serial port and is addressed as port number 8. The amount of memory that can be allocated for the buffer is limited by the available memory, but the last location, or top of the buffer, is always `&7FFF`. Note that this is also where program variables are stored, so that if you intend to use the `BUFFER` command you must do so before you declare any variables.

When the `BUFFER` command is used, an automatic check will make sure that a program is not overwritten. If there is insufficient room for a buffer of the specified size, an error message will be given.

The command:

```
BUFFER &400
```

will reserve 1K of RAM for the buffer. The maximum buffer size is 20K.

The `PUT` and `PRINT` commands are used to *store* data into the buffer:

```
PRINT#8, "text "
```

```
PUT#8, Ch
```

To *retrieve* data from the buffer use the `GET`, `KEY` or `INPUT` commands.

In the following example the next byte of data is read from the buffer and placed in the variable X:

```
X=GET#8
```

It is acceptable to use PUT#8, INPUT#8, GET#8, etc., to manipulate the buffer contents; but when frequent buffer access is required, it is more efficient to use IPORT and OPORT to select port 8 as the default port. The port number can then be omitted from the I/O commands.

The buffer can also be used in conjunction with the LINK command as if it were a normal port. For example:

```
LINK#1, #8
```

```
LINK#8, #2
```

Here, the input from port 1 will be placed in the buffer before being extracted and passed to port 2. This is particularly useful if you are speed-buffering between ports. When using the buffer in this way it is not possible to implement handshaking between the ports, so that if port 1 is very much faster than port 2 the buffer may overflow, and characters will be lost.

The CLEAR#8 or CLEAR BUFFER command is used to empty the buffer by resetting the input and output buffer pointers to the buffer start address. With a 1K buffer this will be &8000 - &400 = &7C00. Any data still in the buffer when the CLEAR command is used will be lost. However with the command RESTORE BUFFER all the data can be read again.

The BUFFER OFF command releases the space allocated for the buffer for other use. The command CLEAR is then executed automatically, so be careful here.

The buffer contents are not affected by the STOP or RESET buttons.

If no buffer space has been allocated, an attempt to access port 8 will not generate an error message, but will cause the program to 'hang'.

For more complex operations on the buffer, the programmer can use PEEK and POKE on the following buffer-related addresses:

```
256*PEEK&4A4+PEEK&4A5 = buffer start address
256*PEEK&4A6+PEEK&4A7 = buffer end address
256*PEEK&4A8+PEEK&4A9 = buffer start pointer
256*PEEK&4AA+PEEK&4AB = buffer ebd pointer
PEEK&43 AND &80      = buffer full flag
PEEK&44 AND &80      = buffer empty flag
```

Important:

When a modem connection is made and MNP5 is active, a buffer of 3K (&C00) must be defined. This space is used by MNP5 and must not be overwritten during the connection.

When using file transfer with the 1-K block option (Ymodem, Xmodem-1K) together with MNP5, then a buffer of at least 4K (&1000) must be defined.

Examples:

```
BUFFER 1000
```

```
BUFFER Size
```

```
BUFFER OFF
```

Syntax:

```
BUFFER [num]
```

BUFFER OFF

See also:

IPOINT, OPOINT, LINK, RESTORE BUFFER

BUTTON**read the S/A button position**

BUTTON is used to read the position of the sync/async button from within your program. When the button is in the 'in' position, the function will return '1'; and when the button is in the 'out' position, the function will return '0'.

Examples:

```
IF BUTTON THEN GOSUB Syn_Com  
  
PRINT BUTTON
```

Syntax:

```
[num-var] = BUTTON
```

See also:

```
ON BUTTON
```

CALL**jump to machine code subroutine**

The `CALL` command is used to execute a machine language subroutine. Such routines are normally only used where high speed operations are required for which BASIC would be too slow. (Which is not very often, because Argus BASIC is fast!)

Each `CALL` command must be followed by the start address of the routine:

```
CALL &4000
```

In addition to the call address, parameters can be passed to the routine in the 16-bit accumulator and in the 16-bit X and Y registers.

```
CALL &4000, A, X, Y
```

where the variables A, X and Y have been set to the appropriate values before calling.

```
CALL &4000, 1294, 7, 128
```

Here, the accumulator is set to 1294, the X register to 7 and the Y register to 128.

The last instruction of the machine code routine must be `RTS` in order that control is passed back to the BASIC program. On return, the contents of the accumulator and the X and Y registers are stored in locations `&2E00`, `&2E02` and `&2E04` and can be accessed using the `PEEK` function.

When more than three bytes of data must be passed to a machine code routine, the `POKE` command should be used to place the data in memory at a convenient location. The safest method is to reserve an area of memory with the `DIM` command, e.g.:

```
DIM Block 200
```

As the modem does not include a built-in assembler, development of machine code routines must be carried out on another computer. The assembled code may then be uploaded into the modem using the appropriate form of the `RECEIVE` command.

Examples:

```
CALL &3000
```

```
CALL Measure, 2, X, Y
```

```
CALL Calculate, G1, G2
```

Syntax:

```
CALL [address] ( , [Accu], [X-reg], [Y-reg] )
```

See also:

```
PEEK, POKE, RECEIVE
```

The CHR\$ function converts an integer in the range 0-255 to its equivalent ASCII character. It is commonly used to include control characters (ASCII values from 0-31) in PRINT commands. For example, to sound a warning at the terminal when an error occurs, the Bell character ASCII 07 (entered at the keyboard as [Ctrl-G]), could be incorporated into the error message as follows:

```
500 E$="Error !"+CHR$7
510 PRINT E$
```

Examples:

```
X$=CHR$(48+X)
PRINT CHR$75
```

Syntax:

```
[string-var] = CHR$ [integer 0..255]
```

See also:

ASC

CLEAR can be used in one of nine ways:

1. CLEAR

When used on its own, it will clear all BASIC program variables.

2. CLEAR ALL

Will clear the input and output buffers of all ports.

3. CLEAR ([port])

When followed by one or more port numbers, it will clear the input and output buffers for those ports.

4. CLEAR INPUT ([port])

CLEAR followed by INPUT empties the input buffer for the specified port. If no port is specified, the default input port as defined by IPORT will be assumed.

5. CLEAR OUTPUT ([port])

CLEAR followed by OUTPUT empties the output buffer for the specified port. If no port is specified, the default output port as defined by OPORT will be assumed.

6. CLEAR BUFFER

This will empty the buffer you have defined, but not disable the buffer. This function will do the same as CLEAR#8 but takes less program space.

7. CLEAR PRINTER

This will clear the printer port. CLEAR#7 will also clear the printer port.

8. CLEAR! or CLEAR FILE

This deletes all files from the RAM/FLASH-disk and should therefore be used with care! If you accidentally type in this command, you should use RESTORE! immediately to recover the RAM-disk contents. Recovering the FLASH-disk is not possible.

9. CLEAR ROM

Is used when a RAM-disk is in socket 2 to clear the chip.

Examples:

```
CLEAR
```

```
CLEAR ALL
```

```
CLEAR OUTPUT
```

```
CLEAR INPUT#1 , #3
```

```
CLEAR PORT 1
```


CLEAR BUFFER

CLEAR FILE

CLEAR !

Syntax:

CLEAR

CLEAR ALL

CLEAR#[port] (, ...)

CLEAR INPUT (#[port]) (, ...)

CLEAR OUTPUT (#[port]) (, ...)

CLEAR BUFFER

CLEAR PRINTER

CLEAR (!/FILE)

CLEAR ROM

See also:

RESTORE, RESTORE ROM, ROM

CLOCK\$ is used to read the time and date from the internal real-time clock. Output is given in the format:

```
DAY      dd-mm-yy
        HH:MM:SS
```

DAY is day of the week (e.g. SUN); dd is the day of the month, mm is the month, yy is the year, HH is the hour, MM is the minute and SS is the second.

CLOCK\$ can be used to set the date and time, but this can usually be done more easily using the individual DAY, DDAY, MONTH, YEAR, HOUR, MIN and SEC commands.

CLOCK\$ can be copied into another string to store the time and date of a certain event permanently.

Examples:

```
C$=CLOCK$
CLOCK$
```

Syntax:

```
[string-var] = CLOCK$
```

See also:

```
DDAY, DAY, MONTH, YEAR, HOUR, MIN, SEC,
DATE$, TIME$
```

CONNECT**connect to remote modem**

CONNECT is identical to the modem command O (ATO). The modem will take the line, wait for an incoming carrier and try to connect to it. The connection will only be established, when the configuration of the relevant S-registers matches the remote carrier.

The ON CONNECT event can be used to detect a successful connection. This can also be determined by reading DCD#3 on exit of the CONNECT command.

The system variable RESULT can be read on exit to determine the status of the modem (NO CARRIER, CONNECT).

Examples:

```
CONNECT
```

```
HOOK OFF : WAIT 20 : IF FREQ > 500 CONNECT
```

Syntax:

```
CONNECT
```

See also:

```
ANSWER, HANGUP
```

This command is used to copy one file on the disk to another file. The new file is identical to the original one. Using wildcards (* or ?) is not possible.

Examples:

```
COPY "MYPROG" , "MYPROG.BAK"
```

```
COPY "MYPROG" TO "MYPROG.BAK"
```

Syntax:

```
COPY (FILE|!) [string] (TO|,) (FILE|!) [string]
```

See also:

```
LOAD, SAVE, SEND, RECEIVE, DIR
```

CTS is one of the hardware handshaking signals defined in the RS232 standard, and each serial port on the modem is capable of controlling a CTS signal as an output.

CTS is a signal provided by modems and other DCEs, to inform the terminal (DTE) that it is ready to accept data for transmission. It is used in conjunction with another signal, RTS (Ready To Send) as part of the RS232 handshaking sequence.

When the CTS signal is taken low by the modem, a terminal or computer must respond by stopping further transmission of data until CTS goes high again. If the terminal does not respond, transmitted characters may be lost.

After the `HANDSHAKE CTS` command has been issued, the modem will respond correctly to RTS and control CTS automatically.

The CTS function is used to read or set the value of the CTS signal on the currently selected output port. When the CTS line is high, the CTS function will return the value 1; if it is low, the value 0 will be returned. If no port is specified, the output port defined by the last used `OPORT` command is assumed when writing (setting); and the last used `IPORT` is used when reading.

If the modem is unable to handle any more incoming characters because its input buffer is nearly full, it will request the sending device to wait by taking its CTS output low. This feature can be switched on or off with the `HANDSHAKE` command.

The modem also supports software handshaking using the XON/XOFF protocol. Refer to the commands `XOFF`, `XON`, and `HANDSHAKE` for more information.

Examples:

```
X=CTS  
  
SWITCH(1)=CTS#1  
  
PRINT CTS#2  
  
IF CTS PRINT "READY"  
  
CTS ON  
  
CTS OFF
```

Syntax:

```
[num-var] = CTS ( #[port] )  
  
CTS (#[port])=[numeric 0...1]  
  
CTS ON/OFF
```

See also:

`HANDSHAKE`, `XOFF`, `XON`, `RTS`

The `DATA` command is used to store numeric and string constants within a program. Such values may be read one or more times using the `READ` command.

`DATA` commands may appear anywhere in the program, but must be the first and only command appearing on a line. As many data items as will fit on a line may be included, with individual data items being separated by commas.

The following example contains 5 items (3 numbers and two strings):

```
70 DATA 5, 10, 50, Apple, Orange
```

Where a program contains more than one `DATA` command, they should be thought of as comprising a single continuous list of data items; i.e. having read the last item of data on a particular line, the next `READ` command will take the first item of data from the next `DATA` command in the program.

To include leading spaces within a string it must be enclosed in double quotes:

```
50 DATA "      This string contains leading spaces"
```

As the comma is used as a separator in `DATA` commands, you may only include a comma as an item of data by enclosing it within double quotes:

```
60 DATA <, >, ?, ", ", /
```

To include a double quote mark in a `DATA` command you must use four sets:

```
80 DATA "" ""
```

```
90 READ x$
```

After line 90 has been executed `x$` will have the value `''`

Examples:

```
DATA 1,2,3,5,7,11,13,17,19
```

```
DATA Peter,15, Clare,12, Robert,13, James,18, Alison,15
```

Syntax:

```
DATA [string] | [integer] ( , ... )
```

See also:

```
READ, RESTORE
```

DATE\$ is used to read the date from the internal clock/calendar, and format it into a text string in the form DD-MM-YY. When I wrote this text on the 23rd of July 1990, PRINT DATE\$ reported 23-07-90.

The date can also be obtained in integer format by using the individual functions DAY, DDAY, MONTH and YEAR.

DATE\$ can also be used to change the system date.

Examples:

```
D$(I)=DATE$  
Today$=DATE$  
PRINT DATE$  
DATE$ = "23-07-90"
```

Syntax:

```
[string-var] = DATE$  
DATE$ = [string]
```

See also:

CLOCK\$, TIME\$, DAY, DDAY, MONTH, HOUR, MIN, SEC, YEAR

DAY is used to read or set the 'day of week' on the internal clock/calendar.

When *reading* it returns an integer from 1 to 7, day 1 being Sunday. Similarly, when *setting* the day of week, the day number from 1 to 7 must be specified. An incorrect value will cause the current setting to remain unchanged.

Examples:

```
DAY=5  
  
IF DAY=1 THEN Day$="Sunday"  
  
PRINT DAY
```

Syntax:

```
DAY = [integer 1..7]  
  
[num-var] = DAY
```

See also:

```
DATE$, DDAY, MONTH, YEAR
```


DCD is the RS232 signal used by modems to indicate that a connection has been established with a remote system. It is activated by the modem when a connection occurs, and is normally used as an output from the modem. However within the BASIC environment, it can also be used to read the connect status of modem port #3.

With port #1 or #2 the DCD command is used by the modem to control or read the status of the DCD line on the specified port. If no port is specified, the default port defined by `OPORT` will be assumed. The DCD signals of port 1 and port 2 are connected via hardware; thus DCD signals on both ports will influence each other, and it is impossible to set them separately.

With port #3 it is only possible to read the status. For example, the following command causes program execution to jump to the label `%Connect2` if the DCD input on port 2 is active:

```
IF DCD#3 GOTO %Connect2
```

However, in order to detect incoming calls correctly using the DCD function, a program must continuously check the appropriate DCD signal in a loop, until its status changes from `FALSE` to `TRUE`. This is because a single DCD call gives only a momentary reading of DCD status. For this reason it is more efficient to use the `ON CONNECT` and `ON HANGUP` event handlers to achieve the same result.

It is sometimes necessary to control the use of DCD as an output from the modem - for example, where you have protected entry to your host computer by using auto-dial back from your modem. In this case, you would tell the computer that a connection has been established only after the user has been correctly identified and authorized to proceed. Before that you would keep DCD low, and your computer would not know that any communication was already going on. To keep DCD low on the ports while your modem is connected, `AT&C` has to be set first, and afterwards DCD has to be enabled by your program:

```
DCD = 1 or DCD ON      will activate DCD
DCD = 0 or DCD OFF    will clear DCD
```

Examples:

```
IF DCD#3 GOTO %Connection
PRINT DCD#3
DCD=1
DCD OFF
```

Syntax:

```
[num-var] = DCD(#[port])
DCD = [integer 0..1]
DCD ON/OFF
```

See also:

```
DSR, ON CONNECT, ON HANGUP
```

DDAY is used to read or set the day within the month for the internal clock/calendar. When *reading* it returns an integer from 1 to 31. Similarly when *setting*, the day number from 1 to 31 must be specified; an incorrect value will cause the current setting to remain unchanged.

Examples:

```
DDAY=15  
  
IF (DDAY=1 OR DDAY=21 OR DDAY=31) Suffix="st"  
  
PRINT DDAY
```

Syntax:

```
DDAY = [integer 1..31]  
  
[num-var] = DDAY
```

See also:

```
DATE$, DDAY, MONTH, YEAR
```

The `DELETE` command, in its various forms, is used to delete lines from a program and to delete files from the disk filing system.

Deleting program lines

`DELETE` may be followed by two line numbers which define the range of lines to be removed. The command:

```
DELETE 50,100 or DELETE 50 TO 100
```

will remove all program lines from 50 to 100 (inclusive).

If the first parameter is omitted all lines from the start of the program to the specified last line will be deleted:

```
DELETE ,50
```

To delete a single line, just type the line number followed by [Return].

Labels can be used to specify the range in place of line numbers:

```
DELETE %Dial, %Dialend
```

Deleting files from the RAM-, FLASH- or Hard-disk

`DELETE` can also be used to remove files from the disk filing system. In this case the command is followed by `!`, `FILE` and/or a string.

To delete the program `"TEST.PROG"` you could alternatively use the following:

```
DELETE !"TEST.PROG"
```

```
DELETE "TEST.PROG"
```

```
DELETE FILE "TEST.PROG"
```

```
F$="TEST.PROG" : DELETE F$
```

To delete multiple files the `'*'` may be used as a wildcard character. Note however, that unread files will not be deleted by this form of the command, and a report will be printed of the number of files deleted and unread. For example, if there were 7 files with the suffix `"MSG"`, two of which were unread, the command:

```
DELETE "*.MSG"
```

would result in the following report:

```
2 Unread
```

```
5 Deleted
```

To delete unread messages the `DELETE` command must be followed by `' , U '`:

```
DELETE "*.MESS" ,U
```

After deleting files, the system variable `MATCH` holds the number of deleted files.

Examples:

```
DEL 10,200
```

```
DELETE %Search, 500
```

```
DEL "*.JOHN"
```

```
DELETE "*.ANN",U
```

Syntax:

Program lines:

```
DELETE [line-num | label] TO|, [line-num | label]
```

RAM-, FLASH - or Hard-disk files:

```
DELETE (!/FILE) "[string]" ( ,U )
```

See also:

MATCH

DIAL**dial number**

This command will dial a number for you. It is the same as ATD in modem mode, and is provided mainly to make your program easier to read.

After the command DIAL, the system variable RESULT can be read to determine the status of the modem (BUSY, NO ANSWER, CONNECT, etc.). When a successful modem connection is made, DCD#3 will be set. This can also be detected with the ON CONNECT event.

Examples:

```
DIAL TONE "2702719"
```

```
DIAL PULSE VIDICODE$
```

Syntax:

```
DIAL (TONE|PULSE) [string]
```

`DIM` (short for dimension), is used to reserve space for single dimension arrays of integers or strings. The maximum number of elements allowed in integer arrays is theoretically 32767, but is in practice limited by the amount of memory available for variables after the program has been loaded. The limitation of a maximum string length of 254 bytes still current in the Argus Communicator has now been removed.

The command:

```
DIM A(100)
```

defines `A` as being an array of 100 integers (indexed from 0099); i.e. the first element is numbered 0. To access the fifth element use:

```
A(4)
```

Similarly:

```
DIM A$(8)
```

reserves space for the array `A$` with 8 strings of up to 255 characters. The actual amount of space used by the array will depend upon the values assigned to the elements.

A very different use of the `DIM` command is to find and reserve free space in memory. If you need free space just enter:

```
DIM [num-var] [integer]
```

The number of bytes as given by the integer is reserved, and the numeric variable will point to the first location of the memory space reserved for you.

Examples:

```
DIM Names$(20) , ID$(5)
```

```
DIM Primes(10)
```

```
DIM A 256
```

Syntax:

```
DIM [num-var] ( [integer] ) ( , ... )
```

```
DIM [string-var] ( [integer 0..254] ) ( , ... )
```

```
DIM [num-var] [integer]
```

The `DIR` command is used in its various forms to list and inform you about the contents of the RAM-disk.

Because of the need to retain compatibility with earlier versions of Argus BASIC, `DIR` might be followed by 'FILE' or '!'. This is optional however, and thus will not normally be used.

The `DIR` command will list files stored in the RAM-disk. The format and level of information given in the listing depends on the parameters used in the command. The following options are available:

<code>DIR ([string])</code>	standard listing
<code>DIR ([string]),A</code>	All available information
<code>DIR ([string]),B</code>	information Block listing
<code>DIR ([string]),D</code>	standard listing plus file creation Date
<code>DIR ([string]),F</code>	special File information listing
<code>DIR ([string]),N</code>	file Names only listing
<code>DIR ([string]),R</code>	Restricted listing

In each case the `[string]` parameter provides a template for filenames to be included in the listing. The asterisk, '*', can be used as a wildcard to match any valid character.

If no `[string]` parameter is given, all files will be listed.

The formats of the various directory listings are as follows:

Standard	[FN] [Size] [Name+Ext]
,B (Blocks)	[Information block contents]
,A (All)	[FN] [Name+Ext] [Date] [Time] [Read] [Type]
,D (Date)	[FN] [Size] [Name+Ext] [Date] [Time]
,F (special)	[FN] [Size] [Name] [Date] [Read] [Info]
,N (Name)	[Name]
,R (Restricted)	[FN] [Size] [Name]

where the following definitions are used:

[FN]	the file number
[Size]	the number of 256-byte RAM disk blocks occupied by the file
[Name+Ext]	the full name of the file (including extensions)
[Name]	the first part of the file name (e.g. MESSAGE1)
[Date]	the creation date of the file (DD/MM)
[Time]	the creation time of the file (HH/MM)
[Read]	R for Read, U for Unread
[Type]	the type of file

After these options there are two more, which can always be added

,U	(Unread only)
,M	(more function)

The 'more' function will, after sending a screenful of lines, ask you for confirmation before it will send the next screenful.

There are four possible file types:

- B** : BASIC program
- T** : Text file; created by `RECEIVE . . . , T`
- X** : binary file created by `RECEIVE . . . , X` (X-MODEM or Y-MODEM)

The directory command lists files in date order, oldest first, and prints the number of free blocks remaining at the end of the list (not with `DIR, N`).

The command `DIR, B` is used to list the contents of the information block associated with the file. This block contains 32 bytes of application-dependent information and is terminated with [CR] (see `SEND` and `RECEIVE` for further information).

Directories are listed to the specified port, or if this is omitted, to the default output port as defined by `OPORT`.

Examples:

```
DIR, U
DIR, M
DIR#2
DIR#1 "*.MSG?", N
DIR !"LETTER"+VAL$(nn), D
DIR !A$
DIR FILE "PETE."+Ext$, N
```

Syntax:

```
DIR ( #[port] , ) (FILE|!) ( [string] ) ( , [option] )
[option]=A|B|D|F|N|R|U|M
```

See also:

```
LOAD, SAVE, RECEIVE, SEND
```


The `DIR ROM` command is used to access the ROM or RAM disk in socket 2. The command is used in its various forms the same as listed in the `DIR` command.

However, to access a RAM disk in socket 2, the command `CLEAR ROM` must be entered first.

See also:

`DIR`, `ROM`

The `DIV` command is used to carry out integer division operations. As the modem does not support 'real' numbers, `DIV` is exactly equivalent to division using the `/` operator.

Examples:

```
Result = X DIV 5
```

```
Q1 = (Q2*4) DIV (Q3+2)
```

Syntax:

```
[num-var] = [integer] DIV [integer]
```

See also:

MOD

DSR is an RS232 signal used by modems to indicate that they are powered-up and ready to transfer data. DSR will be *active* dependant on the setting with the modem command `&S`. The signal is an output to the modem.

The `DSR` command is used to set or read the status of the DSR line on the specified port. When used for reading it returns `TRUE` if DSR is active, and `FALSE` otherwise.

Examples:

```
DSR ON  
  
DSR = 0  
  
Status = DSR  
  
PRINT DSR
```

Syntax:

```
DSR= [num-var]  
  
DSR ON/OFF  
  
[num-var] = DSR
```

See also:

DCD, DTR

DTMF ON turns on the DTMF receiver. Reading the first DTMF value and the command VOICE will also turn it on. Therefore, use of the command, is optional.

DTMF OFF turns off the DTMF receiver.

In most cases, the modem has 2 DTMF receivers build in. They can be selected with S-register 23 bit6 as follows:

SREG23=SREG23 AND &BF : Select External DTMF-chip (default)

SREG23=SREG23 OR &40 : Select Modem-chip

Example:

DTMF ON

Syntax:

DTMF ON | OFF, VOICE

Reading this function will return 0 or an ASCII number. When it is used for the first time, it will also turn on the DTMF receiver.

Values:

0 =	No DTMF detected
48 =	'0' detected
49 =	'1' detected
50 =	'2' detected
51 =	'3' detected
52 =	'4' detected
53 =	'5' detected
54 =	'6' detected
55 =	'7' detected
56 =	'8' detected
57 =	'9' detected
42 =	** detected
35 =	# detected
65 =	'A' detected
66 =	'B' detected
67 =	'C' detected
68 =	'D' detected

Examples:

```
PRINT DTMF
Var=DTMF
REPEAT D=DTMF : UNTIL D
IF DTMF=52 GOTO %TONE4
```

Syntax:

```
[var] = DTMF
```

See also:

```
DTMF ON/OFF, PLAY [filename], VOICE
```

DTR is the RS232 signal used by data terminals - also (host-) computers - to indicate that they are powered up and ready to transfer data. The modem will automatically control the response to DTR according to the instructions given with the modem command &D.

The DTR function is used to read the status of this signal on the specified port. If no port is specified the default port defined by IPORT will be assumed.

We would suggest that if you wish to make use of the DTR signal from within your BASIC application, you disable the automatic response to DTR.

For enabling or disabling the automatic response (and also the ON DTR and ON DTD events), we suggest that you use the modem commands &D0 and &D1, &D2, &D3.

Examples:

```
PRINT DTR

IF DTR Welcome_User

X=DTR
```

Syntax:

```
[num-var] = DTR ( #[port] )
```

See also:

```
CTS, DCD, DSR, RTS
```

DUTCH

select language and screen mode

The command `DUTCH` will, like `ENGLISH`, select the language for the few written messages that are part of the system. The reason for this is that these messages will also occasionally be sent over the telephone line to remote users. This is especially the case when the modem is used as an electronic mail computer.

The 5 possible parameters behind `DUTCH` will have effect on the `SEND`, `T` and `FAX SEND` commands.

```
DUTCH <1>, <2>, <3>, <4>, <5>
```

```
<1> = 0-255 = Characters per Line (0=OFF)
<2> = 0-255 = Lines per Page (0=OFF)
<3> = 0/1 = No/Yes Left Margin
<4> = 0/1 = No/Yes Page Fillout
<5> = 0/1 = No/Yes Clear Screen (ASCII 12) after 'Verder'
```

Examples:

```
DUTCH
```

```
DUTCH 80, 23, 1, 0, 1
```

```
DUTCH 255, 66, 1, 1, 0
```

Syntax:

```
DUTCH ( [int 0..255] ), ( [int 0..255] ), ( [int 0..1] ), ( [int 0..1] ), ( [int 0..1] )
```

See also:

```
ENGLISH
```

ECHO

enable/disable character echo

ECHO is used to turn local character echo on or off. While in command mode or when using the INPUT or AT commands, characters received via the input port are echoed back so that the sender can see what has been received (or, in the case of a terminal, what he is typing). This is known as *local echo*. If both the modem and the user's terminal provide echo in this way, the user will see two of everything he types, and echo should then be disabled on either the modem or the terminal.

Examples:

```
ECHO ON
```

```
ECHO OFF
```

Syntax:

```
ECHO (# [PORT] , ) (# . . . , ) ON|OFF
```

See also:

```
AT, INPUT, RECEIVE
```


END

end of program

The `END` command is used to terminate program execution and return to command mode. For example:

```
50 Ch$=GET$  
60 IF Ch$="Q" or Ch$="q" THEN END
```

`END` is also used in conjunction with the `TIMEOUT` command to indicate the end of a timeout subroutine.

Example:

```
END
```

Syntax:

```
END
```

See also:

```
ON TIMEOUT, STOP
```

The command `ENGLISH` will, like `DUTCH`, select the language for the few written messages that are part of the system. The reason for this is that these messages will also occasionally be sent over the telephone line to remote users. This is especially the case when the modem is used as an electronic mail computer.

The 5 possible parameters behind `ENGLISH` will have effect on the `SEND`, `T` and `FAX SEND` commands.

```
ENGLISH <1>, <2>, <3>, <4>, <5>
```

```
<1> = 0-255 = Characters per Line (0=OFF)
<2> = 0-255 = Lines per Page (0=OFF)
<3> = 0/1 = No/Yes Left Margin
<4> = 0/1 = No/Yes Page Fillout
<5> = 0/1 = No/Yes Clear Screen (ASCII 12) after 'More'
```

Examples:

```
ENGLISH
```

```
ENGLISH 80, 23, 1, 0, 1
```

```
ENGLISH 255, 66, 1, 1, 0
```

Syntax:

```
ENGLISH ( [int 0..255] ), ( [int 0..255] ), ( [int 0..1] ), ( [int 0..1] ), ( [int 0..1] )
```

See also:

```
DUTCH
```

EOR**logical Exclusive OR**

EOR performs a bitwise logical 'Exclusive OR' operation on two numeric values; i.e. each bit of the first operand is Exclusive ORed with the equivalent bit in the second operand. The result of `12 EOR 5` is therefore calculated as follows:

```
operand1 12    =    00000000 00001100
operand2  5    =    00000000 00000101
result   9     =    00000000 00001001
```

The most common use of EOR is to toggle a bit value within an item of data. For example, to toggle the state of user port line 5 you would use:

```
OLINE(5) = OLINE(5) EOR 1
```

If the state of the line was previously high, the command is the same as:

```
1 EOR 1
```

which gives 0 - i.e. OLINE(5) goes low.

If OLINE(5) was already low, the command is the same as:

```
0 EOR 1
```

which gives 1 and sets OLINE(5) high.

Examples:

```
DCD=DCD EOR 1
```

```
PRINT 15 EOR 4
```

Syntax:

```
[num-var] = [integer] EOR [integer]
```

See also:

```
AND, OR, NOT
```

ERL is a system variable that contains the number of the line that is currently being executed. As a result of this, when an error occurs, ERL contains the line number in which the error occurred. Therefore you should be careful. ERL should be stored or printed in the first line of the error handler routine.

Examples:

```
ON ERROR PRINT ERL

ON ERROR PRINT "Program halted due to error at
line ";ERL : STOP
```

Syntax:

```
[num-var] = ERL
```

See also:

```
ERN, ON ERROR, REPORT
```

ERN is a system variable that contains the number of the last error that occurred. A complete list of error messages and the associated codes generated by the Argus BASIC interpreter is given in the **Error Messages** section of this manual.

ERN is also an important variable for the filing system.

Example:

```
PRINT ERN
```

Syntax:

```
[num-var] = ERN
```

See also:

```
ERL, ON ERROR, RECEIVE, REPORT, SEND
```

The `ESCAPE` command is used to define the ASCII character that the modem treats as an 'Escape' code (this is only sometimes related to the [Esc] key on your keyboard).

An Escape is used to interrupt the execution or listing of a program.

The default value for `ESCAPE` is [Ctrl-C].

For instance, to use ASCII character 04 ([Ctrl-D]), as the `ESCAPE` character you would use the command:

```
ESCAPE=4
```

On reception of the `ESCAPE` character, error number 3 will be generated and any pending `ON ERROR` routine will be executed. All I/O buffers will be emptied, all RTS lines will be activated, and all output buffers will be enabled.

There can be only one escape character for all ports at any time.

Escape recognition can be disabled using `ESCAPE OFF`, and is re-enabled using `ESCAPE ON`. Enabling and disabling of escape recognition can be per port. When no port number is specified, the default port as defined by `OPORT` will be assumed. In any other case you must specify the port; e.g. `ESCAPE#3 OFF`.

After pressing the STOP button, the Escape character for each port will be reset to [Ctrl-C], with handling of `ESCAPE` characters being enabled for port 1, and disabled for all other ports.

Examples:

```
ESCAPE 3
```

```
ESCAPE=27
```

```
ESCAPE#1, #2, #3, 1
```

```
ESCAPE ON
```

Syntax:

```
ESCAPE ( #[port], ) (#...) (=) [integer 1..128]
```

```
ESCAPE ( #[port], ) (#...) ON|OFF
```

FALSE**Boolean system variable**

`FALSE` is a system variable which returns the value 0, and it is used in many logical operations to make programs more readable. A further advantage of using `FALSE` (in the correct context), is that it uses only 1 byte of storage, whereas the integer 0 occupies 2 bytes.

Examples:

```
Ready=FALSE  
  
REPEAT ... UNTIL FALSE  
  
IF X=FALSE
```

Syntax:

```
[num-var] = FALSE
```

See also:

```
NOT, TRUE
```

This is used in the programming environment to enable/disable the Fax <> Modem selector. Same as AT " *F3 " .

The selector is used during the first stage after answering a call. It will wait some time for a Fax or Modem Calling Tone (1100 or 1300 Hz). Default 5 seconds is waited and this can be changed by setting SREG62 per 100 ms. If no Calling Tone is 'seen' the modem continues as a Fax (bit0 in SREG62 is 0) or as a modem (bit0 in SREG62 is 1).

Example:

```
ON HANGUP FAX MODEM ON
```

Syntax:

```
FAX MODEM [ON|OFF]
```

See also:

```
ANSWER, FAX
```


With `FAX ON` the modem will wait for an answer tone after dialing and then return with result code 19 (=ESCAPE). The `FAX SEND` and `FAX RECEIVE` commands will both automatically set `FAX ON`.
With `FAX OFF` the modem tries to connect as a modem.

Examples:

```
FAX ON  
  
DIAL TONE "123456"  
  
IF RESULT=19 GOTO Continue ELSE GOTO Error
```

Syntax:

```
FAX [ON|OFF]
```

See also:

```
FAX SEND, FAX RECEIVE
```

The `FAX RECEIVE` command is a powerful tool that allows the Argus to receive a fax directly onto the disk filing system. Before using the `FAX` command a RAM buffer of at least 1024 bytes big must be defined with the `BUFFER` command.

The `FAX RECEIVE` command consists of three parts.

FAX RECEIVE [file]

Filename can have a maximum of 10 characters. No wildcards are allowed. After connecting, a file is stored which starts with page number '01'. Also the resolution is added to the file name. Example:

```
FAX RECEIVE "test"
```

will receive file 'TEST.01.S' or ('TEST.01.F')
next page 'TEST.02.S' or ('TEST.02.F')
etc.

There is 1 exceptions on the file name, namely when the name starts with a '#' (hash):

```
FAX RECEIVE "#"
```

the modem will first connect and then send all the received G3 data through the fax printer driver. It will then be printed out on the printer on port 1, port 2 or port 7 (depending on `OPORT`).

FAX RECEIVE [file], [option]

These are the same options as for `FAX SEND`. The ', U' and ', T' and ', N' options have no effect.

If you only want to receive in standard resolution you can add ', S'. Example:

```
FAX RECEIVE "test",S
```

If you add ', F', it depends on the sending fax whether it will be received as a standard or fine resolution file.

The speed register (S51) can also be set to ensure reception at a known speed. If it is set at 0 (S51=0) then the highest speed, after negotiation with the other fax machine, is used.

During the handshaking process, the two faxes exchange their ID's. The ID numbers are strings with a maximum length of 20 characters and may only contain numbers and the characters '+' and '-'. The ID of the sending fax machine will be stored at address &2950 after leaving the `FAX RECEIVE` command, while your own ID must be stored at address &2970 before executing the `FAX RECEIVE` command. The commands `POKE$` and `PEEK$` are used to write and read the ID strings.

FAX RECEIVE [file] TO [number]

A number can be added in a special case, namely polling. Example:

```
FAX RECEIVE "test" TO "01223-1227"
```

The modem will dial up another fax machine and will look for a ready to send document (polling). If there is a document it will receive it and store it in the usual way.

See also:

ERN, FAX SEND, MATCH, RESULT

The `FAX SEND` command is a powerful tool that allows the Argus to send a fax from the disk filing system. Before using the `FAX` command a RAM buffer of at least 1024 bytes big must be defined with the `BUFFER` command.

The `FAX SEND ... TO ...` command consists of three parts.

FAX SEND [file]

The file name can be any name up to 16 characters long. If the file name is 10 characters or less, the system will automatically add ".01*.*" as the first page number. If found, the sending continues with ".02*.*", ".03*.*", etc until no more files/pages are found. Example:

```
(file) = "TEST" will send 'TEST.01'
```

```
'TEST.02'
```

```
etc.
```

or

```
'TEST.01.S'
```

```
'TEST.02.S'
```

```
etc.
```

If the file name is more than 10 characters long OR a wildcard is used (* or ?), the system just sends the matching files, without adding page numbers. If you are using wildcards in the file name, many files can be sent with one command. Example:

```
(file) = "TEST*.*" will send 'TEST'
```

```
'TEST1'
```

```
'TEST999'
```

```
'TEST-ALL'
```

```
'TEST-TEXT.01'
```

```
'TEST-TEXT.02'
```

```
'TEST-BIN.S'
```

```
'TEST-BIN.F'
```

```
'TEST.T'
```

```
'TEST50.01.T'
```

When a file name ends with '.T' or nothing, it will be treated as a text-file. Conversion to G3 format is done on the fly during sending.

When a file name ends with '.S' it will be treated as a Standard resolution G3 file. Standard resolution can not be converted to Fine resolution by the modem!

When a file name ends with '.F' it will be treated as a Fine resolution G3 file. Fine resolution is automatically converted to Standard during sending if the receiving fax doesn't support Fine or if sending is fixed to Standard.

When a file name ends with ' .G' it will be treated as a G3 file with an unknown resolution. How the file is sent depends on the negotiation of the 2 fax machines and on the way the resolution is set with S-register 17.

Page numbers can be maximum of '99' (after that it will continue with '.AO', '.A1',til '.Z9').

FAX SEND [file], [option]

Some options can be added to the command:

```
,U ; Unread files only
,S ; Send in Standard resolution only
,F ; Send in Fine resolution only
,T ; Include Top Line and strip lines of a 'F' or 'S' file
,N ; Don't send Top Line and don't strip from a G3 file
```

Examples:

```
FAX SEND Text$,U,T,S TO Tel$
```

```
FAX SEND Text$,N TO Tel$
```

```
FAX SEND Text$,F TO Tel$
```

These options can also be selected by S-register 17:

S17 ; Fax flags

```
b7      =0/1= Fax connection disabled/enabled (FAX OFF/ON)
b6      =0/1= No/Yes send own Top Line (,N/,T)
b5      =0/1= Standard/Fine resolution (,S/,F)
b4      =0/1= No/Yes strip Top Line of bin file (,N/,T)
b2+b3   = Select line-time: 0,10,20,40 ms
b0+b1   = Don't alter (not for faxing)!
```

Stripping the Top Line from a G3 file will remove the first 16 (Standard) or 32 (Fine) scan-lines during the sending of that file. This is equal to 2 text-lines when compared to a text-fax.

Adding your own Top Line is enabled by setting bit6.

Bit7 is automatically set/cleared by the system when using the FAX command.

Selecting the line-time is only used when receiving faxes and normally doesn't have to be changed.

After '&F' the register is set on 83, which means:

Standard resolution and Top Line included (, S , T).

The speed at which the modem is transmitting depends on S-register 51. It also depends on the negotiation with the other fax-machine.

```
S51=0 ; Transmit at highest speed possible after negotiation
S51=11 ; Transmit at 9600 (V29) only
S51=7 ; Transmit at 7200 (V29) only
S51=5 ; Transmit at 4800 (V27ter) only
S51=6 ; Transmit at 2400 (V27ter) only

S51=28 ; Transmit at 14400 (V17) only
S51=29 ; Transmit at 12000 (V17) only
```

The V17 speeds can be disabled (excluded from the negotiation) by setting bit3 in S-register 23 (SREG23=SREG23 OR 8).

After sending or receiving a fax, the used speed can be read from S-register 50 (X=SREG50).

During the handshaking process, the two faxes exchange their ID's. The ID numbers are strings with a maximum length of 20 characters and may (officially) only contain numbers and the characters '+' and '-'. The ID of the receiving fax machine will be stored at address &2950 after leaving the `FAX SEND` command, while your own ID must be stored at address &2970 before executing the `FAX SEND` command. The commands `POKE$` and `PEEK$` are used to write and read the ID strings.

The top line text must be stored at address &2990. This text is a string with a maximum length of 100 characters and can be written and read using the commands `POKE$` and `PEEK$`.

FAX SEND ... TO [number]:

This can be a telephone number. A 'D' (Dial) is automatically put in front of the string, and is treated as a normal 'ATD' command. Examples:

```
FAX "TEXT" TO "0-123456"
```

```
FAX "TEXT" TO "T018282"
```

```
FAX "TEXT" TO "p0w111111"
```

If the modem is already on-line, then this string will be ignored.

There are three (3) special cases:

When the command starts with a less than sign ("`<`"):

After dialing, connecting and sending the file(s), the modem will not stop sending, but will return with result code 19 (= `ESCAPE`). A second `FAX`-command can then be given and sending will continue with another file on the same page. This is especially useful for adding footers.

When the command starts with a greater than sign ("`>`"):

After dialing, connecting and sending the file(s), the modem will not stop sending, but will return with result code 19 (= `ESCAPE`). First, however, it will end its current page and start a new page on the other fax-machine. A second `FAX`-command can then be given and sending will continue with another file on a new page. This is especially useful for adding documents.

When the command starts with a hash ("`#`"):

The converted fax data is now routed through the fax printer driver. It will then be printed out on the port selected by `OPORT`. No dialing is needed or done. For example:

```
FAX SEND "TEXTFILE" TO "#"
```

Program example:

```
FAX ON ; Enable fax connection
DIAL Tel$ ; Dial number and wait for Answer tone
IF RESULT<>19 GOTO Err ; Result = ESCAPE (Answer tone found)?
FAX Head$ TO "<" ; Go back on-line, connect/negotiate and send header
IF RESULT<>19 GOTO Err ; Successful so far?
FAX Text$ TO "<" ; Go back on-line, and continue sending text file on same page
IF RESULT<>19 GOTO Err ; Still successful?
FAX Foot$ TO ">" ; Go back on-line and continue sending footer on same page
IF RESULT<>19 GOTO Err ; Still successful?
FAX Dcmt$ TO "" ; Go back on-line and continue sending document on next page
IF RESULT GOTO Err ; Overall successful?
```

There are 3 system variables which can be read or checked after the `FAX SEND/RECEIVE` command: `MATCH`, `RESULT` and `ERN`.

`MATCH` is a function which acts as a counter:

After FAX SEND, it contains the total number of files sent.

After FAX RECEIVE, it contains the number of pages received.

RESULT is a function which holds the status of the last modem activity:

RESULT =	0	OK	:	Sending or Receiving was successful
RESULT =	2	RING	:	File not found with FAX SEND (special case)
RESULT =	3	NO CARRIER	:	Initiate handshake with other fax has failed
RESULT =	4	ERROR	:	No confirmation from other fax after sending a page or failure during receiving a page
RESULT =	6	NO DIALTONE	:	No dialtone found before dialing out
RESULT =	7	BUSY	:	Other fax found busy after dialing out or disk is full during receive (special case)
RESULT =	8	NO ANSWER	:	No answer (answer tone) found after dialing out
RESULT =	18	OFF-HOOK	:	Modem is not connected to telephone system or line is already in use
RESULT =	19	ESCAPE	:	Used when sending, meaning 'continue'
RESULT =	20	VOICE	:	Human voice (or much noise) found after dialing

ERN is only set when a filing system error has occurred:

ERN =	0	OK	:	No error
ERN =	22	Out of memory	:	Disk is (became) full during FAX RECEIVE
ERN =	73	Not found	:	File not found during FAX SEND <filename> RESULT is set to 2 also in this case

The only real BASIC errors which can occur are:

'No filing system'	:	No filing system ROM available
'No fax system'	:	No fax system ROM available
'No/Bad BUFFER'	:	No or too small RAM buffer is defined (must be at least BUFFER &400, 1k bytes, 1024 bytes)

Sending TEXT faxes

ASCII text files are converted to G3 format on the fly. The output on the remote fax machine depends on the 4 parameters set with the DUTCH or ENGLISH command:

1. Characters per line:

0	=	Word Warp disabled, 10 chars/inch, max. 80 chars on a line
1-88	=	Word Wrap enabled, 10 chars/inch, max. 80 chars on a line
89-112	=	Word Wrap enabled, 12 chars/inch, max. 96 chars on a line
113-254	=	Word Wrap enabled, 17 chars/inch, max. 136 chars on a line

Word Warp means that a whole word is started on a new line if it doesn't fit on the current line.

A new line is always generated after a character with the value equal to S-register 3 which defaults to control-character 13 (=Carriage Return). All characters with a value equal to S-register 4 are ignored; default control-character 10 (=Line Feed).

2. Lines per page:

The maximum number of lines per page can be set with this parameter. If more lines are in the file, then a new page is negotiated first with the remote fax before sending continues. For standard A4 format this parameter must be set to 68 if the top line is enabled or 70 if the top line is disabled.

Setting this parameter to 0 will disabled the automatic page-break generation. A new page is also forced if the file contains a control-character 12 (=CTRL-L=Form Feed). If a control-character 26 (=CTRL-Z=End Of File) is encountered the sending stops.

3. Left margin:

Enabling the left margin will add some spaces in front of a line, if that line is less than 80, 96 or 136 characters.

4. Page fillout:

Enabling the page fillout will fill the page with empty lines on the bottom until the 'lines per page' is reached. This is done before starting a new page and at the end of the file.

For use inside a text file, a few embedded commands are recognized:

- @D+ = Use 12 chars/inch
- @D- = Use 10 chars/inch
- @C+ = Use 17 chars/inch
- @C- = Use 10 chars/inch
- @I+ = Use 8 lines/inch vertical
- @I- = Use 6 lines/inch vertical (default)
- @B+ = Bold on
- @B- = Bold off
- @U+ = Underlined on
- @U- = Underlined off

See also:

ERN, FAX RECEIVE, MATCH, RESULT

The Argus BASIC FOR . . . NEXT loop construct is very similar to that used in most versions of BASIC, and allows a sequence of commands to be executed a fixed number of times depending upon the value of a 'loop variable'.

For example:

```
10 PRINT "Start"
20 FOR X=1 TO 10 : PRINT X;" "; : NEXT X
30 PRINT "End"
```

will display:

```
Start
1 2 3 4 5 6 7 8 9 10
End
```

i.e. the first time the FOR command is encountered, the numeric variable X is set to 1 and printed. The loop is then repeated, X being incremented to 2 and printed again. This procedure is repeated until X is 10, after which line 30 is executed and the program ends.

By default, the loop variable X, is incremented by 1, but this can be changed by including the optional STEP clause in the FOR command:

```
10 PRINT "Start"
20 FOR X=1 TO 10 STEP 2 : PRINT X;" "; : NEXT X
30 PRINT "End"
```

Here, X is incremented by 2 each time the loop is executed, so only the numbers 1, 3, 5, 7, 9 are printed.

More complicated loops can be constructed by including multiple commands between FOR and NEXT:

```
50 FOR X=1 TO 5 : PRINT X : PRINT X*X :
PRINT X*X*X : NEXT X
```

This program line prints X, the square of X and the cube of X for each value of X between 1 and 5. The number of commands executed is limited only by the maximum line length of 255 characters.

Up to sixteen FOR . . . NEXT loops can 'nested' within each other, each operating on a different loop variable:

```
FOR A=1 TO 10 : FOR B=5 TO 15 :
PRINT A + B : NEXT B : NEXT A
```

Here, there are two loops. This example could in fact be shorter (and more efficient) by combining the two NEXT commands:

```
FOR A=1 TO 10 : FOR B=5 TO 15 :
PRINT A + B : NEXT B, A
```

It is also possible to omit the variable following NEXT completely; in which case the variables are incremented innermost first. The example above would then become:

```
FOR A=1 TO 10 : FOR B=5 TO 15 :
PRINT A + B : NEXT ,
```


The trailing comma in this example is necessary to ensure that the outer NEXT loop variable is incremented.

Examples:

```
FOR Code = 32 TO 126 : PRINT CHR$(Code) : NEXT
```

```
FOR X = 0 TO -10 STEP -2 : PRINT X : NEXT
```

Syntax:

```
FOR [num-var] = [integer] TO [integer] (STEP [integer]) : [command] :  
.... NEXT ([num-var]) (,)
```

See also:

```
REPEAT ... UNTIL, WHILE ... WEND
```

The frequency of the signal on the telephone line is permanently measured by the modem. When the modem is off-hook, the command `FREQ` will read this value. This is only useful for specialist applications; but we can think of some very special ways of using it for access control of your computer.

The value is updated every 100ms by the operating system. Reading the value faster than 100ms will return the last/same values and has no use.

Examples:

```
PRINT FREQ
```

```
Tone = FREQ
```

Syntax:

```
[num-var] = FREQ
```

See also:

```
ADC
```

GET**get byte (with wait)**

GET is used to obtain the next byte from the specified input buffer, and returns an integer between 0 and 255. Program execution is suspended until data becomes available. If no port is specified, the default input port as defined by IPORT is assumed.

Examples:

```
Code=GET  
Char=GET#1  
REPEAT UNTIL GET=13  
PRINT GET#2
```

Syntax:

```
[num-var] = GET ( #[port] )
```

See also:

GET\$, KEY, KEY\$

GET\$**get character (with wait)**

GET\$ is used to obtain the next character from the specified input buffer, and returns a 1-character string. Program execution is suspended until data becomes available. If no port is specified, the default input port as defined by IPORT is assumed.

Examples:

```
Ch$=GET$  
A$=GET#2  
REPEAT UNTIL GET$#2="Y"  
PRINT GET$#4
```

Syntax:

```
[string-var] = GET$ ( #[port] )
```

See also:

GET, KEY, KEY\$

The GOSUB command causes program execution to branch to the specified line number or label. Execution proceeds from this line until a RETURN command is encountered, at which point execution resumes at the command immediately following the GOSUB. The section of code executed by the GOSUB is referred to as a *subroutine*, and may be executed any number of times within the program. For example:

```
10 PRINT "Main program start"
20 GOSUB 500
30 ...
400 END

500 PRINT "Now in subroutine"
510 WAIT 5
520 PRINT "Leaving subroutine"
530 RETURN
```

In many versions of BASIC, subroutines are referred to by the line number of their first line, as in the example above. In Argus BASIC however, it is better to make use of the label facility, and precede each subroutine by a label that indicates the function of that routine. You could then rewrite the example as:

```
10 PRINT "Main program start"
20 GOSUB Demo
30 ...
400 END

500 %Demo
510 PRINT "Now in subroutine"
520 WAIT 5
530 PRINT "Leaving subroutine"
540 RETURN
```

Proper use of labels can make the main program much easier to read and understand. The use of % preceding a label with GOSUB is optional. Thus GOSUB Dial is identical to GOSUB %Dial.

A subroutine may also be called from within another subroutine, and up to sixteen calls may be 'nested' in this way.

Examples:

```
GOSUB 1000

GOSUB Dial
```

Syntax:

```
GOSUB [line-num | label]
```

See also:

ON GOSUB

GOTO**unconditional jump**

GOTO causes an unconditional branch to the specified line number or label.

Examples:

```
GOTO 70
```

```
GOTO %Check_Password
```

Syntax:

```
GOTO [line-num | label]
```

See also:

```
ON ... GOTO, GOSUB
```

HANDSHAKE is used to select the action taken when an input buffer is almost overflowing during receipt of data. Correct selection of the type of handshaking to be used is necessary to avoid loss of data.

Each port on the modem has a 256-byte input buffer. When characters are received at a higher speed than they can be processed, the input buffer will gradually fill. In order to avoid loss of data due to input buffer overflow, when there are only 5 bytes left free in a buffer, the modem will check to see if a handshake protocol has been selected; and if it has will take the appropriate action as follows:

RTS/CTS selected:

The modem will take CTS low until the input buffer is half empty and then take CTS high again to restart transmission.

XON/XOFF selected:

The modem will send an XOFF character to the remote system, wait until the input buffer is half empty, and then send an XON character to restart transmission.

To select the type of handshaking used there are two commands:

```
HANDSHAKE CTS
```

and

```
HANDSHAKE XOFF
```

which select *hardware* or *software* handshaking respectively.

It is also possible to use both software and hardware handshaking at the same time. Use:

```
HANDSHAKE CTS, XOFF
```

A further command is used to disable the previously selected handshaking method:

```
HANDSHAKE OFF
```

Each of these commands can specify the appropriate port. If no port is specified, the default input port as defined by `IPORT` will be assumed. Note that the modem will always obey the RTS/CTS handshake protocol on ports defined as outputs.

Handshaking via LINKed ports

The situation becomes more complex when ports have been connected using a `LINK` command.

To visualize this situation, assume that ports 1 and 2 are `LINKed`, and that the two character streams are crossed; i.e. all characters received by port 1 are sent to the port 2 output buffer, and all characters received by port 2 are sent to the port 1 output buffer.

Assume also that port 1 is operating at a higher baud rate than port 2 (e.g. 9600 vs. 1200). All characters received via port 1 will be re-transmitted via port 2 at a lower speed, so that the output buffer for port 2 will gradually fill up. When it is almost full, port 1 should take steps to stop the receipt of data by either issuing an XOFF to the connected system, or by taking CTS low. To ensure that this happens, a further command is used:

```
HANDSHAKE LINK ON
```

To summarize, handshake should always be defined for input ports. Check always whether a linked port can cause the buffers of output ports to overflow. This is especially important when baud rate conversion is used.

The handshake of the input ports has to be set with the `HANDSHAKE` command. This is normally `HANDSHAKE RTS`. When the data stream on a linked port is not to be monitored by a program, then the command `HANDSHAKE LINK OFF` should be used.

The `HANDSHAKE` command may specify one or more port numbers, but if these are omitted, the default port as defined by `OPORT` will be assumed.

After the `STOP` button has been pressed, `HANDSHAKE RTS` will be set for each port, and `HANDSHAKE LINK` will be set to `OFF`.

Note that the `XON` and `XOFF` characters can be configured using the `XON` and `XOFF` commands.

Examples:

```
HANDSHAKE RTS
HANDSHAKE#1, #2, XOFF
HANDSHAKE#3, OFF
HANDSHAKE LINK OFF
HANDSHAKE#1, #2, LINK ON
```

Syntax:

```
HANDSHAKE #[port] (, #...) CTS|XOFF|OFF
HANDSHAKE #[port] (, #...) LINK ON|OFF
```

See also:

```
LINK, RTS, XOFF, XON
```

HANGUP

disconnect a call

HANGUP is used to terminate a call and is identical to the command HOOK ON and the modem command AT "H0".

The command itself will not cause an ON HANGUP event.

HANGUP will also put the modem-chip in sleep-mode, if bit6 in S-register 64 is set (see RESET MODEM 2).

Examples:

```
HANGUP  
MATCH#3 "ABC" , 10 : IF MATCH HANGUP
```

Syntax:

```
HANGUP
```

See also:

```
ANSWER, CONNECT, ON HANGUP, RESET MODEM 2
```

HOOK is used to control the modem relay that puts your modem on-line.

HOOK OFF is the same as the modem command ATH1.

HOOK ON is the same as the command HANGUP and the the modem command ATH0.

HOOK on its own can be used to toggle between the two states of the relay.

Examples:

HOOK

HOOK OFF

HOOK ON

Syntax:

HOOK (ON|OFF)

HOUR**read/set hour of day**

HOUR is used to read or set the hour of day on the internal clock/calendar.

The clock uses the 24 hour system, so that when *reading*, HOUR will return an integer from 0 to 23.

Similarly, when *setting*, the hour number from 1 to 23 must be specified; an incorrect value will cause the current setting to remain unchanged.

Examples:

```
HOUR=14  
  
IF HOUR > 12 THEN D$="PM" ELSE D$="AM"  
  
PRINT HOUR
```

Syntax:

```
HOUR = [integer 0..23]  
  
[num-var] = HOUR
```

See also:

```
CLOCK$, MIN, SEC, TIME$
```

IF ... THEN ... ELSE is used to make a decision depending upon the value of a logical expression. To illustrate:

```
20 IF X>40 THEN GOTO 250
```

When this line is executed, the expression $X > 40$ is evaluated, and if the result is true, program execution branches to line 250; otherwise execution continues at the line following line 20.

In Argus BASIC the THEN clause is optional; and in fact it is more efficient to write the above example as:

```
20 IF X>40 GOTO 250
```

An IF command may not extend over more than one program line, but it is possible to 'nest' up to sixteen IF commands in a single line. The following example illustrates two 'nested' IFs:

```
IF X=1 THEN IF Y>2 PRINT "Ok"
```

You must be careful to ensure that the logic is correct with multiple IF commands.

Examples:

```
IF Temp > 20 THEN OLINE (1)=0
```

```
IF Inp$=Password$ PRINT "Ok" ELSE PRINT "Illegal password"
```

```
IF (KEY#1=13 OR KEY#2=13) THEN %Finish
```

Syntax:

```
IF [expr] ( THEN ( [line-num | label] ) ) ... (ELSE ( [line-num | label]  
| [command] ) )
```

ILINE is used to read the status of the specified input line.

The Programmable Modem has sixteen possible lines for external input, numbered 1-16, which are normally configured for use as a parallel printer port and the jumper port. When one of the lines is addressed using the ILINE command, that line is automatically configured as a digital input. This line will then accept a TTL level input voltage between 0 and +5 volts. Signals between 0.8 and 2 volts will give an undefined result; signals below 0.8 volts will give logic 0, and those above 2 volts will give logic 1.

The T.C.Lite has no extra external input lines and ILINE1-16 will therefor return invalid values.

The ILINE command returns either 0 or 1, according to the voltage level.

The status of digital I/O lines is not affected by a reset (START button or power failure).

It is recommended that input signals are buffered using a suitable IC such as a 74LS244, to prevent damage to the modem due to excessive current/voltage.

With the user port, jumper settings can also be seen with ILINE.

ILINE will return a value of 1 if a jumper is set. Thus it is also very easy to connect additional button switches to the modem, or to monitor contacts.

A special case is ILINE(0) - that is, the off-hook detect of the modem. When ILINE0 is true, either the line is busy, or the modem is not connected to a telephone line at all.

If you have a telephone handset connected, we suggest that you enter:

```
REPEAT UNTIL ILINE0 : SOUND
```

and then pick up your phone, to demonstrate the principle.

The second special case ILINE(-1) will read a random number between 0 and 255.

The ILINE command is especially useful when you want to monitor equipment and send data accordingly.

Examples:

```
Switch=ILINE(2)

Count=Count + ILINE(Line)

PRINT ILINE15

IF ILINE16 THEN Fire-Alarm

REPEAT UNTIL ILINE0=0    (waits for line to become free)
```

Syntax:

```
[num-var] = ILINE [integer -1..16]
```

See also:

OLINE

INPUT is used to read string or integer values from ports #1 to #3. A prompt message may be included in the command, and input is terminated when a [CR] is received. A port number can be specified, but if this is omitted the default input port as defined by IPORT will be assumed.

The following example displays a prompt message, and then waits for the user to enter some data and press [Return]:

```
50 INPUT "Enter a number :"; Num
```

A question mark is automatically appended to the prompt message, and the number, when entered, is stored in the integer variable Num.

One or more Line-feeds may be inserted before the prompt message by preceding it with the requisite number of single quote marks:

```
50 INPUT '"Enter a number :"; Num
```

Here, two blank lines are printed before the prompt message.

Multiple values may be read using a single INPUT command, and each of these may have its own prompt string:

```
60 INPUT "First name : "; FName$; " and Surname :"; SName$;
```

Here, two string variables, FName\$ and SName\$, are read using INPUT. The prompt for the SName\$ is not displayed until the user has entered his first name and pressed [Return].

If individual prompts are not required, commas can be used to separate variables:

```
INPUT "Enter three numbers : "; X,Y,Z
```

A question mark will be printed after the prompt message, and the user may type all three items on the same line separated by commas; or alternatively, on separate lines by pressing [Return] after each.

In the latter case a '?' will be printed to prompt for each value.

Invalid input

When reading a numeric variable, invalid input (i.e. numbers out of range, non numeric characters, etc.), will result in the variable being assigned a value of 0. Valid characters appearing at the start of an input line will be accepted. For example, entering:

```
56afg
```

in response to a prompt for an integer, would result in the variable being assigned the value 56.

Pressing [Return] on its own in response to a prompt for a string variable will assign a null string to the variable; i.e. a string with no characters.

A special case is INPUT ALL. INPUT ALL will accept all data received until a carriage return is received, inclusive of control characters and leading spaces.

Another special case is INPUT AT (SLASH).

When this command is executed the following will happen:

- The user can enter an AT command (and optionally also A/).
- The command will be passed transparently to the modem.
- The command will be executed by the modem.
- The result will be printed to the screen.

- The result code will be stored as an integer in the systems variable RESULT.

This is the same as leaving BASIC to give just one command to the modem.

Examples:

```
INPUT X

INPUT#3 "Name ";Na$

INPUT "ID Number "; Id; "Password : ";Pword$

INPUT#2 "Time : (HH MM SS) ",Ho, Mi, Se

INPUT ALL "Screencode ";S$

INPUT AT SLASH
```

Syntax:

```
INPUT ( #[port], ) (ALL) (') ("prompt") (,) (;) [num- var] | [string-
var]...
```

```
INPUT AT (SLASH) (,#[port])
```

See also:

```
CLEAR, ECHO, PRINT
```


I**PORT** is an important system variable that defines the default input port number. The following commands use the value of I**PORT** as the default port when no port is specified:

ACTIVE	DTR	KEY	RTS
AT	GET	KEY\$	SEND
CTS	GET\$	MATCH	
DCD	INPUT	ORI	
DSR	IRI	RECEIVE	

Thus to read characters one at a time from port 1 there are two options:

```
Byte=GET#1
```

or

```
IPORT=1 : Byte=GET
```

The second method is more efficient when repeated input from the same port is necessary.

I**PORT** will operate on the following ports:

- Port 1 is serial port 1.
- Port 2 is serial port 2 (called Aux on a T.C.Lite).
- Port 3 is the modem (telephone line).
- Port 6 is the keyboard (if available).
- Port 7 is the parallel printer port.
- Port 8 is the user/memory buffer (see **BUFFER** command).

Examples:

```
IPORT=2
```

```
IPORT=Port
```

```
Inp=IPORT
```

Syntax:

```
IPORT = [port]
```

```
[num-var] = IPORT
```

See also:

O**PORT**

KEY**get byte (no wait)**

KEY is used like GET, to read a byte from a serial input buffer. Unlike GET, KEY does not wait for data to become available, but will return a 0 if the input buffer is empty.

A port number may be specified, but if this is omitted the default input port as defined by IPORT will be assumed.

Examples:

```
Keystroke=KEY  
X=KEY#2  
IF KEY#3 THEN 1000
```

Syntax:

```
[num-var] = KEY ( #[port] )
```

See also:

```
GET, GET$, KEY$
```

KEY\$**get character (no wait)**

KEY\$ is used like GET\$ to read a character from a serial input buffer. Unlike GET\$, KEY\$ does not wait for data to become available, but will return a null string if the input buffer is empty. In this respect, GET\$ is useful for checking whether or not the input buffer is empty - GET cannot be used because there may be [Ctrl-@] characters in the input buffer which would appear as ASCII 0.

A port number may be specified, but if this is omitted the default input port as defined by IPORT will be assumed.

Examples:

```
Ch$=KEY$  
  
REPEAT UNTIL KEY$#3="E"  
  
IF KEY$="" THEN %Wait
```

Syntax:

```
[string-var] = KEY$ ( #[port] )
```

See also

GET, GET\$, KEY

LCASE**convert ASCII to lowercase**

LCASE is used to calculate the lower case ASCII value of a character.

Examples:

```
PRINT LCASE 65
```

```
B=LCASE66
```

Syntax:

```
[num-var] = LCASE [num]
```

See also:

```
LCASE$, UCASE, UCASE$
```

LCASE\$**convert string to lowercase**

LCASE\$ is used to convert all characters in a string to lower case. Characters that are already in lower case remain unchanged.

This command (or UCASE\$) is frequently needed to standardize keyboard input.

Examples:

```
PRINT LCASE$ "QwErTy"  
  
result$ = LCASE$ input$
```

Syntax:

```
[string-var] = LCASE$ [string]
```

See also:

LCASE, UCASE, UCASE\$

LED is used to control the LEDs on the front of a Programmable Modem (if fitted). The T.C.Lite never has LEDs fitted.

You can control all user and other modem LEDs, except the POWER LED.

```
LED0   - BUSY
LED1   - USER LED1
LED2   - USER LED2
LED3   - USER LED3
LED4   - USER LED4
LED5   - USER LED5
LED6   - AA
LED7   - DCD
LED8   - OL
LED9   - S/A
LED10  - CTS
LED11  - DTR
```

Normally you will turn the LED on by making it equal to a non-zero number, and turn it off by making it equal to 0 again.

LED8 is a bit different, because it is connected via hardware to the on-line relay of the modem. It will operate in reverse and will also influence the relay. We therefore recommend that you do not normally control LED8 from within your programs.

Normally the modem itself controls the User LEDs, so to have them under program control, you will first have to disable control by the modem. From within the BASIC environment:

```
AT "*L0" or SREG32=0 (LEDs 1 till 5 off)
```

The User LEDs (LED1-LED5) can also be set with SREG32: bit0=LED1 - bit4=LED5. So, SREG32=10 will turn on LEDs 2 and 4 and turn off LEDs 1, 3 and 5.

Examples:

```
LED2 ON

LED3=1

LEDUser=A

IF LED6 PRINT "Auto Answer LED is still on!"

PRINT LED4
```

Syntax:

```
LED [num] ON|OFF

LED [num] = [num]

[num-var] = LED [num]
```

LEFT\$**extract left part of a string**

LEFT\$ extracts the specified number of characters from the left-hand side of a string. The command:

```
L$=LEFT$ ("ABCDEFGH", 3)
```

will assign the string "ABC" to the string variable A\$.

The number of characters extracted must be between 0 and 255. Attempts to extract more characters than exist in the string will return the entire string.

Examples:

```
Firstname$=LEFT$(Name$, 6)
```

```
Da$=LEFT$(CLOCK$, 2)
```

```
PRINT LEFT$("Now is the time for all good men..", 10)
```

Syntax:

```
[string-var] = LEFT$ ( [string], [integer  
0.255] )
```

See also:

```
LEN, MID$, RIGHT$
```

LEN**get length of string**

LEN returns the length of the specified string, including spaces and non-printing characters.

Examples:

```
Size=LEN(Name$)
IF LEN(Name$) >30 THEN PRINT "Name too long .."
PRINT LEN(Message$)
```

Syntax:

```
[num-var] = LEN( [string] )
```

See also:

```
LEFT$, MID$, RIGHT$
```


LENGTH

set or read serial port word length

The `LENGTH` function is used to set or read the number of data bits used during serial data transfer.

The number of bits used can be specified separately for each port, and may be set to 7 or 8 bits. The most common format for ASCII systems is 8 data bits, no parity and 1 stop bit, which is the default format used by this modem. This format must be used for sending data or machine language programs using the Xmodem protocol.

If a user is connected to another terminal configuration, you might want to advise the other person that he will not be able to see what the application is expecting him to see, so it is very useful to read `LENGTH` at a certain stage in your program.

After pressing the STOP button, the length for all ports will be configured to the default value of 8 bits.

If no port is specified, the default output port as defined by `OPORT` is assumed.

Examples:

```
LENGTH 7

LENGTH#1, #2, #3, 8

LENGTH#Port, Bits

IF LENGTH=7 GOSUB %WARNING
```

Syntax:

```
LENGTH ( #[port] ) (, #...) [integer 5..8]
```

See also:

```
BAUD, PARITY, SBITS
```

Most terminals and printers accept a combination of Carriage Return (ASCII 13) and Line Feed (ASCII 10) characters to indicate the end of a line, while others require only a Carriage Return, and insert a Line Feed automatically. The `LINEFEED` command is used to switch Line Feeds on or off, as appropriate.

`LINEFEED ON` causes the modem to insert a Line Feed after every Carriage Return in a transmitted data stream for the specified port.

`LINEFEED OFF` disables transmission of Line Feeds for the specified port.

If no port is specified, the default port as defined by `OPORT` will be assumed.

With respect to the printer port (#7), there are two forms of the command that may be used:

```
LINEFEED#7, ON/OFF
```

or

```
PRINTER LINEFEED ON/OFF
```

There is a second syntax of `LINEFEED`.

```
LINEFEED [num-var] .
```

This command allows you to change the ASCII value of the linefeed character, which is normally set to the default value of 10.

Examples:

```
LINEFEED ON
```

```
LINEFEED#2, #3, OFF
```

```
LINEFEED#7, ON
```

```
LINEFEED 10
```

Syntax:

```
LINEFEED ( #[port] ) ( , #... ) ON/OFF
```

```
LINEFEED (=) [num]
```

See also:

```
PRINTER
```

LINK is used to create transparent links between two serial ports. The advantage of linking ports can be explained with the following example.

Assume that a terminal is connected to port 1, and a computer to port 2, and that both are set for 9600 baud operation. There are various ways of connecting the two devices. The first is to write a program that simply takes characters from each port and puts them to the other:

```
10 BAUD#1, #2, 12
20 REPEAT
30   PUT#2, KEY$#1
50   PUT$#1, KEY$#2
60 UNTIL FALSE
```

Unfortunately however, this short loop ties up the modem permanently, so that no other operation can be carried out. Furthermore, Argus BASIC, though fast, is not fast enough to keep up with two 9600 baud data streams, and software or hardware handshaking must be used to prevent loss of data.

The alternative is to use the LINK command:

```
10 BAUD#1, #2, 12
20 LINK#1, #2
30 LINK#2, #1
40 REM - Remainder of program starts here
```

Here, the two LINK commands are used to establish a bi-directional, transparent link between ports 1 and 2. Such a link will not interfere with the rest of the program, and will remain effective even when program execution ceases.

You may ask whether it would be simpler to couple the terminal direct to the computer using an appropriate cable. In this example that is probably the case, but there are situations where this is not practical; e.g. when the data format of the two devices is different. With LINKed ports:

- the baud rate and data format of the linked devices need not be the same.
- it is possible to monitor the data flow and respond to specified character sequences.
- data flow can be switched between different ports.
- incoming data can be routed to one or more output ports.

Data monitoring carried out using the INPUT, GET, KEY and MATCH commands can be extremely useful, but will slow the link down. A further command is therefore provided to enable handshaking over the link:

```
HANDSHAKE LINK ON
```

This will prevent loss of data due to buffer overflow.

```
HANDSHAKE LINK OFF
```

may be used to disable handshaking if speed is not likely to cause problems.

The next example program uses the MATCH command to monitor the link for the string 'PASSWORD' being received at port 1:

```
10 LINK#1, #2
```

```

20 LINK#2, #1

30 HANDSHAKE#1 LINK ON

40 HANDSHAKE#2 LINK OFF

50 MATCH#1, "PASSWORD"

60 ...

```

Note that the first port number specified in the LINK command is that of the input port. One or more output ports then follow:

```
LINK#1, #2, #7
```

Here port 1 is the input port; ports 2 and 7 are outputs.

The following will create a bi-directional link between ports 2 and 3 while allowing port 1 to monitor data received via port 2:

```
LINK#2, #3, #1
```

```
LINK#3, #2
```

A port can also be linked to itself:

```
LINK#1, #1
```

The effect of this is to provide character echo.

If no port numbers are specified, the default input port defined by IPORT will be used as the input port, and the default output port defined by OPORT as the output port.

The command:

```
LINK OFF
```

may be used to disconnect linked ports. Pressing RESET will clear the input and output buffers, and also terminate any existing links.

Port 6 (keyboard and LCD display) can not be linked.

Examples:

```
LINK#1, #2
```

```
LINK PORT2, PORT7
```

```
LINK#1, #2, #3
```

```
LINK#In, #Out
```

```
LINK PORT8, PORT1
```

```
LINK OFF
```

Syntax:

```
LINK ( #[port] (TO|,) (#[port] ) (, ...)
```

```
LINK OFF
```

See also:

```
HANDSHAKE
```


LIST is used to list the currently loaded program to the specified port.

When used without a parameter, the entire program will be listed to the default output port as defined by OPORT.

A range of line numbers can be specified; thus the following example will only list those lines from 500 to 1000 inclusive:

```
LIST 500,1000
```

To list from the start of the program up to a specific line number, just omit the first parameter; e.g. to list all lines up to and including line 300 you would use:

```
LIST , 300
```

Output is normally formatted so that:

- BASIC keywords are translated to upper case
- variable names have their first letter capitalized
- spaces are inserted to make programs more readable
- conditional commands such as IF, REPEAT, etc. are indented.

The formatting feature can be enabled or disabled, using the commands LIST OFF and LIST ON respectively.

There are a number of ways in which LIST can be used to obtain a printed copy of a program. The first is to specify port number 7 which is the parallel printer port (assuming you have a printer attached):

```
LIST#7
```

Alternatively, you may use the PRINTER ON command, followed by LIST.

Yet another option is to set OPORT to 65 before printing.

A very useful variant of LIST is the LIST AUTO command which is used for capturing programs that are going to be edited.

Examples:

```
LIST#7,500
```

```
LIST 70,150
```

```
LIST Label1 TO Label2
```

```
LIST OFF
```

Syntax:

```
LIST ( #[port] ) ( , #...) (line-num | label ( TO|, ) line- num |  
label)LIST ON | OFF
```

See also:

```
OPORT, PRINTER ON/OFF
```

This is used to capture a special listed program for editing.

This command lists programs without line numbers included. Programs captured in this way can be edited offline and ASCII uploaded back to the modem using the `AUTO` command.

When performing an ASCII upload to the modem, the `AUTO` command will automatically assign numbers to program lines.

Syntax:

```
LIST AUTO
```

See also:

```
AUTO, LIST
```

LOAD**load BASIC program or data**

The principal use of `LOAD` is to copy programs from disk into program memory, at the location specified by the system variable `PBOT`.

To load from the disk, use `LOAD` (followed by an optional exclamation mark) and the filename within double quotes:

```
LOAD "DIAL.PROG"
```

By changing the value of `PBOT` it is possible to `LOAD` a number of programs into memory at once:

```
LOAD "Prog1"
```

```
Base1=PBOT
```

```
PBOT=PTOP+2
```

```
Base2=PBOT
```

```
LOAD "Prog2"
```

By manipulating the value of `PBOT` one program can then call another. Each program loaded in this way will have access to the same variables; i.e. the variable `Name$` in one program will be the same in another, so that all variables are 'global'.

The second use of `LOAD` is to copy `DATA` from the disk to a certain address. In this case you have to specify the address.

Examples:

```
LOAD Prog$
```

```
LOAD &4000, "MODEM.PROG"
```

```
LOAD &6800, Bytes$
```

```
LOAD !"SETUP.PROG"
```

```
LOAD FILE "SETUP.PROG"
```

Syntax:

```
LOAD (!|FILE) (address) [string]
```

See also:

```
RUN, SAVE
```


LOAD ROM**load BASIC programs from socket 2**

This is used to load a file from ROM or RAM in socket 2 into the program memory. This is only possible in an Argus Programmable Modem with RAM-disk filing system. To initialize the RAM chip in socket 2, the command `CLEAR ROM` must be entered first.

Example:

```
LOAD ROM "FAX.PROG"
```

Syntax:

```
LOAD ROM [FILE]
```

See also:

```
LOAD, ROM
```

LTRIM\$**remove leading spaces**

LTRIM\$ is used to remove all leading spaces from a string.

Examples:

```
PRINT LTRIM$ "      Test      "  
Adjusted$ = LTRIM$ Input$
```

Syntax:

```
[string-var] = LTRIM$ [string]
```

See also:

RTRIM\$, TRIM\$

There are two main forms of the `MATCH` command, which provides the means for searching data streams and file directories for keywords or names.

Searching for strings

The first form of the command is used to search incoming data from the specified port for occurrences of a specified string. For example:

```
50 MATCH#1, "Password"
```

will search the incoming data stream for the string 'Password', and suspend operation until a match is found. The program will then resume at the next command. An optional timeout period may be specified by following the command with a number, e.g.

```
50 MATCH#1, "Password", 10
```

will cause the program to wait for up to 10 seconds for the string 'Password' to be received, before resuming execution at the next line.

Within the string to be searched for, wildcards may be used. There are two types of wildcards:

```
' .'    for one character only; and  
' * '   for any possible string.
```

```
MATCH "A*B"
```

will find 'AxxB' as well as 'AxB'

while

```
MATCH "A.B"
```

will only find 'AxB'.

If you need to find a string that includes '*' or '.', then you will have to use the command:

```
MATCH#n, ALL
```

because this command is similar to `MATCH`, but will not accept wildcards.

If `ECHO` is added after the command, all characters received while searching for a string will be echoed.

As the `MATCH` command returns a logical `TRUE` or `FALSE` value, an `IF` test can be used to check the result of the operation.

```
MATCH "PRINTER", 10 : IF MATCH=0 GOTO %AGAIN
```

If no port number is specified, the default input port as defined by `IPORT` will be assumed. The `MATCH` command does not affect the operation of any active `ON . . .` event handlers.

Searching for files

When the `MATCH` command is followed by an exclamation mark or the word `FILE` and a string, the string is treated as a filename and a search is made of the file directory to determine if the file exists. For instance, the command:

```
MATCH FILE "TELEX"
```

will search for a file called 'TELEX'. As with the previous form of the command, the result is a logical value which can be acted upon by a suitable `IF` test:

```
MATCH FILE "TELEX" : IF MATCH=0 GOTO %NO
```

With this syntax of match "*" is the only possible wildcard character. The wildcard character can be used to search for multiple filenames, in which case MATCH will return the total number of matches found. To show a count of all stored files you could use the command:

```
MATCH FILE "*. *.*"
```

```
PRINT "Total number of files found: "; MATCH
```

Finally, the 'U' parameter can be used to specify a match with only unread files; e.g.:

```
MATCH FILE "*. *",U
```

```
PRINT "Number of Unread messages : "; MATCH
```

Examples:

```
MATCH "CONNECT"
```

```
MATCH "STATION*AUT"
```

```
MATCH#2, X$
```

```
MATCH Message$, 600
```

```
MATCH FILE File$+".INP", U
```

Syntax:

strings:

```
MATCH ( #[port], ) (ALL) [string/string with wildcards] ( , [integer] )  
( , ECHO)
```

files:

```
MATCH [FILE|!][string] ( , U )
```

See also:

```
FAX RECEIVE, FAX SEND, MATCH ROM
```

MATCH ROM**find matching files in socket 2**

This is used to search the ROM or RAM in socket two for matching files, file directories, keywords or names. This is only possible in an Argus Programmable Modem with RAM-disk filing system. To initialize the RAM chip in socket 2, the command `CLEAR ROM` must be entered first.

Examples:

```
MATCH ROM "CONNECT"
```

```
MATCH ROM "STATION*AUT"
```

Syntax:

```
MATCH ROM [FILENAME]
```

See also:

```
MATCH, ROM
```

MID\$**extract middle portion of string**

MID\$ is used to extract a number of characters from the middle of a string.

Three parameters are required. The first is the original string, the second is a number indicating the start position within the string, and the third specifies the number of characters to be extracted. In the following example the variable Num\$ is set to 'TWO':

```
50 Num$=MID$ ("ONE TWO THREE", 5, 3)
```

If the specified start position is greater than the length of the string, a null string is returned.

If the number of characters requested is greater than those to the right of the start position, all remaining characters are returned.

If the last parameter is omitted, all characters to the right of the start position are returned.

Examples:

```
Mi$=MID$(CLOCK$, 4, 2)
```

```
X$=MID$(Y$, 6)
```

Syntax:

```
[string-var] = MID$ ( [string], [integer 0..255], ([integer 0..255] ))
```

See also:

```
LEFT$, RIGHT$
```

MIN**set/read minute value**

MIN is used to read or set the minute value of the internal clock/calendar.

When *reading* it returns an integer from 0 to 59; similarly, when *setting*, the specified value must be between 0 and 59. An incorrect value will leave the current setting unchanged.

Examples:

```
MIN=15  
  
Duration=(HOUR * 60)+MIN  
  
PRINT MIN
```

Syntax:

```
MIN = [integer 0..59]  
  
[num-var] = MIN
```

See also:

```
CLOCK$, HOUR, SEC, TIME$
```

This is used in the programming environment to turn MNP error correction and compression on or off and has the same effect as the modem commands &E0, &E4 and &E5. To turn MNP2 only on use &E2 and to turn MNP3 only on use &E3,

There are no BASIC commands to turn on or off V42 and V42bis, which is a more modern protocol. This must be done with the modem commands &E0, &E6 and &E7.

To turn on MNP and V42 (no compression) use &E1C0, which is also the default. Using compression (MNP5 or V42bis) is not recommended in BASIC mode, because there is not enough system memory.

Examples:

```
MNP4 ON
```

```
MNP OFF
```

Syntax:

```
MNP (4 | 5) ON | OFF
```


MOD**integer remainder operator**

MOD returns the integer remainder from an integer division operation.
For example the result of

17 MOD 5 would be 2.

Examples:

```
Res=Total MOD Fact
```

```
PRINT 12 MOD 5
```

Syntax:

```
[num-var] = [integer] MOD [integer]
```

See also:

DIV

MODEM

select standard modem mode

MODEM is used to go from the BASIC environment to the standard modem environment. The modem can only return to BASIC with help of the command AT*B, or because of an ON RESET event. The only other method to go from BASIC to standard modem mode is to press the STOP button while the S/A button is in the in position.

To jump to modem mode in a T.C.Lite the keys > + >> + >>> on the keyboard must be pressed at the same. This can be disabled by setting bit7 on address &49D (POKE &49D, &80), but care must be taken because the modem can lock-up if an error occurs in the program and the ON ERROR command is not defined correct.

Examples:

```
MODEM  
  
ON DTR MODEM
```

Syntax:

```
MODEM
```

See also:

```
ON RESET
```

MONTH**set/read month of year**

MONTH is used to read or set the month value for the internal clock/calendar.

When *reading* it returns an integer from 1 to 12, month 1 being January. Similarly, when *setting*, the specified value must be between 1 and 12. An incorrect value will leave the current setting unchanged.

Examples:

```
MONTH=9
```

```
Quarter=MONTH/4
```

Syntax:

```
MONTH = [integer 1..12]
```

```
[num-var] = MONTH
```

See also:

```
DATE$, DAY, DDAY, YEAR
```

The Argus LAN (Local Area Network) is only available for the Argus Programmable Modem, not for the T.C.Lite. A network modem must have a special operating system in ROM with the network filing system build in. A network is made with (at least) 1 network server and 1 or more network stations. They are all connected to one another with RJ45 plugs. The network operates at a clock speed of 921.6 kbits per second (115.2 kbytes per second). This clock is generated by the network server.

A network server is an Argus with Hard-disk and network interface and no modem-chip and telephone interface. More than 1 server can be connected to the network, but a more complex application is needed to get things right. Optional, a backup server can be connected to the network that monitors the network and stores all data (mirror function).

A network station can be an Argus with network interface and modem-chip or an Argus with only a network interface. The last is used as a serial interface between the network and a computer. Depending on the application, there may be 10 till 20 stations connected to a network.

In the case that 1 server is used, it always has the network address 255. This address is also used by the stations to access the server. A backup server has the network address 244. The server has not much intelligents and only serves incoming commands from the stations. It can deal with only 1 command at the time. A following command is only handled if the previous one completely has finished. A serial connection between the server and a computer is not recommended, because a server is not a multi-tasking device; it can not handle the serial port and the network at the same time.

Each station in the network must have a unique network address from 1 till 127. Each station is a complete Argus Programmable Modem with no restrictions compared to a stand alone modem. All BASIC filing system commands are available. The network operating system deals with the data flow over the network (commands to the server, collision detection, etc.).

After a power-up, the first thing the stations will do is loading the BOOT program file from the server and then run it in system memory.

There are 2 BASIC system errors related to the network:

ERN = 101 ("No server") is generated after a timeout of 2 minutes if the network was free but the server couldn't be accessed. It is also generated if a station executes a NETWORK command that is only ment to be used on a server.

ERN = 102 ("No clock") is generated if a station doesn't see a network clock. This can mean that the network cables are not correct or that the server is not connected.

The only network related command is the NETWORK command. There are various forms:

NETWORK

The command NETWORK on its own will load the system variable MATCH with the network address. This is used by a program if it wants to know on which station it is running. Example:

```
30 NETWORK
40 Address=MATCH
```

NETWORK=<var>

This command is used to set the server network address in a station. After a reset, the server address is always set to 255. So, the BOOT program file is always loaded by the stations from server 255. The command is only used if more than 1 server is connected to the network. Example: (copy a file from server 254 to server 255, using the buffer)

```
120 NETWORK=254
130 SEND#8 , F$, F
140 NETWORK=255
```

```
150 RECEIVE#8, F$, F
```

NETWORK ON/OFF

This is an important command. The `NETWORK ON` command claims the network for own use by a station. This means that no other stations can reach the server until the network is freed with the `NETWORK OFF` command. Other stations, accessing the network at the same time, will wait forever until the network becomes free again. So, care must be taken that `NETWORK OFF` always follows `NETWORK ON` after a short period. This feature must be used to update files on the server which can be accessed by more than 1 station at the time. Example:

```
100 NETWORK ON
110 CLEAR BUFFER
120 PRINT#8, Update$
130 RECEIVE#8, Dfile$, F, A
140 NETWORK OFF
```

NETWORK CLEAR

This command will initialise the network chip again, the same as it is done after a reset. The command is not intended to be used in a program.

NETWORK HANDSHAKE ON/OFF

This command can be important if a serial connection to a computer exists on a station. Because the speed of the network, the serial ports and the network can not be handled at the same time. This means that data coming from a serial port is missed by a station if that station is accessing the network at the same time. In those cases where it is needed, the handshake between station and computer must be defined well (see `HANDSHAKE` command). Each time a station accesses the network it sets and clears the handshake to the serial ports automatically after `NETWORK HANDSHAKE ON` (default).

`NETWORK HANDSHAKE OFF` is only needed if the application requires it.

NETWORK LOAD/SAVE CLOCK\$

Each station has its own real time clock on board, but does not have a battery backup. Only the server has a real time clock with battery backup. `NETWORK LOAD CLOCK$` is therefor used to synchronize the real time clock on a station with the one from the server. This is necessary after each reset and must be included in the `BOOT` program file. `NETWORK SAVE CLOCK$` is used to set the real time clock on the server the same as the one in the station executing this command.

NETWORK SCAN

This command is executed on a station to monitor the network activity and is used for test purposes. It will show all the network addresses from the stations and server that are sending commands over the network at the time.

NETWORK RESET

Executing this command from a station will hard reset the network server (power LED off>on).

NETWORK OUTPUT <string>

This command will send a string from a station to the server. The server outputs this string to one of the serial ports (depending on `OPORT` of the sending station).

NETWORK AT <string>

This command will send a string from a station to the server. The server will pass this string to its AT-commands interpreter.

NETWORK RUN/ACTIVE

NETWORK RUN can only be used on a server and puts it into serve-mode. The BOOT program file for the server must therefor always contain this command. The server will stay in this command until a hard reset. Some ON-event handlers can also take it out of serve-mode.

NETWORK ACTIVE has the same effect, but will exit after 1 command from a station has been executed.

REPEAT NETWORK ACTIVE : UNTIL FALSE is the same as NETWORK RUN.

NETWORK TRACE ON/OFF

This command can only be used on a server and will set network trace-mode on or off. If trace-mode is on, the server will output a line of information after each received network command. The output is done to the ports defined by the system variable OPORT. It can be handy during development phase where a serial connection between server and computer monitors all the network activity.

Examples:

```
NETWORK : Addr=MATCH
```

```
NETWORK=255
```

```
NETWORK ON
```

```
NETWORK LOAD CLOCK$
```

```
NETWORK SCAN
```

```
NETWORK RUN
```

Syntax:

```
NETWORK (ON|OFF) (HANDSHAKE ON|OFF) (LOAD|SAVE CLOCK$) (SCAN)
```

```
NETWORK (RUN) (ACTIVE) (TRACE ON|OFF)
```

NEW**new program**

NEW prepares the modem BASIC interpreter to accept a new program. All variables are cleared, and the program start and end are set to the value of the system variable PBOT. Any program already stored at the value of PBOT will be lost (but may be recovered using OLD up to the point at which new program lines are entered).

Syntax:

NEW

See also:

CLEAR, OLD, PBOT

OLD is used to restore a program in system memory that has been lost following use of the NEW command. OLD will only work if no new program lines have been entered.

Syntax:

OLD

See also:

NEW, PBOT

OLINE is used to set or read the status of the specified User I/O line.

The modem has eight programmable output lines numbered 1 to 8, which are normally configured for use as a parallel printer port. To configure one of these lines as a digital output, OLINE is followed by the number of the line to be configured, an equal sign, and 0 for low or 1 for high; e.g. to switch line 5 high use:

```
OLINE5=1 or OLINE5 ON
```

Similarly, to toggle the same line low again, you would use:

```
OLINE5=0 or OLINE5 OFF
```

OLINE can also be used to read the status of a line, even if the line in question has previously been defined as an input; i.e. the current status of an input line can be read using either ILINE or OLINE.

The status of the I/O lines is not affected by a reset.

Lines 1 to 8 are automatically configured as outputs as soon as the printer is addressed.

OLINE0 is the *on-line relay* of the modem and can thus be used to switch the modem on/off line. It directly controls the relay and is not the same as using the HOOK ON/OFF and HANGUP commands. Those commands will set/clear more than the relay only.

OLINE (-1) =var is used to set the seed of the random number generator.

var=OLINE (-1) is used to read the random number generator.

Parentheses () are optional.

Examples:

```
OLINE3=1  
OLINE3 OFF  
OLINECpu=FALSE  
IF ILINE0=0 THEN OLINE0=1  
OLINE3=ILINE6  
Status(Line)=OLINELine  
PRINT OLINE6
```

Syntax:

```
[var] = OLINE(-1)  
OLINE [integer 0..8] = [integer]  
OLINE [integer] = [var] (ON|OFF)  
[num-var] = OLINE [integer 0..8]  
OLINE(-1) = [num]
```

See also:

ILINE

ON AT is used to react to the character string 'AT' typed at one of the serial ports. The ON AT event will only happen if 'AT' is detected on a port which has been previously set to AT scan using the command AT.

When the ON AT event occurs, the port number of the port which received the 'AT' can be found in the system variable PORT.

The command ON AT OFF will turn off the ON AT event but the AT scan on the ports set with the AT command will remain active.

Examples:

```
ON AT GOSUB Menu
```

```
ON AT OFF
```

Syntax:

```
ON AT . . . .
```

```
ON AT OFF
```

See also:

```
AT, ON SLASH, PORT
```

BRK (a *break*) is a signaling procedure where the data line is pulled low by the terminal for a certain time, normally caused by somebody pressing the [Break] key on the keyboard.

This is a very different way of signalling a break which has nothing to do with data. The use of ON BRK as a procedure to interrupt a program when needed has the enormous advantage that full data transparency can be maintained while it is still possible to interrupt whatever the modem is doing from a remote terminal.

For example you might use ON BRK to switch from a link with one port to a link with another port while on line:

```
.  
200 ON BRK GOSUB Switch  
.  
1050 %Switch  
1060 INPUT "Enter port nr. ", x  
1070 LINK#3,#x  
1080 LINK#x,#3  
1090 RETURN
```

This will allow you to maintain full data transparency and still switch between ports when needed. Of course many other useful effects can be programmed using ON BRK.

When needed, the systems variable PORT will tell you on which port the event happened.

Examples:

```
ON BRK GOSUB Switch  
ON BRK OFF
```

Syntax:

```
ON BRK ....  
ON BRK OFF
```

See also:

BRK

ON BUTTON is used to automatically detect the operation of the USER buttons on the modem's front panel. Pressing one of these buttons will result in the immediate execution of the command following the ON BUTTON command.

```
10 REM Button demo
20 ON BUTTON1 GOTO %Keypress
30 %Start
40 REPEAT
50 PRINT GET$;
60 UNTIL FALSE
70 %Keypress
80 PRINT "BUTTON1 PRESSED"
90 GOTO %Start
```

Pressing of the button can be detected up to 4 times a second. Communication will not be affected by operation of this function, and buffers are not emptied.

ON BUTTON_n will remain active after the program is finished, and must therefore be disabled explicitly using ON BUTTON_n OFF.

Button3 is the S/A switch, which has many useful functions in the modem environment.

You can use Button3 for your own purposes also; but because this switch remains in the 'in' position when pressed once, a second event will occur when it is brought back to the 'out' position.

Examples:

```
ON BUTTON1 GOTO %Stop
ON BUTTON2 PRINT "DISCONNECT" : DCD=0 : RUN 100
ON BUTTON3 OFF
```

Syntax:

```
ON BUTTON[integer 1...3] ...
ON BUTTON[integer 1...3] OFF
```

This event handler can be used to automatically detect when a connection has been established with a remote system. Thus waiting for an incoming call can be a background process for your modem. This means that your modem can be doing many other things while waiting for a call. The only thing you have to do is to tell your modem once which parts of your program have to be executed when a call is detected (and when a connection is lost).

```
10 ON CONNECT GOSUB Answ
20 REPEAT
30   PUT GET
40 UNTIL FALSE
50%Answ
60 PRINT "Incoming call received.."
70 REPEAT UNTIL SREG50
80 ...
```

In the example above, the `ON CONNECT` command causes execution of the subroutine starting at the label `%Answ`.

After connection, the modem will always fill S-register 50 with the currently connected line-speed. After a hangup it will clear `SREG50` again. The line-speed is in bits 0 till 5 of this register; see `SPEED` for the possible values. If bit 6 in `SREG50` is set then error correction is used (V42 or MNP) and if bit 7 is set then data compression is active (V42bis or MNP5).

To disable the operation of `ON CONNECT` use `ON CONNECT OFF`.

Pressing `STOP` or `RESET` will also disable `ON CONNECT`.

`ON CONNECT` is generally used in conjunction with `ON HANGUP`, but some care is needed to ensure correct program operation. Further details are given under `ON HANGUP`.

Examples:

```
ON CONNECT
ON CONNECT GOTO %Phone
ON CONNECT OFF
```

Syntax:

```
ON CONNECT ...
ON CONNECT OFF
```

See also:

`ANSWER`, `CONNECT`, `ON HANGUP`

There are two associated events, *DTD* (Data Terminal *[Ready]* Drop) and *DTR* (Data Terminal Ready). These events help your program respond to the status of your terminal.

For more information please refer to the description of `ON DTR` on the following page.

Examples:

```
ON DTD GOSUB Mailbox-entry
```

```
ON DTD OFF
```

Syntax:

```
ON DTD ...
```

```
ON DTD OFF
```

See also:

```
ON DTR, DTR
```

When you enable a terminal program, or switch a terminal on, DTR goes high on the port to the modem. Your modem can react to this event with whatever procedure you like. Like any event handler, this is a background process.

A likely use of this command is to let your modem go from whatever it is normally doing (e.g. operating as a 'mailbox') to modem mode as soon as you start your terminal; and then return to its original function when you disable your terminal.

The following short program routine will make your modem flash its LEDs when your terminal is switched off, while remaining in modem mode when your terminal is switched on (or your terminal program is enabled):

```
10 AT "&D3"  
  
20 ON RESET RUN  
  
30 REPEAT  
  
40 FOR X=1 TO 5  
  
50 WAIT 10  
  
60 LEDX=LEDX EOR 1  
  
70 NEXT  
  
80 UNTIL DTR#1=1 OR DTR#2=1  
  
90 MODEM
```

The systems variable `PORT` will always tell you on what port the event happened.

The `ON DTR` event handler can be disabled with `ON DTR OFF`.

Examples:

```
ON DTR GOSUB Hayes-Modem  
  
ON DTR OFF
```

Syntax:

```
ON DTR ...  
  
ON DTR OFF
```

See also:

```
ON DTD, DTR
```


The `ON ERROR` event handler allows errors that occur during the execution of a program to be 'trapped' so that the appropriate corrective action can be taken. When `ON ERROR` is active, a run-time error will cause program execution to jump to the first command following `ON ERROR`.

`ON ERROR` is often used with `ERL`, `ERN` and `REPORT` to obtain precise information about when an error occurred and the type of error.

`ON ERROR OFF` is used to disable error handling so that any error causes immediate termination of the program. In this case the modem will automatically print the appropriate error message.

`ON ERROR` cannot be used to trap the following:

- `STOP`
- `NMI/STOP`
- `RESET`
- errors that occur within an `ON TIMEOUT` routine.

Examples:

```
ON ERROR GOTO %Mistake  
  
ON ERROR REPORT : PRINT " at line ";ERL : END  
  
ON ERROR OFF
```

Syntax:

```
ON ERROR ...  
  
ON ERROR OFF
```

See also:

```
ERL, ERN, REPORT
```

This event handler can be compared with `ON ERROR`, but will trap an escape only; and will only work when your BASIC program is running.

`ON ERROR` will also trap an escape, but this event handler is easier to use if you want to trap escapes only.

`ESCAPE` events can only occur when escapes have been enabled with the `ESCAPE` command.

Examples:

```
ON ESCAPE GOSUB Switch
```

```
ON ESCAPE OFF
```

Syntax:

```
ON ESCAPE ...
```

```
ON ESCAPE OFF
```

See also:

```
ON ERROR, ESCAPE
```

ON...GOSUB and ON...GOTO are used to jump to another section of code depending on the value of a variable. For instance, if the user is asked to choose one item from a list of options, an ON...GOSUB command could be used to execute the appropriate section of code for each option:

```
500%Menu

510 PRINT "1) Read message"

520 PRINT "2) Send message"

530 PRINT "3) List program"

540 PRINT "4) Set password"

550 PRINT "5) Exit"

560 ON GET-48 GOSUB 600, 700, 800, 900,

1000 ELSE %Menu

600 ...
```

If the user types 1, the subroutine starting at line 600 will be executed; if the user types 2 the subroutine at line 700 will be executed; and so on. If an invalid selection is made, then the ELSE clause of the ON...GOSUB command will be executed and the menu will be re-displayed.

Examples:

```
ON Season GOSUB %Spring, %Summer, %Autumn, %Winter : ELSE %Error

ON Type GOTO 100,150, 200
```

Syntax:

```
ON [integer] GOTO [line-num | label] (, ...) : ( ELSE[line-num | label ]
)

ON [integer] GOSUB [line-num | label] (, ...) : ( ELSE[line-num | label
] )
```

See also:

```
GOSUB ... RETURN, GOTO
```

When the modem is used to process incoming calls the ON HANGUP event handler can be used to automatically detect when the connection has been lost.

Both ON HANGUP and ON CONNECT remain active after program execution has ceased. ON HANGUP OFF should therefore always be used to disable the event handling at the end of a program.

Pressing the STOP button will always terminate all ON functions except ON RESET RUN. The same applies for a RESET or power interruption.

Examples:

```
ON HANGUP RESET
```

```
ON HANGUP GOTO %Bye
```

```
ON HANGUP OFF
```

Syntax:

```
ON HANGUP . . .
```

```
ON HANGUP OFF
```

See also:

```
ON CONNECT
```

ON OFF

disable events

The command ON OFF will disable all ON-events except ON RESET and ON ERROR.

Example:

ON OFF

Syntax:

ON OFF

ON RESET provides one of the most useful facilities in the modem, by allowing program operation to resume automatically at a predetermined point, following a RESET or power failure. The result of the reset procedure is that you can depend on your modem to do exactly what you want it to when you power it up.

There are four ways in which a reset may occur:

- the RESET button is pressed;
- a RESET command is executed;
- there is a power failure, or power surge;
- the internal 'watchdog' circuit times out due to severe errors in programming.

If an ON RESET command is included in the program, any one of these events will cause it to be executed with the following actions being taken:

- all ports will be re-initialized
- all buffers will be emptied
- ECHO will be switched on
- RTS and DTR will be set to 1 (high) for all serial ports
- PRINTER will be turned off
- ON BUTTON will be disabled
- ON ERROR will be disabled
- ON TIMEOUT will be disabled.

All communications settings including BAUD, LENGTH, PARITY and LINK will retain their previous settings, and BASIC variables will retain their values.

Because the reset function has the highest priority, the ON RESET procedures will be effective even if the modem was in **Modem** mode. Thus you can give priority to operation under BASIC after a power failure.

Read the small demonstration program given under ON DTR, to see the effect of ON RESET RUN.

When a reset occurs, four different situations are possible. You have entered:

- ON RESET OFF (or the reset event is not yet enabled)
- ON RESET RUN FILENAME
- ON RESET RUN
- ON RESET

When ON RESET OFF is valid (no reset event has been enabled yet) the modem will stay in the same mode as before the reset happened - either modem-mode or BASIC.

However, when in BASIC mode, there are three further possible situations:

(a) A !BOOT program is found on the RAM-disk

The modem will load this program at &3000 (the default value of PBOOT) and RUN it.

(b) No program running at moment of reset

If a reset occurs while no program is running, the program present in memory at the default value of PBOT will be executed.

This is similar to typing RUN, except that program variables will not be reset.

(c) Program running at moment of reset

If a reset occurs while a program is running, program execution will halt and the error message 'RESET/START at line ...' will be given

This error cannot be trapped by an ON ERROR command.

ON RESET RUN [filename]

This form of the command has the highest priority, and following a reset, it will cause the specified file in User memory to be loaded at &3000 and RUN.

This event has priority over modem operation and will select BASIC automatically when a RESET event happens.

ON RESET RUN

This form of the command has the second highest priority and is only of use when in modem-mode. It will then select BASIC and run either the !BOOT program, or, if there no such program, run the program currently loaded at PBOT.

Pressing the STOP button will disable the normal ON RESET command, but will not affect ON RESET RUN [filename] or ON RESET RUN. This form of the command can only be disabled from within a program using ON RESET OFF.

ON RESET ...

With this form of the command the command immediately following the ON RESET is executed. This type of RESET event handler will not interrupt modem operation when a RESET occurs and will be disabled by pressing the STOP button.

A program using ON RESET can be debugged by using the RESET button to simulate a reset.

Remember that any program containing an ON RESET command must be executed at least once before it will respond to a RESET.

When a program is to be restarted from the beginning after a reset you should use ON RESET CLEAR, in order to avoid re-dimensioning of arrays (which would otherwise cause an error). If your program also uses DATA commands, you should place a RESTORE command after CLEAR to reset the READ pointer.

Examples:

```
ON RESET PRINT "RESET" : RUN  
  
ON RESET OFF
```

Syntax:

```
ON RESET ...  
  
ON RESET OFF
```

See also:

PBOT, RESET

The RING event lets you know when a call comes in, allowing your program time to do whatever you want it to before answering.

Try entering these 3 lines:

```
10 ON RING SOUND 2000,100
20 REPEAT UNTIL FALSE
RUN
```

If you now disconnect your phone from your modem, and dial the number of your line from another location, your modem will 'RING' for you!

Detecting an incoming call is a background process for the modem and could disturb a running program in some cases. To disabled the detection set S-register 44 to 0 (SREG44=0). To enable again set S44 to 1.

Examples:

```
ON RING ANSWER
ON RING SOUND 2000,100
ON RING OFF
```

Syntax:

```
ON RING . . . .
ON RING OFF
```

See also:

```
ON CONNECT
```

The escape sequence event allows you to interrupt whatever the modem is doing by use of an escape sequence. This event will only occur if the escape sequence is also enabled by `SEQUENCE ON`.

The modem S-registers `S2` and `S12` also influence the escape sequence.

When an escape sequence has occurred, you can read the port number where it happened from the systems variable `PORT`.

The following short demonstration program will write characters to your terminal, but will stop doing so as soon as you enter an escape sequence:

```
10 ON SEQUENCE END
20 SEQUENCE ON
30 REPEAT : PRINT "A"; : UNTIL FALSE
```

Examples:

```
ON SEQUENCE MODEM
ON SEQUENCE OFF
```

Syntax:

```
ON SEQUENCE . . . .
ON SEQUENCE OFF
```

See also:

```
SEQUENCE, ON ESCAPE
```

ON SLASH is used to react to the character string 'A/' typed at one of the serial ports and works similar to the ON AT event. The ON SLASH event will only happen if 'A/' is detected on a port which has been previously set to A/-scan using the command AT SLASH.

When the ON SLASH event occurs, the port number of the port which received the 'A/' can be found in the system variable PORT.

The command ON SLASH OFF will turn off the ON SLASH event, but the A/-scan on the ports set with the AT SLASH command will remain active.

This command is very similar to the ON AT command.

Examples:

```
ON SLASH GOSUB Menu
```

```
ON SLASH OFF
```

Syntax:

```
ON SLASH . . . .
```

```
ON SLASH OFF
```

See also:

```
AT, ON AT, PORT
```

ON TIMEOUT can be used to execute a particular sequence of commands at regular time intervals, and therefore provides the basis for carrying out 'background' tasks.

A single numeric parameter is used to specify the time interval in seconds from 1 to 65535. When executed, the timer will count down until the value is 0, and then reset to the original value and execute the commands which follow the ON TIMEOUT command. The end of the timeout routine is marked by a TIMEOUT END command. On completion of a timeout routine, program execution is resumed at the point at which it was interrupted.

```
10 ON TIMEOUT 5 PRINT 'CLOCK$ : TIMEOUT END
20 REPEAT
40   PUT$ GET$
50 UNTIL FALSE
```

Lines 20 to 40 in this example will echo all characters received at the default input port, while the timeout routine on line 10 prints the time at 5 second intervals.

Timeout routines will continue to operate even after the main program terminates unless they are explicitly disabled using ON TIMEOUT OFF. This can be used in isolation at the appropriate point in the program, or it can be used in place of TIMEOUT END:

```
10 ON TIMEOUT 10 PRINT "OK" : ON TIMEOUT OFF
30 REPEAT : PRINT "Looping..." : UNTIL FALSE
```

If you need to terminate program execution within a timeout routine, you must use TIMEOUT OFF (not ON TIMEOUT OFF), followed by END.

```
10 ON TIMEOUT 4 PRINT "ok" : TIMEOUT OFF : END
30 REPEAT UNTIL FALSE
```

The TIMEOUT OFF in line 10 terminates the timeout routine, and is followed by an END command to terminate program execution. END on its own would not be sufficient, because program execution cannot end within an active timeout handler. TIMEOUT OFF will always clear the FOR-NEXT, REPEAT-UNTIL, WHILE-DO and GOSUB stacks and can also be used outside a timeout routine.

TIMEOUT routines can be 'nested', but each time a new timeout is defined the previous one is terminated. This will terminate the original TIMEOUT command as if it were a TIMEOUT END and then take over. In the following example the first timeout will occur after 2 seconds, after which the second timeout will take 5 seconds.

```
10 ON TIMEOUT 2 PRINT "T1" : ON TIMEOUT 5 PRINT "T2" : TIMEOUT END
30 REPEAT : UNTIL FALSE
```

ON TIMEOUT event handlers are disabled when an error is encountered in the timeout routine itself.

Examples:

```
ON TIMEOUT 60 GOTO %Tsub
ON TIMEOUT 1 PRINT CLOCK$ : TIMEOUT END
ON TIMEOUT Interval TIMEOUT END : GOTO %Out
```

Syntax:

ON TIMEOUT [integer] ...

TIMEOUT END

ON TIMEOUT OFF

TIMEOUT OFF

OPORT**default output port**

OPORT is a system variable that defines the default output port number(s).

The following commands will use the port(s) defined by OPORT if no port is specified in the command itself:

ACTIVE	DSR	OUTPUT	REPORT
BAUD	ESCAPE	PARITY	SBITS
CLEAR INPUT	HANDSHAKE	PRINT	SEND
CLEAR OUTPUT	INPUT	PUT	XOFF
CTS	LENGTH	PUT\$	XON
DIR	LINK	RECEIVE	

OPORT differs from IPORT in that more than one port can be defined for simultaneous output. The correct value for OPORT can be calculated from the following table:

PORT	OPORT BIT	OPORT VALUE
1 - Serial port	0	1
2 - Serial port	1	2
3 - Modem	2	4
6 - LCD display	5	32
7 - PRINTER	6	64
8 - BUFFER	7	128

To transmit characters one at a time from serial port 1, you could use either of the following:

```
REPEAT
```

```
    PUT#1,byte
```

```
UNTIL FALSE
```

or

```
OPORT=1
```

```
REPEAT
```

```
    PUT byte
```

```
UNTIL FALSE
```

Similarly, to send text to port 2 and to the printer simultaneously, you could use the following:

```
PRINT#1,#7,"Test message..."
```

but it is more efficient to use OPORT:

```
OPORT=1+64 : PRINT "Test message..."
```

Examples:

```
OPORT=1
```

```
OPORT=135
```

```
OPORT=Port+64
```

PRINT OPORT

Syntax:

OPORT = [integer 0..255]

[num-var] = OPORT

See also:

IPOINT

RI is the RS232 signal used by modems to indicate that an incoming 'ring' has been detected; i.e. that a call has been received.

RI is also an output from the modem on the RS232 ports.

There is of course only one RI for both serial ports, because a modem can either receive a ring or not receive a ring.

The RING output is controlled by the internal processes of the modem. However if you wish, you can also set the RING output yourself.

In a T.C.Lite the RI output on the serial port can not be controlled!

Examples:

```
ORI OFF
```

```
ORI=1
```

```
ring=ORI
```

Syntax:

```
[num-var] = ORI ( #[integer 1..2] )
```

```
ORI = [integer]
```

```
ORI ON|OFF
```


OUTPUT is identical in operation to the PRINT command, and is provided as an alternative to PRINT where program clarity may be improved.

See also:

PRINT

PARITY is used to read or set the type of parity checking used during the transmission and receipt of data via the serial ports. This must be chosen to suit the parity of the system with which you are communicating, otherwise data will be garbled. On the serial ports, parity is usually determined automatically by use of the AT scan. On the modem port however, no AT scan is possible, and you will have to set the parity as required by the remote user.

There are four types of parity; known as EVEN, ODD, MARK and SPACE; with a fifth option being to omit the use of parity checking completely. To select one of the options, the PARITY command is used as follows:

0	Disabled
1	Odd
2	Even

The PARITY command can be followed by one or more port numbers. If no port number is specified, the default output port as defined by OPORT will be assumed.

Parity checking will be disabled on all ports when the STOP button is pressed.

Examples:

```
PARITY 2  
  
PARITY#2, X  
  
PARITY#1, #2, #3, 0  
  
PARITY#3 = PARITY#1
```

Syntax:

```
PARITY ( #[port], ) (#...) [integer 0..2]  
  
[num-var] = PARITY ([port])
```

See also:

BAUD, LENGTH, SBITS

This is an important system variable that contains the address of the start of a BASIC program.

The default value for PBOT is &3000, but this can be changed so that more than one program can be stored in system RAM at once. For example:

```
PBOT=&4000
```

```
NEW
```

```
10 %Sub1
```

```
20 PRINT "SUBROUTINE 1"
```

```
30 RETURN
```

```
PBOT=&4300
```

```
NEW
```

```
10 %Sub2
```

```
20 PRINT "SUBROUTINE 2"
```

```
30 RETURN
```

```
PBOT=&3000
```

```
NEW
```

```
10 REM Main program
```

```
20 REM Press key 1-3 (ESCAPE to stop)
```

```
40%Waiting
```

```
50 ON GET-48 GOSUB %Sub1,%Sub2 ELSE %Waiting
```

```
60 PBOT=&3000 : GOTO %Waiting
```

```
80%Sub1
```

```
90 PBOT=&4000 : GOTO %Sub1
```

```
100%Sub2
```

```
110 PBOT=&4300 : GOTO %Sub2
```

Here, the various programs are used as subroutines which can be called by setting PBOT to the appropriate address.

Examples:

```
PBOT=&4000
```

```
PBOT=PBOT+&1000
```

PRINT ~PBOT

Syntax:

PBOT = [integer]

[num-var] = PBOT

See also:

P'TOP

PEEK**read byte from memory**

PEEK reads a single byte from the specified memory location into an integer variable.

Examples:

```
Byte=PEEK &4000  
X=PEEK Address  
X=PEEK(&2000+Ptr1)  
PRINT PEEK &9000
```

Syntax:

```
[num-var] = PEEK [integer]
```

See also:

```
PEEK$, POKE, POKE$
```

PEEK\$**read string from memory**

PEEK\$ reads a series of bytes starting from the specified memory location up to the first Carriage Return character (ASCII 13) into a string variable.

Examples:

```
code$=PEEK$ &2000  
X$=PEEK$ Address  
name$=PEEK$ (&Base+Offset)  
PRINT PEEK$ &9000  
ON BUTTON1 AT PEEK$ BUTTON1
```

Syntax:

```
[string-var] = PEEK$ [integer]
```

See also:

PEEK, POKE, POKE\$

PLAY [filename]**playback a recorded file**

This command is used to playback a recorded file. Wildcards in the filename are allowed to playback a sequence of speech.

Before entering the `PLAY` command, the following points are important to know:

1. The modem must be in voice-mode. This is done with the `VOICE` command.
2. A RAM buffer of at least 1024 bytes big must be defined with `BUFFER`.
3. The format of the playback file must be set the same as it was when recording that file; the bits per sample in S-register 67 must be set correctly if changed earlier (see `VOICE`).
4. All microphone functions must be disabled, otherwise nothing is heard.
5. The volume of the speaker and headset can be set with `VOLUME 1/2/3`. The internal speaker can be turned on or off with `SPEAKER ON/OFF`.
6. DTMF detection is active during playback, except if bit7 in Sregister 67 is set. After detecting a DTMF the playback is stopped (see `ERN` below).
7. Detection of the busy-tone (hangup) will also stop playback, except if bit7 in S-register 39 is cleared (default).
8. If `PLAY` exits through an event-handler (`ON ESCAPE`, `ON TIMEOUT`, `ON BUTTON`, etc.) the `PLAY OFF` command must be executed!

On exit the system variable `ERN` can have the following values:

<code>ERN = 0</code>	<code>OK</code>	:	Finished
<code>ERN = 6</code>	<code>Cancelled</code>	:	Busy-tone/hangup detected
<code>ERN = 35</code>	<code>-</code>	:	DTMF # detected
<code>ERN = 42</code>	<code>-</code>	:	DTMF * detected
<code>ERN = 48</code>	<code>-</code>	:	DTMF 0 detected
<code>ERN = 49</code>	<code>-</code>	:	DTMF 1 detected
<code>ERN = 50</code>	<code>-</code>	:	DTMF 2 detected
<code>ERN = 51</code>	<code>-</code>	:	DTMF 3 detected
<code>ERN = 52</code>	<code>-</code>	:	DTMF 4 detected
<code>ERN = 53</code>	<code>-</code>	:	DTMF 5 detected
<code>ERN = 54</code>	<code>-</code>	:	DTMF 6 detected
<code>ERN = 55</code>	<code>-</code>	:	DTMF 7 detected
<code>ERN = 56</code>	<code>-</code>	:	DTMF 8 detected
<code>ERN = 57</code>	<code>-</code>	:	DTMF 9 detected
<code>ERN = 64</code>	<code>-</code>	:	DTMF A detected
<code>ERN = 65</code>	<code>-</code>	:	DTMF B detected
<code>ERN = 66</code>	<code>-</code>	:	DTMF C detected
<code>ERN = 67</code>	<code>-</code>	:	DTMF D detected
<code>ERN = 73</code>	<code>Not found</code>	:	Filename not found

Example program using `RECORD` & `PLAY`:

```
10 VOICE
20 RECORD TIMEOUT=15
30 PRINT "Pick up your phone...";
40 REPEAT UNTIL ILINE0
50 OLINE0 ON
60 WAIT 100
70 REPEAT
```

```
80 PRINT "Speak now...";
90 RECORD "TEST.VOI"
100 REPORT ERN
110 PRINT "Listen now...";
120 PLAY "TEST.VOI"
130 IF ERN PRINT "DTMF: ";ERN; ELSE REPORT ERN
140 UNTIL FALSE
```

Syntax:

```
PLAY [FILENAME]
```

See also:

PLAY ON/OFF, PLAY VOLUME, RECORD, RESET MODEM, VOICE

The command `PLAY ON` is used during testing only. It sets the play mode without sending a file. Data can be written to the decoder through port 3 (`PUT#3`, `PRINT#3`, etc.).

The command `PLAY OFF` is used to turn off the play mode. This command must be issued when a playback is aborted by an event (`TIMEOUT`, `ESCAPE`, etc.).

Syntax:

```
PLAY ON|OFF
```

See also:

```
RECORD, RESET MODEM, VOICE
```

Only one playback parameter can be set.

The default is: `PLAY VOLUME 3`.

[num] = Playback Attenuation

This sets the speech signal level and is the same as setting S-register 59. Value 0 is the loudest (not recommended) and value 15 is the most attenuated level.

Syntax:

`PLAY VOLUME [num]`

See also:

`RECORD`, `RESET MODEM`, `VOICE`

POKE**put byte into memory**

POKE is used to write single bytes of data into memory, and is most commonly used to pass values to and from machine code routines, or to save information which can be retrieved later with the PEEK command. Two parameters are required, which specify the address and the value to be stored.

Examples:

```
POKE Address, Byte
```

```
POKE &4000+Pointer, GET
```

```
POKE &7000, B
```

Syntax:

```
POKE [integer], [integer 0..255)
```

See also:

```
PEEK, PEEK$, POKE$
```

POKE\$**put string into memory**

POKE\$ is used to write strings into memory which can be retrieved later using PEEK\$. Two parameters are required which specify the start address and the string to be stored. Each character in the string is stored in the next sequential memory location from the specified start address. A Carriage Return is automatically appended to denote the end of the string.

As an example of the use of POKE\$, the following program stores lines of text one after the other in memory:

```
20 DIM Address 500 : Pointer=0
30 REPEAT
40   INPUT Line$
50   POKE$ Address+Pointer, Line$
60   Pointer=Pointer+LEN(Line$)+1
70 UNTIL Pointer=5
```

This is more efficient than using string arrays because each string takes up the minimum amount of memory possible. Text stored in this way could be retrieved with the following program:

```
10 DIM Address 500 : Pointer=0
20 REPEAT
30   Line$=PEEK$(Address+Pointer)
40   PRINT Line$
50   Pointer=Pointer+LEN(Line$)+1
60 UNTIL Pointer=5
```

Refer to POKE for a description of access to User memory with POKE\$.

Examples:

```
POKE$ Address,String$
POKE$ &A000+I,X$
```

Syntax:

```
POKE$ [integer] , [string]
```

See also:

```
PEEK, PEEK$, POKE
```

PORT is used to read the port number on which the last event happened.

PORT is thus associated with the ON . . . event handlers.

PORT can also be part of many command lines to indicate the port number (instead of '#').

Examples:

```
IF PORT=3 THEN GOSUB Password-control
PRINT PORT 2 "Hello"
```

Syntax:

```
[num-var] = PORT
```

See also:

```
ON BRK, ON DTD, ON DTR, ON ESCAPE, ON SEQUENCE
```

PRINT is used to output numbers and/or strings to the output ports. There are several major advantages over PUT and PUT\$:

- a) PUT and PUT\$ can only send a single value/character at a time. PRINT can send multiple characters or values.
- b) Any output formatting required must be carried out explicitly using PUT and PUT\$, but PRINT provides a variety of formatting options.

If no port number is specified, the default output port as defined by OPORT will be assumed.

All printing from memory is buffered and interrupt driven; i.e. on execution of the PRINT command, the data to be printed is immediately copied to a buffer. Program control then returns to the command following PRINT, while the buffered data is output by a background interrupt routine. Care should be taken not to clear the output buffer before it has been fully emptied - in the next example, the buffer may be cleared before the complete text has been transmitted, and some data may be lost:

```
10 PRINT "VIDICODE Datacommunications - ARGUS Programmable Modem"
20 CLEAR OUTPUT
```

PRINT positioning & formatting

Various characters can be used within the PRINT command to alter the format of the output by inserting or suppressing Carriage Returns:

; a semi-colon following an item to be printed *suppresses* the automatic addition of [CR], and causes printing to commence immediately following the last printed value.

, a comma following a value/string to be printed causes printing to commence at the start of the *next* horizontal *field* position. This is 8 characters away from the last horizontal field position or the start of the line.

' a single quote mark before or after a value/string to be printed causes the *insertion* of an additional [CR].

~ a tilde causes numeric values to be printed in *hexadecimal* format.

Command	Characters sent
PRINT "X"	X[CR]
PRINT "X";	X
PRINT '	[CR][CR]
PRINT "'X"	[CR]X[CR]
PRINT 1	1[CR]
PRINT -1234	-1234[CR]
PRINT ~11	B[CR]
PRINT ~-1	FFFF[CR]
PRINT ;1;	1
PRINT ;~-1;	FFFF
PRINT "X",1	X 1[CR]
PRINT "X" '1	X[CR] 1[CR]
PRINT 1,1;	1 1

Examples:

```
PRINT "Message number : ";M_Nr;
PRINT#1,#7,12+9*A(I)
PRINT ~1000*R/P
PRINT ;&100+PEEK(&2000)
PRINT#1,"DTR=";DTR#2,"RTS=";RTS#2
PRINT E1$+E2$+CHR$7
```

Syntax:

```
PRINT ( #[port] , ) (#...) ( [string] ) (') (,) (;) (~) ( [integer] )
(;) )
```

See also:

OUTPUT, PUT, PUT\$

PRINTER provides an alternative means of controlling the printer port. The parallel printer port is only available in an Argus Programmable Modem, not in a T.C.Lite. There are four forms of the command:

PRINTER and PRINTER ON

The PRINTER and PRINTER ON commands will enable the printer by adding 64 to the value of OPORT.

PRINTER OFF

This subtracts 64 from the value of OPORT, thereby disabling the printer.

PRINTER LINEFEED ON/OFF

This causes the modem to insert or suppress the insertion of a Linefeed character after every Carriage Return sent to the printer.

PRINTER INPUT/OUTPUT

This command is used to configure the parallel port for Input or Output.

Examples:

```
PRINTER
```

```
PRINTER ON
```

```
PRINTER LINEFEED OFF
```

```
PRINTER INPUT
```

Syntax:

```
PRINTER ON|OFF
```

```
PRINTER LINEFEED ON|OFF
```

```
PRINTER OUTPUT
```

See also:

```
LINEFEED, OPORT
```


Selecting between different protocols is done by the `PROTOCOL` command. Before sending or receiving using the `SEND`, `X` or `RECEIVE`, `X` commands you must set the right options with this command.

`PROTOCOL` expects a number between 0 and 255. The bits in the number are organized as follows:

- bit0 =0/1= Use checksum/CRC-16 for error detection
- bit1 =0/1= Use 128/1024 bytes in a block (Xmodem-1K or Ymodem-1K)
- bit2 =0/1= No/Yes batch option (Xmodem/Ymodem)
- bit3 =0/1= No/Yes stream option (Xmodem-g or Ymodem-g)
- bit4 =0/1= Yes/No end of transfer with batch
- bit5 =0/1= Use last number before first dot/last in name with Ymodem
- bit6 =0/1= Use real/info-block for filenames with Ymodem
- bit7 =0/1= No/Yes CTRL-Z (EOT) at end of file in last block with Xmodem

Bit0: Checksum or CRC-16.

Checksum is an old way to detect errors in a block. These days CRC-16 is always used. Some old systems, however, may still be using checksum. The modem will automatically detect that the other system is using checksum, when CRC-16 is selected. So, it is best that this bit is always high.

Bit1: 128 or 1024 bytes

Using 128 bytes in a block is normal for Xmodem and using 1024 bytes in a block is normal for Ymodem. When sending big files with Xmodem, then 1024 bytes can be useful. Sending in Ymodem over a bad line, then 128 bytes would be useful.

Note: In order to be able to use 1024 bytes in a block, a buffer must be defined with the `BUFFER` command of at least 1K (&400) bytes. Otherwise, the error message 'No/Bad Buffer' (`ERN=92`) is generated. When using MNP5 together with file transfer, a buffer of at least 4K (&1000) bytes must be defined. So, the command `BUFFER &1000` is suitable for all situations.

Bit2: Batch option

With this you select between Xmodem or Ymodem. Xmodem only sends the contents of the file. Ymodem also sends the filename and file length.

Bit3: Stream option

With this bit you select the stream option, also known as Xmodem-g or Ymodem-g. This type of transfer can only be used over reliable lines, because there is no error recovery. The sending side keeps on sending and the receiving side keeps on receiving. When the receiver detects an error, it aborts the transfer immediately. It is the fastest way to transfer, because no ACK's and NAK's are sent, only data blocks.

Bit4: End-of-transfer with batch.

This bit only has effect when Ymodem-send is selected. When it is set, a `SEND` command will return without sending the end-of-transfer to remote. This means that a next `SEND` command (with Ymodem) can be entered. The last file must be sent with bit4 cleared.

Example:

```

PROTOCOL 7+16
SEND "FILE1",X           : 1st file
SEND "FILE2",X           : 2nd file
SEND "FILE3",X           : 3rd file
    
```

```

PROTOCOL 7
SEND "FILE4", X           : 4th and last file

```

Bit5: Ymodem-receive file name counter.

This bit only has an effect when Ymodem-receive is selected and when bit6 is 1. It determines how the filename is used when receiving multiple files with Ymodem. If bit5 is set, then the last number in the filename is incremented. If bit5 is cleared, then the last number before the first dot (if any), is incremented.

Example:

```

PROTOCOL 7+64
RECEIVE "TEST0001.01.Y", X
           ^^^^
RECEIVE "1234-0001.01.Y", X
           ^^^^
RECEIVE "1234.01.Y", X
           ^^^^

```

or

```

PROTOCOL 7+32+64
RECEIVE "TEST0001.01.Y", X
           ^^
RECEIVE "1234-0001.01.Y", X
           ^^
RECEIVE "1234.01.Y", X
           ^^

```

or

```

PROTOCOL 7+64 (or PROTOCOL 7+32+64)
RECEIVE "TEST001", X
           ^^^
RECEIVE "TEST01-001", X
           ^^^
RECEIVE "1234", X
           ^^^^
RECEIVE "TEST.01", X
           ^^

```

Bit6: Ymodem filename

The modem can select between 2 modes, when using Ymodem.

The first mode (bit6=0) uses real filenames, which is the normal way.

Example:

```

Computer sends file "LETTER.TXT"
Modem receives file "LETTER.TXT"

```

or

```

Modem sends file "LETTER.TXT"
Computer receives file "LETTER.TXT"

```

Note: When receiving files with Ymodem in this way, a dummy filename must be used with the RECEIVE command (RECEIVE "dummy", X).

The second mode (bit6=1) is specially implemented for use inside the modem because some applications will need it. The outside world won't notice the difference when implemented correctly.

Instead of using the real filename, the modem uses the Information Block (IB) for the received or sent filenames. First some explanation about IB.

Each file in the internal filing system of the modem has 32 bytes of free space attached to that file (the IB). These bytes can be accessed or made visible with the following commands.

```

SEND [name], B

```

```
RECEIVE [name] , B
```

```
DIR [name] , B
```

```
DIR [name] , F
```

Exchanging data between the IB and the file is done with PEEK and POKE on addresses &2BE0 til &2BFF.

When the command SEND [name] , B is given, the IB of that file is placed on &2BE0.

When the command RECEIVE [name] , B is given, the 32 bytes on &2BE0 are placed in the IB of that file.

When a new file is created with RECEIVE, the contents of &2BE0 is used to fill the IB of that new file.

The SEND and LOAD commands are also filling &2BE0 with their IB.

The contents of the IB can be read with DIR [name] , B. DIR [name] , F will also show the contents, but only when there is a legal ASCII string in it, terminated with <CR>, ASCII 13.

When receiving with Ymodem, the modem uses the filename given with the RECEIVE command. The real filename from the other side is stored in the IB. Before storing the name, the modem first looks for a <CR> on &2BE0 til &2BFF. If there is a <CR> at the start, the filename will be stored at the beginning of the IB. Also, when there is no <CR> found, the filename will be stored at the beginning. If it finds a <CR> after a string, the filename will be stored after this string and the <CR> will be replaced by a '\'.
The filename used by the modem to store it, must contain an ASCII number. The number nearest to the dot (if any) is taken. This number is automatically incremented by the modem when more than 1 file is received.

Examples:

```
PROTOCOL 7+64          (or PROTOCOL 71)
```

```
POKE &2BE0,13
```

```
RECEIVE "FILE0001",X    (3 files are received)
```

```
DIR "FILE*",F
```

```
1 0005 FILE0001 24-1 U : LETTER.TXT
2 0020 FILE0002 24-1 U : GAME.EXE
3 0009 FILE0003 24-1 U : MODEM.DOC
```

```
POKE &2BE0,"LETTERS"
```

```
RECEIVE "LET08",X (4 files are received)
```

```
DIR "LET*",F
```

```
1 0005 LET08      24-1 U : LETTERS\ORDER.TXT
2 0030 LET09      24-1 U : LETTERS\ALWIN1.TXT
3 0009 LET10      24-1 U : LETTERS\ALWIN2.TXT
4 0009 LET11      24-1 U : LETTERS\HAWAI.TXT
```

When sending, the modem gets the real filename from the IB and no ASCII number is incremented automatically.

Examples:

```
PROTOCOL 7+64
```

```
SEND "FILE*",X
```

```
SEND "LET*",X
```

Bit7: CTRL-Z (=EOT= End of Text)

If bit7=0 the file length will not be correct, since remaining bytes in the last block are filled with 0 bytes (=NUL). When bit7=1, first a CTRL-Z is placed at the end of the file, the rest is filled with 0. This is sometimes needed when sending text files.

Syntax:

PROTOCOL [integer 0-255]

This is an important (read-only) system variable that contains the address where a BASIC program ends. The length of the program can be determined by subtracting the value of `PBOT` from `PTOP`:

```
PRINT PTOP-PBOT
```

`PTOP` is useful when more than one program is stored in memory:

```
start1=PBOT
```

```
PBOT=PTOP-1
```

```
(NEW)
```

```
LOAD ...
```

```
PBOT=start1
```

```
RENUM
```

Examples:

```
End=PTOP
```

```
PRINT ~PTOP
```

Syntax:

```
[num-var] =PTOP
```

See also:

```
PBOT
```

PULSE is used to select pulse dialing in dial commands, or to dial just one or more digits.

Examples:

```
DIAL PULSE "123456"
```

```
PULSE 8
```

```
PULSE 2,6,4
```

Syntax:

```
DIAL PULSE [string]
```

```
PULSE [integer] ([,...])
```

See also:

TONE

PUT**write byte value to port**

PUT is used to write single byte values to the specified output port(s).

If the port number is omitted, the default output port as defined by `OPORT` will be assumed. The character to be written is given as a single byte which represents its ASCII value.

For example, the command:

```
PUT#1, 65
```

will write ASCII character 65 (the letter 'A'), to port 1 and is therefore equivalent to:

```
PRINT#1, CHR$(65)
```

Multiple-byte data can be sent with a single PUT command by separating each ASCII value with a comma:

```
PUT#2, 12, 9, 9, 10, 10, 65
```

When the PUT function is used to send a Carriage Return, the modem does NOT automatically append a Linefeed.

PUT is often used in conjunction with GET as a simple means of passing characters received from one port as output to another:

```
10 REPEAT
```

```
20 PUT#2, GET#1
```

```
30 UNTIL FALSE
```

In this example, the byte value returned from GET#1 is used as the parameter for the PUT function, so that all characters received from port 1 are written to port 2.

Examples:

```
PUT 12
```

```
PUT#2, 32, 32, 32, 13, 10
```

```
PUT 30, 13, 13, 255
```

```
PUT#2, #4, 12
```

Syntax:

```
PUT ( #[port] , ) ( #... ) [integer 0..255]
```

See also:

```
PRINT, PUT$
```

PUT\$**write character to port**

PUT\$ is used to write single ASCII characters to the specified output port(s).

If the port number is omitted, the default output port as defined by OPORT is assumed. The character to be written is given as a single character string.

For example:

```
PUT$#1, "A"
```

will print the character "A" to port 1. If the parameter string is longer than a single character, only the first character in the string is used so that:

```
PUT#1, Name$
```

is equivalent to:

```
PRINT#1, LEFT$(A$, 1) ;
```

PUT\$ may be used in conjunction with KEY\$ as a simple means of passing received characters from one port as output to another:

```
10 REPEAT  
  
20   PUT$#1, KEY$#2  
  
30   PUT$#2, KEY$#1  
  
40 UNTIL FALSE
```

Examples:

```
PUT$ KEY$#1  
  
PUT#1, #7, Id$
```

Syntax:

```
PUT$ ( #[port], ) (#...) [string var]
```

See also:

```
PRINT, PUT
```


READ is used to read information stored in DATA commands into integer or string variables. As many items of data are read as is necessary to assign to the variables given in the READ command.

If there is insufficient data to satisfy the READ, an 'Out of Data' error message will be given.

Examples:

```
READ Name$, Code
```

```
READ Port(I), Rate(I), Wlen(I)
```

```
READ X, X$(X), Y, Y$(Y)
```

Syntax:

```
READ ( [num-var] | [string-var] ) , ...
```

See also:

```
DATA, RESTORE
```

RECEIVE**receive file or data**

RECEIVE is one of the most powerful commands available with this modem, and provides a simple means of transferring data into the modem under program control, with minimum effort.

There are 3 basic forms of the command:

RECEIVE [port] , [address] : receive *data* using Xmodem

RECEIVE [port] , [filename] , [type] : receive *file*

RECEIVE ALL : receive a backup of the disk

In addition, the second form can be used with one of a number of options which are dealt with individually in the following sections.

Here, `RECEIVE` is used to transfer *data* as opposed to a file, using the Xmodem file transfer protocol to prevent the occurrence of errors.

The command is followed by a port number (1 - 3 only) and an address, which are both optional.

If the parameters are omitted, the default input port as defined by `IPORT`, and the start address defined by `PBOT`, will be used.

Examples:

```
RECEIVE#2           : receive data and store at PBOT
RECEIVE &3000       : receive data and store at address &3000.
RECEIVE Location    : receive data and store at address defined by variable
                    : 'Location'.
RECEIVE#3, &4321    : receive data via modem (port 3) and store at address
                    : &4321
```

This form of the `RECEIVE` command cannot be used to download BASIC programs in ASCII format into the modem, because during receive, data is simply copied into the modem's memory, and would not undergo the tokenization process required by the BASIC interpreter.

During a data `RECEIVE`, the `ESCAPE` and `XOFF` settings for the specified port will be temporarily disabled, and the character length will be set to 8 bits. The original settings will be restored after completion of file reception.

An error message (28) will be generated if, after 180 seconds, no characters have been received. If this is likely to occur, you should trap the error using the `ON ERROR` command. The same error message will be given if more than nine Xmodem transfer errors occur during reception of the same block.

Examples:

```
RECEIVE
RECEIVE#2
RECEIVE &4000
RECEIVE#Port, Address
```

Syntax:

```
RECEIVE ( #[port], ) ( [integer] )
```

See also:

```
SEND#[port], [address]
```

When followed by a string, the `RECEIVE` command is used to receive and store complete *files* in the filing system:

```
RECEIVE [filename]
```

All types of files discussed under the `DIR` command may be received, and there are various options which are discussed below.

RECEIVE [filename] (,T/N)

`RECEIVE` without a parameter, or with the `, T` parameter, prepares the modem to receive a text file. On executing the command a file is opened and the message:

```
Please enter your message:
```

will be transmitted.

You may suppress this message by using `ECHO OFF` before `RECEIVE`, if necessary. Any text now received by the modem will be stored in the specified file (if the file already exists it will be overwritten). Text storage is terminated and the file closed when a single line containing "NNNN" or "NNNNE" is received. At the same time the file directory will be updated to include the date and time at which the file was received.

The sender may cancel the transfer at any time by entering "CCCC" on a line by itself, in which case none of the received text will be stored. The transfer will also be aborted if an `ESCAPE` is entered, or the `RESET` buttons is pressed.

All event handlers such as `ON TIMEOUT` will remain active during the transfer.

On completion of the transfer, the `DIR` command can be used to check whether the file has been successfully received, how much memory it occupies, etc. The message can be read back out of the modem by using the appropriate form of the `SEND` command.

One further variations is used to receive text files from systems which do not support the `[CR]NNNN[CR]` convention:

```
RECEIVE [filename],N
```

In both cases the prompt message is suppressed and the modem will wait for the first character to be received before initiating a countdown timer. Whenever a character is received it is stored in the file and the timer is reset. The file is closed when no characters have been received after a certain time (specified with `RECEIVE TIMEOUT`); or when either of the `[CR]CCCC[CR]` or `[CR]NNNE[CR]` sequences have been received; or when `ESCAPE` has been entered.

On exit the system variable `ERN` can have the following values:

```
ERN =    0  OK                : File stored
ERN =    6  Cancelled         : CCCC sequence detected
ERN =   22  Out of memory     : Disk full
```

RECEIVE [filename],F

`RECEIVE, F` is used to capture all incoming data, including control codes, into a file of the specified name. Received characters are not echoed, and capture will terminate automatically when no data has been received after a timeout that is specified with `RECEIVE TIMEOUT`. Detecting `ESCAPE` is disabled during reception with the `, F` option.

`RECEIVE, F` can also be used to copy or manipulate files within the modem by using the `BUFFER` (port 8). The following program demonstrates how this can be done:

```
10 CLEAR
20 BUFFER &1000
```

```

30 REPEAT
40   INPUT "Enter Source file : "; Src$
50   MATCH Src$
60   IF NOT MATCH PRINT "No such file" : GOTO 40
70   INPUT "Enter Destination file : "; Dest$
80   CLEAR#8
90   SEND#8, Src$, F
100 RECEIVE#8, Dest$, F
110 Print "File copied OK"
120 UNTIL FALSE

```

In this case, no timeout condition is applied and the `RECEIVE` will terminate automatically when the buffer is empty. The buffer size specified must be large enough to accommodate the entire file. If the file size is greater than 16K (i.e. larger than the maximum buffer size), copies must be made by using `SEND [source], P (part)` and `RECEIVE [dest], A (append)`.

Of course, you will normally use the command `COPY` to copy files!

RECEIVE [filename](,T/F),A

This command allows you to append data to the end of an existing file. The file must be the last file in a RAM-disk otherwise `ERN 93` ("Not last") will be generated. For a Hard-disk filing system it doesn't matter which file on the disk is used. When the file does not exist, it is created first.

The input is determined by the `, T` or `, F` options (text or binary). When the disk becomes full, the system variable `ERN` is set to 22 ("Out of memory") and the `RECEIVE` is aborted.

RECEIVE [filename],X

`RECEIVE, X` is used to transfer files using the Xmodem / Ymodem error correction protocol. Any type of file may be received from a compatible Xmodem / Ymodem system with complete safety, and although binary files may contain data from other computers and cannot be used by the modem itself, they can be stored for access by other systems using the `SEND, X` command.

The `PROTOCOL` command must be set first to select the right transfer protocol. If Ymodem-Batch is used, the filename is included in the transfer and can be a dummy filename in the `RECEIVE, X` command.

After receiving, the system variable `ERN` must be checked to see if the receive was successful. For example:

```

PROTOCOL 7

RECEIVE "dummy", X

IF ERN GOTO Err ELSE GOTO Ok

```

On exit the system variable `ERN` can have the following values:

<code>ERN =</code>	<code>0</code>	<code>OK</code>	<code>:</code>	File stored
<code>ERN =</code>	<code>7</code>	<code>Failed</code>	<code>:</code>	Timeout or too many errors
<code>ERN =</code>	<code>22</code>	<code>Out of memory</code>	<code>:</code>	Disk full
<code>ERN =</code>	<code>23</code>	<code>Subscript</code>	<code>:</code>	File too big (>16 Mbyte)
<code>ERN =</code>	<code>33</code>	<code>Aborted</code>	<code>:</code>	CTRL-X detected twice
<code>ERN =</code>	<code>72</code>	<code>Bad name</code>	<code>:</code>	Filename not correct (Ymodem-Batch)

If Ymodem-Batch is used, the system variable `MATCH` contains the number of files received.

RECEIVE [filename],C

The `RECEIVE, C` command is very useful in combination with the `SEND, C` command to create and restore file copies. The used filename is not important (but must be filled in), because the received file contains all the original Argus filenames. All other Argus-filing-system related information is also included in the copy for each file. The Xmodem-CRC protocol must be used and is set automatically with `RECEIVE, C (PROTOCOL 1)`. Example:

```
RECEIVE "dummy", C
```

On exit, the system variables `ERN` and `MATCH` can have the same values as with `RECEIVE, X`.

RECEIVE [filename],B

For every file stored in the modem, a 32-byte information block is created. This block is provided for use by the programmer in storing additional information about the file, and may contain items such as telephone numbers, user names, date & time, etc.

The information block is edited by using `POKE`, to put the appropriate data into memory locations `&2BE0` to `&2BFF`. This 32-byte memory block is always saved together with the file after a successful `RECEIVE` command (this is also true for `SAVE` and `RECORD`). Setting the memory block before the `RECEIVE` command is more efficient, because in that case the `RECEIVE, B` command can be omitted. `RECEIVE, B` is only used to incorporate the block into an existing file.

The block on memory locations `&2BE0` to `&2BF0` and the block in a file are never changed or used by the filing system itself. So, the block is completely free for use by the programmer.

`SEND . . . , B` is used to extract the information block from a file and store it back into memory at locations `&2BE0` to `&2BFF`. The information can then be read using `PEEK`.

`DIR, B` can be used to send the information block to a port (a Carriage Return will be automatically appended to the end of the block).

For example:

```
10 RECEIVE "TESTMESSAGE", T
20 POKE$ 240, "INFORMATION"+SPC$(21)
30 RECEIVE "TESTMESSAGE", B
40 SEND "TESTMESSAGE", B
50 PRINT 'PEEK$ 240'
60 DIR "TESTMESSAGE", B
```

Line 10 is used to create the test message, which must be terminated with `[CR] NNNN [CR]` as usual.

Lines 20 and 30 are then used to fill the information block afterwards.

Line 40 extracts the information block from the file, and lines 50 and 60 illustrate the two methods of displaying the block.

RECEIVE ROM [filename], [type]

`RECEIVE ROM` is used to receive one file or a group of files to a RAM chip in socket 2 of an Argus Programmable Modem. See `ROM` for further details.

The type and options are used in the same manner as with the normal `RECEIVE` command.

Examples:

```
RECEIVE "MESSAGE"  
RECEIVE B$+".TXT",T  
RECEIVE "BINARY",X  
RECEIVE "PROG"+VAL$(Nr)+".BIN",X  
RECEIVE#3, Name$+".JAN"  
RECEIVE Page$,B  
RECEIVE ROM "TEXT",T
```

Syntax:

```
RECEIVE ( #[port] , ) [filename] ( ,T ) ( ,F ) ( ,A ) ( ,X ) ( ,B )  
RECEIVE ROM ...
```

See also:

DIR, PROTOCOL, ROM, SEND

The command `RECEIVE ALL` is normally used to receive a RAM-disk backup. It overwrites the complete contents of the RAM chip and only works if the file is created with `SEND ALL`.

In a RAM-disk filing system this command differs from the command `RECEIVE, C` which expects a file created with `SEND, C`.

In a Hard-disk filing system, this command is exactly the same as the command `RECEIVE, C`.

The Xmodem-CRC protocol is used for this purpose; `PROTOCOL 1` is set automatically.

Example:

```
RECEIVE ALL
```

Syntax:

```
RECEIVE (ROM) ALL
```

See also:

```
RECEIVE, ROM
```


RECEIVE TIMEOUT = [int]	set Sreg66 timeout
--------------------------------	---------------------------

This command is used to set the timeout in S-register 66. This is used with `RECEIVE, F, N`. Note that time is set in 1/10 seconds.

Example:

```
RECEIVE TIMEOUT = 50
```

This will set the receive timeout to 5 seconds.

Syntax:

```
RECEIVE TIMEOUT = [integer]
```

See also:

```
RECEIVE
```

The command `RECORD [filename]` is used to store an encoded voice.

Before entering the `RECORD` command, the following points are important to know:

1. The modem must be in voice-mode. This is done with the `VOICE` command.
2. A RAM buffer of at least 1024 bytes big must be defined with `BUFFER`.
3. The speech quality, bits per sample, must be set in S-register 67 (see `VOICE`).
4. Microphone control, volume of speaker/headset and speaker on or off must be set right.
5. DTMF detection is active during recoding, except if bit6 in Sregister 67 is set. DTMF's are not detected, if recording is done off-line with a microphone. After detecting a DTMF the recording is stopped (see `ERN` below). The DTMF itself is stripped from the file, except if bit0 in S-register 66 is set (`RECORD TIMEOUT` has an odd value).
6. Recording is also stopped after a certain time of silence. This time can be set per 100ms with `RECORD TIMEOUT` or by setting S-register 66. The system variable `ERN` will be 0 in this case. Detecting silence can be adjusted with bits0+1 in S-register 69; 0=Most sensitive (default) and 3=Less sensitive.
7. Detection of the busy-tone (hangup) will also stop the recording; `ERN` is set to 6 in that case. Busy-tone detection can be disabled by setting S-register 39 to 0 (`SREG39=0`).
8. An overall timeout can also stop the recording. This timeout is set per second with the command `ON TIMEOUT <num>` in front of `RECORD`. The timeout event itself is disabled again, only the value from `<num>` is used. `ERN` is set to 7 after the timeout.
9. Recording is also stopped if a character from a serial port is received. This feature must be enabled first by setting bit2 in S-register 28 (`SREG28=SREG28 OR 4`). This is used when recording is done until control of a serial connected computer. `ERN` will be 0 on exit.
10. The programmer must take into account that a recorded file can be maximal 16Mbyte big. If 4 bits per sample is used (3600 bytes per second), the recording can last about 4500 seconds. If 2 bits per sample is used (1800 bytes per second), the recording can last about 9000 seconds.
11. If `RECORD` exits through an event-handler (`ON ESCAPE`, `ON BUTTON`, etc.) the `RECORD OFF` command must be executed!

On exit the system variable `ERN` can have the following values:

<code>ERN = 0</code>	OK	:	Stored (silence timeout)
<code>ERN = 6</code>	Cancelled	:	Busy-tone/hangup detected
<code>ERN = 7</code>	Failed	:	Overall timeout
<code>ERN = 22</code>	Out of memory	:	Disk full
<code>ERN = 23</code>	Subscript	:	File too big (>16Mbyte)
<code>ERN = 35</code>	-	:	DTMF # detected
<code>ERN = 42</code>	-	:	DTMF * detected
<code>ERN = 48</code>	-	:	DTMF 0 detected
<code>ERN = 49</code>	-	:	DTMF 1 detected
<code>ERN = 50</code>	-	:	DTMF 2 detected
<code>ERN = 51</code>	-	:	DTMF 3 detected
<code>ERN = 52</code>	-	:	DTMF 4 detected
<code>ERN = 53</code>	-	:	DTMF 5 detected

ERN =	54	-	:	DTMF 6 detected
ERN =	55	-	:	DTMF 7 detected
ERN =	56	-	:	DTMF 8 detected
ERN =	57	-	:	DTMF 9 detected
ERN =	64	-	:	DTMF A detected
ERN =	65	-	:	DTMF B detected
ERN =	66	-	:	DTMF C detected
ERN =	67	-	:	DTMF D detected

Example:

```
RECORD "VOICE"
```

Syntax:

```
RECORD [FILENAME]
```

See also:

```
ERN, HOOK, RESET MODEM, VOICE
```

The command `RECORD ON` is used during testing only. It sets the record mode, without receiving a file. Data from the encoder can be read from port 3 (`GET#3`, `LINK#3`, etc.).

The command `RECORD OFF` is used to turn off the record mode. This command must be issued when a recording is aborted by an event (`TIMEOUT`, `ESCAPE`, etc.).

Syntax:

```
RECORD ON|OFF
```

See also:

```
RESET MODEM, VOICE
```

RECORD TIMEOUT [num]**set record timeout**

An important variable used with the `RECORD` command is the Silent Timeout. With `RECORD TIMEOUT` you can change this. It is measured in 1/10th of a second. Changing S-register 66 (`SREG66 = (num)`) has the same effect.

The Silent Timeout determines when the recording is stopped, after a given period of silence. The Silent Timeout is started as soon as the Silent Period is detected. When no new speech is received, the Silent Timeout runs out and the recording is stopped.

Syntax:

```
RECORD TIMEOUT [num]
```

See also:

```
RECORD, RESET MODEM, VOICE
```

REM	remark
------------	---------------

REM provides the means to include comments within a program. All text following the REM command up to the end of the line is treated as a comment, and is not executed.

Examples:

```
10 REM Protocol Conversion program V2.0  
400 GOSUB %Answer : REM Execute Auto-answer routine
```

Syntax:

```
REM text...
```

This command will change the name of any file in the filing system. Using wildcards is not allowed.

Examples:

```
RENAME "TEST" , "MANUAL.TXT.ASC"
```

```
RENAME "TEST" TO "MANUAL.TXT.ASC"
```

Syntax:

```
RENAME (FILE|!) [string] TO|, (FILE|!) [string]
```

See also:

COPY

RENUM is used to renumber program lines.

Used on its own, RENUM renumbers the entire program using a default start line, and an increment of 10.

Using RENUM with the following program stored in memory:

```
10 CLEAR
15 INPUT "Enter password :";p$
20 IF p$<>"TULIP" THEN END
25 PRINT "Welcome to Argus..."
30 ....
```

would result in:

```
10 CLEAR
20 INPUT "Enter password :";p$
30 IF p$<>"TULIP" THEN END
40 PRINT "Welcome to Argus..."
50 ....
```

Line numbers and labels referenced in ELSE, THEN, GOTO, GOSUB, ON ... GOTO, ON ... GOSUB, ON ERROR, RESTORE and ERL commands will be renumbered correctly.

RENUM may also be used with two parameters which specify the start line and the increment to be used; e.g.:

```
RENUM 20, 2
```

Examples:

```
RENUM
```

```
RENUM 100
```

```
RENUM ,50
```

```
RENUM 20,20
```

Syntax:

```
RENUM ( [line-num] ) ( , [line-num] )
```

See also:

```
AUTO, LIST
```


REPEAT ... UNTIL provides a conditional loop construct in which all commands within the loop are executed at least once before a test condition is checked to determine whether or not the loop should be repeated. If the test condition evaluates to be TRUE, then the commands within the loop are repeated; otherwise execution continues at the next command following the UNTIL test.

A REPEAT command may span more than one program line.

The construct:

```
REPEAT : ... : UNTIL FALSE
```

is commonly used to carry out a series of commands indefinitely. This command is effectively the same as:

```
REPEAT : ... : UNTIL FALSE=TRUE
```

Obviously FALSE never will equal TRUE, and so the commands within the loop are executed repeatedly until the program is halted.

Examples:

```
REPEAT PRINT GET$; : UNTIL FALSE
```

```
REPEAT PUT#7, GET#1 : UNTIL KEY$#2=27
```

```
REPEAT UNTIL ILINE(3)
```

Syntax:

```
REPEAT .... UNTIL [test]
```

See also:

```
FOR..TO..NEXT..(STEP), WHILE...DO
```

REPORT

report result or error

REPORT is used to send a result code or error message to the specified port.

REPORT ERN

This command is used to report BASIC error messages.

Two parameters may be given; the port number(s) and the number corresponding to the error message required. If *no error number* is specified, the number of the last error encountered will be used, and the corresponding message given. If *no port* is specified, the default output port as defined by OPORT will be assumed.

REPORT or REPORT RESULT

This command is used to report modem result messages.

Two parameters may be given; the port number(s) and the number corresponding to the result messages required. If *no number* is specified, the number of the last result message will be used, and the corresponding message given. If *no port* is specified, the default output port as defined by OPORT will be assumed.

Examples:

```
REPORT
```

```
REPORT RESULT
```

```
REPORT#2 ERN 3
```

```
REPORT#1,#2 ERN Err
```

Syntax:

```
REPORT ( #[port] , ) ( #... ) (RESULT/ERN) ([integer 1..79])
```

See also:

```
ERL, ERN, ON ERROR, RESULT
```

RESET**generate hardware reset**

RESET is used to generate a hardware reset from within a program, the result of which is equivalent to pressing the RESET button on the front panel. It is generally used in conjunction with an ON RESET command to restart the program at a particular point.

Because there is a special chip in your modem (known as the 'watchdog & power monitor'), the RESET command will effectively reset all ICs in the modem.

Syntax:

```
RESET
```

See also:

```
ON RESET
```

RESET ALL SREG**reset all S-registers**

Used to reset all S-registers to factory settings (same as AT"&F").

Syntax:

```
RESET ALL SREG
```

See also:

```
SREG
```

This command is used to software reset the modem-chip only. If the modem-chip was in sleep-mode it will be waken up first. The command is used to switch between the different modes; modem, fax or voice. After `RESET MODEM 2` the modem is in modem-mode.

Sleep-mode brings the modem-chip in low power mode. This feature is default enabled. The commands `ANSWER`, `CONNECT`, `DIAL`, `FAX`, `RESET MODEM 2` and `VOICE` will wake up the modem-chip so it is ready for use. This takes about 500 ms. The `HANGUP` and `HOOK ON` commands will put the modem-chip back into sleep-mode.

Sleep-mode can be disabled by clearing bit6 in S-register 64 (`SREG64=SREG64 AND &BF`).

Syntax:

```
RESET MODEM 2
```

RESTORE is used in conjunction with the DATA and READ commands to reset the data pointer to a particular DATA command which is identified by its line number or label, and therefore allows data to be read in a non-sequential manner. The RESTORE command is followed by the line number or label of the line containing the requisite DATA command.

RESTORE BUFFER will set one of the user buffer pointers back to the beginning, so that the last data read may be read again. For example:

```
10 BUFFER 100
20 PRINT#8, "HELLO"
30 INPUT#8, A$
40 PRINT A$
50 RESTORE BUFFER
60 GOTO 30
```

The command RESTORE FILE or RESTORE! will recover a RAM-disk from CLEAR!. This can only be done if no other files have been written to the RAM-disk between the deletion and the restoring. The command is not valid in a Hard-disk filing system.

The command RESTORE ROM is used in an Argus Programmable Modem when a RAM chip is fitted in socket 2. If accidental erasure or corruption occurs, use this command before new data is stored in this chip.

Examples:

```
RESTORE 500
RESTORE %NameList
RESTORE!
RESTORE ROM
```

Syntax:

```
RESTORE [line-num| label]
RESTORE BUFFER
RESTORE FILE
RESTORE ROM
```

See also:

```
CLEAR, DATA, READ, ROM
```

RESULT is a function which holds the status of the last modem activity after a modem command.

RESULT	DESCRIPTION
0 OK	: Modem command was successful
1 CONNECT	: Connection at 300 baud (or fax connection)
2 RING	: A ring is being received
3 NO CARRIER	: Handshake with other modem or fax has failed
4 ERROR	: Error in executing a modem command
5 CONNECT 1200	: Connection at 1200 baud
6 NO DIALTONE	: No dial tone found before dialing out
7 BUSY	: Other modem or fax found busy after dialing out
8 NO ANSWER	: No answer after dialing out
9 CONNECT 75	: Connection at 75/1200 split baud
10 CONNECT 2400	: Connection at 2400 baud
11 CONNECT 4800	: Connection at 4800 baud
12 CONNECT 9600	: Connection at 9600 baud
13 CONNECT 14400	: Connection at 14400 baud
14 CONNECT 19200	: Connection at 14400 baud
15 CONNECT 600	: Connection at 600 baud
16 CONNECT 7200	: Connection at 7200 baud
17 CONNECT 12000	: Connection at 12000 baud
18 OFF-HOOK	: Modem is not connected to telephone system : or line is already in use
19 ESCAPE	: Escape detected in modem mode or 'continue' for : FAX command
20 VOICE	: Human voice (or noise) found after dialing out
26 CONNECT 1275	: Connection at 1200/75 split baud
32 CONNECT 16800	: Connection at 16800 baud
33 DATA	: Message to terminal
34 CONNECT 24000	: Connection at 24000 baud
35 FAX	: Message to terminal
36 CONNECT 28800	: Connection at 28800 baud
38 CONNECT 38400	: Message to terminal
39 CONNECT 57600	: Message to terminal
40 CONNECT 115200	: Message to terminal
41 CONNECT 21600	: Connection at 21600 baud
42 CONNECT 26400	: Connection at 26400 baud
43 VCON	: Voice Class 8 message
44 CONNECT 31200	: Connection at 31200 baud (V34plus only)
45 CONNECT 33600	: Connection at 33600 baud (V34plus only)

Example:

```
R=RESULT
```

```
IF RESULT GOTO Error
```

Syntax:

```
RESULT
```


RETURN

Return from procedure/set return character

The command RETURN is used for two purposes:

1. Return from procedure

(see GOSUB ... RETURN)

2. Set value of RETURN character.

The ASCII value of the return character as sent by the modem can be set with use of the command RETURN or the modem S-register S3.

Examples:

```
RETURN
```

```
RETURN=13
```

Syntax:

```
RETURN
```

```
RETURN = [num-var]
```

See also:

```
LINEFEED
```

RIGHT\$**extract right portion of string**

RIGHT\$ extracts the specified number of characters from the right-hand side of a string. The command:

```
R$=RIGHT$ ("ABCDEFGH" , 3)
```

will assign the string "FGH" to the string variable A\$.

The number of characters extracted must be between 0 and 255. Attempts to extract more characters than exist in the string will return the entire string.

Examples:

```
Lastname$=RIGHT$ (Name$ , 6)
```

```
Secs=RIGHT$ (CLOCK$ , 2)
```

```
PRINT RIGHT$ ("and so, the story ends." , 10)
```

Syntax:

```
[string-var] = RIGHT$ ( [string], [integer 0.255] )
```

See also:

```
LEFT$, LEN, MID$
```

The ring counter is increased with 1 on every ring that is received. The ring counter is the same as the modem S-register 1.

You can read or set the value of the ring counter, but of course, it is fairly meaningless to set the value, unless you have a specific reason for doing so.

Examples:

```
IF RING=8 THEN GOTO %Answer  
  
PRINT RING  
  
Beep=RING
```

Syntax:

```
[num-var] = RING
```

See also:

```
ON RING
```

The `ROM` command is used to access the RAM/ROM-disk in socket 2 of an Argus Programmable Modem with RAM-disk filing system.

Socket Organization

There are 2 sockets to put a RAM chip into

Socket 1 is the one nearest the 32k system RAM.

Socket 2 is the one with the jumpers above it.

If you want to have a RAM-disk filing system, there must always be a RAM chip in socket 1. A RAM chip in socket 2 is optional and the jumpers must be set to RAM in this case.

The RAM chips used can be any combination of a 32k, 128k, 256k or 512k chip. So, the smallest possible RAM disk is 32k (a 32k chip in socket 1) and the largest possible RAM disk is 1Mb (2 x 512k chips in sockets 1 and 2).

If a ROM is put into socket 1, then the filing system can access it only with `LOAD` and `READ` commands (`LOAD`, `RUN`, `SEND`, `MATCH`, `DIR`). This is handy for putting in a BASIC application program, which requires no further RAM disk.

If a ROM is put into socket 2 and the jumpers are set correctly, the filing system can access it only with `LOAD` and `READ` commands. This is handy for putting in a BASIC application program, which requires a RAM-disk in socket 1.

The ROM commands:

The word 'ROM' can be used in all existing filing system commands:

<code>SEND ROM</code>	<code>RECEIVE ROM</code>
<code>LOAD ROM</code>	<code>SAVE ROM</code>
<code>DIR ROM</code>	<code>COPY ROM</code>
<code>MATCH ROM</code>	<code>RENAME ROM</code>
<code>RUN ROM</code>	<code>CLEAR ROM</code>
	<code>RESTORE ROM</code>

The commands in the **first** column can be used if a **ROM** chip is in **socket 2**.

The commands in **both** columns can be used if a **RAM** chip is in **socket 2**.

To initialize a RAM chip in socket 2, the command `CLEAR ROM` must be entered first. After that all other commands can be used.

If there is a RAM chip in socket 2, there are two (2) RAM disk filing systems at that time.

The first will access the RAM in socket 1, and

The second will access the RAM in socket 2.

The first filing system uses all the normal commands, without the word ROM in it.

The second filing system uses these commands as listed above. Normally, the second filing system is only used for developing software for a ROM chip, which can be used later. Or the second filing system is never used at all, in which case all the RAM can be used by the first filing system.

The first filing system uses both RAMS in socket 1 and 2. So, the number of blocks free, shown with `DIR`, is for both sockets. If you are using the second filing system in socket 2, you must be careful that the RAM in socket 1 doesn't become full and overwrite data from the first filing system in socket 2.

All commands work exactly the same as the command without 'ROM' in it, except the command `LOAD ROM`. `LOAD ROM` will look first in socket 1 and if it can't find the file there it will look in socket 2, when there is a ROM in it. Software can be developed then in such a way that the program in ROM can be overwritten or replaced by a program in RAM.

See also:

`CLEAR`, `COPY`, `DIR`, `LOAD`, `MATCH`, `RECEIVE`, `RENAME`, `RESTORE`, `RUN`, `SAVE`, `SEND`

RPT\$ is used to produce a string containing multiple repetitions of another string. The result of the following example will be to set X\$ to "*****":

```
X$=RPT$ ("*", 10)
```

Similarly,

```
ab$=RPT$ ("ab", 5)
```

will set ab\$ to "ababababab".

The resulting string must not be longer than 255 characters.

Examples:

```
Line$=RPT$ ("-", 80)
```

```
H$=RPT$ ("H", 20)  
PRINT RPT$ (">", 15)
```

Syntax:

```
[string-var] = RPT$ ( [string] , [integer 0..255] )
```

See also:

SPC\$

RTRIM\$

remove trailing spaces

RTRIM\$ is used to remove all trailing spaces of a string.

Examples:

```
PRINT RTRIM$ "      Test      "  
Adjusted$ = RTRIM$ Input$
```

Syntax:

```
[string-var] = RTRIM$ [string]
```

See also:

LTRIM\$, TRIM\$

RTS is one of the hardware handshaking signals defined in the RS232 standard, and the two serial ports of the modem normally respond to the RTS input by stopping the outgoing data stream.

The RTS signal is used in conjunction with CTS (Clear to Send) as part of the RS232 handshaking sequence which prevents loss of data when a receiving piece of equipment cannot handle data from the sender at full speed.

The `RTS` function is used to read the status of RTS on a specified port. If no port is specified when reading `RTS`, the default port as defined by `IPORT` will be assumed. If no port is specified when setting `RTS`, the default port as defined by `OPORT` will be assumed.

Automatic control of RTS by the modem can be enabled using the command `HANDSHAKE RTS`. This will take the RTS line low when there are only 5 free bytes remaining in the input buffer. It will only be set high when 50% of the input buffer is free.

If hardware handshaking using RTS/CTS is not available for some reason, software handshaking using XON/XOFF can be enabled using the command `HANDSHAKE XOFF`.

If the modem is to ignore the RTS signal, then use the command `RTS OFF`.

Examples:

```
Req=RTS
```

```
Hand=RTS#2
```

```
RTS#1 ON|OFF
```

Syntax:

```
[num-var] = RTS ( #[port] )
```

```
RTS ( #[port] ) ( , #... ) ON|OFF
```

See also:

```
CTS, HANDSHAKE, XOFF
```


RUN is used to initiate the execution of a program.

There are three forms of the command, the first of which is used to execute the program currently residing in memory at PBOT. In this case no parameter is required, and before execution commences all variables are cleared, and the data pointer restored to the first DATA command.

The second form allows programs to be executed from a specified line number or label, in which case variables are *not* cleared. This is particularly useful when RUN is used in conjunction with STOP for debugging programs; i.e. a program can be stopped to allow the current value of variables to be examined, and then restarted by typing RUN with the line number of the line following the STOP command.

Finally, RUN can be used to execute programs stored in the filing system. For example:

```
RUN "MAILBOX.PRG"
```

will cause the program stored on disk to be copied on PBOT and executed. Variables are *not* cleared.

Examples:

```
RUN
```

```
RUN 50
```

```
RUN Menu
```

```
RUN "START.PRG"
```

```
RUN Name$+" .BASIC"
```

Syntax:

```
RUN ( [line-num] | [label] )
```

```
RUN [string]
```

See also:

```
PBOT
```

RUN ROM**run a program in ROM/RAM**

This command is used to run a program stored in a ROM/RAM in socket 2, in an Argus Programmable Modem.

Example:

```
RUN ROM "MYPROG.BAS"
```

Syntax:

```
RUN ROM [FILENAME]
```

See also:

ROM, RUN

SAVE**save program or data to memory**

SAVE is used to store BASIC programs on the disk. The command must be followed by the filename.

It is recommended that you use extensions like .PRG or .BAS, to keep your programs separate from data files, faxes or messages. You can then use something like DIR "*" .PRG" to list all your BASIC programs.

SAVE is also used to save a block of memory to the disk. You will have to specify the start address and end address of the memory block to be saved, as well as the filename.

Examples:

```
SAVE Name$+" .PROG"
```

```
SAVE "PROG1.BASIC"
```

```
SAVE &7000, &7FFF, "EDI12.DATA"
```

Syntax:

```
SAVE (FILE|!) [string]
```

```
SAVE (FILE|!), [num], [num], [string]
```

See also:

```
DELETE, DIR, LOAD, RESTORE
```

SAVE ROM**save files in RAM disk in socket 2**

This command is used to save files to a RAM-disk in socket 2 in an Argus Programmable Modem. To initialize a RAM chip in socket 2, the command `CLEAR ROM` must be entered first.

Syntax:

```
SAVE ROM [FILENAME]
```

See also:

ROM, SAVE

SCAN**set answering speed/scan**

In the programming environment, the *SCAN* command is used to configure the answering procedure of the modem. The command does the same as setting S-register 53:

```
0 = Speed is determined by S-register 51
1 = Scan: V22bis >> V23 >> V21 >> V23 (orig)
2 = Scan: V22bis >> V23 >> V21
3 = Scan: V23 >> V21 >> V22bis >> V34/V32bis
4 = Scan: V23 >> V22bis >> V34/V32bis
5 = Scan: V21 >> V22bis >> V34/V32bis
6 = Auto-mode V8 (default)
7 = Scan: V22bis >> V34/V32bis
```

Examples:

```
SCAN 6
```

```
SCAN OFF (=0)
```

Syntax:

```
SCAN [integer]
```

SEC is used to read or set the seconds value on the internal clock/calendar.

When *reading* it returns an integer from 0 to 59.

Similarly, when *setting*, a value from 0 to 59 must be specified.

An incorrect value will leave the current setting unchanged.

Examples:

```
SEC=40  
  
PRINT SEC
```

Syntax:

```
SEC = [integer 0..99]  
  
[num-var] = SEC
```

See also:

CLOCK\$, HOUR, MIN, TIMES\$

SEND**send file or data**

`SEND` is one of the most powerful commands available for use by the modem, and provides a simple means of transferring data from the modem under program control, with the minimum of effort.

There are 3 basic forms of the command:

`SEND [port] , [address]` : send *data* using Xmodem

`SEND [port] , [filename] , [type] , [option]` : send *file*

`SEND ALL` : send a backup of the disk

In addition, the second form can be used with one of a number of file type specifiers and options, which are dealt with individually in the following sections.

SEND #[port], [address], [address]**send data**

This form of the `SEND` command is used to transfer data from the modem's memory to other systems. The Xmodem error correction protocol is used to prevent errors occurring during the transfer, so the facility cannot be used if the destination system does not support Xmodem.

A port number may be specified (1 to 3 only), but if this is omitted the default port as defined by `OPORT` will be assumed. Following this, two integer parameters are used to specify the start and end addresses for the data to be transmitted. If these are omitted it is assumed that you wish to send the currently loaded BASIC program; i.e. the area of memory starting from `PBOT` and extending to `PTOP`.

During the `SEND`, `ESCAPE` and `XOFF` are disabled for the specified port, and the character length is set to 8 bits. On completion the original settings are restored.

Error number 27 will be generated if no data is transferred for a period of 180 seconds, or when more than nine sequential block errors have occurred.

A common use for this form of the `SEND` command is to upload the current BASIC program in system memory into a PC to use as a backup copy.

Examples:

```
SEND
```

```
SEND#2
```

```
SEND &8000, &C000
```

```
SEND#Port, Addr1, Addr2
```

Syntax:

```
SEND ( #[port], ) ( [integer], [integer] )
```

See also:

```
RECEIVE
```


[With filing system software only.]

When followed by an exclamation mark and a filename, the `SEND` command is used to transmit complete files from the disk filing system. Wildcards may be used in the filename, so that a single `SEND` command may be used to transfer more than one file:

```
SEND "*.TXT"  
SEND "MESSAGE*. *"
```

The system variable `MATCH` can be read afterwards to check how many file there were send.

If a file with the specified filename does not exist, the message "Not found" will be given (system variable `ERN` is set to 73). You may of course use `MATCH FILE` to check for the existance of the file before using `SEND`.

The following types of transfer may be initiated:

```
SEND#[port],[filename],B : read file information block only  
SEND#[port],[filename],C : make copy of file using Xmodem protocol  
SEND#[port],[filename],F : binary output  
SEND#[port],[filename],N : text output with NNNN protocol  
SEND#[port],[filename],P : read part of a file  
SEND#[port],[filename],T : text output  
SEND#[port],[filename],X : use Xmodem or Ymodem protocol
```

If no port number is specified, the default output port as defined by `OPORT` will be assumed. More than one output port may be specified for transfers other than those using Xmodem or Ymodem..

Data may be copied into the `BUFFER` by specifying port 8. In this case you must ensure that the buffer is large enough to accommodate the file. Once in the buffer, the file contents may be manipulated using commands such as `INPUT#8`, `KEY#8`, `PUT#8`, etc. `KEY$#8` is particularly useful for reading the file as it simplifies end-of-file detection:

```
IF KEY$#8="" THEN Eof=TRUE
```

If wildcards are used to specify multiple files, each will be separated from the next by four `[CR]` characters, and the `MATCH` command may be used on completion to show the number of files transferred.

Options

Two further options may be used in conjunction with some of the above types of transfer. These are:

```
SEND ... , [type] , U      : send Unread files only
SEND ... , [type] , M      : send with 'More?' prompt (text only)
```

In most cases options may be used in conjunction with each other; i.e Unread files may be sent with the `M` option.

SEND [filename]

`SEND` without a type or an option will use the text output and is therefor the same as `SEND , T`. As `SEND` uses the `CTRL-Z` character (ASCII 26) to determine the end of file, a binary file containing a `CTRL-Z` will not transmit correctly, and should be sent using the `F` or `X` type specifier.

Both the `U` and `M` options may be used.

SEND [filename],B

`SEND , B` is used to copy the 32-byte information block, available in each file, into memory locations `&2BE0` to `&2BFF`. This block may then be manipulated using `PEEK` and `POKE`. It can be written back to the file with the `RECEIVE , B` command. Wildcards are allowed, but only the first encountered file is taken.

The `U` option can be specified.

SEND [filename],C

`SEND , C` takes one or more files and sends it as one file with Xmodem-CRC (`PROTOCOL 1` is set automatically). The contents of that file is unique to an Argus filing system; for each file it will hold the name, type, date & time, length and information block. It is used to create backups and to transfer files from one Argus system to another as an exact copy. For example, to transfer a copy of all the program files:

```
SEND "*" .PRG" , C
```

SEND [filename],F

`SEND , F` can be used to transfer any type of file as a stream of binary data, i.e. no formatting is carried out, and no error correction is provided. It is recommended that this type of transfer is used for local purposes on a serial port or, if connected as a modem, in conjunction with MNP or V42 error correction.

The `U` option can be specified.

SEND [filename],N

This is similar in operation to `SEND , T` but there are differences in the way multiple files are handled when using wildcards in the filename. Individual files will be separated by `[CR] NNNN [CR]`, and the last file in sequence will be terminated with `[CR] NNNNE [CR]`.

Both the `M` and `U` options may be used.

SEND [filename] ([,type]) ,P,<var=part> ([,option])

This `SEND` command allows you to send a part of a file. Each part contains between 1 and 2048 bytes, depending on the filing system (RAM/FLASH-disk or Hard-disk/LAN).

The number of bytes in the requested part is returned in the system variable `MATCH`. When the last part is sent, the system variable `ERN` is set to 92 ("End Of File"), else `ERN` will be 0 ("OK").

`ERN` can also be set to 73 ("Not found"), when the file is not found in the filing system. A none existing part will cause `ERN` 91 ("Bad part").

A part can be sent with the `T` (Text) or `F` (Binary) type specifier, if not specified then Text is taken as the default. Also the `U` and `M` options may be used.

Example program:

```
10 Num=0
20 REPEAT
30   Num=Num+1
40   SEND "file",T,P,Num
50 UNTIL ERN
```

To sort out a binary file:

```
10 BUFFER &1000
20 Numb=0
30 REPEAT
40   Numb=Numb+1
50   SEND#8,"file",F,P,Numb
60   Part=MATCH : Last=ERN
70   FOR Coun=1 TO Part
80     Byte=KEY#8
90     REM do some stuff here with 'Byte'
100  NEXT
110 UNTIL Last
```

SEND [filename],T

`SEND, T` is used to send a file as text to a port. A [Ctrl-Z] character (ASCII 26) in the file will be treated as an end-of-file character, and will terminate transmission. Both the `U` and `M` options may be used.

The format of the output to the specified port(s) depends on the 5 parameters set with the `DUTCH` or `ENGLISH` command:

1. Characters per line:

- 0 = Word Warp disabled
- 1-255 = Word Wrap enabled for 1 till 255 characters per line

Word Warp means that a whole word is started on a new line if it doesn't fit on the current line.

A new line is always generated after a character with the value equal to S-register 3 which defaults to control-character 13 (=Carriage Return). After each new line, a linefeed character is sent with the value equal to S-register 4 which defaults to control-character 10 (=LineFeed). Automatic linefeeds can be enabled or disabled with `LINEFEED ON/OFF`.

2. Lines per page:

The maximum number of lines per page can be set with this parameter. This is only of importance if the file is sent with the `M` option or if 'page fillout' is enabled.

3. Left margin:

Enabling the left margin will add some spaces in front of a line, if that line is less than the 'characters per line' parameter.

4. Page fillout:

Enabling the page fillout will fill the page with empty lines on the bottom until the 'lines per page' is reached. This is done for each page if the file is sent with the `M` option and always at the end of the text.

5. FormFeed:

If a file is sent with the `M` option, a control-character 12 (=FormFeed) is sent after the 'lines per page' is reached.

SEND [filename],X

`SEND, X` can be used to send any type of file using Xmodem or Ymodem error correction. The `PROTOCOL` command must be set first to select the transfer protocol.

As Xmodem breaks files down into 128- or 1024-byte blocks, the last block will be automatically padded with NULs (zero bytes) to the correct length.

The Ymodem-Batch protocol does not include the date & time of the files, only filenames and length.

SEND ROM [filename], [type], [options]

`SEND ROM` is used to send one or more file on the ROM or RAM chip in socket 2 of an Argus Programmable Modem. See `ROM` for further details.

The type and options are used in the same manner as with the normal `SEND` command.

Examples:

```
SEND "TELEX.TXT"

SEND "*.TXT"

SEND "*" + VAL$(Nr) + ".CCI"

SEND#3, Name$ + CHR$(Nr) + ".VIEW"

SEND "LETTER*.TXT", X

SEND#1, "*.*", U

SEND#7, Name$, U

SEND "2000a.VIEW", B
```

Syntax:

SEND (#[port] ,) (#...) [filename] (, [type]) (, [option])

SEND ROM ...

See also:

MATCH, RECEIVE

SEND ALL**make a backup of RAM disk**

The `SEND ALL` command is normally used to make a backup of the RAM-disk. It sends the complete contents of the RAM chip as one file. This file can be used later to restore a RAM-disk with `RECEIVE ALL` or the file can be used directly to fill an EEPROM chip that can be put into socket 2 of an Argus Programmable Modem.

In a RAM-disk filing system this command differs from the command `SEND , C` which creates file copies.

In a Hard-disk filing system, this command is exactly the same as `SEND " *.*.*.*.*.*.*" , C`.

The Xmodem-CRC protocol is used for this purpose; `PROTOCOL 1` is set automatically.

Example:

```
SEND ALL
```

Syntax:

```
SEND (ROM) ALL
```

See also:

```
RECEIVE ALL, ROM
```

The escape sequence is part of the 'Hayes compatible' aspect of modem operation.

It is explained in more detail in the *Modem User Guide*, and also with the sequence event handler `ON SEQUENCE`.

The possibility of an escape sequence occurring can be enabled with `SEQUENCE ON`; disabled with `SEQUENCE OFF`.

The escape sequence character can be set with `SEQUENCE [num]`. This is the same as setting S-register 2 (`SREG2= [num]`).

Examples:

```
SEQUENCE ON
```

```
SEQUENCE 37
```

Syntax:

```
SEQUENCE ON/OFF
```

```
SEQUENCE [num]
```

See also:

```
ON SEQUENCE, ESCAPE
```

SGN is used to determine the sign of a number and returns three possible values, depending on whether the test value is negative, zero or positive.

The following apply:

SGN 56 returns 1

SGN 0 returns 0

SGN -45 returns -1

Examples:

Dif=SGN(X1-X2)

X=SGN(y)

Syntax:

[num-var] = SGN [integer]

See also:

ABS

`SOUND` has two optional parameters *frequency* and *duration*, set in hundredths of a second. If the parameters are left out, then 2000 Hz and 250 ms are taken by default.

The command `SOUND`, without `MODEM` behind, will produce a sound on the speaker only, by toggling the speaker-mute-bit (see `VOICE`) with the desired frequency.

The command `SOUND MODEM` will produce a sound through the modem-chip and can therefor be hearded on the telephone line. The modem-chip must not be in sleep-mode (see `RESET MODEM 2`). After `SOUND MODEM`, the modem-chip is always put back in voice-mode, except if a duration of zero (0 ms) is defined, in which case the modem-chip is left in sound-mode and immediatly returns.

Examples:

```
SOUND 2000,25
```

```
SOUND MODEM 1250,100
```

Syntax:

```
SOUND (MODEM) ( [num] ) ( , [num] )
```

See also:

```
RESET MODEM 2
```

SPC\$**repeat spaces**

The `SPC$` string function produces a string containing the specified number of ASCII Space characters.

Examples:

```
Title$="Name"=SPC$(10)+"Age"+"SPC$(5)+"Sex"
```

```
PRINT "Port number"+SPC$(20)+VAL$(port)
```

Syntax:

```
[string-var]=SPC$[integer 0..255]
```

See also:

`RPT$`

The `SPEAKER` command is used to enable or disable the modem's internal loudspeaker. Same as `AT "M2/0"`.

If a headset is connected to the Argus, the `SPEAKER OFF` command disables the headset as well and is therefore not suitable to control the speaker while the headset is active. In that case the speaker-mute-bit must be set or cleared with `POKE` (see `VOICE`).

Examples:

```
SPEAKER ON
```

```
SPEAKER OFF
```

Syntax:

```
SPEAKER ON|OFF
```

See also:

```
VOICE, VOLUME
```

SPEED [number]**line speed select**

This command is used to set S-register 51 within the programming environment. Same as AT "S51=x" or SREG51=[num]. SPEED is used when connecting as a modem or fax.

After connection, S-register 50 (bits0-5) is set to the connected speed as listed below:

1 = 1200 (V22)	38 = 28800 (Vfast)
2 = 2400 (V22bis)	39 = 26400 (Vfast)
3 = 300 (V21)	40 = 24000 (Vfast)
4 = 4800 (V32)	41 = 21600 (Vfast)
5 = 4800 (V27ter) Fax	42 = 19200 (Vfast)
6 = 2400 (V27ter) Fax	43 = 16800 (Vfast)
7 = 7200 (V29) Fax	44 = 14400 (Vfast)
8 = 2400 (V26bis) Synchronous	45 = Vfast all speeds
9 = 9600 (V32 TCM)	46 = 4800 (Bell 208)
10 = 9600 (V32 U)	47 = 1200 (Bell 212A)
11 = 9600 (V29) Fax	48 = 300 (Bell 103)
12 = 12000 (V32bis)	49 = 33600 (V34plus)
13 = 12000 (V33) Half Duplex	50 = 31200 (V34plus)
14 = 14400 (V32bis)	51 = 28800 (V34)
15 = 14400 (V33) Half Duplex	52 = 26400 (V34)
16 = 600 (V22)	53 = 24000 (V34)
17 = 1200/75 (V23 Answ)	54 = 21600 (V34)
18 = 4800 (V29) Half Duplex	55 = 19200 (V34)
19 = 7200 (V32bis)	56 = 16800 (V34)
20 = V8 Auto-mode (all speeds)	57 = 14400 (V34)
21 = V32bis Auto-mode	58 = 12000 (V34)
22 = V8 High speed only	59 = 9600 (V34)
23 = 75/1200 (V23 Orig)	60 = 7200 (V34)
25 = 9600 (V33) Half Duplex	61 = 4800 (V34)
26 = 4800 (V33) Half Duplex	62 = 2400 (V34)
27 = 300 (V21 channel 2) Fax	63 = V34(plus) all speeds
28 = 14400 (V17) Fax	
29 = 12000 (V17) Fax	
30 = 9600 (V17) Fax	
31 = 7200 (V17) Fax	

Example:

```
SPEED 10
```

```
SPEED Line
```

Syntax:

```
SPEED [number]
```

See also:

```
FAX, ON CONNECT, SCAN
```

The command is another way to set S-register 52 within the programming environment. Same as AT "S52=1/0" or SREG52=1/0.

When speed buffering is enabled (default), there is no relationship between the modem-speed on the line and the speed at the serial ports of the modem. The ports will only change their speed when AT is entered at a new baud rate or after using the BAUD command.

When speed buffering is disabled, the speed at the port will automatically change to the speed on the line after the 'CONNECT' message has been sent.

Example:

```
SPEED BUFFER ON
```

Syntax:

```
SPEED BUFFER ON|OFF
```

See also:

```
SCAN
```

SREG is used as another way to set or read the 80 available S-registers that the modem keeps in memory. These registers are holding various types of parameters which are used by the modem operating system for all sorts of purposes. Most of them are mentioned throughout this manual. For further information, see the *Modem User Guide*.

This way of manipulating the S-registers is very useful within the programming environment, because it then enables you to treat these registers as system variables.

The command `RESET ALL SREG` will set all the registers back to factory settings.

Examples:

```
SREG0=1  
  
PRINT SREG3  
  
IF RING AND NOT SREG0 THEN SOUND 2000,200
```

Syntax:

```
SREG[integer0...79] = [num]  
  
[num-var] = SREG
```

See also:

```
AT [string], RESET ALL SREG
```

STOP**halt program execution**

`STOP` terminates program execution, and is most often used during program debugging to temporarily halt a program so that variables can be examined.

Execution may be restarted by entering a `GOTO` command for the line following the line containing the `STOP` command.

A special command to control the `STOP` button is:

```
STOP ON/OFF
```

With `STOP ON`, the button will function normally. With `STOP OFF`, the button will only work when the S/A-button is in the in-position. This prevents that BASIC programs are stopped if the `STOP` button is pressed by accident.

Examples:

```
IF x>500 THEN STOP
```

```
ON ERROR STOP
```

```
STOP OFF
```

Syntax:

```
STOP
```

```
STOP ON/OFF
```

See also:

```
RUN
```

TIME\$**read or set time**

TIME\$ is used to read the current time as a string or to set the clock.

The string has the format *hh:mm:ss* (hour, minutes, seconds) for input as well as for output.

Examples:

```
TIME$ = "14:20:00"
```

```
Tstamp$ = TIME$
```

Syntax:

```
TIME$ = [string]
```

```
[string-var] = TIME$
```

See also:

```
DATE$
```


TRACE**trace program execution**

TRACE provides a debugging aid which monitors program execution by printing the number of each line as it is executed. Line numbers are printed in square brackets to the port requested, or the default output port as defined by OPORT.

Events and errors are printed as text between the line numbers.

For example:

```
10 TRACE ON
20 READ x
30 FOR i=1 TO x
40     PRINT i
50 NEXT
60 DATA 3
```

Output will be:

```
[20] [60] [30] [40] [50] [30] [40] [50] [30] [40] [50]
```

Examples:

```
TRACE#7
TRACE OFF
```

Syntax:

```
TRACE ( #[port] , ) (...) ([ON | OFF])
```

TRIM\$ is used to remove all spaces from a string.

Examples:

```
PRINT TRIM$ "      Test      "
```

```
Adjusted$ = TRIM$ Input$
```

Syntax:

```
[string-var] = TRIM$ [string]
```

See also:

```
LTRIM$, RTRIM$
```

UCASE**convert ASCII to uppercase**

UCASE is used to calculate the upper case ASCII value of a character.

Examples:

```
PRINT UCASE 97
```

```
B=UCASE98
```

Syntax:

```
[num-var] = UCASE [num]
```

See also:

```
LCASE$, LCASE, UCASE$
```

UCASE\$**convert string to uppercase**

UCASE\$ is used to convert all characters in a string to upper case.

Characters that are already in upper case remain unchanged.

This command (or LCASE\$) is frequently needed to standardize keyboard input.

Examples:

```
PRINT UCASE$ "QwErTy"  
result$ = UCASE$ input$
```

Syntax:

```
[string-var] = UCASE$ [string]
```

See also:

LCASE, LCASE\$, UCASE\$

VAL**convert string to number**

VAL is used to convert a string expression representing a numeric value into the actual value it represents.

The string may comprise any expression resulting in a numeric value, providing that this does not exceed 32767 - the largest integer value that may be used on the modem.

In the following example the integer variable Code is set to 345:

```
Code=VAL ("345")
```

Examples:

```
F=VAL (W1$+W2)
```

```
Nr(i)=VAL (Code$(i))
```

```
PRINT VAL Tel$
```

Syntax:

```
[num-var] = VAL [string]
```

See also:

```
VAL$
```

VAL\$ is used to convert a numeric value or expression into its string form. This is the reverse of the VAL function described earlier.

In the following example the string variable Code\$ is set to 276:

```
code$=VAL$(300-24)
```

The tilde symbol '~' may be used to convert numbers to hexadecimal format before being converted into strings.

Examples:

```
Title$="Number "+VAL$(Codenum)
```

```
PRINT VAL$~(D1+D2)
```

Syntax:

```
[string-var] = VAL$ (~) [integer]
```

See also:

VAL

VOICE**set modem to receive voice**

The VOICE command puts the modem-chip in voice-mode and prepares it for playback or recording. If the modem-chip was in sleep-mode (low power), it is first waken up (see RESET MODEM 2). After VOICE, the commands PLAY, RECORD or SOUND MODEM can be used.

Controlling the microphone, Automatic Gain Control (AGC) and other features is done with the PEEK and POKE commands. The related addresses and their actions are listed below (Argus Programmable Modem and T.C.Lite):

POKE &49B,2 : POKE &311,5 : POKE &311,&6A	: Microphone local on (when on-hook).
POKE &49B,2 : POKE &381,5 : POKE &381,&6A	
POKE &49B,0 : POKE &311,5 : POKE &311,&68	: Microphone local off.
POKE &49B,0 : POKE &381,5 : POKE &381,&68	
POKE &49C,&C0 : POKE &180,&C0	: Microphone to line on (when off-hook).
POKE &49C,&40 : POKE &100,&40	
POKE &49C,0 : POKE &180,0	: Microphone to line off.
POKE &49C,0 : POKE &100,0	
1.POKE &49B,2 : POKE &311,5 : POKE &311,&6A	: Automatic Gain Control on. This is only
2.POKE &FE,PEEK&FE OR &40	available when recording directly from the line
3.POKE &104,PEEK&104 OR &80	(microphone must be off).
1.POKE &49B,2 : POKE &381,5 : POKE &381,&6A	
2.POKE &8,PEEK&8 OR &10	
3.POKE &304,PEEK&304 OR &80	
1.POKE &49B,0 : POKE &311,5 : POKE &311,&68	: Automatic Gain Control off.
2.POKE &FE,PEEK&FE AND &BF	
3.POKE &104,PEEK&104 AND &7F	
1.POKE &49B,0 : POKE &381,5 : POKE &381,&68	
2.POKE &8,PEEK&8 AND &EF	
3.POKE &304,PEEK&304 AND &7F	
POKE &FE,PEEK&FE AND &FB	: Speaker enabled.
POKE &8, PEEK&8 AND &DF	
POKE &FE,PEEK&FE OR &04	: Speaker disabled.
POKE &8, PEEK&8 OR &20	
POKE &101,PEEK&101 AND &3F	: Speaker & Headset mute.
POKE &301,PEEK&301 AND &3F	
POKE &101, (PEEK&101 AND &3F) OR &C0	: Speaker & Headset Volume 1.
POKE &301, (PEEK&301 AND &3F) OR &C0	
POKE &101, (PEEK&101 AND &3F) OR &40	: Speaker & Headset Volume 2.
POKE &301, (PEEK&301 AND &3F) OR &40	
POKE &101, (PEEK&101 AND &3F) OR &80	: Speaker & Headset Volume 3.
POKE &301, (PEEK&301 AND &3F) OR &80	
POKE &FC,PEEK&FC OR &1	: Off-hook relay active (or use OLINE0 ON or HOOK OFF).
POKE &0,PEEK&0 OR &40	
POKE &FC,PEEK&FC AND &FE	: Off-hook relay non-active (or use OLINE0 OFF or HOOK ON/HANGUP).
POKE &0,PEEK&0 AND &BF	
POKE &0,PEEK&0 OR &40	: Shorten-line relay active.
POKE &0,PEEK&0 AND &BF	: Shorten-line relay non-active.
POKE &49A,2 : POKE &310,5 : POKE &310,&6A	: Earth-pulse relay active.
POKE &49A,0 : POKE &310,5 : POKE &310,&68	: Earth-pulse relay non-active.
POKE &49A,&80 : POKE &310,5 : POKE &310,&EA	: High-impedance relay active.
POKE &49A,0 : POKE &310,5 : POKE &310,&68	: High-impedance relay non-active.

The last 3 relays are not available in a T.C.Lite. The shorten-line relay is used during pulse dialing and also when a hook-flash is generated. The earth-pulse relay is only used during the earth-pulse connect-through method. The high-impedance relay is used when the modem must 'listen' to the line, without taking it (modem must be on-hook).

Syntax:

VOICE

See also:

PLAY, RECORD, RESET MODEM 2, SPEAKER, VOLUME

VOLUME**set loudspeaker volume**

This command is used to set the volume level of the modem's internal loudspeaker and the volume in the headset at the same time. The levels 1, 2 or 3 can be set only.

The `VOLUME` command itself never has immediate effect. It must always be followed by `SPEAKER ON`. Controlling the volume of the headset directly must be done with a `POKE` command (see `VOICE`).

Example:

```
VOLUME 3
```

Syntax:

```
VOLUME (=) [integer 1, 2, 3]
```

See also:

```
SPEAKER, VOICE
```

WAIT**pause program execution**

WAIT is used to introduce a delay in program execution. A single parameter is used to specify the length of the delay in one hundredths of a second. For example:

```
WAIT 120
```

will pause program execution for 1.2 seconds before execution resumes normally at the command following WAIT.

Alternatively the delay may be specified in seconds or minutes. Then the delay given has to be followed by SEC or MIN.

Note:

A maximum delay of 32767 (% of a second, seconds, or minutes) may be specified.

WAIT commands do not affect the operation of any ON function, or LINK.

The second way to use WAIT is to wait for an "AT" or "A/". Here, you should enter the command:

```
WAIT AT SLASH
```

and the program will stop and scan for an AT or A/ on the input port. If ECHO is set to ON the command will respond with AT or A/ after the proper baud rate is set.

To determine whether an AT or an A/ was received, you can read the system variable MATCH. MATCH returns 1 if it was AT, and 0 if it was A/.

Examples:

```
WAIT 10
```

```
WAIT Pause SEC
```

```
WAIT FOR 1 MIN
```

```
WAIT 2 * I
```

```
WAIT FOR AT SLASH#1
```

Syntax:

```
WAIT (FOR) [integer] (SEC|MIN)
```

```
WAIT (FOR) AT (SLASH) ([port]) (#...)
```

The `WHILE ... DO ... WEND` construct is a conditional loop construct which allows the conditional execution of one or more program lines according to the result of a test carried out at the beginning of the loop. 16 loops can be nested.

If the result of the test condition is `FALSE`, the code within the loop (i.e. between the test condition and the `WEND`), is skipped, and execution proceeds with the command immediately following the `WEND`.

If the result of the test condition is `TRUE`, all commands within the loop are executed before the test is re-applied. This process is repeated until the test condition becomes `FALSE`.

For example:

```
10 A=1
20 WHILE A<17 DO
30   PRINT A
40   A=A*2
50 WEND
```

will cause lines 30 and 40 to be repeated until `A` exceeds 32, producing the output:

```
1
2
4
8
16
```

before exiting the loop when `A` is set to 64.

Examples:

```
WHILE code<100 DO PRINT code*4/3: READ code
WHILE KEY$<>"Y" AND KEY$ <> "N" DO
WHILE TRUE DO PUT GET: WEND
```

Syntax:

```
WHILE [test] DO ... WEND
```

See also:

```
FOR ... TO ... NEXT, REPEAT .. UNTIL
```

The `XOFF` command is used in a number of ways to control software handshaking during data transfer.

There are 4 variations:

XOFF#[port]

When used on its own, `XOFF` halts data transmission from the specified serial port. This provides a way to control the output data stream which otherwise can only be controlled via the CTS line. `XON` is used to re-enable transmission.

XOFF#[port], OFF

This is used to disable software handshaking on the specified port.

XOFF ON

This is used to enable software handshaking on the specified port. Transmission of data will be halted when an `XOFF` character is received, and will not start again until an `XON` has been received.

XOFF [integer 0...255]

The default value for the `XOFF` character is ASCII 19 ([Ctrl-S]). This form of the `XOFF` command is used to redefine this value so that an alternative character may be used for software flow control. It is the same as setting S-register 21 (`SREG21 = [num]`).

Note that after pressing the STOP button, the `XOFF` character for each port will be set to 19 ([Ctrl-S]) and `XON` to 17 ([Ctrl-Q]). All ports are also set to `XOFF ON`.

Examples:

```
XOFF
```

```
XOFF#1, ON
```

```
XOFF#2, #3, #4, OFF
```

```
XOFF=19
```

Syntax:

```
XOFF ( #[port] , ) ( #... )
```

```
XOFF ( #[port] , ) ( #... ) ON/OFF
```

```
XOFF = [integer 0..255]
```

See also:

```
HANDSHAKE, XON
```

The `XON` command is used in a number of ways to control software handshaking during data transfer.

There are three variations:

XON #[port]

When used on its own, `XON` re-enables data transmission after it has been halted using `XOFF`. If no port is specified, the default port as defined by `OPORT` is assumed.

Note that `CLEAR OUTPUT` and `ESCAPE` will also re- enable transmission, as will pressing either of the `RESET` and `STOP` buttons.

XON [integer 0 - 255]

The default value for the `XON` character is ASCII 17 ([Ctrl- Q]). This form of the `XON` command is used to re-define this value so that an alternative character may be used for software flow control. . It is the same as setting S-register 22 (`SREG22= [num]`).

Note that after pressing the `STOP` button, the `XOFF` character for each port will be set to 19 ([Ctrl-S]) and `XON` to 17 ([Ctrl-Q]).

Examples:

```
XON
```

```
XON#1 , #2
```

```
XON=17
```

Syntax:

```
XON ( #[port] , ) ( #... )
```

```
XON = [integer 0..255]
```

See also:

```
XOFF
```

YEAR is used to read or set the year value on the internal clock/calendar.

When *reading* it returns an integer from 0 to 99.

When *setting*, the year number from 0 to 99 must be specified.

An incorrect value will leave the current setting unchanged.

Examples:

```
YEAR=89
```

```
PRINT YEAR
```

Syntax:

```
YEAR = [integer 0..99]
```

```
[num-var] = YEAR
```

See also:

```
DATE$, DAY, DDAY, MONTH
```

Error Messages

The ARGUS can generate about 100 different error messages.

The associated text can be requested via the `REPORT ERN` command.

The number of the last error message can be obtained via `ERN`; and the line which caused this error with `ERL`.

Almost all error messages can be trapped and handled with `ON ERROR`, except:

- `NMI/STOP`
- `STOP`
- Out of memory
- Error in `TIMEOUT-SUB` (`ON ERROR` must be redefined
in timeout subroutine)
- `RESET` (can be handled with `ON RESET`)

A list of all error numbers and messages is given on the following pages.

<i>No.</i>	<i>message</i>	<i>cause</i>
0	OK	File transfer was successful.
2	BASIC 6.xx	BASIC version number.
3	ESCAPE	Escape character received and ESCAPE was ON.
4	RESET	A reset has occurred (power failure, button, RESET).
5	STOP	Program execution has stopped by command STOP.
6	Cancelled	File transfer cancelled with SEND or RECEIVE.
7	Failed	FAX RECEIVE not successful or too many errors in SEND/RECEIVE file transfer or timed out.
8	Calculation error	Error in calculation expression.
9	Division by zero	Division by zero not allowed.
10	Bad string	Error in a string.
11	Bad button	A button command out of the range 1-3.
12	String expected	No string found where one was expected
13	No DATA	RESTORE on a line which does not start with DATA
14	Out of DATA	A READ where no more DATA is available.
15	Mistake	Something is not recognized as a command.
16	Bad line	A line number for a GOTO is not in range (32767)
17	Bad program address	PBOT has been set to an invalid address
18	Syntax error	All non-specific errors in commands.
19	Not allowed	Command is not allowed inside a program (NEW, DELETE, RENUM, AUTO).
20	Bad LED	LED number not in range.
21	Memory corrupted	Program or variables or RAM-disk damaged.
22	Out of memory	No more space for new variables or lines or files (try BUFFER OFF). During RECEIVE, disk became full.
23	Subscript	Array element out of range.
24	Array	A non-existent array is accessed.
25	Bad time/date	Wrong value for time or date.
26	Bad setting	A communication parameter has wrong value.
27	No RAM expansion	No extra memory is plugged in for filing system usage.
28	No filing system	No filing system software (ROM) on board.
29	Printer using line	One of the I/O lines is accessed while printer is active.
30	Bad I/O line	I/O line number not in range (ILINE/OLINE).
31	Bad port	Port number not in range.
32	Bad S-register	S-register not in range (SREG). Must be 0-79.
33	Aborted	File transfer aborted by remote.
34	ON out of range	An ON GOTO/GOSUB line number out of range.

35	Array exist	DIM of an already existing array.
36	ON/OFF expected	ON or OFF missing after a command.
37	Type mismatch	Integer and string in same expression.
38	No number/variable	A number or variable is expected or missing.
39	Variable expected	Variable name expected.
40	Bad DIM	Syntax error in DIM command.
41	Label/Line expected	A label or line number is expected.
42	= missing	Equal sign missing.
43	Line too long	Length of a program line is longer than 255 bytes.
44	Number too big	A number exceeds 32767.
45	Variable not found	Integer variable does not exist.
46	Bad HEX	Wrong format for HEX number.
47	String not found	String variable does not exist.
48	Line not found	Line number in GOTO/GOSUB does not exist.
49	[missing	Left bracket missing.
50] missing	Right bracket missing.
51	" " missing	Signs with text missing.
52	, missing	Sign missing.
53	Label not found	Label does not exist.
54	FOR not found	A NEXT followed by an unknown variable.
55	No FOR	A NEXT without FOR.
56	Too many FOR's	More than 16 nested FORs.
57	No REPEAT	UNTIL without REPEAT.
58	Too many REPEAT's	More than 16 nested REPEATs.
59	No GOSUB	A RETURN without GOSUB.
60	Too many GOSUB's	More than 16 nested GOSUBs.
61	No WHILE	WEND without WHILE.
62	Too many WHILE's	More than 16 nested WHILEs.
63	No WEND	A WHILE without WEND.
64	TO expected	TO forgotten in FOR command.
65	DO expected	DO forgotten in WHILE command.
66	Error in TIMEOUT sub:	Error encountered in ON TIMEOUT subroutine.
67	TIMEOUT END expected	A program ends in a TIMEOUT subroutine.
68	No ON TIMEOUT	TIMEOUT END found outside timeout subroutine.
69	(Timeout .	Used during trace mode.
70	End)	Used during trace mode.
71	No FAX system	No fax system ROM available.

72	Bad name	File name not correct.
73	Not found	File(s) not found
74	Duplicate name	File already exists while using RENAME.
75	* RESET *	Used during trace mode.
76	* ERROR *	Used during trace mode.
77	* AT *	Used during trace mode.
78	* SLASH *	Used during trace mode.
79	* RING *	Used during trace mode.
80	* CONNECT *	Used during trace mode.
81	* HANGUP *	Used during trace mode.
82	* BRK *	Used during trace mode.
83	* DTD *	Used during trace mode.
84	* DTR *	Used during trace mode.
85	* SEQUENCE *	Used during trace mode.
86	* ESCAPE *	Used during trace mode.
87	* BUTTON3 *	Used during trace mode.
88	* BUTTON2 *	Used during trace mode.
89	* BUTTON1 *	Used during trace mode.
90	Not implemented	Used by filing system.
91	Bad part	Used by filing system.
92	End of File	Used by filing system.
93	Not last	Used by filing system.
94	File not opened	Used by filing system.
95	File not closed	Used by filing system.
96	No/Bad BUFFER	Too small BUFFER in FAX SEND/RECEIVE or PLAY/RECEORD.
97	No disk	No hard disk found.
100	No net	No LAN filing system (NETWORK).
101	No server	No network server found (doesn't react).
102	No clock	No network clock found.

BASIC Keywords and associated short forms

ABS	AB.	ELSE	EL.
ADC	AD.	END	E.
ALL	AL.	ENGLISH	ENG.
ANSWER	AN.	ERL	ERL
ASC	AS.	ERN	ERN
AT	AT	ERROR	ER.
AUTO	A.	ESCAPE	ES.
BAUD	BA.	FALSE	FA.
BRK	BR.	FILE	FI.
BUFFER	B/	FOR	F.
BUTTON	BUT.	FREQ	FR.
CALL	CA.	GERMAN	GE.
CHR\$	CH.	GET	GET
CLEAR	C.	GET\$	GET.
CLOCK\$	CLO.	GOSUB	GOS.
CONNECT	CO.	GOTO	G.
COPY	COP.	HANDSHAKE	HAND.
CRC	CR.	HANGUP	H.
CTS	CT.	HOOK	HO.
DATA	DA.	HOUR	HOU.
DATES\$	DATE.	IF	IF
DAY	DAY	ILINE	IL.
DCD	DC.	INPUT	I.
DDAY	DD.	IPOINT	IP.
DELETE	DE.	KEY	KEY
DIAL	DI.	KEY\$	KE.
DIM	DIM	LCASE	LCASE
DIR	DIR	LCASE\$	LC.
DO	D.	LED	LED
DSR	DS.	LEFT\$	LE.
DTD	DTD	LEN	LEN
DTR	DT.	LENGTH	LEN.
DUTCH	DU.	LINEFEED	LIN.
ECHO	EC.	LINK	
LIST	.	RIGHT\$	RI.
LOAD	LO.	RING	RIN.
LTRIM\$	LT.	RND	RN.
MATCH	MA.	RPT\$	RP.
MID\$	MI.	RTRIM\$	RT.
MIN	MIN	RTS	RTS
MODEM	M.	RUN	R.
MONTH	MON.	SAVE	SA.
NEW	NEW	SBITS	SB.
NEXT	N.	SEC	SEC
NOT	NO.	SEND	SE.
OFF	OF.	SEQUENCE	SEQ.
OLD	O.	SGN	SG.
OLINE	OLI.	SLASH	SL.
ON	ON	SOUND	SO.
OPORT	OP.	SPC\$	SPC.
ORI	OR.	SPEAKER	SP.
OUTPUT	OU.	SREG	SR.
PARITY	PA.	STEP	STE.
PBOT	PB.	STOP	S.

PEEK	PEEK	THEN	TH.
PEEK\$	PE.	TIME\$	TIME\$
PLAY	PL.	TIMEOUT	TI.
POKE	POKE	TO	TO
POKE\$	PO.	TONE	TO.
PORT	POR.	TRACE	T.
PRINT	?	TRIM\$	TRI.
PRINTER	P.	TRUE	TRU.
PTOP	PT.	UCASE	UCASE
PULSE	PU.	UCASE\$	UC.
PUT	PUT	UNTIL	U.
PUT\$	PU.	VAL	VAL
READ	REA.	VAL\$	VA.
RECEIVE	REC.	VOICE	VO.
RECORD	RECO.	VOLUME	VOL.
REM	REM	WAIT	W.
RENAME	RENA.	WEND	WE.
RENUM	RE.	WHILE	WH.
REPEAT	REP.	XOFF	X.
REPORT	REPO.	XON	XON
RESET	RESE.	YEAR	Y.
RESTORE	RES.		
RESULT	RESU.		
RETURN	RET.		

KEYBOARD & LCD codes

Keyboard codes:

The following codes are returned if port 6 is used for input with `GET`, `GET$`, `KEY`, `KEY$` or `INPUT`:

SoftKey 1	= 65
SoftKey 2	= 66
SoftKey 3	= 67
SK1+SK2	= 68
SK1+SK3	= 69
SK2+SK3	= 70
SK1+SK2+SK3	= 71
0 - 9	= 48 - 57
*	= 42
#	= 35
REC	= 12
<<	= 8
STOP	= 14
>	= 10
>>	= 9
>>>	= 13
Function 1 - 8	= 128 - 135

Pressing `> + >> + >>>` at the same time will directly jump to the modem's AT-command-mode. After that, returning to BASIC-mode is done by entering `AT*B<CR>`. This 'stop' function can be disabled by setting bit7 in address `&49D` (`POKE &49D, &80`), but care must be taken that the program can't crash (define `ON ERROR`).

LCD display codes:

The LCD display is 2 lines of each 40 characters wide, but only 24 characters are visible! The following codes can be used if an output is done to port 6 with `PUT`, `PUT$` or `PRINT`.

0	= NOP (no operation)
1	= Display OFF
2	= Display ON + Cursor OFF + Blink OFF
3	= Display ON + Cursor ON + Blink OFF
4	= Display ON + Cursor OFF + Blink ON
5	= Display ON + Cursor ON + Blink ON
6	= Shift Display Left 1 character
7	= Shift Display Right 1 character
8	= Cursor Left 1 character
9	= Cursor Right 1 character
10	= Cursor Down 1 line
11	= Cursor Up 1 line
12	= Clear Display & Cursor at Home
13	= Cursor at start of line (return)
14	= Cursor at Home
15-31	= reserved
32-127	= Normal ASCII characters
128-255	= Special characters (lots of Japanese)