**Altair BASIC 3.2 (4K) - Annotated Disassembly**

Here is a complete annotated disassembly of Altair BASIC 3.2 (4K). I have faithfully followed the program order, and the code naturally divides into three sections - the Interpreter, the Maths Package, and Initialisation.

Each of these sections is more-or-less logically subdivided into a number of subsections.

**1. The Interpreter**

[**Interpreter (Explanation)**](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_ex.htm) - Explanations and overviews of most of the internal logic used in BASIC. It describes the memory layout, key concepts, structures and variables, and you shouldn't look at a line of code until you've read through this.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Section** | **Base** | **Size** | **Key Functions** |
| 1.1 | [Restarts](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_1.htm) | 0000 | 61 | Start, SyntaxCheck, NextChar, OutChar, CompareHLDE, FTestSign, PushNextWord |
| 1.2 | [Keyword Tables](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_2.htm) | 003D | 189 | [no code in this section] |
| 1.3 | [Error Codes & Globals](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_3.htm) | 00FA | 152 | [no code in this section] |
| 1.4 | [Utility Functions](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_4.htm) | 0192 | 101 | GetFlowPtr, CopyMemoryUp, CheckEnoughVarSpace, CheckEnoughMem, OutOfMemory, SyntaxError, DivideByZero, Error |
| 1.5 | [The BASIC Prompt & Program Storage](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_5.htm) | 01F7 | 375 | Stop, Main, StoreProgramLine, FindProgramLine, New, Run, ResetAll, InputLineWith'?', Tokenize, InputLine |
| 1.6 | [Terminal I/O](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_6.htm) | 036E | 32 | OutChar\_tail, InputChar |
| 1.7 | [LIST Handler](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_7.htm) | 038E | 71 | List |
| 1.8 | [FOR Handler](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_8.htm) | 03D5 | 76 | For |
| 1.9 | [Execution](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_9.htm) | 0421 | 61 | ExecNext, Exec |
| 1.10 | [More Utility Functions](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_10.htm) | 045E | 63 | NextChar\_tail, Restore, TestBreakKey, CharIsAlpha, GetSubscript |
| 1.11 | [Goto, Gosub, and Return](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_11.htm) | 049D | 101 | LineNumberFromStr, Gosub, Goto, Return, FindNextStatement |
| 1.12 | [Assigning Variables](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_12.htm) | 0502 | 20 | Let, AssignVar |
| 1.13 | [IF Handler](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_13.htm) | 0516 | 63 | If |
| 1.14 | [Printing](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_14.htm) | 0555 | 143 | Print, NewLine, PrintString, ToNextTabBreak, Tab |
| 1.15 | [INPUT & READ Handlers](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_15.htm) | 05E4 | 101 | Input, Read |
| 1.16 | [NEXT Handler](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_16.htm) | 0649 | 65 | Next |
| 1.17 | [Expression Evaluation](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_17.htm) | 068A | 135 | EvalExpression, EvalTerm |
| 1.18 | [Variable Management](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/int_dis_18.htm) | 0711 | 232 | Dim, GetVar, GetArrayVar |

Total size: 2041 bytes

**2. The Math Package**

[**Math package (explained)**](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_ex.htm) - All the concepts needed to understand the math package.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Section** | **Base** | **Size** | **Key Functions** |
| 2.1 | [Utility Functions](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_1.htm) | 07F9 | 17 | FWordToFloat, FAddOneHalf |
| 2.2 | [Addition & Subtraction](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_2.htm) | 080A | 81 | FSub, FAdd |
| 2.3 | [Mantissa Magic](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_3.htm) | 085B | 136 | FNormalise, FMantissaLeft, FMantissaInc, Overflow, FAddMantissas, FNegateInt, FMantissaRtOnce, FMantissaRtMult |
| 2.4 | [Multiplication & Division](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_4.htm) | 08E3 | 247 | FMul, FDiv, FDivByTen, FMulByTen, FZero, FExponentAdd |
| 2.5 | [Sign Magic](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_5.htm) | 09DA | 40 | FTestSign\_tail, InvSignToInt, SignToInt, Sgn, FCharToFloat, Abs, FNegate |
| 2.6 | [Moving FACCUM about](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_6.htm) | 0A02 | 53 | FPush, FLoadFromMem, FLoadFromBCDE, FCopyToBCDE, FLoadBCDE, FCopyToMem |
| 2.7 | [Unpacking & Comparison](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_7.htm) | 0A37 | 64 | FUnpack, FCompare |
| 2.8 | [Converting to Integers](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_8.htm) | 0A77 | 60 | FAsInteger, FMantissaDec, Int |
| 2.9 | [Reading Numbers](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_9.htm) | 0AB3 | 124 | FIn |
| 2.10 | [Printing Numbers](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_10.htm) | 0B2F | 242 | PrintIN, PrintInt, FOut |
| 2.11 | [SQR](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_11.htm) | 0C21 | 62 | Sqr |
| 2.12 | [RND](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_12.htm) | 0C5F | 54 | Rnd |
| 2.13 | [SIN](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/math_dis_13.htm) | 0C95 | 130 | Sin |

Total size: 1307 bytes.

**3. Initialisation**

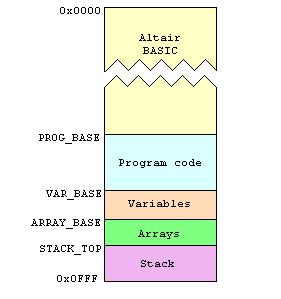
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Section** | **Base** | **Size** | **Key Functions** |
| 3.1 | [IO Detection](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/init_dis_1.htm) | 0D21 | 109 | Init |
| 3.2 | [Configuration](https://web.archive.org/web/20011211233332/www.rjh.org.uk/altair/4k/init_dis_2.htm) | 0D8E | 367 |  |

# The BASIC Interpreter - Explained

blah

## How Basic arranged memory

Here's how Basic arranged the 4K of memory available to it. Most of memory was of course occupied by BASIC itself - from 76% to 82% depending on what optional inline functions had been selected during initialisation - with the remaining few hundred bytes at the top for the program code, space for variables, and of course stack space.



Lets consider the blocks of memory that follow Basic's own code in turn :

The minimum amount of stack space is 18 bytes - at initialisation, after the user has stated the options they want, the amount of space is reported as "X BYTES FREE", where X is 4096 minus (amount needed for Basic, plus 18 bytes for the stack). With all optional inline functions selected - SIN, RND, and SQR - X works out to 727 bytes. With no optional inline functions selected, the amount increases to 973 bytes.

## Program Code

For efficiency, each line of the program would be 'tokenised' before being stored in program space. This tokenisation involved the simple replacement of keywords with keyword IDs. These keyword IDs occupied a single byte, and were easily distinguished from other bytes of the program since they had their top bit set - ie they were in the range of 0x80 to 0xFF.

Consider this line of input :

FOR I=1 TO 10

This would be tokenised to :

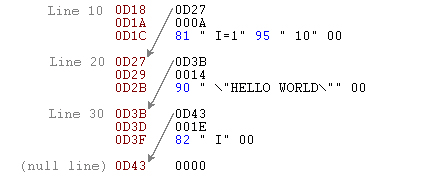
81 " I=1" 95 " 10"

Which is 0x81 (keyword ID for 'FOR') followed by the string " I=1", followed by 0x95 (keyword ID for 'TO') followed by the string " 10". This is 9 bytes, compared to 13 bytes for the untokenised input.

This particular example line of input is meaningless unless it is part of a larger program. As you should know already, each line of a program is prefixed with a line number. These line numbers get stored as 16-bit integers preceding the tokenised line content. Additionally, each line is stored with a pointer to the following line. Let's consider the example input again, this time as a part of a larger program :

10 FOR I=1 TO 10  
20 PRINT "HELLO WORLD"  
30 NEXT I

Assuming that the beginning of program memory was at 0D18, this program would be stored in memory like this:



So as you can see, each program line has three components :

* Pointer to the next line
* Line number
* Tokenised line content

The final line of the program - the last one in the above diagram - is always present and is always a null pointer to the non-existent next line. This null line, just two bytes long, is there to mark the end of the program.

## Variables

The variable support in this version of Basic is rather limited. There only permitted type of variables is numeric - no strings, structs, and of course no distinction between integers and floating-point numbers. All variables are stored and treated as floating-point.

The second restriction is that variable names were a maximum of two characters in length : the first (mandatory) character had to be alphabetic, and the second (optional) character had to be a digit. Thus the following declarations are invalid :

|  |  |
| --- | --- |
| LET FOO=1 | cross |
| LET A="HELLO" | cross |
| LET AB=A | cross |

Whereas these declarations are valid :

|  |  |
| --- | --- |
| LET A=1 | tick |
| LET B=2.5 | tick |
| LET B2=5.6 | tick |
|  |  |

The fixed-length of variable names greatly simplified their storage. Each variable occupies 6 bytes : two bytes for the name, and four bytes for the floating-point value (fixme: link to fp).

## Arrays

Arrays are stored seperately in their own block which immediately follows normal variables and is pointed to by VAR\_ARRAY\_BASE. An array is declared with the DIM keyword, and this version of Basic has the curious property where declaring an array of *n* elements results in *n+1* elements being allocated, addressable with subscript values from 0 to *n* inclusive. Thus the following is quite legal :

DIM A(2)  
A(0) = 1  
A(1) = 2  
A(2) = 3

but :

A(3) = 4

results in a Bad Subscript (BS) error.

An array is stored similarly to normal variables in that we lead with the two-byte variable name. This is followed by a 16-bit integer denoting the size in bytes of the array elements; and finally the array elements themselves (4 bytes each). The example array A(2) shown above, if stored at address 0D20, would appear like this :

|  |  |  |  |
| --- | --- | --- | --- |
| **Address** | **Bytes** | **Value** | **Description** |
| 0D20 | 0x4100 | 'A\0' | Variable name |
| 0D22 | 0x000C |  | Total size, in bytes, of the array elements. |
| 0D24 | 0x81000000 | 1 | Element 0 value |
| 0D28 | 0x82000000 | 2 | Element 1 value |
| 0D2C | 0x82400000 | 3 | Element 2 value |

## Program Flow

When a program is RUN, execution begins on the first line of the program. When a line has finished, execution passes to the next line and so on, until the end of the program or a END or STOP instruction is reached.

This is too simple for all but the simplest programs - there are two mechanisms in Basic for altering program flow so that code can run in loops and subroutines be called. These mechanisms are FOR/NEXT for looping, and GOSUB/RETURN for subroutines.

In both FOR and GOSUB cases, the stack is used to store specific information about the program line to return to.

## 1.1 Restarts

One useful feature of the 8080 was the ability to call a handful of addresses in low memory with a single-byte instuction, as opposed to the usual three bytes needed for CALL and other branch instructions. These addresses were known as the "Restart" addresses, and memory-conscious programmers would always put their most-commonly called functions here, thus saving two bytes on every call elsewhere in the program. There are 7 restart addresses, spaced at 8 byte intervals from 0000 to 0030 inclusive. (NB: There is support for an eighth restart function, RST 7, but Basic makes no use of it).

### Start (RST 0)

Once the loader had finished loading Basic into memory from paper tape it would jump to address 0000, the very start of the program. Not much needs to be done here - just disable interrupts and jump up to the [Init](http://altairbasic.org/init.htm) section in upper memory. Notice that the jump address here is coloured red, indicating the code is modified by code elsewhere. In this case, the jump address is changed to point to Ready once Init has run successfully. (fixme: not yet it isnt).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0000 | F3 | Start | DI |  |
| 0001 | C3210D |  | JMP [Init](http://altairbasic.org/init_dis_1.htm#Init) |  |

Fixme: What are these two addresses for?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0004 | 9004 |  | DW 0490 |  |
| 0006 | F907 |  | DW 07F9 |  |

### SyntaxCheck (RST 1)

Here is a truly beautiful piece of code, it's Golden Weasel richly deserved. It's used at run-time to check syntax in a very cool way : the byte immediately following an RST 1 instruction is not the following instruction, but the keyword or operator ID that's expected to appear in the program at that point. If the keyword or operator is not present, then it Syntax Errors out, but if it is present then the return address is fixed-up - ie advanced one byte - and the function falls into NextChar so the caller has even less work to do. I honestly doubt syntax checks could be done more efficiently than this. Sheer bloody genius.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 0008 | 7E | SyntaxCheck | MOV A,M | A=Byte of BASIC program. | | 0009 | E3 |  | XTHL | HL=return address. | | 000A | BE |  | CMP M | Compare to byte expected. | | 000B | 23 |  | INX H | Return address++; | | 000C | E3 |  | XTHL |  | | 000D | C2D001 |  | JNZ [SyntaxError](http://altairbasic.org/int_dis_4.htm#SyntaxError) | Error if not what was expected. | |  |

### NextChar (RST 2)

Return next character of input from the buffer at HL, skipping over space characters. The Carry flag is set if the returned character is not alphanumeric, also the zero flag is set if a null character has been reached.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0010 | 23 | NextChar | INX H |  |
| 0011 | 7E |  | MOV A,M |  |
| 0012 | FE3A |  | CPI 3A |  |
| 0014 | D0 |  | RNC |  |
| 0015 | C35E04 |  | JMP [NextChar\_tail](http://altairbasic.org/int_dis_10.htm#NextChar_tail) |  |

### OutChar (RST 3)

Prints a character to the terminal.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0018 | F5 | OutChar | PUSH PSW |  |
| 0019 | 3A2700 |  | LDA [TERMINAL\_X](http://altairbasic.org/int_dis_1.htm#TERMINAL_X) |  |
| 001C | C36E03 |  | JMP [OutChar\_tail](http://altairbasic.org/int_dis_6.htm#OutChar_tail) |  |
| 001F | 00 |  | NOP |  |

### CompareHLDE (RST 4)

Compares HL and DE with same logical results (C and Z flags) as for standard eight-bit compares.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0020 | 7C | CompareHLDE | MOV A,H |  |
| 0021 | 92 |  | SUB D |  |
| 0022 | C0 |  | RNZ |  |
| 0023 | 7D |  | MOV A,L |  |
| 0024 | 93 |  | SUB E |  |
| 0025 | C9 |  | RET |  |

### TERMINAL\_X and TERMINAL\_Y

Variables controlling the current X and Y positions of terminal output.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0026 | 01 | TERMINAL\_Y | DB 01 |  |
| 0027 | 00 | TERMINAL\_X | DB 00 |  |

### FTestSign (RST 5)

Tests the state of FACCUM. This part returns with A=0 and zero set if FACCUM==0, the tail of the function sets the sign flag and A accordingly (0xFF is negative, 0x01 if positive) before returning.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0028 | 3A7201 | FTestSign | LDA [FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 002B | B7 |  | ORA A |  |
| 002C | C2DA09 |  | JNZ [FTestSign\_tail](http://altairbasic.org/math_dis_5.htm#FTestSign_tail) |  |
| 002F | C9 |  | RET |  |

### PushNextWord (RST 6)

Effectively PUSH (HL). First we write the return address to the JMP instruction at the end of the function; then we read the word at (HL) into BC and push it onto the stack; lastly jumping to the return address.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0030 | E3 | PushNextWord | XTHL |  |
| 0031 | 223B00 |  | SHLD 003B |  |
| 0034 | E1 |  | POP H |  |
| 0035 | 4E |  | MOV C,M |  |
| 0036 | 23 |  | INX H |  |
| 0037 | 46 |  | MOV B,M |  |
| 0038 | 23 |  | INX H |  |
| 0039 | C5 |  | PUSH B |  |
| 003A | C33A00 |  | JMP 003A |  |

**1.2 Keywords**

There are three groups of keywords :

* General keywords. These typically start a statement; examples are LET, PRINT, GOTO and so on.
* Supplementary keywords. Used in statements but not as part of an expression, eg TO, STEP, TAB
* Inline keywords. Only used in expressions, eg, SIN, RND, INT.

**KW\_INLINE\_FNS**

A table of function pointers for the inline keywords.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 003D | E409 | KW\_INLINE\_FNS | DW [Sgn](http://altairbasic.org/math_dis_5.htm#Sgn) |  |
| 003F | A20A |  | DW [Int](http://altairbasic.org/math_dis_8.htm#Int) |  |
| 0041 | F809 |  | DW [Abs](http://altairbasic.org/math_dis_5.htm#Abs) |  |
| 0043 | 9804 |  | DW Usr |  |
| 0045 | 210C |  | DW [Sqr](http://altairbasic.org/math_dis_11.htm#Sqr) |  |
| 0047 | 5F0C |  | DW [Rnd](http://altairbasic.org/math_dis_12.htm#Rnd) |  |
| 0049 | 950C |  | DW [Sin](http://altairbasic.org/math_dis_13.htm#Sin) |  |

**KW\_ARITH\_OP\_FNS**

A table of function pointers for the arithmetic operator functions. Four entries of three bytes each; the first entry byte is for operator precedence and the second and third bytes are function pointers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 004B | 791008 | KW\_ARITH\_OP\_FNS | DB 0x79, DW [FAdd](http://altairbasic.org/math_dis_2.htm#FAdd) | + |
| 004E | 790A08 |  | DB 0x79, DW [FSub](http://altairbasic.org/math_dis_2.htm#FSub) | - |
| 0051 | 7CE308 |  | DB 0x7C, DW [FMul](http://altairbasic.org/math_dis_4.htm#FMul) | \* |
| 0054 | 7C2F09 |  | DB 0x7C, DW [FDiv](http://altairbasic.org/math_dis_4.htm#FDiv) | / |

**KEYWORDS**

String constants for all keywords, including arithmetic operators. Note that the last character of each keyword has bit 7 set to denote that it is the last character; also that the whole table is terminated with a single null byte.

|  |
| --- |
| General keywords |
| 0057 | 454EC4 | KEYWORDS | "END" | 80 |
| 005A | 464FD2 |  | "FOR" |  |
| 005D | 4E4558D4 |  | "NEXT" | 82 |
| 0061 | 444154C1 |  | "DATA" | 83 |
| 0065 | 494E5055D4 |  | "INPUT" | 84 |
| 006A | 4449CD |  | "DIM" | 85 |
| 006D | 524541C4 |  | "READ" | 86 |
| 0071 | 4C45D4 |  | "LET" | 87 |
| 0074 | 474F54CF |  | "GOTO" | 88 |
| 0078 | 5255CE |  | "RUN" | 89 |
| 007B | 49C6 |  | "IF" | 8A |
| 007D | 524553544F52C5 |  | "RESTORE" | 8B |
| 0084 | 474F5355C2 |  | "GOSUB" | 8C |
| 0089 | 5245545552CE |  | "RETURN" | 8D |
| 008F | 5245CD |  | "REM" | 8E |
| 0092 | 53544FD0 |  | "STOP" | 8F |
| 0096 | 5052494ED4 |  | "PRINT" | 90 |
| 009B | 4C4953D4 |  | "LIST" | 91 |
| 009F | 434C4541D2 |  | "CLEAR" | 92 |
| 00A4 | 4E45D7 |  | "NEW" | 93 |
| Supplementary keywords |
| 00A7 | 544142A8 |  | "TAB(" | 94 |
| 00AB | 54CF |  | "TO" | 95 |
| 00AD | 544845CE |  | "THEN" | 96 |
| 00B1 | 535445D0 |  | "STEP" | 97 |
| Arithmetic and logical operators |
| 00B5 | AB |  | "+" | 98 |
| 00B6 | AD |  | "-" | 99 |
| 00B7 | AA |  | "\*" | 9A |
| 00B8 | AF |  | "/" | 9B |
| 00B9 | BE |  | ">" | 9C |
| 00BA | BD |  | "=" | 9D |
| 00BB | BC |  | "<" | 9E |
| Inline keywords |
| 00BC | 5347CE |  | "SGN" | 9F |
| 00BF | 494ED4 |  | "INT" | A0 |
| 00C2 | 4142D3 |  | "ABS" | A1 |
| 00C5 | 5553D2 |  | "USR" | A2 |
| 00C8 | 5351D2 |  | "SQR" | A3 |
| 00CB | 524EC4 |  | "RND" | A4 |
| 00CE | 5349CE |  | "SIN" | A5 |
| Null terminator. |
| 00D1 | 00 |  |  |  |

**KW\_GENERAL\_FNS**

Pointers to the functions for the 20 general keywords at the start of the KEYWORDS table above.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00D2 | F701 | KW\_GENERAL\_FNS | DW [Stop](http://altairbasic.org/int_dis_5.htm#Stop) | END |
| 00D4 | D503 |  | DW [For](http://altairbasic.org/int_dis_8.htm#For) | FOR |
| 00D6 | 4906 |  | DW [Next](http://altairbasic.org/int_dis_16.htm#Next) | NEXT |
| 00D8 | F504 |  | DW Data | DATA |
| 00DA | E405 |  | DW [Input](http://altairbasic.org/int_dis_15.htm#Input) | INPUT |
| 00DC | 1607 |  | DW [Dim](http://altairbasic.org/int_dis_18.htm#Dim) | DIM |
| 00DE | F605 |  | DW [Read](http://altairbasic.org/int_dis_15.htm#Read) | READ |
| 00E0 | 0205 |  | DW [Let](http://altairbasic.org/int_dis_12.htm#Let) | LET |
| 00E2 | CF04 |  | DW [Goto](http://altairbasic.org/int_dis_11.htm#Goto) | GOTO |
| 00E4 | A102 |  | DW [Run](http://altairbasic.org/int_dis_5.htm#Run) | RUN |
| 00E6 | 1605 |  | DW [If](http://altairbasic.org/int_dis_13.htm#If) | IF |
| 00E8 | 6904 |  | DW [Restore](http://altairbasic.org/int_dis_10.htm#Restore) | RESTORE |
| 00EA | BE04 |  | DW [Gosub](http://altairbasic.org/int_dis_11.htm#Gosub) | GOSUB |
| 00EC | DF04 |  | DW [Return](http://altairbasic.org/int_dis_11.htm#Return) | RETURN |
| 00EE | F704 |  | DW [Rem](http://altairbasic.org/int_dis_11.htm#Rem) | REM |
| 00F0 | F701 |  | DW [Stop](http://altairbasic.org/int_dis_5.htm#Stop) | STOP |
| 00F2 | 5705 |  | DW [Print](http://altairbasic.org/int_dis_14.htm#Print) | PRINT |
| 00F4 | 8E03 |  | DW [List](http://altairbasic.org/int_dis_7.htm#List) | LIST |
| 00F6 | A602 |  | DW Clear | CLEAR |
| 00F8 | 9502 |  | DW [New](http://altairbasic.org/int_dis_5.htm#New) | NEW |

## .3 Error Codes & Globals

A table of two-character error codes for the 12 errors.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00FA | 4EC6 | ERROR\_CODES | "NF" | NEXT without FOR. |
| 00FC | 53CE |  | "SN" | Syntax Error |
| 00FE | 52C7 |  | "RG" | RETURN without GOSUB. |
| 0100 | 4FC4 |  | "OD" | Out of Data |
| 0102 | 46C3 |  | "FC" | Illegal Function Call |
| 0104 | 4FD6 |  | "OV" | Overflow. |
| 0106 | 4FCD |  | "OM" | Out of memory. |
| 0108 | 55D3 |  | "US" | Undefined Subroutine |
| 010A | 42D3 |  | "BS" | Bad Subscript |
| 010C | 44C4 |  | "DD" | Duplicate Definition |
| 010E | 2FB0 |  | "\0" | Division by zero. |
| 0110 | 49C4 |  | "ID" | Invalid in Direct mode. |

### LINE\_BUFFER

Buffer for a line of input or program, 73 bytes long.

|  |
| --- |
| The line buffer is prefixed with this comma. It's here because the INPUT handler defers to the READ handler, which expects items of data (which the line buffer is treated as) to be prefixed with commas. Quite a neat trick! |
| 0112 | 2C |  | DB ',' |  |
| 0113 | 0000 | LINE\_BUFFER | DW 0000 |  |
| .... | .... |  | ....... |  |
| 015A | 0000 |  | DW 0000 |  |

### Globals

A bunch of variables BASIC needs to run.

|  |
| --- |
| A flag used by GetArrayVar to determine whether we are declaring an array with DIM, or whether we are accessing an element of an existing array as might be done by EvalTerm. |
| 015B | 00 | DIM\_OR\_EVAL | DB 00 |  |
| A flag used by [Read](http://altairbasic.org/int_dis_3.htm#read) to determine whether the INPUT or READ keyword invoked it. |
| 015C | 00 | INPUT\_OR\_READ | DB 00 |  |
| A temporary prog ptr, used by the NEXT handler and other places (fixme: identify). |
| 015D | 0000 | PROG\_PTR\_TEMP | DW 0000 |  |
| A temporary prog ptr used by the Expression Evaluator |
| 015F | 0000 |  | DW 0000 |  |
| Holds the line number of the program line currently being executed |
| 0161 | 0000 | CURRENT\_LINE | DW 0000 |  |
| Holds the highest possible memory word address, used for the top of the stack. |
| 0163 | 1A0F | STACK\_TOP | DW 0F1A |  |
| Pointer to the base of program storage. |
| 0165 | 0000 | PROGRAM\_BASE | DW 0000 |  |
| Points to the start of storage available for programs. Always immediately follows program storage. |
| 0167 | 0000 | VAR\_BASE | DW 0000 |  |
| Points to the start of array storage. This immediately follows the block allocated for normal variables pointed to by VAR\_BASE. |
| 0169 | 0000 | VAR\_ARRAY\_BASE | DW 0000 |  |
| Points to top of variable and array variable storage. |
| 016B | 0000 | VAR\_TOP | DW 0000 |  |
| Points to the point within the program where DATA can be READ from. |
| 016D | 0000 | DATA\_PROG\_PTR | DW 0000 |  |
| The (F)loating (Accum)ulator. See the [Floating Point section](http://altairbasic.org/math_ex.htm) for a full explanation. |
| 016F | 00000000 | FACCUM | DD 0 |  |
| A temporary byte used by floating point to temporarily hold signs+exponents during calculations. |
| 0173 | 00 | FTEMP | DB 00 |  |

### 

### FBUFFER

Small buffer (12 bytes) used by the math package functions FOut and Sqr.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0174 | 0000 | FBUFFER | DW 0000 |  |
| .... | .... |  | ....... |  |
| 017F | 0000 |  | DW 0000 |  |

### Status Strings

Some string constants, used when reporting errors and printing OK.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0181 | 204552524FD200 | szError | " ERROR\0" |  |
| 0188 | 209E00 | szIn | " IN \0" |  |
| 018D | 0DFBD0 | szOK | "\rOK\r\0" |  |

## 1.4 Utility Functions

Some useful functions.

### GetFlowPtr

Sets HL to point to the appropriate flow struct on the stack. On entry, if this was called by the NEXT keyword handler then DE is pointing to the variable following the NEXT keyword.

|  |
| --- |
| The first four bytes on the stack are (or rather, should be) two return addresses. We're not interested in them, so the first thing to do is set HL to point to SP+4. |
| 0192 | 210400 | GetFlowPtr | LXI H,0004 | HL=SP+4 (ie get word |
| 0195 | 39 |  | DAD SP | just past return addr) |
| Get the keyword ID, the byte that precedes the flow struct. Then we increment HL so it points to (what should be) the flow struct, and return if the keyword ID is not 'FOR'. |
| 0196 | 7E |  | MOV A,M |  |
| 0197 | 23 |  | INX H |  |
| 0198 | FE81 |  | CPI 81 | 'FOR'? |
| 019A | C0 |  | RNZ | Return if not 'FOR' |
| Special treatment for FOR flow structs. Here we check that we've got the right one, ie the one required by the NEXT statement which called us. When we're called by NEXT, it sets DE to point to the variable in the NEXT statement. So here we get the first word of the FOR flow struct which is the address of the FOR variable, and compare it to the one we've been given in DE. If they match, then we've found the flow struct wanted and we can safely return. If not then we jump 13 bytes up the stack - 13 bytes is the size of the FOR flow struct - and loop back to try again. |
| 019B | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) | PUSH (HL) |
| 019C | E3 |  | XTHL | POP HL (ie HL=(HL)) |
| 019D | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) | HL==DE? |
| 019E | 010D00 |  | LXI B,000D |  |
| 01A1 | E1 |  | POP H | Restore HL |
| 01A2 | C8 |  | RZ | Return if var ptrs match. |
| 01A3 | 09 |  | DAD B | HL+=000D |
| 01A4 | C39601 |  | JMP [GetFlowPtr](http://altairbasic.org/int_dis_4.htm#GetFlowPtr)+4 | Loop |

### 

### CopyMemoryUp

Copies a block of memory from BC to HL. Copying is done backwards, down to and including the point where BC==DE. It goes backwards because this function is used to move blocks of memory forward by as little as a couple of bytes. If it copied forwards then the block of memory would overwrite itself.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01A7 | CDC301 | CopyMemoryUp | CALL [CheckEnoughMem](http://altairbasic.org/int_dis_4.htm#CheckEnoughMem) |  |
| Exchange BC with HL, so HL now points to the source and BC points to destination. |
| 01AA | C5 |  | PUSH B | Exchange BC with HL. |
| 01AB | E3 |  | XTHL |  |
| 01AC | C1 |  | POP B |  |
| 01AD | E7 | CopyMemLoop | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) | HL==DE? |
| 01AE | 7E |  | MOV A,M |  |
| 01AF | 02 |  | STAX B |  |
| 01B0 | C8 |  | RZ | Exit if DE reached. |
| 01B1 | 0B |  | DCX B |  |
| 01B2 | 2B |  | DCX H |  |
| 01B3 | C3AD01 |  | JMP [CopyMemLoop](http://altairbasic.org/int_dis_4.htm#CopyMemLoop) |  |

### CheckEnoughVarSpace

Checks that there is enough room for C\*4 bytes on top of (VAR\_TOP) before it intrudes on the stack. Probably varspace.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01B6 | E5 | CheckEnoughVarSpace | PUSH H |  |
| 01B7 | 2A6B01 |  | LHLD [VAR\_TOP](http://altairbasic.org/int_dis_3.htm#VAR_TOP) |  |
| 01BA | 0600 |  | MVI B,00 | BC=C\*4 |
| 01BC | 09 |  | DAD B |  |
| 01BD | 09 |  | DAD B |  |
| 01BE | CDC301 |  | CALL [CheckEnoughMem](http://altairbasic.org/int_dis_4.htm#CheckEnoughMem) |  |
| 01C1 | E1 |  | POP H |  |
| 01C2 | C9 |  | RET |  |

### CheckEnoughMem

Checks that HL is more than 32 bytes away from the stack pointer. If HL is within 32 bytes of the stack pointer then this function falls into OutOfMemory.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01C3 | D5 | CheckEnoughMem | PUSH D |  |
| 01C4 | EB |  | XCHG |  |
| 01C5 | 21DEFF |  | LXI H,FFDE | HL=-34 (extra 2 bytes for return address) |
| 01C8 | 39 |  | DAD SP |  |
| 01C9 | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) |  |
| 01CA | EB |  | XCHG |  |
| 01CB | D1 |  | POP D |  |
| 01CC | D0 |  | RNC |  |

### Three common errors.

Notice use of LXI trick.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01CD | 1E0C | OutOfMemory | MVI E,0C |  |
| 01CF | 01.... |  | LXI B,.... |  |
|  |  |  |  |  |
| 01D0 | 1E02 | SyntaxError | MVI E,02 |  |
| 01D2 | 01.... |  | LXI B,.... |  |
|  |  |  |  |  |
| 01D3 | 1E14 | DivideByZero | MVI E,14 |  |
|  |  |  |  |  |
|  |  |  |  |  |

### Error

Resets the stack, prints an error code (offset into error codes table is given in E), and stops program execution.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01D5 | CDB502 | Error | CALL [ResetStack](http://altairbasic.org/int_dis_5.htm#ResetStack) |  |
| 01D8 | CD8A05 |  | CALL [NewLine](http://altairbasic.org/int_dis_14.htm#NewLine) |  |
| 01DB | 21FA00 |  | LXI H,[ERROR\_CODES](http://altairbasic.org/int_dis_3.htm#ERROR_CODES) |  |
| 01DE | 57 |  | MOV D,A |  |
| 01DF | 3E3F |  | MVI A,'?' | Print '?' |
| 01E1 | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 01E2 | 19 |  | DAD D | HL points to error code. |
| 01E3 | 7E |  | MOV A,M |  |
| 01E4 | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) | Print first char of code. |
| 01E5 | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 01E6 | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) | Print second char of code. |
| 01E7 | 218101 |  | LXI H,[szError](http://altairbasic.org/int_dis_3.htm#szError) | Print " ERROR". |
| 01EA | CDA305 |  | CALL [PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |
| 01ED | 2A6101 |  | LHLD [CURRENT\_LINE](http://altairbasic.org/int_dis_3.htm#CURRENT_LINE) |  |
| 01F0 | 7C |  | MOV A,H |  |
| 01F1 | A5 |  | ANA L |  |
| 01F2 | 3C |  | INR A |  |
| 01F3 | C42F0B |  | CNZ [PrintIN](http://altairbasic.org/math_dis_10.htm#PrintIN) |  |
| 01F6 | 01.... |  | LXI B,.... | LXI over Stop and fall into Main |

## 1.5 The BASIC Prompt & Program Storage

### STOP / END

The keywords STOP and END are synonymous. We don't need to do anything other than lose the return address and fall into Main.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01F7 | C0 | Stop | RNZ | Syntax Error if args. |
| 01F8 | C1 |  | POP B | Lose return address. |

### Main

Here's where a BASIC programmer in 1975 spent most of their time : typing at an "OK" prompt, one line at a time. A line of input would either be exec'd immediately (eg "PRINT 2+2"), or it would be a line of a program to be RUN later. Program lines would be prefixed with a line number. The code below looks for that line number, and jumps ahead to Exec if it's not there.

|  |
| --- |
| Print "OK" |
| 01F9 | 218D01 | Main | LXI H,[szOK](http://altairbasic.org/int_dis_3.htm#szOK) |  |
| 01FC | CDA305 |  | CALL [PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |
| Set current line number to -1, indicating we're in immediate mode. |
| 01FF | 21FFFF | GetNonBlankLine | LXI H,FFFF |  |
| 0202 | 226101 |  | SHLD [CURRENT\_LINE](http://altairbasic.org/int_dis_3.htm#CURRENT_LINE) |  |
| Get a line of input |
| 0205 | CD3C03 |  | CALL [InputLine](http://altairbasic.org/int_dis_5.htm#InputLine) |  |
| Get first char of input. Note that carry will be set if it's a digit. |
| 0208 | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| If line is blank (ie A is the null byte terminating the input buffer) then loop back to get a new input line. Notice the interesting method of testing for 0 - inc followed by dec. This is done so as not to affect the carry flag, which CPI 0 and OR A would have both done. |
| 0209 | 3C |  | INR A |  |
| 020A | 3D |  | DCR A |  |
| 020B | CAFF01 |  | JZ [GetNonBlankLine](http://altairbasic.org/int_dis_5.htm#GetNonBlankLine) |  |
| Preserve first-char-is-digit flag (carry). |
| 020E | F5 |  | PUSH PSW |  |
| If the line of input begins with a line number, this call will read that number into DE. |
| 020F | CD9D04 |  | CALL [LineNumberFromStr](http://altairbasic.org/int_dis_11.htm#LineNumberFromStr) |  |
| Preserve line number on stack and tokenize the rest of the line. |
| 0212 | D5 |  | PUSH D |  |
| 0213 | CDCC02 |  | CALL [Tokenize](http://altairbasic.org/int_dis_5.htm#Tokenize) |  |
| Tokenize returns with the length of the tokenized line content in C and with A=0. Here we load B with 0, so as to have the line length in the 16-bit register BC. |
| 0216 | 47 |  | MOV B,A |  |
| Restore line number to DE. |
| 0217 | D1 |  | POP D |  |
| Restore first-char-is-digit flag (carry), and if the first char was /not/ a digit, then execute the line of input now. |
| 0218 | F1 |  | POP PSW |  |
| 0219 | D23E04 |  | JNC [Exec](http://altairbasic.org/int_dis_9.htm#Exec) |  |
| First char was a digit, therefore it's a line of program which needs to be stored, so we can fall into StoreProgramLine... |

### StoreProgramLine

Here's where a program line has been typed, which we now need to store in program memory.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 021C | D5 | StoreProgramLine | PUSH D | Push line number |
| 021D | C5 |  | PUSH B | Push line length |
| 021E | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) | Get first char of line |
| 021F | B7 |  | ORA A | Zero set if line is empty (ie removing a line) |
| 0220 | F5 |  | PUSH PSW | Preserve line-empty flag |
| 0221 | CD7D02 |  | CALL [FindProgramLine](http://altairbasic.org/int_dis_5.htm#FindProgramLine) | Get nearest program line address in BC. |
| 0224 | C5 |  | PUSH B | Push line address. |
| 0225 | D23902 |  | JNC [InsertProgramLine](http://altairbasic.org/int_dis_5.htm#InsertProgramLine) | If line doesn't exist, jump ahead to insert it. |
| Carry was set by the call to FindProgramLine, meaning that the line already exists. So we have to remove the old program line before inserting the new one in it's place. To remove the program line we simply move the remainder of the program (ie every line that comes after it) down in memory. |
| 0228 | EB | RemoveProgramLine | XCHG | DE=Next line address. |
| 0229 | 2A6701 |  | LHLD [VAR\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_BASE) |  |
| 022C | 1A |  | LDAX D | Move byte of program remainder down |
| 022D | 02 |  | STAX B | in memory. |
| 022E | 03 |  | INX B |  |
| 022F | 13 |  | INX D |  |
| 0230 | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) | Loop until DE==VAR\_BASE, ie whole |
| 0231 | C22C02 |  | JNZ RemoveLine+4 | program remainder done. |
| 0234 | 60 |  | MOV H,B |  |
| 0235 | 69 |  | MOV L,C | Update VAR\_BASE from BC. |
| 0236 | 226701 |  | SHLD [VAR\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_BASE) |  |
| To insert the program line, firstly the program remainder (every line that comes after the one to be inserted) must be moved up in memory to make room. |
| 0239 | D1 | InsertProgramLine | POP D | DE=Line address (from 224) |
| 023A | F1 |  | POP PSW | Restore line-empty flag (see above) |
| 023B | CA6002 |  | JZ [UpdateLinkedList](http://altairbasic.org/int_dis_5.htm#UpdateLinkedList) | If line is empty, then we don't need to insert it so can jump ahead. |
| 023E | 2A6701 |  | LHLD [VAR\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_BASE) |  |
| 0241 | E3 |  | XTHL | HL = Line length (see 21D) |
| 0242 | C1 |  | POP B | BC = VAR\_BASE |
| 0243 | 09 |  | DAD B | HL = VAR\_BASE + line length. |
| 0244 | E5 |  | PUSH H |  |
| 0245 | CDA701 |  | CALL [CopyMemoryUp](http://altairbasic.org/int_dis_4.htm#CopyMemoryUp) | Move remainder of program so there's enough space for the new line. |
| 0248 | E1 |  | POP H |  |
| 0249 | 226701 |  | SHLD [VAR\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_BASE) | Update VAR\_BASE |
| 024C | EB |  | XCHG | HL=Line address, DE=VAR\_BASE |
| 024D | 74 |  | MOV M,H | ??? |
| 024E | 23 |  | INX H | Skip over next line ptr (updated below) |
| 024F | 23 |  | INX H |  |
| 0250 | D1 |  | POP D | DE = line number (see 21C) |
| 0251 | 73 |  | MOV M,E | Write line number to program line memory. |
| 0252 | 23 |  | INX H |  |
| 0253 | 72 |  | MOV M,D |  |
| 0254 | 23 |  | INX H |  |
| 0255 | 111301 | CopyFromBuffer | LXI D,[LINE\_BUFFER](http://altairbasic.org/int_dis_3.htm#LINE_BUFFER) | Copy the line into the program. |
| 0258 | 1A |  | LDAX D |  |
| 0259 | 77 |  | MOV M,A |  |
| 025A | 23 |  | INX H |  |
| 025B | 13 |  | INX D |  |
| 025C | B7 |  | ORA A |  |
| 025D | C25802 |  | JNZ [CopyFromBuffer](http://altairbasic.org/int_dis_5.htm#CopyFromBuffer)+3 |  |
| Now the program line has been inserted/removed, all the pointers from each line to the next need to be updated. |
| 0260 | CDA202 | UpdateLinkedList | CALL [ResetAll](http://altairbasic.org/int_dis_5.htm#ResetAll) |  |
| 0263 | 23 |  | INX H |  |
| 0264 | EB |  | XCHG |  |
| 0265 | 62 |  | MOV H,D |  |
| 0266 | 6B |  | MOV L,E |  |
| 0267 | 7E |  | MOV A,M | If the pointer to the next line is a null |
| 0268 | 23 |  | INX H | word then we've reached the end of the |
| 0269 | B6 |  | ORA M | program, job is done, and we can jump back |
| 026A | CAFF01 |  | JZ [GetNonBlankLine](http://altairbasic.org/int_dis_5.htm#GetNonBlankLine) | to let the user type in the next line. |
| 026D | 23 |  | INX H | Skip over line number. |
| 026E | 23 |  | INX H |  |
| 026F | 23 |  | INX H |  |
| 0270 | AF |  | XRA A |  |
| 0271 | BE |  | CMP M |  |
| 0272 | 23 |  | INX H |  |
| 0273 | C27102 |  | JNZ 0271 |  |
| 0276 | EB |  | XCHG |  |
| 0277 | 73 |  | MOV M,E |  |
| 0278 | 23 |  | INX H |  |
| 0279 | 72 |  | MOV M,D |  |
| 027A | C36502 |  | JMP 0265 |  |

### FindProgramLine

Given a line number in DE, this function returns the address of that progam line in BC. If the line doesn't exist, then BC points to the next line's address, ie where the line could be inserted. Carry flag is set if the line exists, otherwise carry reset.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 027D | 2A6501 | FindProgramLine | LHLD [PROGRAM\_BASE](http://altairbasic.org/int_dis_3.htm#PROGRAM_BASE) |  |
| 0280 | 44 |  | MOV B,H | BC=this line |
| 0281 | 4D |  | MOV C,L |  |
| 0282 | 7E |  | MOV A,M | If we've found two consecutive |
| 0283 | 23 |  | INX H | null bytes, then we've reached the end |
| 0284 | B6 |  | ORA M | of the program and so return. |
| 0285 | 2B |  | DCX H |  |
| 0286 | C8 |  | RZ |  |
| 0287 | C5 |  | PUSH B | Push this line address |
| 0288 | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) | Push (next line address) |
| 0289 | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) | Push (this line number) |
| 028A | E1 |  | POP H | HL = this line number |
| 028B | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) | Compare line numbers |
| 028C | E1 |  | POP H | HL = next line address |
| 028D | C1 |  | POP B | BC = this line address |
| 028E | 3F |  | CMC |  |
| 028F | C8 |  | RZ | Return carry set if line numbers match. |
| 0290 | 3F |  | CMC |  |
| 0291 | D0 |  | RNC | Return if we've reached a line number greater than the one required. |
| 0292 | C38002 |  | JMP [FindProgramLine](http://altairbasic.org/int_dis_5.htm#FindProgramLine)+3 |  |

### 

### New

Keyword NEW. Writes the null line number to the bottom of program storage (ie an empty program), updates pointer to variables storage, and falls into RUN which just happens to do the rest of the work NEW needs to do.

|  |
| --- |
| No arguments allowed for the NEW keyword. |
| 0295 | C0 | New | RNZ |  |
| Write the two null bytes program terminator to the start of program storage. |
| 0296 | 2A6501 |  | LHLD [PROGRAM\_BASE](http://altairbasic.org/int_dis_3.htm#PROGRAM_BASE) |  |
| 0299 | AF |  | XRA A |  |
| 029A | 77 |  | MOV M,A |  |
| 029B | 23 |  | INX H |  |
| 029C | 77 |  | MOV M,A |  |
| 029D | 23 |  | INX H |  |
| And set the base of variable storage to immediately follow the null program. |
| 029E | 226701 |  | SHLD [VAR\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_BASE) |  |

### Run

Runs the program. We don't actually need to do anything here, except check that no arguments have been supplied! We can just fall into ResetAll which sets everything up ready to run the program, and we then return to ExecNext.

|  |
| --- |
| No arguments allowed for the RUN keyword. |
| 02A1 | C0 | Run | RNZ |  |

### ResetAll

Resets everything.

|  |
| --- |
| Set PROG\_PTR\_TEMP to just before the start of the program. |
| 02A2 | 2A6501 | ResetAll | LHLD [PROGRAM\_BASE](http://altairbasic.org/int_dis_3.htm#PROGRAM_BASE) |  |
| 02A5 | 2B |  | DCX H |  |
| 02A6 | 225D01 |  | SHLD [PROG\_PTR\_TEMP](http://altairbasic.org/int_dis_3.htm#PROG_PTR_TEMP) |  |
| Reset the data pointer |
| 02A9 | CD6904 |  | CALL [Restore](http://altairbasic.org/int_dis_10.htm#Restore) |  |
| Reset variable pointers |
| 02AC | 2A6701 |  | LHLD [VAR\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_BASE) |  |
| 02AF | 226901 |  | SHLD [VAR\_ARRAY\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_ARRAY_BASE) |  |
| 02B2 | 226B01 |  | SHLD [VAR\_TOP](http://altairbasic.org/int_dis_3.htm#VAR_TOP) |  |
| Get return address in BC and reset the stack pointer to it's top. |
| 02B5 | C1 | ResetStack | POP B |  |
| 02B6 | 2A6301 |  | LHLD [STACK\_TOP](http://altairbasic.org/int_dis_3.htm#STACK_TOP) |  |
| 02B9 | F9 |  | SPHL |  |
| Push address of stack top module 256. Fixme - why??? |
| 02BA | AF |  | XRA A |  |
| 02BB | 6F |  | MOV L,A |  |
| 02BC | E5 |  | PUSH H |  |
| Put return address back on stack, set HL to ??? and return. |
| 02BD | C5 |  | PUSH B |  |
| 02BE | 2A5D01 |  | LHLD [PROG\_PTR\_TEMP](http://altairbasic.org/int_dis_3.htm#PROG_PTR_TEMP) |  |
| 02C1 | C9 |  | RET |  |

### InputLineWith'?'

Gets a line of input at a '? ' prompt.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 02C2 | 3E3F | InputLineWith'?' | MVI A,'?' | Print '?' |
| 02C4 | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 02C5 | 3E20 |  | MVI A,' ' | Print ' ' |
| 02C7 | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 02C8 | CD3C03 |  | CALL [InputLine](http://altairbasic.org/int_dis_5.htm#InputLine) |  |
| 02CB | 23 |  | INX H |  |

### 

### Tokenize

Tokenises LINE\_BUFFER, replacing keywords with their IDs. On exit, C holds the length of the tokenised line plus a few bytes to make it a complete program line.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 02CC | 0E05 | Tokenize | MVI C,05 | Initialise line length to 5. |
| 02CE | 111301 |  | LXI D,[LINE\_BUFFER](http://altairbasic.org/int_dis_3.htm#LINE_BUFFER) | ie, output ptr is same as input ptr at start. |
| If char is a space, jump ahead to write it out. |
| 02D1 | 7E |  | MOV A,M |  |
| 02D2 | FE20 |  | CPI ' ' |  |
| 02D4 | CA0203 |  | JZ [WriteChar](http://altairbasic.org/int_dis_5.htm#WriteChar) |  |
| If char is a " (indicating a string literal) then freely copy up to the closing ". Obviously we don't want to tokenize string literals. |
| 02D7 | 47 |  | MOV B,A |  |
| 02D8 | FE22 |  | CPI '\"' |  |
| 02DA | CA1503 |  | JZ [FreeCopy](http://altairbasic.org/int_dis_5.htm#FreeCopy) |  |
| If char is null then we've reached the end of input, and can exit this function. |
| 02DD | B7 |  | ORA A |  |
| 02DE | CA2903 |  | JZ [Exit](http://altairbasic.org/int_dis_5.htm#Exit) |  |
| Here's where we start to see if we've got a keyword. |
| 02E1 | D5 |  | PUSH D | Preserve output ptr. |
| 02E2 | 0600 |  | MVI B,00 | Initialise Keyword ID to 0. |
| 02E4 | 115600 |  | LXI D,[KEYWORDS](http://altairbasic.org/int_dis_2.htm#KEYWORDS)-1 |  |
| 02E7 | E5 |  | PUSH H | Preserve input ptr. |
| 02E8 | 3E.. |  | MVI A,.. | LXI over get-next-char |
| fixme. |
| 02E9 | D7 | KwCompare | RST SyntaxCheck0 | Get next input char |
| 02EA | 13 |  | INX D |  |
| 02EB | 1A |  | LDAX D | Get keyword char to compare with. |
| 02EC | E67F |  | ANI 7F | Ignore bit 7 of keyword char. |
| 02EE | CAFF02 |  | JZ [NotAKeyword](http://altairbasic.org/int_dis_5.htm#NotAKeyword) | If keyword char==0, then end of keywords reached. |
| 02F1 | BE |  | CMP M | Keyword char matches input char? |
| 02F2 | C21C03 |  | JNZ [NextKeyword](http://altairbasic.org/int_dis_5.htm#NextKeyword) | If not, jump to get next keyword. |
| OK, so input char == keyword char. Now we test bit 7 of the keyword char : if it's 0 then we haven't yet reached the end of the keyword and so have to loop back to continue comparing. |
| 02F5 | 1A |  | LDAX D |  |
| 02F6 | B7 |  | ORA A |  |
| 02F7 | F2E902 |  | JP [KwCompare](http://altairbasic.org/int_dis_5.htm#KwCompare) |  |
| Matched a keyword! First thing we do is remove input ptr from the stack, as since we're matched to a keyword we don't need to go back and try to match another keyword - HL is already the correct input ptr. Then we set A to the keyword ID which gets written out in the next block but one (notice we LXI over the next block). |
| 02FA | F1 |  | POP PSW | Remove input ptr from stack. We don't need it. |
| 02FB | 78 |  | MOV A,B | A=Keyword ID |
| 02FC | F680 |  | ORI 80 | Set bit 7 (indicates a keyword) |
| 02FE | F2.... |  | JP .... | LXI trick again. |
| Here we have found that the input does not lead with a keyword, so we restore the input ptr and write out the literal character. |
| 02FF | E1 | NotAKeyword | POP H | Restore input ptr |
| 0300 | 7E |  | MOV A,M | and get input char |
| Write character, and advance buffer pointers. |
| 0301 | D1 |  | POP D | Restore output ptr |
| 0302 | 23 | WriteChar | INX H | Advance input ptr |
| 0303 | 12 |  | STAX D | Store output char |
| 0304 | 13 |  | INX D | Advance output ptr |
| 0305 | 0C |  | INR C | C++ (arf!). |
| If we've just written the ID of keyword REM then we need to freecopy the rest of the line. Here we test for REM (8E) and jump back to the outer loop if it isn't. Note that if it is REM, then we set B to 0 so the freecopy won't stop prematurely. |
| 0306 | D68E |  | SUI 8E | If it's not the |
| 0308 | C2D102 |  | JNZ [Tokenize](http://altairbasic.org/int_dis_5.htm#Tokenize)+5 |  |
| 030B | 47 |  | MOV B,A | B=0 |
| Free copy loop. This loop copies from input to output without tokenizing, as needs to be done for string literals and comment lines. The B register holds the terminating character - when this char is reached the free copy is complete and it jumps back |
| 030C | 7E | FreeCopyLoop | MOV A,M | A=Input char |
| 030D | B7 |  | ORA A | If char is null then exit |
| 030E | CA2903 |  | JZ [Exit](http://altairbasic.org/int_dis_5.htm#Exit) |  |
| 0311 | B8 |  | CMP B | If input char is term char then |
| 0312 | CA0203 |  | JZ [WriteChar](http://altairbasic.org/int_dis_5.htm#WriteChar) | we're done free copying. |
| 0315 | 23 | FreeCopy | INX H |  |
| 0316 | 12 |  | STAX D |  |
| 0317 | 0C |  | INR C |  |
| 0318 | 13 |  | INX D |  |
| 0319 | C30C03 |  | JMP [FreeCopyLoop](http://altairbasic.org/int_dis_5.htm#FreeCopyLoop) |  |
| NextKeyword. Advances keyword ptr in DE to point to the next keyword in the table, then jumps back to KwCompare to see if it matches. Note we also increment the keyword ID. |
| 031C | E1 | NextKeyword | POP H | Restore input ptr |
| 031D | E5 |  | PUSH H |  |
| 031E | 04 |  | INR B | Keyword ID ++; |
| 031F | EB |  | XCHG | HL=keyword table ptr |
| 0320 | B6 | NextKwLoop | ORA M | Loop until |
| 0321 | 23 |  | INX H | bit 7 of previous |
| 0322 | F22003 |  | JP [NextKwLoop](http://altairbasic.org/int_dis_5.htm#NextKwLoop) | keyword char is set. |
| 0325 | EB |  | XCHG | DE=keyword ptr, HL=input ptr |
| 0326 | C3EB02 |  | JMP [KwCompare](http://altairbasic.org/int_dis_5.htm#KwCompare)+2 |  |
| Exit. Restore LINE\_BUFFER to HL, null-terminated the tokenized line buffer (three times in fact - why?) and return. |
| 0329 | 211201 | Exit | LXI H,[LINE\_BUFFER](http://altairbasic.org/int_dis_3.htm#LINE_BUFFER) |  |
| 032C | 12 |  | STAX D |  |
| 032D | 13 |  | INX D |  |
| 032E | 12 |  | STAX D |  |
| 032F | 13 |  | INX D |  |
| 0330 | 12 |  | STAX D |  |
| 0331 | C9 |  | RET |  |

### InputLine

Gets a line of input into LINE\_BUFFER.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0332 | 05 | Backspace | DCR B | Char count--; |
| 0333 | 2B |  | DCX H | Input ptr--; |
| 0334 | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) | Print backspace char. |
| 0335 | C24103 |  | JNZ [InputNext](http://altairbasic.org/int_dis_5.htm#InputNext) |  |
| 0338 | DF | ResetInput | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 0339 | CD8A05 |  | CALL [NewLine](http://altairbasic.org/int_dis_14.htm#NewLine) |  |
| 033C | 211301 | InputLine | LXI H,[LINE\_BUFFER](http://altairbasic.org/int_dis_3.htm#LINE_BUFFER) |  |
| 033F | 0601 |  | MVI B,01 |  |
| Get a character and jump out of here if user has pressed 'Enter'. |
| 0341 | CD8203 | InputNext | CALL [InputChar](http://altairbasic.org/int_dis_6.htm#InputChar) |  |
| 0344 | FE0D |  | CPI '\r' |  |
| 0346 | CA8505 |  | JZ [TerminateInput](http://altairbasic.org/int_dis_14.htm#TerminateInput) |  |
| If user has not given a printable character, then loop back until they do. |
| 0349 | FE20 |  | CPI ' ' | If < ' ' |
| 034B | DA4103 |  | JC [InputNext](http://altairbasic.org/int_dis_5.htm#InputNext) | or |
| 034E | FE7D |  | CPI 7D | > '}' |
| 0350 | D24103 |  | JNC [InputNext](http://altairbasic.org/int_dis_5.htm#InputNext) | then loop back. |
| Deal with line-abort. The character for this key was '@'. |
| 0353 | FE40 |  | CPI '@' |  |
| 0355 | CA3803 |  | JZ [ResetInput](http://altairbasic.org/int_dis_5.htm#ResetInput) |  |
| Deal with backspace. The character for this key was '\_'. |
| 0358 | FE5F |  | CPI '\_' |  |
| 035A | CA3203 |  | JZ [Backspace](http://altairbasic.org/int_dis_5.htm#Backspace) |  |
| A normal character has been pressed. Here we store it in LINE\_BUFFER, only we don't if the terminal width has been exceeded. If the terminal width is exceeded then we ring the bell (ie print ASCII code 7) and ignore the char. Finally we loop back for the next input character. |
| 035D | 4F |  | MOV C,A |  |
| 035E | 78 |  | MOV A,B |  |
| 035F | FE48 |  | CPI 48 |  |
| 0361 | 3E07 |  | MVI A,07 |  |
| 0363 | D26A03 |  | JNC 036A |  |
| 0366 | 79 |  | MOV A,C | Write char to LINE\_BUFFER. |
| 0367 | 71 |  | MOV M,C |  |
| 0368 | 23 |  | INX H |  |
| 0369 | 04 |  | INR B |  |
| 036A | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 036B | C34103 |  | JMP [InputNext](http://altairbasic.org/int_dis_5.htm#InputNext) |  |

## 1.6 Terminal I/O

### OutChar\_tail

Prints a character to the terminal. On entry, the char to be printed is on the stack and A holds TERMINAL\_X. If the current line is up to the maximum width then we print a new line and update the terminal position. Then we print the character - to do this we loop until the device is ready to receive a char and then write it out.

|  |
| --- |
| If terminal width reached then print new line. Note that NewLine returns with A=0, so TERMINAL\_X gets reset properly. |
| 036E | FE4E | OutChar\_tail | CPI 48 |  |
| 0370 | CC8A05 |  | CZ [NewLine](http://altairbasic.org/int_dis_14.htm#NewLine) |  |
| 0373 | 3C |  | INR A |  |
| 0374 | 322700 |  | STA [TERMINAL\_X](http://altairbasic.org/int_dis_1.htm#TERMINAL_X) |  |
| Wait for terminal device to become ready for output. |
| 0377 | DB00 | WaitTermReady | IN 00 |  |
| 0379 | E680 |  | ANI 80 |  |
| 037B | C27703 |  | JNZ [WaitTermReady](http://altairbasic.org/int_dis_6.htm#WaitTermReady) |  |
| Get char off stack and write it out. |
| 037E | F1 |  | POP PSW |  |
| 037F | D301 |  | OUT 01 |  |
| 0381 | C9 |  | RET |  |

### InputChar

Gets one char of input from the user.

|  |
| --- |
| Firstly loop until device ready |
| 0382 | DB00 | InputChar | IN 00 |  |
| 0384 | E601 |  | ANI 01 |  |
| 0386 | C28203 |  | JNZ [InputChar](http://altairbasic.org/int_dis_6.htm#InputChar) |  |
| Read char from IO bus, reset bit 7, and return. |
| 0389 | DB01 |  | IN 01 |  |
| 038B | E67F |  | ANI 7F |  |
| 038D | C9 |  | RET |  |

## 1.7 LIST Handler

### List

Lists the program. As the stored program is in tokenised form (ie keywords are represented with single byte numeric IDs) LIST is more complex than a simple memory dump. When it meets a keyword ID it looks it up in the keywords table and prints it.

|  |
| --- |
| Get the line number argument into DE and error back if a non-numeric argument was given. |
| 038E | CD9D04 | List | CALL [LineNumberFromStr](http://altairbasic.org/int_dis_11.htm#LineNumberFromStr) |  |
| 0391 | C0 |  | RNZ |  |
| 0392 | C1 |  | POP B | ?why get return address? |
| From the line number in DE, get the address of the starting program line onto the stack. |
| 0393 | CD7D02 |  | CALL [FindProgramLine](http://altairbasic.org/int_dis_5.htm#FindProgramLine) |  |
| 0396 | C5 |  | PUSH B |  |
| Pop the current program line address into HL, and get the address of the \*next\* program line into BC.. |
| 0397 | E1 | ListNextLine | POP H |  |
| 0398 | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) |  |
| 0399 | C1 |  | POP B |  |
| If we've reached the null line at the end of the program, then exit. |
| 039A | 78 |  | MOV A,B |  |
| 039B | B1 |  | ORA C |  |
| 039C | CAF901 |  | JZ [Main](http://altairbasic.org/int_dis_5.htm#Main) |  |
| Allow user a chance to stop the program listing. |
| 039F | CD7304 |  | CALL [TestBreakKey](http://altairbasic.org/int_dis_10.htm#TestBreakKey) |  |
| 03A2 | C5 |  | PUSH B |  |
| 03A3 | CD8A05 |  | CALL [NewLine](http://altairbasic.org/int_dis_14.htm#NewLine) |  |
| Get current program line number into HL, and push current program line ptr onto the stack. |
| 03A6 | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) |  |
| 03A7 | E3 |  | XTHL |  |
| Print the line number and prepare to print a space.. |
| 03A8 | CD370B |  | CALL [PrintInt](http://altairbasic.org/math_dis_10.htm#PrintInt) |  |
| 03AB | 3E20 |  | MVI A,' ' |  |
| Restore current line ptr to HL, print current character, advance current line ptr and |
| 03AD | E1 |  | POP H |  |
| 03AE | DF | ListChar | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 03AF | 7E |  | MOV A,M |  |
| 03B0 | B7 |  | ORA A |  |
| 03B1 | 23 |  | INX H |  |
| 03B2 | CA9703 |  | JZ [ListNextLine](http://altairbasic.org/int_dis_7.htm#ListNextLine) |  |
| 03B5 | F2AE03 |  | JP [ListChar](http://altairbasic.org/int_dis_7.htm#ListChar) |  |
| Bit 7 of A is set, indicating a keyword ID. So we need to look the keyword up in the table and print it. |
| 03B8 | D67F |  | SUI 7F | A is now keyword index + 1. |
| 03BA | 4F |  | MOV C,A |  |
| 03BB | E5 |  | PUSH H |  |
| 03BC | 115700 |  | LXI D,[KEYWORDS](http://altairbasic.org/int_dis_2.htm#KEYWORDS) |  |
| 03BF | D5 |  | PUSH D |  |
| Find the start of the next keyword. |
| 03C0 | 1A | ToNextKeyword | LDAX D |  |
| 03C1 | 13 |  | INX D |  |
| 03C2 | B7 |  | ORA A |  |
| 03C3 | F2C003 |  | JP [ToNextKeyword](http://altairbasic.org/int_dis_7.htm#ToNextKeyword) |  |
| Decrement keyword index and restore start of previous keyword to HL. If this is not yet the keyword we want, then loop back. |
| 03C6 | 0D |  | DCR C |  |
| 03C7 | E1 |  | POP H |  |
| 03C8 | C2BF03 |  | JNZ [ToNextKeyword](http://altairbasic.org/int_dis_7.htm#ToNextKeyword)-1 |  |
| Print the keyword. Note that printing of the last character is deferred to ListChar in the main loop. |
| 03CB | 7E | PrintKeyword | MOV A,M |  |
| 03CC | B7 |  | ORA A |  |
| 03CD | FAAD03 |  | JM [ListChar](http://altairbasic.org/int_dis_7.htm#ListChar)-1 |  |
| 03D0 | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 03D1 | 23 |  | INX H |  |
| 03D2 | C3CB03 |  | JMP [PrintKeyword](http://altairbasic.org/int_dis_7.htm#PrintKeyword) |  |

## 1.8 FOR Handler

### For

Although FOR indicates the beginning of a program loop, the handler only gets called the once. Subsequent iterations of the loop return to the following statement or program line, not the FOR statement itself.

|  |
| --- |
| First we call LET to assign the initial value to the variable. On return, HL points to the next bit of program (the TO clause with any luck) |
| 03D5 | CD0205 | For | CALL [Let](http://altairbasic.org/int_dis_12.htm#Let) |  |
| Stick program ptr onto stack. We lose the return address, since we don't need it as this function conveniently falls into ExecNext by itself. |
| 03D8 | E3 |  | XTHL |  |
| 03D9 | CD9201 |  | CALL [GetFlowPtr](http://altairbasic.org/int_dis_4.htm#GetFlowPtr) |  |
| Get program ptr into DE. |
| 03DC | D1 |  | POP D |  |
| 03DD | C2E203 |  | JNZ 03E2 |  |
| 03E0 | 09 |  | DAD B |  |
| 03E1 | F9 |  | SPHL |  |
| HL=prog ptr, DE=stack. Here we check we've at least 8\*4 bytes of space to use for the flow struct. |
| 03E2 | EB |  | XCHG |  |
| 03E3 | 0E08 |  | MVI C,08 |  |
| 03E5 | CDB601 |  | CALL [CheckEnoughVarSpace](http://altairbasic.org/int_dis_4.htm#CheckEnoughVarSpace) |  |
| Get pointer to end of statement (or end of program line) onto stack. This is the prog ptr that NEXT will return to. |
| 03E8 | E5 |  | PUSH H |  |
| 03E9 | CDF504 |  | CALL [FindNextStatement](http://altairbasic.org/int_dis_11.htm#FindNextStatement) |  |
| 03EC | E3 |  | XTHL |  |
| Push current line number onto stack. |
| 03ED | E5 |  | PUSH H |  |
| 03EE | 2A6101 |  | LHLD [CURRENT\_LINE](http://altairbasic.org/int_dis_3.htm#CURRENT_LINE) |  |
| 03F1 | E3 |  | XTHL |  |
| Syntax check that TO clause is next. |
| 03F2 | CF |  | RST [SyntaxCheck](http://altairbasic.org/int_dis_1.htm#SyntaxCheck) |  |
| 03F3 | 95 |  | KWID\_TO |  |
| Evaluate expression following 'TO', and push the result of that expression (a floating point number of course) on the stack |
| 03F4 | CD8A06 |  | CALL [EvalExpression](http://altairbasic.org/int_dis_17.htm#EvalExpression) |  |
| 03F7 | E5 |  | PUSH H |  |
| 03F8 | CD1D0A |  | CALL [FCopyToBCDE](http://altairbasic.org/math_dis_6.htm#FCopyToBCDE) |  |
| 03FB | E1 |  | POP H |  |
| 03FC | C5 |  | PUSH B |  |
| 03FD | D5 |  | PUSH D |  |
| Initialise the STEP value in BCDE to 1. |
| 03FE | 010081 |  | LXI B,8100 |  |
| 0401 | 51 |  | MOV D,C |  |
| 0402 | 5A |  | MOV E,D |  |
| If a STEP clause has not been given, skip ahead with the direction byte (in A) as 0x01. |
| 0403 | 7E |  | MOV A,M |  |
| 0404 | FE97 |  | CPI KWID\_STEP |  |
| 0406 | 3E01 |  | MVI A,01 |  |
| 0408 | C21404 |  | JNZ [PushStepValue](http://altairbasic.org/int_dis_8.htm#PushStepValue) |  |
| STEP clause has been given so we evaluate it and get it into BCDE. The sign of this value becomes the direction byte (0x01 for fowards, 0xFF for backwards). |
| 040B | CD8B06 |  | CALL [EvalExpression](http://altairbasic.org/int_dis_17.htm#EvalExpression)+1 |  |
| 040E | E5 |  | PUSH H |  |
| 040F | CD1D0A |  | CALL [FCopyToBCDE](http://altairbasic.org/math_dis_6.htm#FCopyToBCDE) |  |
| 0412 | EF |  | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |
| 0413 | E1 |  | POP H |  |
| Push the STEP value onto the stack. |
| 0414 | C5 | PushStepValue | PUSH B |  |
| 0415 | D5 |  | PUSH D |  |
| Push A onto stack. (A=1 if no step clause, else ???) |
| 0416 | F5 |  | PUSH PSW |  |
| 0417 | 33 |  | INX SP |  |
| Push the prog ptr to the end of the FOR statement (kept on PROG\_PTR\_TEMP) on the stack. |
| 0418 | E5 |  | PUSH H |  |
| 0419 | 2A5D01 |  | LHLD [PROG\_PTR\_TEMP](http://altairbasic.org/int_dis_3.htm#PROG_PTR_TEMP) |  |
| 041C | E3 |  | XTHL |  |
| Push KWID\_FOR onto the stack, and fall into ExecNext |
| 041D | 0681 | EndOfForHandler | MVI B,KWID\_FOR |  |
| 041F | C5 |  | PUSH B |  |
| 0420 | 33 |  | INX SP |  |

## 1.9 Execution

### ExecNext

Having exec'd one statement, this block moves on to the next statement in the line or the next line if there are no more statements on the current line.

|  |
| --- |
| Give user a chance to break the execution. |
| 0421 | CD7304 | ExecNext | CALL [TestBreakKey](http://altairbasic.org/int_dis_10.htm#TestBreakKey) |  |
| If we have a ':' then that's a statement seperator (which allows multiple statements on the same line) so we jump to Exec to run it. |
| 0424 | 7E |  | MOV A,M |  |
| 0425 | FE3A |  | CPI ':' |  |
| 0427 | CA3E04 |  | JZ [Exec](http://altairbasic.org/int_dis_9.htm#Exec) |  |
| Well it wasn't a statement seperator, therefore it must be the null byte terminating the line otherwise it's a syntax error. |
| 042A | B7 |  | ORA A |  |
| 042B | C2D001 |  | JNZ [SyntaxError](http://altairbasic.org/int_dis_4.htm#SyntaxError) |  |
| The next two bytes should be the address of the next line. We can ignore this, since lines are stored consecutively, but must jump back to Main if we've reached the end of the program |
| 042E | 23 |  | INX H |  |
| 042F | 7E |  | MOV A,M |  |
| 0430 | 23 |  | INX H |  |
| 0431 | B6 |  | ORA M |  |
| 0432 | 23 |  | INX H |  |
| 0433 | CAF901 |  | JZ [Main](http://altairbasic.org/int_dis_5.htm#Main) |  |
| Get the number of the following program line, store it in CURRENT\_LINE and fall into Exec to run it. |
| 0436 | 5E |  | MOV E,M |  |
| 0437 | 23 |  | INX H |  |
| 0438 | 56 |  | MOV D,M |  |
| 0439 | EB |  | XCHG |  |
| 043A | 226101 |  | SHLD [CURRENT\_LINE](http://altairbasic.org/int_dis_3.htm#CURRENT_LINE) |  |
| 043D | EB |  | XCHG |  |

### Exec

Executes a statement of BASIC code pointed to by HL.

|  |
| --- |
| Get first character of statement. |
| 043E | D7 | Exec | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| Stick address of ExecNext onto stack so we can return to it. |
| 043F | 112104 |  | LXI D,[ExecNext](http://altairbasic.org/int_dis_9.htm#ExecNext) |  |
| 0442 | D5 |  | PUSH D |  |
| Return immediately if this is an empty statement. |
| 0443 | C8 |  | RZ |  |
| See if we lead with a keyword, ie first byte is >=0x80. If it isn't then it might be an implicit LET statement (ie where the LET has not been typed) and so we defer to the LET handler to deal with it. |
| 0444 | D680 |  | SUI 80 |  |
| 0446 | DA0205 |  | JC [Let](http://altairbasic.org/int_dis_12.htm#Let) |  |
| If it's not a general keyword then that's a syntax error. |
| 0449 | FE14 |  | CPI 14 |  |
| 044B | D2D001 |  | JNC [SyntaxError](http://altairbasic.org/int_dis_4.htm#SyntaxError) |  |
| Ok so we have a general keyword, so here we get the address of the handling function into HL, preserving the program ptr in DE. |
| 044E | 07 |  | RLC | BC = A\*2 |
| 044F | 4F |  | MOV C,A |  |
| 0450 | 0600 |  | MVI B,00 |  |
| 0452 | EB |  | XCHG |  |
| 0453 | 21D200 |  | LXI H,[KW\_GENERAL\_FNS](http://altairbasic.org/int_dis_2.htm#KW_GENERAL_FNS) |  |
| 0456 | 09 |  | DAD B |  |
| Read the keyword handler function address into BC. |
| 0457 | 4E |  | MOV C,M |  |
| 0458 | 23 |  | INX H |  |
| 0459 | 46 |  | MOV B,M |  |
| Stick keyword handler function address on stack, restore program ptr to HL, get the next character of the statement, and return to the keyword handler function. |
| 045A | C5 |  | PUSH B |  |
| 045B | EB |  | XCHG |  |
| 045C | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 045D | C9 |  | RET |  |

## 1.10 More Utility Functions

### NextChar\_tail

|  |
| --- |
| Jump back if we've got a space char, since we're not interested in spaces. |
| 045E | FE20 | NextChar\_tail | CPI ' ' |  |
| 0460 | CA1000 |  | JZ [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| Set carry flag if char >= '0'. |
| 0463 | FE30 |  | CPI '0' |  |
| 0465 | 3F |  | CMC |  |
| Test for null char without affecting carry flag. |
| 0466 | 3C |  | INR A |  |
| 0467 | 3D |  | DCR A |  |
| 0468 | C9 |  | RET |  |

### Restore

Resets the data pointer to just before the start of the program.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0469 | EB | Restore | XCHG |  |
| 046A | 2A6501 |  | LHLD [PROGRAM\_BASE](http://altairbasic.org/int_dis_3.htm#PROGRAM_BASE) |  |
| 046D | 2B |  | DCX H |  |
| 046E | 226D01 |  | SHLD [DATA\_PROG\_PTR](http://altairbasic.org/int_dis_3.htm#DATA_PROG_PTR) |  |
| 0471 | EB |  | XCHG |  |
| 0472 | C9 |  | RET |  |

### TestBreakKey

Apparently the Altair had a 'break' key, to break program execution. This little function tests to see if the terminal input device is ready, and returns if it isn't. If it is ready (ie user has pressed a key) then it reads the char from the device, compares it to the code for the break key (0x03) and jumps to Stop. Since the first instruction at Stop is RNZ, this will return at once if the user pressed some other key.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0473 | DB00 | TestBreakKey | IN 00 | Exit if no key pressed. |
| 0475 | E601 |  | ANI 01 |  |
| 0477 | C0 |  | RNZ |  |
| 0478 | CD8203 |  | CALL [InputChar](http://altairbasic.org/int_dis_6.htm#InputChar) |  |
| 047B | FE03 |  | CPI 03 | Break key? |
| 047D | C3F701 |  | JMP [Stop](http://altairbasic.org/int_dis_5.htm#Stop) |  |

### CharIsAlpha

If character pointed to by HL is alphabetic, the carry flag is reset otherwise set.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0480 | 7E | CharIsAlpha | MOV A,M |  |
| 0481 | FE41 |  | CPI 'A' |  |
| 0483 | D8 |  | RC |  |
| 0484 | FE5B |  | CPI 'Z'+1 |  |
| 0486 | 3F |  | CMC |  |
| 0487 | C9 |  | RET |  |

### GetSubscript

Gets the subscript of an array variable encountered in an expression or a DIM declaration. The subscript is returned as a positive integer in CDE.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0488 | D7 | GetSubscript | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0489 | CD8A06 |  | CALL [EvalExpression](http://altairbasic.org/int_dis_17.htm#EvalExpression) |  |
| If subscript is negative then jump to FC error below. |
| 048C | EF |  | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |
| 048D | FA9804 |  | JM [FunctionCallError](http://altairbasic.org/int_dis_10.htm#FunctionCallError) |  |
| Likewise, if subscript is >32767 then fall into FC error, otherwise exit to FAsInteger. |
| 0490 | 3A7201 |  | LDA [FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 0493 | FE90 |  | CPI 90 |  |
| 0495 | DA770A |  | JC [FAsInteger](http://altairbasic.org/math_dis_8.htm#FAsInteger) |  |
| Invalid function call (FC) error.. |
| 0498 | 1E08 | FunctionCallError | MVI E,08 |  |
| 049A | C3D501 |  | JMP [Error](http://altairbasic.org/int_dis_4.htm#Error) |  |

## 1.11 Jumping to Program Lines

### LineNumberFromStr

Gets a line number from a string pointer. The string pointer is passed in in HL, and the integer result is returned in DE. Leading spaces are skipped over, and it returns on finding the first non-digit. The largest possible line number is 65529 - it syntax errors out if the value of the first four digits is more then 6552.

One interesting feature of this function is that it returns with Z set if it found a valid number (or the string was empty), or NZ if the string didn't lead with a number.

|  |
| --- |
| Decrement string ptr (so we're pointing at preceding character) and initialise result to 0. |
| 049D | 2B | LineNumberFromStr | DCX H |  |
| 049E | 110000 |  | LXI D,0000 |  |
| Get next character and exit if it's not alphanumeric. |
| 04A1 | D7 | NextLineNumChar | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 04A2 | D0 |  | RNC |  |
| 04A3 | E5 |  | PUSH H |  |
| 04A4 | F5 |  | PUSH PSW | Preserve flags |
| Syntax Error out if line number is already > 6552. This is really erroring out of the line number is >65529, since the next digit has not been counted in yet. |
| 04A5 | 219819 |  | LXI H,1998 | Decimal 6552 |
| 04A8 | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) |  |
| 04A9 | DAD001 |  | JC [SyntaxError](http://altairbasic.org/int_dis_4.htm#SyntaxError) |  |
| Multiply result by 10. |
| 04AC | 62 |  | MOV H,D |  |
| 04AD | 6B |  | MOV L,E |  |
| 04AE | 19 |  | DAD D |  |
| 04AF | 29 |  | DAD H |  |
| 04B0 | 19 |  | DAD D |  |
| 04B1 | 29 |  | DAD H |  |
| 04B2 | F1 |  | POP PSW |  |
| Add this digit's value to the result and jump back. |
| 04B3 | D630 |  | SUI '0' |  |
| 04B5 | 5F |  | MOV E,A |  |
| 04B6 | 1600 |  | MVI D,00 |  |
| 04B8 | 19 |  | DAD D |  |
| 04B9 | EB |  | XCHG |  |
| 04BA | E1 |  | POP H |  |
| 04BB | C3A104 |  | JMP [NextLineNumChar](http://altairbasic.org/int_dis_11.htm#NextLineNumChar) |  |

### Gosub

Gosub sets up a flow struct on the stack and then falls into Goto. The flow struct is KWID\_GOSUB, preceded by the line number of the gosub statement, in turn preceded by prog ptr to just after the gosub statement.

|  |
| --- |
| Check we've got at least 12 bytes of memory spare. |
| 04BE | 0E03 | Gosub | MVI C,03 |  |
| 04C0 | CDB601 |  | CALL [CheckEnoughVarSpace](http://altairbasic.org/int_dis_4.htm#CheckEnoughVarSpace) |  |
| Preserve return address in BC. |
| 04C3 | C1 |  | POP B |  |
| Push prog ptr. |
| 04C4 | E5 |  | PUSH H |  |
| Push the current line number on the stack so we can RETURN to it later. |
| 04C5 | E5 |  | PUSH H |  |
| 04C6 | 2A6101 |  | LHLD [CURRENT\_LINE](http://altairbasic.org/int_dis_3.htm#CURRENT_LINE) |  |
| 04C9 | E3 |  | XTHL |  |
| Push keyword ID of 'GOSUB' onto stack. Note we only push the single byte. |
| 04CA | 168C |  | MVI D,KWID\_GOSUB |  |
| 04CC | D5 |  | PUSH D |  |
| 04CD | 33 |  | INX SP |  |
| Push return address preserved in BC, and fall into GOTO. |
| 04CE | C5 |  | PUSH B |  |

### Goto

Sets program execution to continue from the line number argument.

|  |
| --- |
| Get line number argument in DE and return NZ indicating syntax error if the argument was a non-number . |
| 04CF | CD9D04 | Goto | CALL [LineNumberFromStr](http://altairbasic.org/int_dis_11.htm#LineNumberFromStr) |  |
| 04D2 | C0 |  | RNZ |  |
| Look up the address of the program line with the provided number, put that into HL, and return if the program line was found (ie it exists). |
| 04D3 | CD7D02 |  | CALL [FindProgramLine](http://altairbasic.org/int_dis_5.htm#FindProgramLine) |  |
| 04D6 | 60 |  | MOV H,B |  |
| 04D7 | 69 |  | MOV L,C |  |
| 04D8 | 2B |  | DCX H |  |
| 04D9 | D8 |  | RC |  |
| Carry flag was not set by FindProgramLine so Undefined Subroutine (US) error. |
| 04DA | 1E0E |  | MVI E,0E |  |
| 04DC | C3D501 |  | JMP [Error](http://altairbasic.org/int_dis_4.htm#Error) |  |

### Return

Returns program execution to the statement following the last GOSUB. Information about where to return to is kept on the stack in a flow struct (see notes).

|  |
| --- |
| No arguments allowed. |
| 04DF | C0 | Return | RNZ |  |
| Get pointer to flow struct on stack |
| 04E0 | 16FF |  | MVI D,FF |  |
| 04E2 | CD9201 |  | CALL [GetFlowPtr](http://altairbasic.org/int_dis_4.htm#GetFlowPtr) |  |
| 04E5 | F9 |  | SPHL |  |
| If the first byte of the flow struct is not KWID\_GOSUB (as placed there by the Gosub handler) then the gosub can't have happened so exit to a Return without Gosub (RG) error. |
| 04E6 | FE8C |  | CPI KWID\_GOSUB |  |
| 04E8 | 1E04 |  | MVI E,04 |  |
| 04EA | C2D501 |  | JNZ [Error](http://altairbasic.org/int_dis_4.htm#Error) |  |
| Get line number of Gosub statement from flow struct into CURRENT\_LINE. |
| 04ED | E1 |  | POP H |  |
| 04EE | 226101 |  | SHLD [CURRENT\_LINE](http://altairbasic.org/int_dis_3.htm#CURRENT_LINE) |  |
| Set return address to ExecNext and pop the prog ptr to just after the Gosub statement into HL. |
| 04F1 | 212104 |  | LXI H,[ExecNext](http://altairbasic.org/int_dis_9.htm#ExecNext) |  |
| 04F4 | E3 |  | XTHL |  |

Safe to fall into FindNextStatement, since we're already at the end of the line!...

### FindNextStatement

Finds the end of the statement or the end of the program line.

**BUG**: There is an interesting bug in this block, although it's harmless as by luck it's impossible to see it. The byte at 04F7 is 0x10, an illegal instruction, which is in turn followed by a NOP. This illegal instruction is almost certainly supposed to be 0x0E, so as to become the two-byte instruction MOV C,00. If this were the case it would make perfect sense as the loop reads ahead until it finds a null byte terminating the line or whatever the C register is loaded with.

04F7 is jumped to in two places - it is the REM handler, and also when an IF statement's condition evals to false and the rest of the line needs to be skipped. Luckily in both these cases, C just happens to be loaded with a byte that cannot occur in the program so the null byte marking the end of the line is found as expected.

|  |
| --- |
| Load C with the statement seperator character, a colon ':', and also LXI over the illegal instruction (which is a bug - see above). |
| 04F5 | 013A.. | FindNextStatement | LXI B,..3A |  |
| 04F7 | 10 | Rem | ??? |  |
| 04F8 | 00 |  | NOP |  |
| 04F9 | 7E | FindNextStatementLoop | MOV A,M |  |
| 04FA | B7 |  | ORA A |  |
| 04FB | C8 |  | RZ |  |
| 04FC | B9 |  | CMP C |  |
| 04FD | C8 |  | RZ |  |
| 04FE | 23 |  | INX H |  |
| 04FF | C3F904 |  | JMP [FindNextStatementLoop](http://altairbasic.org/int_dis_11.htm#FindNextStatementLoop) |  |

## 1.12 Assigning Variables

### Let

Assigns a value to a variable.

|  |
| --- |
| First call GetVar. This returns the address of the variable's value in DE, allocating space for it if it doesn't already exist. |
| 0502 | CD1B07 | Let | CALL [GetVar](http://altairbasic.org/int_dis_18.htm#GetVar) |  |
| Check that '=' follows the variable name. |
| 0505 | CF |  | RST [SyntaxCheck](http://altairbasic.org/int_dis_1.htm#SyntaxCheck) |  |
| 0506 | 9D |  | = |  |

### AssignVar

Assigns the result of the BASIC expression at HL to the variable pointed to by DE.

|  |
| --- |
| Preserve variable address on stack and get the variable value into FACCUM. |
| 0507 | D5 | AssignVar | PUSH D |  |
| 0508 | CD8A06 |  | CALL [EvalExpression](http://altairbasic.org/int_dis_17.htm#EvalExpression) |  |
| Swap (SP) and HL, so HL now holds the address of the variable's value and the program ptr is on the stack. |
| 050B | E3 |  | XTHL |  |
| Store variable address in PROG\_PTR\_TEMP, don't know why. |
| 050C | 225D01 |  | SHLD [PROG\_PTR\_TEMP](http://altairbasic.org/int_dis_3.htm#PROG_PTR_TEMP) |  |
| Store FACCUM (the expression result) in the variable and get the variable value address into DE. |
| 050F | E5 |  | PUSH H |  |
| 0510 | CD290A |  | CALL [FCopyToMem](http://altairbasic.org/math_dis_6.htm#FCopyToMem) |  |
| 0513 | D1 |  | POP D |  |
| Restore program ptr to HL and return. |
| 0514 | E1 |  | POP H |  |
| 0515 | C9 |  | RET |  |

## 1.13 IF Keyword Handler

### If

Evaluates a condition. A condition has three mandatory parts : a left-hand side expression, a comparison operator, and a right-hand side expression. Examples are 'A=2', 'B<=4' and so on.

The comparison operator is one or more of the three operators '>', '=', and '<'. Since these three operators can appear more than once, and in any order, the code does something rather clever to convert them to a single 'comparison operator value'. This value has bit 0 set if '>' is present, bit 1 for '=', and bit 2 for '<'. Thus the comparison operators '<=' and '=<' are both 6, likewise '>=' and '=>' are both 3, and '<>' is 5

You can therefore get away with stupid operators such as '>>>>>' (value 1, the same as a single '>') and '>=<' (value 7), the latter being particularly dense as it causes the condition to always evaluate to true.

|  |
| --- |
| Get left-hand side expression onto stack. Also get first byte of comparison operator into A. |
| 0516 | CD8A06 | If | CALL [EvalExpression](http://altairbasic.org/int_dis_17.htm#EvalExpression) |  |
| 0519 | 7E |  | MOV A,M |  |
| 051A | CD020A |  | CALL [FPush](http://altairbasic.org/math_dis_6.htm#FPush) |  |
| Get value for comparison operator. Fixme: This needs proper working out as I suspect it's brilliant! |
| 051D | 1600 |  | MVI D,00 |  |
| 051F | D69C | GetCompareOpLoop | SUI KWID\_> |  |
| 0521 | DA3205 |  | JC [GotCompareOp](http://altairbasic.org/int_dis_13.htm#GotCompareOp) |  |
| 0524 | FE03 |  | CPI 03 |  |
| 0526 | D23205 |  | JNC [GotCompareOp](http://altairbasic.org/int_dis_13.htm#GotCompareOp) |  |
| 0529 | FE01 |  | CPI 01 |  |
| 052B | 17 |  | RAL |  |
| 052C | B2 |  | ORA D |  |
| 052D | 57 |  | MOV D,A |  |
| 052E | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 052F | C31F05 |  | JMP [GetCompareOpLoop](http://altairbasic.org/int_dis_13.htm#GetCompareOpLoop) |  |
| If no comparison operator value, then syntax error out. |
| 0532 | 7A | GotCompareOp | MOV A,D |  |
| 0533 | B7 |  | ORA A |  |
| 0534 | CAD001 |  | JZ [SyntaxError](http://altairbasic.org/int_dis_4.htm#SyntaxError) |  |
| Preserve comparison operator value on stack and evaluate the right hand side of the condition. |
| 0537 | F5 |  | PUSH PSW |  |
| 0538 | CD8A06 |  | CALL [EvalExpression](http://altairbasic.org/int_dis_17.htm#EvalExpression) |  |
| Syntax check for the THEN keyword. |
| 053B | CF |  | RST [SyntaxCheck](http://altairbasic.org/int_dis_1.htm#SyntaxCheck) |  |
| 053C | 96 |  | KWID\_THEN |  |
|  |
| 053D | 2B |  | DCX H |  |
| 053E | F1 |  | POP PSW |  |
| 053F | C1 |  | POP B |  |
| 0540 | D1 |  | POP D |  |
| 0541 | E5 |  | PUSH H |  |
| 0542 | F5 |  | PUSH PSW |  |
| 0543 | CD4C0A |  | CALL [FCompare](http://altairbasic.org/math_dis_7.htm#FCompare) |  |
| 0546 | 3C |  | INR A |  |
| 0547 | 17 |  | RAL |  |
| 0548 | C1 |  | POP B |  |
| 0549 | A0 |  | ANA B |  |
| 054A | E1 |  | POP H |  |
| 054B | CAF704 |  | JZ 04F7 |  |
| Condition evaluated to True. Here we get the first character of the THEN statement, and if it's a digit then we jump to GOTO's handler as it's an implicit GOTO. Otherwise we jump to near the top of Exec to run the THEN statement. |
| 054E | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 054F | DACF04 |  | JC [Goto](http://altairbasic.org/int_dis_11.htm#Goto) |  |
| 0552 | C34304 |  | JMP [Exec](http://altairbasic.org/int_dis_9.htm#Exec)+5 |  |

## 1.14 Printing

### Print

Prints something! It can be an empty line, a single expression/literal, or multiple expressions/literals seperated by tabulation directives (comma, semi-colon, or the TAB keyword).

|  |
| --- |
| If end of line has been reached than exit via NewLine. Note that the handler function starts on the third line. |
| 0555 | 2B |  | DCX H |  |
| 0556 | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0557 | CA8A05 | Print | JZ [NewLine](http://altairbasic.org/int_dis_14.htm#NewLine) |  |
| 055A | C8 |  | RZ |  |
| If the start of a string literal (ie characters enclosed in double-quotes), then print it. If this function returns having found the end of the line then jump back (instead of simply RZ-ing) to the top so that NewLine gets called. |
| 055B | FE22 |  | CPI '"' |  |
| 055D | CCA205 |  | CZ [PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |
| 0560 | CA5505 |  | JZ [Print](http://altairbasic.org/int_dis_14.htm#Print)-2 |  |
| If TAB( keyword id, then jump to the Tab() handler. |
| 0563 | FE94 |  | CPI KWID\_TAB |  |
| 0565 | CAC705 |  | JZ [Tab](http://altairbasic.org/int_dis_14.htm#Tab) |  |
| 0568 | E5 |  | PUSH H |  |
| If we have a comma, then that means we have to move to the next tab break. |
| 0569 | FE2C |  | CPI ',' |  |
| 056B | CAB305 |  | JZ [ToNextTabBreak](http://altairbasic.org/int_dis_14.htm#ToNextTabBreak) |  |
| If we have a semi-colon, then we don't need to do anything - just jump to the end of Tab() which restores the prog ptr and loops back to Print-2. |
| 056E | FE3B |  | CPI ';' |  |
| 0570 | CADF05 |  | JZ [ExitTab](http://altairbasic.org/int_dis_14.htm#ExitTab) |  |
| We've got an expression to print. First we discard the prog ptr kept on the stack, and evaluate the expression to print. |
| 0573 | C1 |  | POP B |  |
| 0574 | CD8A06 |  | CALL [EvalExpression](http://altairbasic.org/int_dis_17.htm#EvalExpression) |  |
| Preserve the prog ptr and convert the numeric expression result (in FACCUM of course) to a string and print it.. |
| 0577 | E5 |  | PUSH H |  |
| 0578 | CD420B |  | CALL [FOut](http://altairbasic.org/math_dis_10.htm#FOut) |  |
| 057B | CDA305 |  | CALL [PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |
| Print a space. Expressions are printed with a trailing space, don't know why. |
| 057E | 3E20 |  | MVI A,' ' |  |
| 0580 | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| Restore the prog ptr and jump back to continue with the rest of the print statement. |
| 0581 | E1 |  | POP H |  |
| 0582 | C35505 |  | JMP [Print](http://altairbasic.org/int_dis_14.htm#Print)-2 |  |

### TerminateInput

HL points to just beyond the last byte of a line of user input. Here we write a null byte to terminate it, reset HL to point to the start of the input line buffer, then fall into NewLine.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0585 | 3600 | TerminateInput | MVI M,00 |  |
| 0587 | 211201 |  | LXI H,[LINE\_BUFFER](http://altairbasic.org/int_dis_3.htm#LINE_BUFFER)-1 |  |

### NewLine

Prints carriage return + line feed, plus a series of nulls which was probably due to some peculiarity of the teletypes of the day.

|  |
| --- |
| Print carriage return and save the ASCII value 13 to TERMINAL\_X |
| 058A | 3E0D | NewLine | MVI A,'\r' |  |
| 058C | 322700 |  | STA [TERMINAL\_X](http://altairbasic.org/int_dis_1.htm#TERMINAL_X) |  |
| 058F | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| Print line feed. |
| 0590 | 3E0A |  | MVI A,'\n' |  |
| 0592 | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| Print nulls, the number of which is taken from TERMINAL\_Y. TERMINAL\_X is reset to 0. |
| 0593 | 3A2600 |  | LDA [TERMINAL\_Y](http://altairbasic.org/int_dis_1.htm#TERMINAL_Y) |  |
| 0596 | 3D | PrintNullLoop | DCR A |  |
| 0597 | 322700 |  | STA [TERMINAL\_X](http://altairbasic.org/int_dis_1.htm#TERMINAL_X) |  |
| 059A | C8 |  | RZ |  |
| 059B | F5 |  | PUSH PSW |  |
| 059C | AF |  | XRA A |  |
| 059D | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 059E | F1 |  | POP PSW |  |
| 059F | C39605 |  | JMP [PrintNullLoop](http://altairbasic.org/int_dis_14.htm#PrintNullLoop) |  |

### PrintString

Prints a string to the terminal. Printing stops when a null byte or a '"' character is reached.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 05A2 | 23 |  | INX H |  |
| 05A3 | 7E | PrintString | MOV A,M |  |
| 05A4 | B7 |  | ORA A |  |
| 05A5 | C8 |  | RZ |  |
| 05A6 | 23 |  | INX H |  |
| 05A7 | FE22 |  | CPI '"' |  |
| 05A9 | C8 |  | RZ |  |
| 05AA | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 05AB | FE0D |  | CPI '\r' |  |
| 05AD | CC8A05 |  | CZ [NewLine](http://altairbasic.org/int_dis_14.htm#NewLine) |  |
| 05B0 | C3A305 |  | JMP [PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |

### ToNextTabBreak

Calculate how many spaces are needed to get us to the next tab-break then jump to PrintSpaces to do it.

|  |
| --- |
| If current character output position is beyond the last tab-break column, then call NewLine and jump to ExitTab. |
| 05B3 | 3A2700 | ToNextTabBreak | LDA [TERMINAL\_X](http://altairbasic.org/int_dis_1.htm#TERMINAL_X) |  |
| 05B6 | FE38 |  | CPI 38 |  |
| 05B8 | D48A05 |  | CNC [NewLine](http://altairbasic.org/int_dis_14.htm#NewLine) |  |
| 05BB | D2DF05 |  | JNC [ExitTab](http://altairbasic.org/int_dis_14.htm#ExitTab) |  |
| Get the number of spaces required to reach the next tab break. A = (14 - (A % 14)) - 1. |
| 05BE | D60E | CalcSpaceCount | SUI 0E |  |
| 05C0 | D2BE05 |  | JNC [CalcSpaceCount](http://altairbasic.org/int_dis_14.htm#CalcSpaceCount) |  |
| 05C3 | 2F |  | CMA |  |
| Jump to print the required number of spaces. |
| 05C4 | C3D605 |  | JMP [PrintSpaces](http://altairbasic.org/int_dis_14.htm#PrintSpaces) |  |

### Tab

Tabulation. The TAB keyword takes an integer argument denoting the absolute column to print spaces up to.

|  |
| --- |
| Get positive integer argument (the function called errors out if it's not a positive integer) into E |
| 05C7 | CD8804 | Tab | CALL [GetSubscript](http://altairbasic.org/int_dis_10.htm#GetSubscript) |  |
| Syntax check for closing bracket. |
| 05CA | CF |  | RST [SyntaxCheck](http://altairbasic.org/int_dis_1.htm#SyntaxCheck) |  |
| 05CB | 29 |  | ')' |  |
| Preserve prog ptr (minus one cos it gets incremented later). |
| 05CC | 2B |  | DCX H |  |
| 05CD | E5 |  | PUSH H |  |
| Get the number of spaces we need to print. This number is the argument in E minus TERMINAL\_X. If this is 0 or a negative number then we can't print any spaces so jump ahead to ExitTab. Note the math is a bit backward - this is a better way of doing it, memory-wise. |
| 05CE | 3A2700 |  | LDA [TERMINAL\_X](http://altairbasic.org/int_dis_1.htm#TERMINAL_X) |  |
| 05D1 | 2F |  | CMA |  |
| 05D2 | 83 |  | ADD E |  |
| 05D3 | D2DF05 |  | JNC [ExitTab](http://altairbasic.org/int_dis_14.htm#ExitTab) |  |
| Print A+1 spaces. |
| 05D6 | 3C | PrintSpaces | INR A |  |
| 05D7 | 47 |  | MOV B,A |  |
| 05D8 | 3E20 |  | MVI A,' ' |  |
| 05DA | DF | PrintSpaceLoop | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 05DB | 05 |  | DCR B |  |
| 05DC | C2DA05 |  | JNZ [PrintSpaceLoop](http://altairbasic.org/int_dis_14.htm#PrintSpaceLoop) |  |
| Restore the prog ptr, get the next character of program and jump back to the Print keyword handler. |
| 05DF | E1 | ExitTab | POP H |  |
| 05E0 | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 05E1 | C35A05 |  | JMP [Print](http://altairbasic.org/int_dis_14.htm#Print)+3 |  |

## 1.15 INPUT & READ Handlers

### Input

Let user input a number at a '?' prompt.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 05E4 | E5 | Input | PUSH H |  |
| Error out if in direct mode (ie not in a program). To do this we get the current line number + 1. If it's zero, then we're in direct mode. |
| 05E5 | 2A6101 |  | LHLD [CURRENT\_LINE](http://altairbasic.org/int_dis_3.htm#CURRENT_LINE) |  |
| 05E8 | 1E16 |  | MVI E,16 |  |
| 05EA | 23 |  | INX H |  |
| 05EB | 7D |  | MOV A,L |  |
| 05EC | B4 |  | ORA H |  |
| 05ED | CAD501 |  | JZ [Error](http://altairbasic.org/int_dis_4.htm#Error) |  |
| Get the input from the user and jump to Read+5. |
| 05F0 | CDC202 |  | CALL [InputLineWith'?'](http://altairbasic.org/int_dis_5.htm#InputLineWithxQx) |  |
| 05F3 | C3FB05 |  | JMP 05FB |  |

### Read

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 05F6 | E5 | Read | PUSH H |  |
| 05F7 | 2A6D01 |  | LHLD [DATA\_PROG\_PTR](http://altairbasic.org/int_dis_3.htm#DATA_PROG_PTR) |  |
| 05FA | F6AF |  | ORI AF |  |
| 05FB | AF |  | XRA A |  |
| 05FC | 325C01 |  | STA [INPUT\_OR\_READ](http://altairbasic.org/int_dis_3.htm#INPUT_OR_READ) |  |
| Preserve data prog ptr on stack and restore prog ptr to HL. This should point to the name of the variable to read data into. Note we also LXI over the syntax check for a comma that's done on subsequent reads. |
| 05FF | E3 |  | XTHL |  |
| 0600 | 01.... |  | LXI B,.... |  |
| 0601 | CF | ReadNext | RST [SyntaxCheck](http://altairbasic.org/int_dis_1.htm#SyntaxCheck) |  |
| 0602 | 2C |  | ',' |  |
| Get variable value address in DE. |
| 0603 | CD1B07 |  | CALL [GetVar](http://altairbasic.org/int_dis_18.htm#GetVar) |  |
| Preserve prog ptr and get data prog ptr into HL. |
| 0606 | E3 |  | XTHL |  |
| Preserve variable value address on stack. |
| 0607 | D5 |  | PUSH D |  |
| Get byte of data part of program. If this is a comma seperator then we've found our data item and can jump ahead to GotDataItem |
| 0608 | 7E |  | MOV A,M |  |
| 0609 | FE2C |  | CPI ',' |  |
| 060B | CA2006 |  | JZ [GotDataItem](http://altairbasic.org/int_dis_15.htm#GotDataItem) |  |
| If the next byte of data is not a null byte terminating the line then syntax error out. |
| 060E | B7 |  | ORA A |  |
| 060F | C2D001 |  | JNZ [SyntaxError](http://altairbasic.org/int_dis_4.htm#SyntaxError) |  |
| If we're READ'ing data then jump ahead to find the next DATA statement. |
| 0612 | 3A5C01 |  | LDA [INPUT\_OR\_READ](http://altairbasic.org/int_dis_3.htm#INPUT_OR_READ) |  |
| 0615 | B7 |  | ORA A |  |
| 0616 | 23 |  | INX H |  |
| 0617 | C23606 |  | JNZ [NextDataLine](http://altairbasic.org/int_dis_15.htm#NextDataLine)+1 |  |
| We've been called by the INPUT handler, and we have more inputs to take - the interpreter allows 'INPUT A,B,C' -type statement. So here we get the next input, only Bill has made a mistake here - he prints an unnecessary '?' , so the user gets two question marks for all inputs after the first one. |
| 061A | 3E3F |  | MVI A,'?' |  |
| 061C | DF |  | RST [OutChar](http://altairbasic.org/int_dis_1.htm#OutChar) |  |
| 061D | CDC202 |  | CALL [InputLineWith'?'](http://altairbasic.org/int_dis_5.htm#InputLineWithxQx) |  |
| Restore variable address, advance the data ptr so it points to the start of the next data item, and assign the data item to the variable. |
| 0620 | D1 | GotDataItem | POP D |  |
| 0621 | 23 |  | INX H |  |
| 0622 | CD0705 |  | CALL [AssignVar](http://altairbasic.org/int_dis_12.htm#AssignVar) |  |
| Get prog ptr off stack, and push data prog ptr, decrement prog ptr because AssignVar automatically advanced it, which we don't want. |
| 0625 | E3 |  | XTHL |  |
| 0626 | 2B |  | DCX H |  |
| Get next char of READ statement and jump back to ReadNext if the end of the line has not been reached. |
| 0627 | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0628 | C20106 |  | JNZ [ReadNext](http://altairbasic.org/int_dis_15.htm#ReadNext) |  |
| End of READ statement reached. |
| 062B | D1 |  | POP D |  |
| 062C | 3A5C01 |  | LDA [INPUT\_OR\_READ](http://altairbasic.org/int_dis_3.htm#INPUT_OR_READ) |  |
| 062F | B7 |  | ORA A |  |
| 0630 | C8 |  | RZ |  |
| 0631 | EB |  | XCHG |  |
| 0632 | C26E04 |  | JNZ 046E |  |
| Loop to find the next DATA line. Here we get the data prog ptr off the stack, and PushNextWord so the address of the next line is on the stack. |
| 0635 | E1 | NextDataLine | POP H |  |
| 0636 | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) |  |
| PushNextWord has pushed the address of the next line on the stack, but it has also left this address in BC. Here we test to see if this address is null (ie the null line at the end of the program has been reached) and if so then 'Out of Data' (OD) error. |
| 0637 | 79 |  | MOV A,C |  |
| 0638 | B0 |  | ORA B |  |
| 0639 | 1E06 |  | MVI E,06 |  |
| 063B | CAD501 |  | JZ [Error](http://altairbasic.org/int_dis_4.htm#Error) |  |
| Get the first character of the line. If it's not the DATA keyword ID then loop back to try the next line. |
| 063E | 23 |  | INX H |  |
| 063F | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0640 | FE83 |  | CPI KWID\_DATA |  |
| 0642 | C23506 |  | JNZ [NextDataLine](http://altairbasic.org/int_dis_15.htm#NextDataLine) |  |
| Found a DATA line. We remove the prog ptr to the beginning of the line from the stack (we don't need it - we have the ptr in HL) and jump up to GotDataItem. |
| 0645 | C1 |  | POP B |  |
| 0646 | C32006 |  | JMP [GotDataItem](http://altairbasic.org/int_dis_15.htm#GotDataItem) |  |

## 1.16 NEXT Handler

### Next

|  |
| --- |
| The NEXT keyword is followed by the name of the FOR variable, so firstly we get the address of that variable into DE. |
| 0649 | CD1B07 | Next | CALL [GetVar](http://altairbasic.org/int_dis_18.htm#GetVar) |  |
| Save the prog ptr in HL to PROG\_PTR\_TEMP. This currently points to the end of the NEXT statement, and we need to get it back later in case we find that the FOR loop has completed. |
| 064C | 225D01 |  | SHLD [PROG\_PTR\_TEMP](http://altairbasic.org/int_dis_3.htm#PROG_PTR_TEMP) |  |
| GetFlowPtr to get access to the FOR flow struct on the stack. |
| 064F | CD9201 |  | CALL [GetFlowPtr](http://altairbasic.org/int_dis_4.htm#GetFlowPtr) |  |
| 0652 | F9 |  | SPHL |  |
| Push address of FOR variable |
| 0653 | D5 |  | PUSH D |  |
| Load A with first byte of struct (0x01), advance HL, and preserve A. |
| 0654 | 7E |  | MOV A,M |  |
| 0655 | 23 |  | INX H |  |
| 0656 | F5 |  | PUSH PSW |  |
| Push address of FOR variable again. |
| 0657 | D5 |  | PUSH D |  |
| If GetFlowPtr returned without the zero (Z) flag set, then the FOR struct was not found and we error out with Next without For (NF). |
| 0658 | 1E00 |  | MVI E,00 |  |
| 065A | C2D501 |  | JNZ [Error](http://altairbasic.org/int_dis_4.htm#Error) |  |
| The next 4 bytes of the flow struct are the STEP number. We load this into FACCUM here. |
| 065D | CD0F0A |  | CALL [FLoadFromMem](http://altairbasic.org/math_dis_6.htm#FLoadFromMem) |  |
| Get FOR variable address into HL and push the struct ptr |
| 0660 | E3 |  | XTHL |  |
| Add the FOR variable to the STEP number and update the FOR variable with the result. |
| 0661 | E5 |  | PUSH H |  |
| 0662 | CD0408 |  | CALL [FAddMem](http://altairbasic.org/math_dis_1.htm#FAddMem) |  |
| 0665 | E1 |  | POP H |  |
| 0666 | CD290A |  | CALL [FCopyToMem](http://altairbasic.org/math_dis_6.htm#FCopyToMem) |  |
| Restore struct ptr to HL. This now points to the TO number, which we load into BCDE. |
| 0669 | E1 |  | POP H |  |
| 066A | CD200A |  | CALL [FLoadBCDEfromMem](http://altairbasic.org/math_dis_6.htm#FLoadBCDEfromMem) |  |
| Compare the updated FOR variable (in FACCUM) with the TO number (in BCDE). The result of the compare is in A and will be 0xFF if FOR var is less than the TO number, 0x00 if equal, and 0x01 if the FOR variable is greater than the TO number. |
| 066D | E5 |  | PUSH H |  |
| 066E | CD4C0A |  | CALL [FCompare](http://altairbasic.org/math_dis_7.htm#FCompare) |  |
| 0671 | E1 |  | POP H |  |
| Restore the direction byte to B. Remember this is 0x01 for forward iteration, 0xFF for backwards (when there is a -ve STEP number). |
| 0672 | C1 |  | POP B |  |
| This is marvellous! By subtracting the direction byte from the result of FCompare we can tell if the FOR loop has completed (the result of the subtraction will be zero) with the minimum of fuss. Read the two above comments and it should make sense. |
| 0673 | 90 |  | SUB B |  |
| NOT loading a floating point number, this is just a handy way of getting the last four bytes of the struct. BC is loaded with the prog ptr to just beyond the FOR statement, and DE is loaded with the line number of the FOR statement. |
| 0674 | CD200A |  | CALL [FLoadBCDEfromMem](http://altairbasic.org/math_dis_6.htm#FLoadBCDEfromMem) |  |
| If FOR loop is complete (see two comments up) then jump ahead. |
| 0677 | CA8306 |  | JZ [ForLoopIsComplete](http://altairbasic.org/int_dis_16.htm#ForLoopIsComplete) |  |
| FOR loop is not yet complete. Here we save the line number of the FOR statement to the CURRENT\_LINE variable, load HL with the prog ptr to the end of the FOR statement, and jump to EndOfForHandler which pushes the last byte of the for\_struct on the stack and falls into ExecNext. |
| 067A | EB |  | XCHG |  |
| 067B | 226101 |  | SHLD [CURRENT\_LINE](http://altairbasic.org/int_dis_3.htm#CURRENT_LINE) |  |
| 067E | 69 |  | MOV L,C |  |
| 067F | 60 |  | MOV H,B |  |
| 0680 | C31D04 |  | JMP [EndOfForHandler](http://altairbasic.org/int_dis_8.htm#EndOfForHandler) |  |
| The FOR loop is complete. Therefore we don't need the for\_struct on the stack any more, and since HL points just past it we can load the stack pointer from HL to reclaim that bit of stack space. |
| 0683 | F9 | ForLoopIsComplete | SPHL |  |
| 0684 | 2A5D01 |  | LHLD [PROG\_PTR\_TEMP](http://altairbasic.org/int_dis_3.htm#PROG_PTR_TEMP) |  |
| 0687 | C32104 |  | JMP [ExecNext](http://altairbasic.org/int_dis_9.htm#ExecNext) |  |

## 1.17 Expression Evaluation

### EvalExpression

Evaluates an expression, returning with the result in FACCUM. An expression is a combination of terms and operators.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 068A | 2B | EvalExpression | DCX H |  |
| 068B | 1600 |  | MVI D,00 |  |
| 068D | D5 |  | PUSH D |  |
| Check we've got enough space for one floating-point number. |
| 068E | 0E01 |  | MVI C,01 |  |
| 0690 | CDB601 |  | CALL [CheckEnoughVarSpace](http://altairbasic.org/int_dis_4.htm#CheckEnoughVarSpace) |  |
| Evaluate term and store prog ptr in 015f |
| 0693 | CDC406 |  | CALL [EvalTerm](http://altairbasic.org/int_dis_17.htm#EvalTerm) |  |
| 0696 | 225F01 |  | SHLD 015F |  |
| 0699 | 2A5F01 | ArithParse | LHLD 015F |  |
| 069C | C1 |  | POP B |  |
| Get byte following sub-expression. This is where we deal with arithmetic operators. If the byte is less than KWID\_+ then return. |
| 069D | 7E |  | MOV A,M |  |
| 069E | 1600 |  | MVI D,00 |  |
| 06A0 | D698 |  | SUI KWID\_+ |  |
| 06A2 | D8 |  | RC |  |
| Return if A>=4, ie not an arithmetic operator, then convert A to an offset into the KW\_ARITH\_OP\_FNS table |
| 06A3 | FE04 |  | CPI 04 |  |
| 06A5 | D0 |  | RNC |  |
| 06A6 | 5F |  | MOV E,A |  |
| 06A7 | 07 |  | RLC |  |
| 06A8 | 83 |  | ADD E |  |
| 06A9 | 5F |  | MOV E,A |  |
| 06AA | 214B00 |  | LXI H,[KW\_ARITH\_OP\_FNS](http://altairbasic.org/int_dis_2.htm#KW_ARITH_OP_FNS) |  |
| 06AD | 19 |  | DAD D |  |
| Get first byte of table entry. This is the operator precedence indicator byte. Not entirely sure how this works yet, but it seems we basically return if we don't need to evaluate this bit yet. |
| 06AE | 78 |  | MOV A,B |  |
| 06AF | 56 |  | MOV D,M |  |
| 06B0 | BA |  | CMP D |  |
| 06B1 | D0 |  | RNC |  |
| 06B2 | 23 |  | INX H |  |
| Push counter and address of ArithParse onto the stack (the latter so we return to it after the arith fn runs) |
| 06B3 | C5 |  | PUSH B |  |
| 06B4 | 019906 |  | LXI B,[ArithParse](http://altairbasic.org/int_dis_17.htm#ArithParse) |  |
| 06B7 | C5 |  | PUSH B |  |
| Push FACCUM, taking care to preserve the operator precedence byte in D. |
| 06B8 | 4A |  | MOV C,D | ??? |
| 06B9 | CD020A |  | CALL [FPush](http://altairbasic.org/math_dis_6.htm#FPush) |  |
| 06BC | 51 |  | MOV D,C |  |
| Push address of arithmetic fn and jump back to |
| 06BD | F7 |  | RST PushNextWord() |  |
| 06BE | 2A5F01 |  | LHLD 015F |  |
| 06C1 | C38D06 |  | JMP 068D |  |

### EvalTerm

Evaluates a term in an expression. This can be a numeric constant, a variable, an inline function call taking a full expression as an argument, or a bracketed expression.

|  |
| --- |
| Get first character of term, and if it's a digit (as indicated by the carry flag) then jump to FIn |
| 06C4 | D7 | EvalTerm | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 06C5 | DAB30A |  | JC [FIn](http://altairbasic.org/math_dis_9.htm#FIn) |  |
| If the character is alphabetic then we have a variable, so jump ahead to get it. |
| 06C8 | CD8004 |  | CALL [CharIsAlpha](http://altairbasic.org/int_dis_10.htm#CharIsAlpha) |  |
| 06CB | D2F306 |  | JNC [EvalVarTerm](http://altairbasic.org/int_dis_17.htm#EvalVarTerm) |  |
| If the character is a leading '+' then simply ignore it and jump back to EvalTerm. |
| 06CE | FE98 |  | CPI KWID\_+ |  |
| 06D0 | CAC406 |  | JZ [EvalTerm](http://altairbasic.org/int_dis_17.htm#EvalTerm) |  |
| If the character is a leading '.' then that's a decimal point, so jump to FIn |
| 06D3 | FE2E |  | CPI '.' |  |
| 06D5 | CAB30A |  | JZ [FIn](http://altairbasic.org/math_dis_9.htm#FIn) |  |
| If the character is a leading '-' then jump head to EvalMinusTerm |
| 06D8 | FE99 |  | CPI KWID\_- |  |
| 06DA | CAEA06 |  | JZ [EvalMinusTerm](http://altairbasic.org/int_dis_17.htm#EvalMinusTerm) |  |
| If the character is the keyword ID of an inline function them jump ahead to deal with that. |
| 06DD | D69F |  | SUI 9F |  |
| 06DF | D2FD06 |  | JNC [EvalInlineFn](http://altairbasic.org/int_dis_17.htm#EvalInlineFn) |  |
| The only possibility left is a bracketed expression. Here we check for an opening bracket, recurse into EvalExpression, and return. |
| 06E2 | CF | EvalBracketed | RST [SyntaxCheck](http://altairbasic.org/int_dis_1.htm#SyntaxCheck) |  |
| 06E3 | 28 |  | '(' |  |
| 06E4 | CD8A06 |  | CALL [EvalExpression](http://altairbasic.org/int_dis_17.htm#EvalExpression) |  |
| 06E7 | CF |  | RST [SyntaxCheck](http://altairbasic.org/int_dis_1.htm#SyntaxCheck) |  |
| 06E8 | 29 |  | ')' |  |
| 06E9 | C9 |  | RET |  |
| Evaluate a negative term. This is done by simply recursing this block and negating the result. |
| 06EA | CDC406 | EvalMinusTerm | CALL [EvalTerm](http://altairbasic.org/int_dis_17.htm#EvalTerm) |  |
| 06ED | E5 |  | PUSH H |  |
| 06EE | CDFA09 |  | CALL [FNegate](http://altairbasic.org/math_dis_5.htm#FNegate) |  |
| 06F1 | E1 |  | POP H |  |
| 06F2 | C9 |  | RET |  |
| Evaluate a variable. The call to GetVar returns the address of the variable's value in DE, which is then moved to HL then the call to FLoadFromMem loads FACCUM with the variable's value. |
| 06F3 | CD1B07 | EvalVarTerm | CALL [GetVar](http://altairbasic.org/int_dis_18.htm#GetVar) |  |
| 06F6 | E5 |  | PUSH H |  |
| 06F7 | EB |  | XCHG |  |
| 06F8 | CD0F0A |  | CALL [FLoadFromMem](http://altairbasic.org/math_dis_6.htm#FLoadFromMem) |  |
| 06FB | E1 |  | POP H |  |
| 06FC | C9 |  | RET |  |
| Evaluate an inline function. First we get the offset into the KW\_INLINE\_FNS table into BC and stick it on the stack. |
| 06FD | 0600 | EvalInlineFn | MVI B,00 |  |
| 06FF | 07 |  | RLC |  |
| 0700 | 4F |  | MOV C,A |  |
| 0701 | C5 |  | PUSH B |  |
| Evaluate function argument |
| 0702 | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0703 | CDE206 |  | CALL [EvalBracketed](http://altairbasic.org/int_dis_17.htm#EvalBracketed) |  |
| Preserve prog ptr on stack and simultaneously get offset into KW\_INLINE\_FNS into HL |
| 0706 | E3 |  | XTHL |  |
| Set return address to somewhere that just POP H and RETs. |
| 0707 | 11F106 |  | LXI D,06F1 |  |
| 070A | D5 |  | PUSH D |  |
| Get function address by adding the offset in HL to KW\_INLINE\_FNS |
| 070B | 013D00 |  | LXI B,[KW\_INLINE\_FNS](http://altairbasic.org/int_dis_2.htm#KW_INLINE_FNS) |  |
| 070E | 09 |  | DAD B |  |
| Put function address on stack and return to it. |
| 070F | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) |  |
| 0710 | C9 |  | RET |  |

## 1.18 Variable Management

### Dim

Declares an array. Note that the start of this function handler is some way down in the block (at 0716).

|  |
| --- |
| Get next program character and return if null byte at end of line has been reached. |
| 0711 | 2B | DimContd | DCX H |  |
| 0712 | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0713 | C8 |  | RZ |  |
| Syntax check for a comma. |
| 0714 | CF |  | RST [SyntaxCheck](http://altairbasic.org/int_dis_1.htm#SyntaxCheck) |  |
| 0715 | 2C |  | ',' |  |
| Set return address to DimContd above. |
| 0716 | 011107 | Dim | LXI B,[DimContd](http://altairbasic.org/int_dis_18.htm#DimContd) |  |
| 0719 | C5 |  | PUSH B |  |
| Set A to a non-zero number before falling into GetVar+1. This indicates to GetVar that we're declaring rather than accessing an array. |
| 071A | F6AF |  | ORI AF |  |

### GetVar

Given that HL points to a variable name that's been encountered during program interpretation, this function will return in DE a pointer to that variable's value. If the variable does not exist, then (space permitting) this function also allocates and initialises it.

|  |
| --- |
| ??? Signalling this is a real call to GetVar, not dropped in from DIM |
| 071B | AF | GetVar | XRA A |  |
| 071C | 325B01 |  | STA [DIM\_OR\_EVAL](http://altairbasic.org/int_dis_3.htm#DIM_OR_EVAL) |  |
| Get variable name into BC, B=first char and C=(optional) second char. Notice we Syntax Error out if the first character is not alphabetic. |
| 071F | 46 |  | MOV B,M |  |
| 0720 | CD8004 |  | CALL [CharIsAlpha](http://altairbasic.org/int_dis_10.htm#CharIsAlpha) |  |
| 0723 | DAD001 |  | JC [SyntaxError](http://altairbasic.org/int_dis_4.htm#SyntaxError) |  |
| 0726 | AF |  | XRA A |  |
| 0727 | 4F |  | MOV C,A |  |
| 0728 | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0729 | D22E07 |  | JNC 072E |  |
| 072C | 4F |  | MOV C,A |  |
| 072D | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| If next char is '(' then we're dealing with an array and so jump to GetArrayVar (the following block) to deal with it. |
| 072E | D628 |  | SUI '(' |  |
| 0730 | CA8A07 |  | JZ [GetArrayVar](http://altairbasic.org/int_dis_18.htm#GetArrayVar) |  |
| Preserve program ptr on stack, and get VAR\_ARRAY\_BASE into DE and VAR\_BASE into HL. This is where we iterate through the stored variables (ie from VAR\_BASE to VAR\_ARRAY\_BASE) to see if the variable has already been declared. |
| 0733 | E5 |  | PUSH H |  |
| 0734 | 2A6901 |  | LHLD [VAR\_ARRAY\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_ARRAY_BASE) |  |
| 0737 | EB |  | XCHG |  |
| 0738 | 2A6701 |  | LHLD [VAR\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_BASE) |  |
| Loop to find the variable if it's already been allocated. If HL==DE then we've reached VAR\_ARRAY\_BASE without finding it, and so can jump ahead to allocate a new variable. |
| 073B | E7 | FindVarLoop | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) |  |
| 073C | CA5207 |  | JZ [AllocNewVar](http://altairbasic.org/int_dis_18.htm#AllocNewVar) |  |
| 073F | 79 |  | MOV A,C |  |
| 0740 | 96 |  | SUB M |  |
| 0741 | 23 |  | INX H |  |
| 0742 | C24707 |  | JNZ 0747 |  |
| 0745 | 78 |  | MOV A,B |  |
| 0746 | 96 |  | SUB M |  |
| 0747 | 23 |  | INX H |  |
| 0748 | CA8207 |  | JZ 0782 |  |
| 074B | 23 |  | INX H |  |
| 074C | 23 |  | INX H |  |
| 074D | 23 |  | INX H |  |
| 074E | 23 |  | INX H |  |
| 074F | C33B07 |  | JMP [FindVarLoop](http://altairbasic.org/int_dis_18.htm#FindVarLoop) |  |
| Prepare to alloc a new variable, but first we have to do something slightly bizarre... we check the return address to see if it's the expression evaluator that's called us, and if it is then we exit without allocating. Notice that (assuming we haven't been called by the evaluator) the prog ptr on the stack is kept in place. |
| 0752 | E1 | AllocNewVar | POP H | HL=prog ptr |
| 0753 | E3 |  | XTHL | (SP)=prog ptr, HL=ret.addr. |
| 0754 | D5 |  | PUSH D |  |
| 0755 | 11F606 |  | LXI D,06F6 | an address inside EvalTerm |
| 0758 | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) |  |
| 0759 | D1 |  | POP D |  |
| 075A | CA8507 |  | JZ [AlreadyAllocd](http://altairbasic.org/int_dis_18.htm#AlreadyAllocd) |  |
| 075D | E3 |  | XTHL | (SP)=ret.addr, HL=prog ptr. |
| 075E | E5 |  | PUSH H | Prog ptr back on stack |
| Allocate memory for the variable. We need a space of 6 bytes to be inserted before the array variables storage pointed to by VAR\_ARRAY\_BASE, so we copy that block up 6 bytes in memory. |
| 075F | C5 |  | PUSH B | Preserve var name on stack |
| 0760 | 010600 |  | LXI B,0006 |  |
| 0763 | 2A6B01 |  | LHLD [VAR\_TOP](http://altairbasic.org/int_dis_3.htm#VAR_TOP) |  |
| 0766 | E5 |  | PUSH H |  |
| 0767 | 09 |  | DAD B |  |
| 0768 | C1 |  | POP B |  |
| 0769 | E5 |  | PUSH H |  |
| 076A | CDA701 |  | CALL [CopyMemoryUp](http://altairbasic.org/int_dis_4.htm#CopyMemoryUp) |  |
| 076D | E1 |  | POP H |  |
| 076E | 226B01 |  | SHLD [VAR\_TOP](http://altairbasic.org/int_dis_3.htm#VAR_TOP) |  |
| Update VAR\_ARRAY\_BASE cos the array block has been moved up 6 bytes. |
| 0771 | 60 |  | MOV H,B |  |
| 0772 | 69 |  | MOV L,C |  |
| 0773 | 226901 |  | SHLD [VAR\_ARRAY\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_ARRAY_BASE) |  |
| Initialise the new variable to zero. |
| 0776 | 2B | InitVarLoop | DCX H |  |
| 0777 | 3600 |  | MVI M,00 |  |
| 0779 | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) |  |
| 077A | C27607 |  | JNZ [InitVarLoop](http://altairbasic.org/int_dis_18.htm#InitVarLoop) |  |
| Restore variable name to DE and write it to the first 2 bytes of the variable's storage. |
| 077D | D1 |  | POP D |  |
| 077E | 73 |  | MOV M,E |  |
| 077F | 23 |  | INX H |  |
| 0780 | 72 |  | MOV M,D |  |
| 0781 | 23 |  | INX H |  |
| Swap HL and DE so that DE points to the variable value, then restore the prog ptr to HL and return |
| 0782 | EB |  | XCHG |  |
| 0783 | E1 |  | POP H |  |
| 0784 | C9 |  | RET |  |
| Function exit for when called by EvalTerm. Here we set FACCUM to zero (don't know why), restore the prog ptr to HL and return. |
| 0785 | 327201 | AlreadyAllocd | STA [FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 | A was set to zero at 075A. |
| 0788 | E1 |  | POP H |  |
| 0789 | C9 |  | RET |  |

### GetArrayVar

Accesses or allocates an array variable. The contents of DIM\_OR\_EVAL indicate whether we're dealing with an array declaration (ie a DIM statement) or whether an array element is being accessed. In the former case DIM\_OR\_EVAL is 0xEF, otherwise it is 0.

|  |
| --- |
| Preserve variable name on stack. |
| 078A | C5 | GetArrayVar | PUSH B |  |
| Push declare-or-access flag on stag |
| 078B | 3A5B01 |  | LDA [DIM\_OR\_EVAL](http://altairbasic.org/int_dis_3.htm#DIM_OR_EVAL) |  |
| 078E | F5 |  | PUSH PSW |  |
| Get array subscript into CDE |
| 078F | CD8804 |  | CALL [GetSubscript](http://altairbasic.org/int_dis_10.htm#GetSubscript) |  |
| Syntax check for closing bracket ')' |
| 0792 | CF |  | RST [SyntaxCheck](http://altairbasic.org/int_dis_1.htm#SyntaxCheck) |  |
| 0793 | 29 |  | ')' |  |
| Restore declare-or-access flag |
| 0794 | F1 |  | POP PSW |  |
| 0795 | 325B01 |  | STA [DIM\_OR\_EVAL](http://altairbasic.org/int_dis_3.htm#DIM_OR_EVAL) |  |
| Get variable name from stack into DE; HL becomes the subscript previously in DE; previous value of HL goes on stack. |
| 0798 | E3 |  | XTHL |  |
| 0799 | EB |  | XCHG |  |
| Multiply the subscript by 4 and stick it on the stack. |
| 079A | 29 |  | DAD H |  |
| 079B | 29 |  | DAD H |  |
| 079C | E5 |  | PUSH H |  |
| Load HL with the base of array storage and LXI into FindArray+2 |
| 079D | 2A6901 |  | LHLD [VAR\_ARRAY\_BASE](http://altairbasic.org/int_dis_3.htm#VAR_ARRAY_BASE) |  |
| 07A0 | 01C109 |  | LXI B,.... |  |
| Find the array loop. First thing we do is advance the array ptr by {subscript} |
| 07A1 | C1 | FindArray | POP B |  |
| 07A2 | 09 |  | DAD B |  |
| 07A3 | EB |  | XCHG |  |
| If we've reached VAR\_TOP then we know the array has not already been declared, so we can jump forward to allocate it. |
| 07A4 | E5 |  | PUSH H |  |
| 07A5 | 2A6B01 |  | LHLD [VAR\_TOP](http://altairbasic.org/int_dis_3.htm#VAR_TOP) |  |
| 07A8 | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) |  |
| 07A9 | EB |  | XCHG |  |
| 07AA | D1 |  | POP D |  |
| 07AB | CACD07 |  | JZ [AllocArray](http://altairbasic.org/int_dis_18.htm#AllocArray) |  |
| Advance to the next array in the block. |
| 07AE | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) |  |
| 07AF | E3 |  | XTHL |  |
| 07B0 | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) |  |
| 07B1 | E1 |  | POP H |  |
| 07B2 | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) |  |
| 07B3 | C2A107 |  | JNZ [FindArray](http://altairbasic.org/int_dis_18.htm#FindArray) |  |
| Array found. If we're in a DIM statement then this is a duplicate definition error. You can only DIM an array the once. |
| 07B6 | 3A5B01 |  | LDA DIM\_OR\_READ |  |
| 07B9 | B7 |  | ORA A |  |
| 07BA | 1E12 |  | MVI E,12 |  |
| 07BC | C2D501 |  | JNZ [Error](http://altairbasic.org/int_dis_4.htm#Error) |  |
| Array element retrieval. First we ??? |
| 07BF | D1 |  | POP D |  |
| 07C0 | 1B |  | DCX D |  |
| 07C1 | E3 |  | XTHL |  |
| If the subscript is greater than or equal to the number of array elements, then 'Bad Subscript' error. |
| 07C2 | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) |  |
| 07C3 | 1E10 |  | MVI E,10 |  |
| 07C5 | D2D501 |  | JNC [Error](http://altairbasic.org/int_dis_4.htm#Error) |  |
| Return with DE pointing to the element value, and HL the prog ptr. |
| 07C8 | D1 |  | POP D |  |
| 07C9 | 19 |  | DAD D |  |
| 07CA | D1 |  | POP D |  |
| 07CB | EB |  | XCHG |  |
| 07CC | C9 |  | RET |  |
| Allocate space for the array. Here, DE holds the array name and HL points to where it is going to be stored. We start by storing the array name. |
| 07CD | 73 | AllocArray | MOV M,E |  |
| 07CE | 23 |  | INX H |  |
| 07CF | 72 |  | MOV M,D |  |
| 07D0 | 23 |  | INX H |  |
| If DIM\_OR\_EVAL is zero (indicating array access), then jump ahead with DE=???? |
| 07D1 | 112C00 |  | LXI D,002C |  |
| 07D4 | 3A5B01 |  | LDA 015B |  |
| 07D7 | B7 |  | ORA A |  |
| 07D8 | CAE107 |  | JZ 07E1 |  |
| DIM\_OR\_EVAL is not zero, therefore we're declaring the array. Here we restore the array elements size to DE, save it again, add 4, and write that value to follow the array name. |
| 07DB | D1 |  | POP D |  |
| 07DC | D5 |  | PUSH D |  |
| 07DD | 13 |  | INX D |  |
| 07DE | 13 |  | INX D |  |
| 07DF | 13 |  | INX D |  |
| 07E0 | 13 |  | INX D |  |
| Write out array size. |
| 07E1 | D5 |  | PUSH D |  |
| 07E2 | 73 |  | MOV M,E |  |
| 07E3 | 23 |  | INX H |  |
| 07E4 | 72 |  | MOV M,D |  |
| 07E5 | 23 |  | INX H |  |
| 07E6 | E5 |  | PUSH H |  |
| Check the new VAR\_TOP won't interfere with the stack, and update VAR\_TOP. |
| 07E7 | 19 |  | DAD D |  |
| 07E8 | CDC301 |  | CALL [CheckEnoughMem](http://altairbasic.org/int_dis_4.htm#CheckEnoughMem) |  |
| 07EB | 226B01 |  | SHLD [VAR\_TOP](http://altairbasic.org/int_dis_3.htm#VAR_TOP) |  |
| Initialise all array elements to 0. |
| 07EE | D1 |  | POP D |  |
| 07EF | 2B | InitElements | DCX H |  |
| 07F0 | 3600 |  | MVI M,00 |  |
| 07F2 | E7 |  | RST 4 |  |
| 07F3 | C2EF07 |  | JNZ [InitElements](http://altairbasic.org/int_dis_18.htm#InitElements) |  |
| 07F6 | C3BF07 |  | JMP 07BF |  |

# The Math Package

The 'math package' is a reasonably discrete block of code that provides floating-point arithmetic capability for the rest of BASIC. It also includes some math functions, such as SQR (square root) and is probably the hardest part of BASIC to understand. There are three general reasons for this :

* You may have forgotten a lot of the maths you learnt at school. This certainly applied to me : when I began working on this section from first principles, I quickly found myself floundering at the very idea of binary fractions.
* Unless you're a numerical analyst who has reason to distrust conventional hardware/software floating-point support, you probably never needed to think about how floating point worked before now. Modern processors, compilers, and runtime libraries took the pain away years ago, and quite right too.
* Floating point is hard to code. Consider this : Bill Gates is one of the brightest kids in America at the time, but he and his equally brainy pal Paul Allen end up having to hire a third wunderkind, Monte Davidoff, just to do floating point. They needed a specialist to do specialist work, and Monte had done it before.

## Maths Refresher

### The Basics of Bases

Consider an everyday decimal number such as 317.25. The digits that make up this and every other decimal number represent multiples of powers of ten, all added together:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 102 | 101 | 100 | . | 10-1 | 10-2 | | **3** | **1** | **7** | **.** | **2** | **5** | |

So writing 317.25 is basically just shorthand for 3\*102 + 1\*101 + 7\*100 + 2\*10-1 + 5\*10-2. The shorthand form is far more readable, and that's why everybody uses it. At risk of labouring the point, the below table should clarify this.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Digit Position** | **Digit Value for Position** | **Decimal Number** | **Digit value for this number** | | | | 2 | 100 | **3**17.25 | 3 \* 100 | = | 300 | | 1 | 10 | 3**1**7.25 | 1 \* 10 | = | 10 | | 0 | 1 | 31**7**.25 | 7 \* 1 | = | 7 | | -1 | 1/10 | 317.**2**5 | 2 \* .1 | = | .2 | | -2 | 1/100 | 317.2**5** | 5 \* .01 | = | .05 | | **Total:** | | | | **=** | **317.25** | |

Now consider the same number in binary (base two). The decimal number 317.25, expressed in binary, is :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | . | 2-1 | 2-2 | | **1** | **0** | **0** | **1** | **1** | **1** | **1** | **0** | **1** | **.** | **0** | **1** | |

And here's a table like the decimal one above, which should make it completely clear (remember 'bit' is short for 'binary digit') :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Bit Position** | **Bit Value for Position** | **Binary Number** | **Bit value for this number** | | | | 8 | 256 | **1**00111101.01 | 1 \* 256 | = | 256 | | 7 | 128 | 1**0**0111101.01 | 0 \* 128 | = | 0 | | 6 | 64 | 10**0**111101.01 | 0 \* 64 | = | 0 | | 5 | 32 | 100**1**11101.01 | 1 \* 32 | = | 32 | | 4 | 16 | 1001**1**1101.01 | 1 \* 16 | = | 16 | | 3 | 8 | 10011**1**101.01 | 1 \* 8 | = | 8 | | 2 | 4 | 100111**1**01.01 | 1 \* 4 | = | 4 | | 1 | 2 | 1001111**0**1.01 | 0 \* 2 | = | 0 | | 0 | 1 | 10011110**1**.01 | 1 \* 1 | = | 1 | | -1 | 1/2 | 100111101.**0**1 | 0 \* 1/2 | = | 0 | | -2 | 1/4 | 100111101.0**1** | 1 \* 1/4 | = | 0.25 | | **Total:** | | | | **=** | **317.25** | |

### Mantissas, Exponents, and Scientific Notation

Now let's think about decimal numbers again. Another way of representing the number 317.25 is like this : **3.1725 \* 102**. Yes we've split one number into two numbers - we've extracted the number's magnitude and written it seperately. Why is this useful? Well, consider a very small number such as 0.00000000000588. Looking at it now, precisely how small is that? That's a lot of zeros to work through. Also, let's pretend we're using very small numbers like this one in a pen+paper calculation - something like 0.00000000000588 + 0.000000000000291. You'd better be sure you don't miss out a zero when you're working the problem through, or your answer will be off by a factor of 10. It's much easier to have those numbers represented as **5.88 \* 10-12** and **2.91\* 10-13** (yes the second number had an extra zero - did you spot that?). The same principle applies for very large numbers like 100000000 - it's just easier and less human error prone to keep the magnitudes seperated out when working with such numbers.

It's the smallest of small steps to get from this form of number notation to proper *scientific notation*. The only difference is how the magnitude is written - in scientific notation we lose the magnitude's base and only write it's exponent part, thusly : **3.1725 E 2**. The part that's left of the E, the 3.1725, is called the **mantissa**. The bit to the right of the E is the **exponent**.

### Mantissas and Exponents in Binary

Let's go back to considering 317.25 in binary : 100111101.01. Using scientific notation, this is **1.0011110101 E 1000**. Remember that both mantissa and exponent are written in binary that exponent value 1000 is a binary number, 8 in decimal.

## Why *floating* point?

Consider the eternal problem of having a finite amount of computer memory. Not having infinate RAM means we cannot represent an infinite range of numbers. If we have eight bits of memory, we can represent the integers from 0 to 255 only. If we have sixteen, we can raise our range from 0 to 65535, and so on. The more bits we can play with, the larger the range of numbers we can represent. With fractional numbers there is a second problem : *precision*. Many fractions recur : eg one third in decimal is 0.33333 recurring. Likewise, one tenth is 0.1 in decimal but 0.0001100110011 last four bits recurring in binary.

So any method we choose for storing fractional numbers has to take these two problems into consideration. Bearing this in mind, consider the two possible approaches for storing fractional numbers :

* *Fixed point*. Store the integer part in one field, and the fractional part in another field. It's called fixed point representation since the point (binary or decimal) is always in the same place - between the integer and fractional fields.
* *Floating point*. Store the mantissa in one field, and the exponent in another field. This way, the point wouldn't be fixed into place - it could be anywhere, as determined by the binary exponent. It would, in fact be, a floating point.

Why is floating point better than fixed point? Let's say we have 32 bits to play with. Let's use fixed point and assign 16 bits for the integer part and 16 for the fractional part. This allows a range of 0 to 65535.9999 or so, which isn't very good value, range-wise, for 32 bits. OK, lets increase the range - we'll change to using 20 bits for the integer and 12 for the fraction. This gives us a range of 0 to 1,048,575.999ish . Still not a huge range, and since we've only got 12 bits for the fraction we're losing precision - numbers stored this way will be rounded to the nearest 1/4096th.

Now lets try floating point instead. Lets assign a whopping 24 bits for the mantissa and 8 bits for the exponent. 8 bits doesn't sound like much, but this is an exponent after all - with these 8 bits we get a range of -128 to +127 which is roughly 10-38 to to 1038. That's a nice big range! And we get 24 bits of precision too! It's clearly the better choice.

Floating point is not a perfect solution though... adding a very small number to a very large number is likely to produce an erroneous result. For example, go to the BASIC [emulator](http://altairbasic.org/em/em.htm) and try **PRINT 10000+.1**. You get **10000.1** as expected. Now try **PRINT 10000+.01** or **PRINT 100000+.1**. See?

## Normalisation

Normalisation is the process of shifting the mantissa until it is between 0.5 and 1 and adjusting the exponent to compensate. For example, these binary numbers are unnormalised :

* 101.001
* 0.0001
* 0.011 E 101

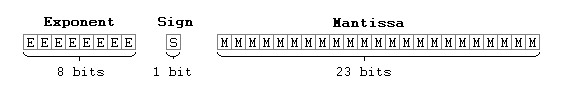
After normalisation these same binary numbers become :

* 0.101001 E 11
* 0.1 E -11
* 0.11 E 100

blah

## How Altair BASIC stored floating point numbers

There was no industry standard for floating-point number representation back in 1975, so Monte had to roll his own. He decided that 32 bits would allow an adequate range, and defined his floating-point number format like this :



The 8-bit exponent field had a *bias* of 128. This just meant that the stored exponent was stored as 'exponent+128'.

Also, the mantissa was really 24 bits long, but squeezed into 23 bits. How did he save an extra bit of precision? By considering zero as a special case, indicated by exponent zero. Any non-zero number will always have a mantissa with a leading 1. And since the first bit is always going to be 1, why bother storing it?

The intermediate storage of unpacked fp numbers is undefined and seems to be generally done on the fly.

*fixme: put example of normalising and denormalising.*

## 2.1 Utility Functions

Two small utility functions.

### FWordToFloat

Converts the integer word in AB (A holds high byte, B holds low byte) to a floating-point number. BUG: This code is completely unused! The address of this function is stored at 0006, but nothing ever uses it. It's probably needed by 8K or other bigger BASICs, but has accidentally been compiled into this 4K version.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 07F9 | 50 | FWordToFloat | MOV D,B |  |
| 07FA | 1E00 |  | MVI E,00 |  |
| 07FC | 0690 |  | MVI B,90 | exponent=2^16 |
| 07FE | C3EA09 |  | JMP [FCharToFloat](http://altairbasic.org/math_dis_5.htm#FCharToFloat)+5 |  |

### FAddOneHalf

Adds 0.5 to FACCUM.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0801 | 210B0C | FAddOneHalf | LXI H,[ONE\_HALF](http://altairbasic.org/math_dis_10.htm#ONE_HALF) | Load BCDE with (float) 0.5. |
| 0804 | CD200A | FAddMem | CALL [FLoadBCDEfromMem](http://altairbasic.org/math_dis_6.htm#FLoadBCDEfromMem) |  |
| 0807 | C31208 |  | JMP [FAdd](http://altairbasic.org/math_dis_2.htm#FAdd)+2 |  |

### 

**2.2 Addition & Subtraction**

blah

**FSub**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 080A | C1 | FSub | POP B | Get lhs in BCDE. |
| 080B | D1 |  | POP D |  |
| 080C | CDFA09 |  | CALL [FNegate](http://altairbasic.org/math_dis_5.htm#FNegate) | Negate rhs and slimily |
| 080F | 21.... |  | LXI H,.... | LXI into FAdd + 2. |

**FAdd**

The lhs is on the stack, the rhs is in FACCUM. The steps for adding the two numbers are :

* Handle special cases where either side is 0.
* Denormalise mantissas.
* Align the two numbers. *fixme: give Monte's decimal example*.
* Add (or subtract) mantissas.
* Normalise result and store in FACCUM.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0810 | C1 | FAdd | POP B | Get lhs in BCDE. |
| 0811 | D1 |  | POP D |  |
| Special cases for when lhs or rhs are zero. |
| 0812 | 78 |  | MOV A,B | If lhs==0 then we don't need |
| 0813 | B7 |  | ORA A | to do anything and can just |
| 0814 | C8 |  | RZ | exit. |
| 0815 | 3A7201 |  | LDA [FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 | If rhs==0 then exit via a copy |
| 0818 | B7 |  | ORA A | of lhs to FACCUM. |
| 0819 | CA120A |  | JZ [FLoadFromBCDE](http://altairbasic.org/math_dis_6.htm#FLoadFromBCDE) |  |
| Get exponents' difference into A. |
| 081C | 90 |  | SUB B | A=rhs.exponent-lhs.exponent. |
| 081D | D22C08 |  | JNC 082C | If rhs' exponent >= lhs'exponent, jump ahead. |
| Swap lhs and rhs if lhs exponent was more than rhs exponent. |
| 0820 | 2F |  | CMA | Two's complement the exponent |
| 0821 | 3C |  | INR A | difference, so it's correct. |
| 0822 | EB |  | XCHG |  |
| 0823 | CD020A |  | CALL [FPush](http://altairbasic.org/math_dis_6.htm#FPush) | Push old rhs |
| 0826 | EB |  | XCHG |  |
| 0827 | CD120A |  | CALL [FLoadFromBCDE](http://altairbasic.org/math_dis_6.htm#FLoadFromBCDE) | rhs = old lhs |
| 082A | C1 |  | POP B | lhs = old rhs. |
| 082B | D1 |  | POP D |  |
| Unpack the mantissas. This loses the signs of both numbers, but we do get back their relationship : the call to FUnpackMantissas leaves A +ve if the signs mismatched, or -ve if the signs were equal. |
| 082C | F5 |  | PUSH PSW | Preserve exponent diff |
| 082D | CD370A |  | CALL [FUnpackMantissas](http://altairbasic.org/math_dis_7.htm#FUnpackMantissas) |  |
| 0830 | 67 |  | MOV H,A | H=sign relationship |
| 0831 | F1 |  | POP PSW | A=exponent diff. |
| Align lhs with rhs. |
| 0832 | CDC908 |  | CALL [FMantissaRtMult](http://altairbasic.org/math_dis_3.htm#FMantissaRtMult) | Shift lhs mantissa right by (exponent diff) places. |
| Decide whether to add or subtract the mantissas. We subtract if the signs were mismatched. |
| 0835 | B4 |  | ORA H | A=0 after last call, so this tests |
| 0836 | 216F01 |  | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM) | the sign relationship. |
| 0839 | F24D08 |  | JP [FSubMantissas](http://altairbasic.org/math_dis_2.htm#FSubMantissas) | Jump ahead if we need to subtract. |
| Add the mantissas. |
| 083C | CDA908 |  | CALL [FAddMantissas](http://altairbasic.org/math_dis_3.htm#FAddMantissas) |  |
| 083F | D27E08 |  | JNC [FRoundUp](http://altairbasic.org/math_dis_3.htm#FRoundUp) | Jump ahead if that didn't overflow. |
| 0842 | 23 |  | INX H | Flip the sign in FTEMP\_SIGN. |
| 0843 | 34 |  | INR M |  |
| 0844 | CAA408 |  | JZ [Overflow](http://altairbasic.org/math_dis_3.htm#Overflow) | Error out if exponent overflowed. |
| 0847 | CDD608 |  | CALL [FMantissaRtOnce](http://altairbasic.org/math_dis_3.htm#FMantissaRtOnce) | Shift mantissa one place right |
| 084A | C37E08 |  | JMP [FRoundUp](http://altairbasic.org/math_dis_3.htm#FRoundUp) | Jump ahead. |
| Subtract lhs mantissa from rhs mantissa. |
| 084D | AF | FSubMantissas | XRA A | B=0-B |
| 084E | 90 |  | SUB B |  |
| 084F | 47 |  | MOV B,A |  |
| 0850 | 7E |  | MOV A,M | E=(FACCUM)-E |
| 0851 | 9B |  | SBB E |  |
| 0852 | 5F |  | MOV E,A |  |
| 0853 | 23 |  | INX H |  |
| 0854 | 7E |  | MOV A,M | D=(FACCUM+1)-D |
| 0855 | 9A |  | SBB D |  |
| 0856 | 57 |  | MOV D,A |  |
| 0857 | 23 |  | INX H |  |
| 0858 | 7E |  | MOV A,M | C=(FACCUM+2)-C |
| 0859 | 99 |  | SBB C |  |
| 085A | 4F |  | MOV C,A |  |

Fall into FNormalise

## 2.3 Mantissa Magic

A group of functions for manipulating mantissas.

### FNormalise

Result mantissa in CDEB is normalised, rounded up to CDE, and stored in FACCUM.

|  |
| --- |
| If carry set then negate the mantissa. Most users of this function call over this step. |
| 085B | DCB508 | FNormalise | CC [FNegateInt](http://altairbasic.org/math_dis_3.htm#FNegateInt) |  |
| Normalise the mantissa : We shift it left until bit 23 is set. |
| 085E | 2600 |  | MVI H,00 |  |
| 0860 | 79 |  | MOV A,C | Test most-significant bit of mantissa |
| 0861 | B7 |  | ORA A | and jump ahead if it's 1. |
| 0862 | FA7E08 |  | JM [FRoundUp](http://altairbasic.org/math_dis_3.htm#FRoundUp) |  |
| 0865 | FEE0 | NormLoop | CPI E0 | If we've shifted 32 times, |
| 0867 | CABE09 |  | JZ [FZero](http://altairbasic.org/math_dis_4.htm#FZero) | then the number is 0. |
| 086A | 25 |  | DCR H |  |
| 086B | 78 |  | MOV A,B | Left-shift extra mantissa byte |
| 086C | 87 |  | ADD A |  |
| 086D | 47 |  | MOV B,A |  |
| 086E | CD9008 |  | CALL [FMantissaLeft](http://altairbasic.org/math_dis_3.htm#FMantissaLeft) | Left-shift mantissa. |
| 0871 | 7C |  | MOV A,H |  |
| 0872 | F26508 |  | JP [NormLoop](http://altairbasic.org/math_dis_3.htm#NormLoop) | Loop |
| Adjust exponent by however many left-shifts had do be done during normalization. |
| 0875 | 217201 |  | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 0878 | 86 |  | ADD M |  |
| 0879 | 77 |  | MOV M,A | Since A was a -ve number, that certainly should |
| 087A | D2BE09 |  | JNC [FZero](http://altairbasic.org/math_dis_4.htm#FZero) | have carried, hence the extra check for zero. |
| 087D | C8 |  | RZ | ?why? |
| Round up the extra mantissa byte. |
| 087E | 78 | FRoundUp | MOV A,B | A=extra mantissa byte |
| 087F | 217201 |  | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 0882 | B7 |  | ORA A | If bit 7 of the extra mantissa byte |
| 0883 | FC9A08 |  | CM [FMantissaInc](http://altairbasic.org/math_dis_3.htm#FMantissaInc) | is set, then round up the mantissa. |
| Set the sign and exit. The XRA C is interesting : remember that bit 7 of C is the most significant bit of the normalised mantissa, which is invariably 1. Also, we need to use this bit for the sign. Well, in FUnpackMantissas the temporary sign in FTEMP\_SIGN was inverted, so an XOR with 1 will get back the correct sign. |
| 0886 | 46 |  | MOV B,M | B=exponent |
| 0887 | 23 |  | INX H |  |
| 0888 | 7E |  | MOV A,M | A=FTEMP\_SIGN |
| 0889 | E680 |  | ANI 80 |  |
| 088B | A9 |  | XRA C | Bit 7 of C is always 1. Thi |
| 088C | 4F |  | MOV C,A |  |
| 088D | C3120A |  | JMP [FLoadFromBCDE](http://altairbasic.org/math_dis_6.htm#FLoadFromBCDE) | Exit via copying BCDE to FACCUM. |

### FMantissaLeft

Shift the mantissa in CDE left by one bit.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0890 | 7B | FMantissaLeft | MOV A,E |  |
| 0891 | 17 |  | RAL |  |
| 0892 | 5F |  | MOV E,A |  |
| 0893 | 7A |  | MOV A,D |  |
| 0894 | 17 |  | RAL |  |
| 0895 | 57 |  | MOV D,A |  |
| 0896 | 79 |  | MOV A,C |  |
| 0897 | 8F |  | ADC A |  |
| 0898 | 4F |  | MOV C,A |  |
| 0899 | C9 |  | RET |  |

### FMantissaInc

Increments the mantissa in CDE and handles overflow.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 089A | 1C | FMantissaInc | INR E |  |
| 089B | C0 |  | RNZ |  |
| 089C | 14 |  | INR D |  |
| 089D | C0 |  | RNZ |  |
| 089E | 0C |  | INR C |  |
| 089F | C0 |  | RNZ |  |
| 08A0 | 0E80 |  | MVI C,80 | Mantissa overflowed to zero, so set it |
| 08A2 | 34 |  | INR M | to 1 and increment the exponent. |
| 08A3 | C0 |  | RNZ | And if the exponent overflows... |

### Overflow

A convenient place for exiting with the overflow (OV) error.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 08A4 | 1E0A | Overflow | MVI E,0A |  |
| 08A6 | C3D501 |  | JMP [Error](http://altairbasic.org/int_dis_4.htm#Error) |  |

### FAddMantissas

Adds the mantissa pointed to by HL to the one in CDE.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 08A9 | 7E | FAddMantissas | MOV A,M |  |
| 08AA | 83 |  | ADD E |  |
| 08AB | 5F |  | MOV E,A |  |
| 08AC | 23 |  | INX H |  |
| 08AD | 7E |  | MOV A,M |  |
| 08AE | 8A |  | ADC D |  |
| 08AF | 57 |  | MOV D,A |  |
| 08B0 | 23 |  | INX H |  |
| 08B1 | 7E |  | MOV A,M |  |
| 08B2 | 89 |  | ADC C |  |
| 08B3 | 4F |  | MOV C,A |  |
| 08B4 | C9 |  | RET |  |

### 

### FNegateInt

Negate the 32-bit integer in CDEB by subtracting it from zero. Also flips the sign in FTEMP. Used by FAsInteger and FAdd.

|  |
| --- |
| Flip the sign byte kept in FTEMP. |
| 08B5 | 217301 | FNegateInt | LXI H,[FTEMP](http://altairbasic.org/int_dis_3.htm#FTEMP) |  |
| 08B8 | 7E |  | MOV A,M |  |
| 08B9 | 2F |  | CMA |  |
| 08BA | 77 |  | MOV M,A |  |
| Negate extended mantissa, ie CDEB = 0 - CDEB. |
| 08BB | AF |  | XRA A |  |
| 08BC | 6F |  | MOV L,A |  |
| 08BD | 90 |  | SUB B |  |
| 08BE | 47 |  | MOV B,A |  |
| 08BF | 7D |  | MOV A,L |  |
| 08C0 | 9B |  | SBB E |  |
| 08C1 | 5F |  | MOV E,A |  |
| 08C2 | 7D |  | MOV A,L |  |
| 08C3 | 9A |  | SBB D |  |
| 08C4 | 57 |  | MOV D,A |  |
| 08C5 | 7D |  | MOV A,L |  |
| 08C6 | 99 |  | SBB C |  |
| 08C7 | 4F |  | MOV C,A |  |
| 08C8 | C9 |  | RET |  |

### FMantissaRtMult

Shifts the mantissa in CDE right by A places. Note that lost bits end up in B, general practice so we can round up from something later should we need to.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 08C9 | 0600 | FMantissaRtMult | MVI B,00 | Initialise extra mantissa byte |
| 08CB | 3C |  | INR A |  |
| 08CC | 6F |  | MOV L,A |  |
| 08CD | AF | RtMultLoop | XRA A |  |
| 08CE | 2D |  | DCR L |  |
| 08CF | C8 |  | RZ |  |
| 08D0 | CDD608 |  | CALL [FMantissaRtOnce](http://altairbasic.org/math_dis_3.htm#FMantissaRtOnce) |  |
| 08D3 | C3CD08 |  | JMP [RtMultLoop](http://altairbasic.org/math_dis_3.htm#RtMultLoop) |  |

### FMantissaRtOnce

Shifts the mantissa in CDE one bit right.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 08D6 | 79 | FMantissaRtOnce | MOV A,C |  |
| 08D7 | 1F |  | RAR |  |
| 08D8 | 4F |  | MOV C,A |  |
| 08D9 | 7A |  | MOV A,D |  |
| 08DA | 1F |  | RAR |  |
| 08DB | 57 |  | MOV D,A |  |
| 08DC | 7B |  | MOV A,E |  |
| 08DD | 1F |  | RAR |  |
| 08DE | 5F |  | MOV E,A |  |
| 08DF | 78 |  | MOV A,B | NB: B is the extra |
| 08E0 | 1F |  | RAR | mantissa byte. |
| 08E1 | 47 |  | MOV B,A |  |
| 08E2 | C9 |  | RET |  |

**2.4 Multiplication & Division**

blah

**FMul**

Multiplying two floating point numbers is theoretically simple. All we have to do is add the exponents, multiply the mantissas, and normalise the result. The only problem is that the 8080 didn't have a MUL instruction. Therefore the fundamental logic of multiplication (shift and add) is done by hand in this function. FMul's logic read something like this :

1. Get lhs and rhs. Exit if rhs=0.
2. Add lhs and rhs exponents
3. Initialise result mantissa to 0.
4. Get rightmost bit of rhs.
5. If this bit is set then add the lhs mantissa to the result mantissa.
6. Shift result mantissa right one bit.
7. Get next bit of rhs mantissa. If not done all 24 bits, loop back to 5.
8. Jump to FNormalise

Alternatively, here's some C++ pseudo-code :

float FMul(float lhs, float rhs)  
{  
  float result = 0;  
  for (int bit=0 ; bit<24 ; bit++) {  
    if (lhs.mantissa & (2^bit)) {  
      result.mantissa += rhs.mantissa;  
    }  
    result.mantissa>>=1;  
  }  
  return FNormalise(result);  
}

(fixme: Show why this works)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 08E3 | C1 | FMul | POP B | Get lhs in BCDE |
| 08E4 | D1 |  | POP D |  |
| 08E5 | EF |  | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) | If rhs==0 then exit |
| 08E6 | C8 |  | RZ |  |
| Add the exponents. |
| 08E7 | 2E00 |  | MVI L,00 | L=0 to signify exponent add |
| 08E9 | CD9B09 |  | CALL [FExponentAdd](http://altairbasic.org/math_dis_4.htm#FExponentAdd) |  |
| Store the lhs mantissa in the operands for FMulInnerLoop |
| 08EC | 79 |  | MOV A,C |  |
| 08ED | 321709 |  | STA [FMulInnerLoop](http://altairbasic.org/math_dis_4.htm#FMulInnerLoop)+13 |  |
| 08F0 | EB |  | XCHG |  |
| 08F1 | 221209 |  | SHLD [FMulInnerLoop](http://altairbasic.org/math_dis_4.htm#FMulInnerLoop)+8 |  |
| Initialise result mantissa CDEB to 0. |
| 08F4 | 010000 |  | LXI B,0000 |  |
| 08F7 | 50 |  | MOV D,B |  |
| 08F8 | 58 |  | MOV E,B |  |
| Set return address to FNormalise |
| 08F9 | 215E08 |  | LXI H,[FNormalise](http://altairbasic.org/math_dis_3.htm#FNormalise)+3 |  |
| 08FC | E5 |  | PUSH H |  |
| A great trick! FMul's outer loop works on one byte of multiplicand at a time. There are three bytes to do, so by pushing the address of the loop onto the stack twice, we can cheaply run the loop 3 times without needing a counter. |
| 08FD | 210509 |  | LXI H,[FMulOuterLoop](http://altairbasic.org/math_dis_4.htm#FMulOuterLoop) |  |
| 0900 | E5 |  | PUSH H |  |
| 0901 | E5 |  | PUSH H |  |
| 0902 | 216F01 |  | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM) |  |
| 0905 | 7E | FMulOuterLoop | MOV A,M | A=FACCUM mantissa byte |
| 0906 | 23 |  | INX H |  |
| 0907 | E5 |  | PUSH H | Preserve FACCUM ptr |
| 0908 | 2E08 |  | MVI L,08 | 8 bits to do |
| Inner loop processes a single byte of multiplicand. |
| 090A | 1F | FMulInnerLoop | RAR | Test lowest bit of mantissa byte |
| 090B | 67 |  | MOV H,A | Preserve mantissa byte |
| 090C | 79 |  | MOV A,C | A=result mantissa's high byte |
| 090D | D21909 |  | JNC 0919 | If that bit of multiplicand was 0, then skip over adding mantissas. |
| Add the lhs mantissa to the result mantissa |
| 0910 | E5 |  | PUSH H |  |
| 0911 | 210000 |  | LXI H,0000 |  |
| 0914 | 19 |  | DAD D |  |
| 0915 | D1 |  | POP D |  |
| 0916 | CE00 |  | ACI 00 | A=result mantissa high byte. This gets back to C |
| 0918 | EB |  | XCHG | in the call to FMantissaRtOnce+1. |
| Shift result mantissa right and loop back to inner loop if we haven't done all 8 bits yet. |
| 0919 | CDD708 |  | CALL [FMantissaRtOnce](http://altairbasic.org/math_dis_3.htm#FMantissaRtOnce)+1 |  |
| 091C | 2D |  | DCR L |  |
| 091D | 7C |  | MOV A,H | Restore mantissa byte and |
| 091E | C20A09 |  | JNZ [FMulInnerLoop](http://altairbasic.org/math_dis_4.htm#FMulInnerLoop) | jump back if L is not yet 0. |
| 0921 | E1 | PopHLandReturn | POP H | Restore FACCUM ptr |
| 0922 | C9 |  | RET | Return to FMulOuterLoop, or if finished that then exit to FNormalise |

**FDivByTen**

Divides FACCUM by 10. Used in FOut to bring the number into range before printing.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0923 | CD020A |  | CALL [FPush](http://altairbasic.org/math_dis_6.htm#FPush) |  |
| 0926 | 012084 |  | LXI B,8420 | BCDE=(float)10; |
| 0929 | 110000 |  | LXI D,0000 |  |
| 092C | CD120A |  | CALL [FLoadFromBCDE](http://altairbasic.org/math_dis_6.htm#FLoadFromBCDE) |  |

**FDiv**

fixme: work out how this works!

|  |
| --- |
| Get lhs into BCDE. |
| 092F | C1 | FDiv | POP B |  |
| 0930 | D1 |  | POP D |  |
| If rhs is zero, then divide-by-zero error. |
| 0931 | EF |  | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |
| 0932 | CAD301 |  | JZ [DivideByZero](http://altairbasic.org/int_dis_4.htm#DivideByZero) |  |
| Subtract exponents. |
| 0935 | 2EFF |  | MVI L,FF |  |
| 0937 | CD9B09 |  | CALL [FExponentAdd](http://altairbasic.org/math_dis_4.htm#FExponentAdd) |  |
| Multiply FACCUM by 4, the easy way. |
| 093A | 34 |  | INR M |  |
| 093B | 34 |  | INR M |  |
| Decrement HL so it points to most-significant byte of FACCUM's mantissa. |
| 093C | 2B |  | DCX H |  |
| Copy FACCUM's mantissa to places in FDivLoop. |
| 093D | 7E |  | MOV A,M |  |
| 093E | 326009 |  | STA 0960 |  |
| 0941 | 2B |  | DCX H |  |
| 0942 | 7E |  | MOV A,M |  |
| 0943 | 325C09 |  | STA 095C |  |
| 0946 | 2B |  | DCX H |  |
| 0947 | 7E |  | MOV A,M |  |
| 0948 | 325809 |  | STA 0958 |  |
| Load B with most significant byte of lhs mantissa, and load HL with 0 (DE was zeroed previously) |
| 094B | 41 |  | MOV B,C |  |
| 094C | EB |  | XCHG |  |
| Initialise A and CDE to 0. |
| 094D | AF |  | XRA A |  |
| 094E | 4F |  | MOV C,A |  |
| 094F | 57 |  | MOV D,A |  |
| 0950 | 5F |  | MOV E,A |  |
| 0951 | 326309 |  | STA 0963 |  |
| Long division loop. |
| 0954 | E5 | FDivLoop | PUSH H |  |
| 0955 | C5 |  | PUSH B |  |
| 0956 | 7D |  | MOV A,L |  |
| 0957 | D600 |  | SUI 00 |  |
| 0959 | 6F |  | MOV L,A |  |
| 095A | 7C |  | MOV A,H |  |
| 095B | DE00 |  | SBI 00 |  |
| 095D | 67 |  | MOV H,A |  |
| 095E | 78 |  | MOV A,B |  |
| 095F | DE00 |  | SBI 00 |  |
| 0961 | 47 |  | MOV B,A |  |
| 0962 | 3E00 |  | MVI A,00 |  |
| 0964 | DE00 |  | SBI 00 |  |
| 0966 | 3F |  | CMC |  |
| 0967 | D27109 |  | JNC 0971 |  |
| 096A | 326309 |  | STA 0963 |  |
| 096D | F1 |  | POP PSW |  |
| 096E | F1 |  | POP PSW |  |
| LXI over the restoration of ?? ? |
| 096F | 37 |  | STC |  |
| 0970 | D2.... |  | JNC .... |  |
| 0971 | C1 |  | POP B |  |
| 0972 | E1 |  | POP H |  |
| 0973 | 79 |  | MOV A,C |  |
| 0974 | 3C |  | INR A |  |
| 0975 | 3D |  | DCR A |  |
| 0976 | 1F |  | RAR |  |
| 0977 | FA7F08 |  | JM [FRoundUp](http://altairbasic.org/math_dis_3.htm#FRoundUp)+1 |  |
| 097A | 17 |  | RAL |  |
| 097B | CD9008 |  | CALL [FMantissaLeft](http://altairbasic.org/math_dis_3.htm#FMantissaLeft) |  |
| DIVTEMP \*= 2 |
| 097E | 29 |  | DAD H |  |
| 097F | 78 |  | MOV A,B |  |
| 0980 | 17 |  | RAL |  |
| 0981 | 47 |  | MOV B,A |  |
| 0982 | 3A6309 |  | LDA 0963 |  |
| 0985 | 17 |  | RAL |  |
| 0986 | 326309 |  | STA 0963 |  |
| If CDE is not zero yet, then continue the long division loop. |
| 0989 | 79 |  | MOV A,C |  |
| 098A | B2 |  | ORA D |  |
| 098B | B3 |  | ORA E |  |
| 098C | C25409 |  | JNZ [FDivLoop](http://altairbasic.org/math_dis_4.htm#FDivLoop) |  |
| Finally divide FACCUM by 2 and loop back unless overflowed. |
| 098F | E5 |  | PUSH H |  |
| 0990 | 217201 |  | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 0993 | 35 |  | DCR M |  |
| 0994 | E1 |  | POP H |  |
| 0995 | C25409 |  | JNZ [FDivLoop](http://altairbasic.org/math_dis_4.htm#FDivLoop) |  |
| 0998 | C3A408 |  | JMP [Overflow](http://altairbasic.org/math_dis_3.htm#Overflow) |  |

**FExponentAdd**

Here is code common to FMul and FDiv and is called by both of them. It's main job is to add (for FMul) or subtract (for FDiv) the binary exponents of the lhs and rhs arguments, for which on entry L=0 for addition or L=FF respectively.

|  |
| --- |
| If BCDE is 0, then we don't need to do anything and can jump to the function exit. |
| 099B | 78 | FExponentAdd | MOV A,B |  |
| 099C | B7 |  | ORA A |  |
| 099D | CABA09 |  | JZ [FExponentAdd](http://altairbasic.org/math_dis_4.htm#FExponentAdd)+31 |  |
| Exponent arithmetic. |
| 09A0 | 7D |  | MOV A,L | A=0 for add, FF for subtract. |
| 09A1 | 217201 |  | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 09A4 | AE |  | XRA M | XOR with FAccum's exponent. |
| 09A5 | 80 |  | ADD B | Add exponents |
| 09A6 | 47 |  | MOV B,A |  |
| 09A7 | 1F |  | RAR | Carry (after the add) into bit 7. |
| 09A8 | A8 |  | XRA B | XOR with old bit 7. |
| 09A9 | 78 |  | MOV A,B |  |
| 09AA | F2B909 |  | JP [FExponentAdd](http://altairbasic.org/math_dis_4.htm#FExponentAdd)+30 | If |
| Add exponent bias, store in FACCUM+3 and if the result is 0 then discard return address and return to caller's caller. |
| 09AD | C680 |  | ADI 80 |  |
| 09AF | 77 |  | MOV M,A |  |
| 09B0 | CA2109 |  | JZ [PopHLandReturn](http://altairbasic.org/math_dis_4.htm#PopHLandReturn) |  |
| 09B3 | CD370A |  | CALL [FUnpackMantissas](http://altairbasic.org/math_dis_7.htm#FUnpackMantissas) |  |
| 09B6 | 77 |  | MOV M,A |  |
| 09B7 | 2B |  | DCX H |  |
| 09B8 | C9 |  | RET |  |
| 09B9 | B7 |  | ORA A |  |
| 09BA | E1 |  | POP H | Ignore return address so we'll end |
| 09BB | FAA408 |  | JM [Overflow](http://altairbasic.org/math_dis_3.htm#Overflow) | up to returning to caller's caller. And fall into FZero... |

**FZero**

Sets FACCUM to zero. Zero is stored in a slightly special way : the exponent is zero'ed without bias.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 09BE | AF | FZero | XRA A |  |
| 09BF | 327201 |  | STA [FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 09C2 | C9 |  | RET |  |

**FMulByTen**

Multiplies FACCUM by 10. Seems to be here for speed reasons, since this could be done very simply with a call to FMul.

|  |
| --- |
| Copy FACCUM to BCDE and return if it's 0. |
| 09C3 | CD1D0A | FMulByTen | CALL [FCopyToBCDE](http://altairbasic.org/math_dis_6.htm#FCopyToBCDE) |  |
| 09C6 | 78 |  | MOV A,B |  |
| 09C7 | B7 |  | ORA A |  |
| 09C8 | C8 |  | RZ |  |
| Multiply BCDE by 4. This is done by adding 2 to BCDE's exponent and erroring out if it overflows (ie exponent > 127 plus bias). |
| 09C9 | C602 |  | ADI 02 |  |
| 09CB | DAA408 |  | JC [Overflow](http://altairbasic.org/math_dis_3.htm#Overflow) |  |
| 09CE | 47 |  | MOV B,A |  |
| Add BCDE to FACCUM. So FACCUM is now the old FACCUM times 5. |
| 09CF | CD1208 |  | CALL [FAdd](http://altairbasic.org/math_dis_2.htm#FAdd)+2 |  |
| Multiply FACCUM by 2, done by incrementing the exponent. Now we have multiplied FACCUM by 10 and can exit, but notice that we also test for exponent overflow again. |
| 09D2 | 217201 |  | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 09D5 | 34 |  | INR M |  |
| 09D6 | C0 |  | RNZ |  |
| 09D7 | C3A408 |  | JMP [Overflow](http://altairbasic.org/math_dis_3.htm#Overflow) |  |

## 2.5 Sign Magic

A group of functions for testing and changing the sign of an fp number.

### 

### FTestSign\_tail

When FACCUM is non-zero, RST FTestSign jumps here to get the sign as an integer : 0x01 for positive, 0xFF for negative.

|  |
| --- |
| Load A with the most-significant byte of FACCUM, the top-most bit of which holds the sign. Then we LXI into SignToInt. |
| 09DA | 3A7101 | FTestSign\_tail | LDA [FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+2 |  |
| 09DD | FE.. |  | CPI .. |  |

### InvSignToInt

Inverts the sign byte in A before falling into SigntoInt.

|  |
| --- |
| Simply invert A. |
| 09DE | 2F | InvSignToInt | CMA |  |

### SignToInt

Converts the sign byte in A to 0x01 for positive, 0xFF for negative.

|  |
| --- |
| Get bit 7 into carry flag and subtract from itself with carry. If A was +ve then it is now 0, whereas if A was -ve then A is now FF. |
| 09DF | 17 | SignToInt | RAL |  |
| 09E0 | 9F |  | SBB A |  |
| Return if A is FF, otherwise return with A=1. |
| 09E1 | C0 |  | RNZ |  |
| 09E2 | 3C |  | INR A |  |
| 09E3 | C9 |  | RET |  |

### 

### Sgn

Returns an integer that indicates FACCUM's sign. We do this by a simple call to FTestSign which gets the answer in A, then fall into FCharToFloat to get that answer back into FACCUM.

|  |
| --- |
| Get FACCUM's sign in A. A will be 0x01 for positive, 0 for zero, and 0xFF for negative. |
| 09E4 | EF | Sgn | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |

### 

### FCharToFloat

Converts the signed byte in A to a floating-point number in FACCUM..

|  |
| --- |
| Get the char value in A as an unnormlised floating-point number. |
| 09E5 | 0688 | FCharToFloat | MVI B,88 | ie 2^8 |
| 09E7 | 110000 |  | LXI D,0000 |  |
| 09EA | 217201 |  | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 09ED | 4F |  | MOV C,A |  |
| ]Set FACCUM's exponent to 2^8, in preparation for a jump into FNormalise. |
| 09EE | 70 |  | MOV M,B |  |
| 09EF | 0600 |  | MVI B,00 |  |
| Set FTEMP to 0x80, another preparation step for FNormalise. |
| 09F1 | 23 |  | INX H |  |
| 09F2 | 3680 |  | MVI M,80 |  |
| Get sign into carry flag and jump to FNormalise. |
| 09F4 | 17 |  | RAL |  |
| 09F5 | C35B08 |  | JMP [FNormalise](http://altairbasic.org/math_dis_3.htm#FNormalise) |  |

### Abs

FACCUM = |FACCUM|.

|  |
| --- |
| Return if FACCUM is already positive, otherwise fall into FNegate to make it positive. |
| 09F8 | EF | Abs | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |
| 09F9 | F0 |  | RP |  |

### FNegate

Negate FACCUM's sign, ie FACCUM = -FACCUM.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 09FA | 217101 | FNegate | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+2 |  |
| 09FD | 7E |  | MOV A,M |  |
| 09FE | EE80 |  | XRI 80 |  |
| 0A00 | 77 |  | MOV M,A |  |
| 0A01 | C9 |  | RET |  |

## 2.6 Moving FACCUM about

A group of functions for loading, copying, and pushing FACCUM.

### FPush

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0A02 | EB | FPush | XCHG |  |
| 0A03 | 2A6F01 |  | LHLD [FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM) |  |
| 0A06 | E3 |  | XTHL |  |
| 0A07 | E5 |  | PUSH H |  |
| 0A08 | 2A7101 |  | LHLD [FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+2 |  |
| 0A0B | E3 |  | XTHL |  |
| 0A0C | E5 |  | PUSH H |  |
| 0A0D | EB |  | XCHG |  |
| 0A0E | C9 |  | RET |  |

### 

### FLoadFromMem

FLoadFromMem loads FACCUM with the fp number pointed to by HL. It does this by calling a function to load BCDE with the in-memory number, then falls into FLoadFromBCDE.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0A0F | CD200A | FLoadFromMem | CALL [FLoadBCDEfromMem](http://altairbasic.org/math_dis_6.htm#FLoadBCDEfromMem) |  |

### FLoadFromBCDE

Loads FACCUM with BCDE.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0A12 | EB | FLoadFromBCDE | XCHG |  |
| 0A13 | 226F01 |  | SHLD [FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM) |  |
| 0A16 | 60 |  | MOV H,B |  |
| 0A17 | 69 |  | MOV L,C |  |
| 0A18 | 227101 |  | SHLD [FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+2 |  |
| 0A1B | EB |  | XCHG |  |
| 0A1C | C9 |  | RET |  |

### FCopyToBCDE and FLoadBCDE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0A1D | 216F01 | FCopyToBCDE | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM) |  |
| 0A20 | 5E | FLoadBCDEfromMem | MOV E,M |  |
| 0A21 | 23 |  | INX H |  |
| 0A22 | 56 |  | MOV D,M |  |
| 0A23 | 23 |  | INX H |  |
| 0A24 | 4E |  | MOV C,M |  |
| 0A25 | 23 |  | INX H |  |
| 0A26 | 46 |  | MOV B,M |  |
| 0A27 | 23 | IncHL+Return | INX H |  |
| 0A28 | C9 |  | RET |  |

### FCopyToMem

Copies FACCUM to another place in memory pointed to by HL.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0A29 | 116F01 | FCopyToMem | LXI D,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM) |  |
| 0A2C | 0604 |  | MVI B,04 |  |
| 0A2E | 1A | FCopyLoop | LDAX D |  |
| 0A2F | 77 |  | MOV M,A |  |
| 0A30 | 13 |  | INX D |  |
| 0A31 | 23 |  | INX H |  |
| 0A32 | 05 |  | DCR B |  |
| 0A33 | C22E0A |  | JNZ [FCopyLoop](http://altairbasic.org/math_dis_6.htm#FCopyLoop) |  |
| 0A36 | C9 |  | RET |  |

## 2.7 Unpacking & Comparison

Two functions : the first is for unpacking the mantissas of two floating-point numbers, the second is for comparing two floating-point numbers.

### 

### FUnpackMantissas

Unpacks the mantissas of FACCUM and BCDE. This is simple enough - we just restore the missing most-significant bit, invariably a 1 (see tech note). Unfortunately, doing this loses the sign bits of both packed numbers.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| To compensate for this, a combination of both signs is returned. Duing the function FACC's sign is negated and later xor'ed with BCDE's sign, and returned in bit 7 of A. The effect of this is when the function returns, A is +ve if the signs mismatched, or -ve if the signs matched. | |  |  |  |  | | --- | --- | --- | --- | | **FACC** | **Negated FACC** | **BCDE** | **Result after XOR** | | + | - | + | - | | + | - | - | + | | - | + | + | + | | - | + | - | - | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0A37 | 217101 | FUnpackMantissas | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+2 |  |
| 0A3A | 7E |  | MOV A,M |  |
| 0A3B | 07 |  | RLC | Move FACCUM's sign to bit 0. |
| 0A3C | 37 |  | STC | Set MSB of FACCUM mantissa, |
| 0A3D | 1F |  | RAR | FACCUM's sign is now in carry. |
| 0A3E | 77 |  | MOV M,A |  |
| 0A3F | 3F |  | CMC | Negate FACCUM's sign. |
| 0A40 | 1F |  | RAR | Bit 7 of A is now FACCUM's sign. |
| 0A41 | 23 |  | INX H | Store negated FACCUM sign at FTEMP\_SIGN. |
| 0A42 | 23 |  | INX H |  |
| 0A43 | 77 |  | MOV M,A |  |
| 0A44 | 79 |  | MOV A,C |  |
| 0A45 | 07 |  | RLC | Set MSB of BCDE mantissa, |
| 0A46 | 37 |  | STC | BCDE's sign is now in carry. |
| 0A47 | 1F |  | RAR |  |
| 0A48 | 4F |  | MOV C,A |  |
| 0A49 | 1F |  | RAR | Bit 7 of A is now BCDE's sign |
| 0A4A | AE |  | XRA M | XORed with FTEMP\_SIGN. |
| 0A4B | C9 |  | RET |  |

### FCompare

Compares FACCUM to BCDE, with the result being returned in A as follows :

FACCUM > BCDE, A = 0x01.  
FACCUM < BCDE, A = 0xFF.  
FACCUM = BCDE, A = 0.

|  |
| --- |
| If BCDE is zero, then we don't need to compare and can just return via FTestSign. |
| 0A4C | 78 | FCompare | MOV A,B |  |
| 0A4D | B7 |  | ORA A |  |
| 0A4E | CA2800 |  | JZ [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |
| Set return address to InvSignToInt |
| 0A51 | 21DE09 |  | LXI H,[InvSignToInt](http://altairbasic.org/math_dis_5.htm#InvSignToInt) |  |
| 0A54 | E5 |  | PUSH H |  |
| Test FACCUM's sign, and return with A=the inverse of BCDE's sign if FACCUM is zero. |
| 0A55 | EF |  | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |
| 0A56 | 79 |  | MOV A,C |  |
| 0A57 | C8 |  | RZ |  |
|  |
| 0A58 | 217101 |  | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+2 |  |
| 0A5B | AE |  | XRA M |  |
| 0A5C | 79 |  | MOV A,C |  |
| 0A5D | F8 |  | RM |  |
| Call function to test for equality. If BCDE and FACCUM are equal, then this function will not return here, but to FCompare's caller. |
| 0A5E | CD640A |  | CALL [FIsEqual](http://altairbasic.org/math_dis_7.htm#FIsEqual) |  |
| Not equal. We get the carry flag (indicating greater/lesser relationship) into bit 7 of A and then XOR that with ??? and return to 09DE. |
| 0A61 | 1F |  | RAR |  |
| 0A62 | A9 |  | XRA C |  |
| 0A63 | C9 |  | RET |  |
| Test for equality between BCDE and FACCUM. |
| 0A64 | 23 | FIsEqual | INX H |  |
| 0A65 | 78 |  | MOV A,B |  |
| 0A66 | BE |  | CMP M |  |
| 0A67 | C0 |  | RNZ |  |
| 0A68 | 2B |  | DCX H |  |
| 0A69 | 79 |  | MOV A,C |  |
| 0A6A | BE |  | CMP M |  |
| 0A6B | C0 |  | RNZ |  |
| 0A6C | 2B |  | DCX H |  |
| 0A6D | 7A |  | MOV A,D |  |
| 0A6E | BE |  | CMP M |  |
| 0A6F | C0 |  | RNZ |  |
| 0A70 | 2B |  | DCX H |  |
| 0A71 | 7B |  | MOV A,E |  |
| 0A72 | 96 |  | SUB M |  |
| 0A73 | C0 |  | RNZ |  |
| Equality, ie BCDE==FACCUM. In this case we can lose the first two return addresses on the stack, and return the caller of FCompare. |
| 0A74 | E1 |  | POP H | Lose 0A5E |
| 0A75 | E1 |  | POP H | Lose 09DE |
| 0A76 | C9 |  | RET | Return to caller |

## 2.8 Converting to Integer

blah

### 

### FAsInteger

Returns the integer part of FACCUM in CDE.

|  |
| --- |
| Return with BCDE=0 if A=0. |
| 0A77 | 47 | FAsInteger | MOV B,A |  |
| 0A78 | 4F |  | MOV C,A |  |
| 0A79 | 57 |  | MOV D,A |  |
| 0A7A | 5F |  | MOV E,A |  |
| 0A7B | B7 |  | ORA A |  |
| 0A7C | C8 |  | RZ |  |
| Preserve HL and copy FACCUM to BCDE |
| 0A7D | E5 |  | PUSH H |  |
| 0A7E | CD1D0A |  | CALL [FCopyToBCDE](http://altairbasic.org/math_dis_6.htm#FCopyToBCDE) |  |
| Unpack FACCUM's mantissa to get the hidden most-significant-bit (see top of page) and get the sign back via an XOR to undo what FUnpackMantissas did with it. Preserve the sign in H |
| 0A81 | CD370A |  | CALL [FUnpackMantissas](http://altairbasic.org/math_dis_7.htm#FUnpackMantissas) |  |
| 0A84 | AE |  | XRA M | Get sign back |
| 0A85 | 67 |  | MOV H,A |  |
| If FACCUM was negative then decrement the mantissa (two's complement?) |
| 0A86 | FC9B0A |  | CM [FMantissaDec](http://altairbasic.org/math_dis_8.htm#FMantissaDec) |  |
| Shift mantissa in CDE right by (24-B) places. This gets the integer part of FACCUM into CDE, which is the whole point of the function. |
| 0A89 | 3E98 |  | MVI A,98 | Shift mantissa right |
| 0A8B | 90 |  | SUB B | by (24-exponent) places? |
| 0A8C | CDC908 |  | CALL [FMantissaRtOnce](http://altairbasic.org/math_dis_3.htm#FMantissaRtOnce) | WHY? |
| If floating point sign is negative (ie bit 7 of H set) then two's complement CDE to get the signed integer. We do two's complement by first adding 1, then negating. |
| 0A8F | 7C |  | MOV A,H |  |
| 0A90 | 17 |  | RAL |  |
| 0A91 | DC9A08 |  | CC [FMantissaInc](http://altairbasic.org/math_dis_3.htm#FMantissaInc) |  |
| 0A94 | 0600 |  | MVI B,00 | Needed for FNegateInt. |
| 0A96 | DCB508 |  | CC [FNegateInt](http://altairbasic.org/math_dis_3.htm#FNegateInt) |  |
| Restore HL and return with the integer result in CDE. |
| 0A99 | E1 |  | POP H |  |
| 0A9A | C9 |  | RET |  |

### FMantissaDec

Decrements the mantissa in CDE.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0A9B | 1B | FMantissaDec | DCX D | DE-- |
| 0A9C | 7A |  | MOV A,D | If DE!=0xFFFF... |
| 0A9D | A3 |  | ANA E |  |
| 0A9E | 3C |  | INR A |  |
| 0A9F | C0 |  | RNZ | ... then return |
| 0AA0 | 0D |  | DCR C | C-- |
| 0AA1 | C9 |  | RET |  |

### Int

Removes the fractional part of FACCUM.

|  |
| --- |
| If FACCUM's exponent is >= 2^24, then it's too big to hold any fractional part - it is already an integer, so we just return. |
| 0AA2 | 217201 | Int | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 0AA5 | 7E |  | MOV A,M |  |
| 0AA6 | FE98 |  | CPI 98 |  |
| 0AA8 | D0 |  | RNC |  |
| Convert FACCUM to an integer in CDE. On returning, HL points to FACCUM's exponent byte (ie FACCUM+3). |
| 0AA9 | CD770A |  | CALL [FAsInteger](http://altairbasic.org/math_dis_8.htm#FAsInteger) |  |
| Now we need to convert the integer in CDE to a proper floating point value. To do this we first set FACCUM's exponent to 2^24. |
| 0AAC | 3698 |  | MVI M,98 |  |
| 0AAE | 79 |  | MOV A,C |  |
| 0AAF | 17 |  | RAL |  |
| 0AB0 | C35B08 |  | JMP 085B |  |

## 2.9 Reading Numbers

Function that reads a floating-point number from ASCII text.

### FIn

Reads a string and converts it to a floating point number in FACCUM. The first thing we do is some initialisation.

|  |
| --- |
| Decrement string ptr so it points to just before the number, also set FACCUM to 0. |
| 0AB3 | 2B | FIn | DCX H |  |
| 0AB4 | CDBE09 |  | CALL [FZero](http://altairbasic.org/math_dis_4.htm#FZero) |  |
| Set B,D,E to 0 and C to 255. |
| 0AB7 | 47 |  | MOV B,A | B=count of fractional digits |
| 0AB8 | 57 |  | MOV D,A | D=exponent sign |
| 0AB9 | 5F |  | MOV E,A | E=exponent |
| 0ABA | 2F |  | CMA | C=decimal\_point\_done (0xFF for no, 0x00 for yes) |
| 0ABB | 4F |  | MOV C,A |  |

This is the head of the loop that processes one character of ASCII text at a time.

|  |
| --- |
| Get next ASCII character and if it's a digit (as determined by carry flag) then jump down to ProcessDigit. |
| 0ABC | D7 | FInLoop | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0ABD | DA040B |  | JC [ProcessDigit](http://altairbasic.org/math_dis_9.htm#ProcessDigit) |  |
| If we have a decimal point then jump down to increment C and loop back up here (unless we've already encountered a decimal point). |
| 0AC0 | FE2E |  | CPI '.' |  |
| 0AC2 | CAE40A |  | JZ 0AE4 |  |
| If it's not an 'E' indicating exponent then assume we've reached the end of the number and jump to ScaleResult. |
| 0AC5 | FE45 |  | CPI 'E' |  |
| 0AC7 | C2E80A |  | JNZ [ScaleResult](http://altairbasic.org/math_dis_9.htm#ScaleResult) |  |

A decimal exponent has been given (eg 'E+23'). Here we read that (signed) exponent value into E. In this block we use D to hold the exponent sign (0xFF for minus and 0x01 for positive) while we loop reading the digits

|  |
| --- |
| Get first character of exponent (following the E). |
| 0ACA | D7 | GetExponent | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0ACB | 15 |  | DCR D |  |
| 0ACC | FE99 |  | CPI KWID\_- |  |
| 0ACE | CAD80A |  | JZ [NextExponentDigit](http://altairbasic.org/math_dis_9.htm#NextExponentDigit) |  |
| 0AD1 | 14 |  | INR D |  |
| 0AD2 | FE98 |  | CPI KWID\_+ |  |
| 0AD4 | CAD80A |  | JZ [NextExponentDigit](http://altairbasic.org/math_dis_9.htm#NextExponentDigit) |  |
| Exponent sign was not given, so it's implicitly +ve (D is 0 after above). We decrement HL so we can get the same character again without jumping. |
| 0AD7 | 2B |  | DCX H |  |
| If the next char is a digit (of the exponent) then jump down to deal with it. |
| 0AD8 | D7 | NextExponentDigit | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0AD9 | DA230B |  | JC [DoExponentDigit](http://altairbasic.org/math_dis_9.htm#DoExponentDigit) |  |
| 0ADC | 14 |  | INR D |  |
| 0ADD | C2E80A |  | JNZ [ScaleResult](http://altairbasic.org/math_dis_9.htm#ScaleResult) |  |
| D was 0xFF (ie negative exponent) therefore E = 0 - E |
| 0AE0 | AF |  | XRA A |  |
| 0AE1 | 93 |  | SUB E |  |
| 0AE2 | 5F |  | MOV E,A |  |
| 0AE3 | 0C |  | INR C | C was 0xFF, so here it |
| 0AE4 | 0C |  | INR C | becomes 0x01. |
| 0AE5 | CABC0A |  | JZ [FInLoop](http://altairbasic.org/math_dis_9.htm#FInLoop) | If C is now zero |

We believe we've read all the characters that make up the number. The last thing that needs to be done is decimally scale our number in FACCUM with respect to the position of the decimal point and the decimal exponent value in E.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0AE8 | E5 | ScaleResult | PUSH H |  |
| The number of decimal positions to shift by is given by E-B, that is the (E)xponent minus the number of fractional digits given. The direction to shift in is also given by this subtraction, in the sign flag. This is preserved in the following loop. |
| 0AE9 | 7B |  | MOV A,E |  |
| 0AEA | 90 |  | SUB B |  |
| If we're shifting up (ie \*ing by 10) then call the function to do that and jump to the end of the loop. |
| 0AEB | F4FC0A | DecimalLoop | CP [DecimalShiftUp](http://altairbasic.org/math_dis_9.htm#DecimalShiftUp) |  |
| 0AEE | F2F70A |  | JP [DecimalLoopEnd](http://altairbasic.org/math_dis_9.htm#DecimalLoopEnd) |  |
| We're shifting down (ie sign flag is set) so call the function to do that and increment A, |
| 0AF1 | F5 |  | PUSH PSW |  |
| 0AF2 | CD2309 |  | CALL FDivByTen |  |
| 0AF5 | F1 |  | POP PSW |  |
| 0AF6 | 3C |  | INR A |  |
| If A is not yet 0 then we have more shifting to do and so loop back. Otherwise we restore the prog ptr and return. |
| 0AF7 | C2EB0A | DecimalLoopEnd | JNZ [DecimalLoop](http://altairbasic.org/math_dis_9.htm#DecimalLoop) |  |
| 0AFA | E1 |  | POP H |  |
| 0AFB | C9 |  | RET |  |

Helper function for shifting the result decimally up one place. We only do this if A !=0, and at the end we decrement A before returning.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0AFC | C8 | DecimalShiftUp | RZ |  |
| 0AFD | F5 |  | PUSH PSW |  |
| 0AFE | CDC309 |  | CALL [FMulByTen](http://altairbasic.org/math_dis_4.htm#FMulByTen) |  |
| 0B01 | F1 |  | POP PSW |  |
| 0B02 | 3D |  | DCR A |  |
| 0B03 | C9 |  | RET |  |

Process an ASCII digit. We multiply the current total by ten, then add this digit value to it.

|  |
| --- |
| Preserve the current exponent (DE) and copy the ASCII digit value to D. |
| 0B04 | D5 | ProcessDigit | PUSH D |  |
| 0B05 | 57 |  | MOV D,A |  |
| Increment the count of fractional digits in B. C is 0xFF if no decimal point has been encountered, and 0 if it has. Adding with carry (which is set before this block is jumped to) makes this work. |
| 0B06 | 78 |  | MOV A,B |  |
| 0B07 | 89 |  | ADC C |  |
| 0B08 | 47 |  | MOV B,A |  |
| 0B09 | C5 |  | PUSH B |  |
| 0B0A | E5 |  | PUSH H |  |
| Multiply the current total in FACCUM by ten and get the digit value in A. |
| 0B0B | D5 |  | PUSH D |  |
| 0B0C | CDC309 |  | CALL [FMulByTen](http://altairbasic.org/math_dis_4.htm#FMulByTen) |  |
| 0B0F | F1 |  | POP PSW |  |
| 0B10 | D630 |  | SUI '0' |  |
| Add this digit's value to FACCUM. To do this we first preserve FACCUM on the stack, then load FACCUM with the digit value (in A) via a call to FCharToFloat. Then we get the old FACCUM off the stack into BCDE and call FAdd. |
| 0B12 | CD020A |  | CALL [FPush](http://altairbasic.org/math_dis_6.htm#FPush) |  |
| 0B15 | CDE509 |  | CALL [FCharToFloat](http://altairbasic.org/math_dis_5.htm#FCharToFloat) |  |
| 0B18 | C1 |  | POP B |  |
| 0B19 | D1 |  | POP D |  |
| 0B1A | CD1208 |  | CALL [FAdd](http://altairbasic.org/math_dis_2.htm#FAdd)+2 |  |
| Restore regpairs and jump back to FInLoop |
| 0B1D | E1 |  | POP H |  |
| 0B1E | C1 |  | POP B |  |
| 0B1F | D1 |  | POP D |  |
| 0B20 | C3BC0A |  | JMP [FInLoop](http://altairbasic.org/math_dis_9.htm#FInLoop) |  |

Deal with a digit of the exponent. This is simple - just multiply the current exponent value (in E) by ten, then add the digit value to it, and finally jump back.

|  |
| --- |
| E=E\*10 + (HL) |
| 0B23 | 7B | DoExponentDigit | MOV A,E |  |
| 0B24 | 07 |  | RLC |  |
| 0B25 | 07 |  | RLC |  |
| 0B26 | 83 |  | ADD E |  |
| 0B27 | 07 |  | RLC |  |
| 0B28 | 86 |  | ADD M |  |
| 0B29 | D630 |  | SUI '0' |  |
| 0B2B | 5F |  | MOV E,A |  |
| 0B2C | C3D80A |  | JMP [NextExponentDigit](http://altairbasic.org/math_dis_9.htm#NextExponentDigit) |  |

## 2.10 Printing Numbers

Functions for printing floating-point numbers.

### PrintIN

Prints "IN " and falls into PrintInt. Used by the error handling code to print stuff like "?SN ERROR IN 50".

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0B2F | E5 | PrintIN | PUSH H |  |
| 0B30 | 218801 |  | LXI H,[szIn](http://altairbasic.org/int_dis_3.htm#szIn) |  |
| 0B33 | CDA305 |  | CALL [PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |
| 0B36 | E1 |  | POP H |  |

### PrintInt

Promotes the integer in HL to a floating-point number in FACC, sets the return address to PrintSz-1, and falls into FOut. The promotion from integer to float is interesting : the integer starts off by occupying the least significant bits of the mantissa CDE. The exponent in B is set to 24 (because, thus giving us an unnormalised but perfectly valid floating-point number in no time at all! Took me a while to see that...

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0B37 | EB | PrintInt | XCHG | DE=integer |
| 0B38 | AF |  | XRA A | A=0 (ends up in C) |
| 0B39 | 0698 |  | MVI B,98 | B (ie exponent) = 24 |
| 0B3B | CDEA09 |  | CALL [FCharToFloat](http://altairbasic.org/math_dis_5.htm#FCharToFloat)+5 |  |
| 0B3E | 21A205 |  | LXI H,PrintSz-1 |  |
| 0B41 | E5 |  | PUSH H |  |

### FOut

Prints a floating point number to the terminal.

|  |
| --- |
| Set HL to FBUFFER, which is where FACCUM gets printed to. |
| 0B42 | 217401 | FOut | LXI H,[FBUFFER](http://altairbasic.org/int_dis_3.htm#FBUFFER) |  |
| 0B45 | E5 |  | PUSH H |  |
| Test FACCUM. If it's positive then write a leading space; if it's negative then write a leading minus sign. |
| 0B46 | EF |  | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |
| 0B47 | 3620 |  | MVI M,' ' |  |
| 0B49 | F24E0B |  | JP [DoZero](http://altairbasic.org/math_dis_10.htm#DoZero) |  |
| 0B4C | 362D |  | MVI M,'-' |  |
| Write a '0', and if FACCUM equals 0 then jump to NullTerm-3, which is a convenient shortcut for null-terminating the output buffer. Jumping to NullTerm-3 means that a spurious byte (in C) gets written immediately following the null-terminator, but this isn't a problem because we're nowhere near the end of the buffer plus doing it this way we save a couple of bytes we would have lost had we insisted on jumping to NullTerm with C explicitly set to 0. If FACCUM is not zero then the '0' gets overwritten a few lines down. |
| 0B4E | 23 | DoZero | INX H |  |
| 0B4F | 3630 |  | MVI M,'0' |  |
| 0B51 | CAF70B |  | JZ [NullTerm](http://altairbasic.org/math_dis_10.htm#NullTerm)-3 |  |
| Make FACCUM a positive number by negating it if it's negative. |
| 0B54 | E5 |  | PUSH H |  |
| 0B55 | FCFA09 |  | CM [FNegate](http://altairbasic.org/math_dis_5.htm#FNegate) |  |
| Initialise Decimal Exponent Adjustment (hereafter shortened to DecExpAdj) to 0. |
| 0B58 | AF |  | XRA A |  |
| 0B59 | F5 |  | PUSH PSW |  |
| Here's where we bring FACCUM into range between 100,000 and 1,000,000 by multiplying or dividing by ten a number of times. The first call ensures FACCUM is less than 1,000,000 and the loop that follows makes it more than or equal to 100,000. The decimal exponent that we had to use on FACCUM to get it into this range (referred to as DecExpAdj) is kept on the stack. |
| 0B5A | CDFD0B |  | CALL [ToUnder1,000,000](http://altairbasic.org/math_dis_10.htm#ToUnder1,000,000) |  |
| 0B5D | 014391 | ToOver100,000 | LXI B,9143 | BCDE=(float)100,000. |
| 0B60 | 11F84F |  | LXI D,4FF8 |  |
| 0B63 | CD4C0A |  | CALL [FCompare](http://altairbasic.org/math_dis_7.htm#FCompare) | If FACCUM >= 100,000 |
| 0B66 | E27A0B |  | JPO [PrepareToPrint](http://altairbasic.org/math_dis_10.htm#PrepareToPrint) | then jump to PrepareToPrint. |
| 0B69 | F1 |  | POP PSW | A=DecExpAdj |
| 0B6A | CDFD0A |  | CALL [DecimalShiftUp](http://altairbasic.org/math_dis_9.htm#DecimalShiftUp) | FACCUM\*=10; DecExpAdj--; |
| 0B6D | F5 |  | PUSH PSW |  |
| 0B6E | C35D0B |  | JMP [ToOver100,000](http://altairbasic.org/math_dis_10.htm#ToOver100,000) |  |
| Divide FACCUM by ten and increment DecExpAdj. |
| 0B71 | CD2309 |  | CALL DecimalShiftDown |  |
| 0B74 | F1 |  | POP PSW |  |
| 0B75 | 3C |  | INR A | DecExpAdj++; |
| 0B76 | F5 |  | PUSH PSW |  |
| 0B77 | CDFD0B |  | CALL [ToUnder1,000,000](http://altairbasic.org/math_dis_10.htm#ToUnder1,000,000) |  |
| Some preparation. We add 0.5 to FACCUM, make it an integer, then finally store the result of that in FACCUM. |
| 0B7A | CD0108 | PrepareToPrint | CALL [FAddOneHalf](http://altairbasic.org/math_dis_1.htm#FAddOneHalf) |  |
| 0B7D | 3C |  | INR A |  |
| 0B7E | CD770A |  | CALL [FAsInteger](http://altairbasic.org/math_dis_8.htm#FAsInteger) |  |
| 0B81 | CD120A |  | CALL [FLoadFromBCDE](http://altairbasic.org/math_dis_6.htm#FLoadFromBCDE) |  |
| fixme. |
| 0B84 | 010602 |  | LXI B,0206 |  |
| 0B87 | F1 |  | POP PSW | A=DecExpAdj+6. |
| 0B88 | 81 |  | ADD C |  |
| 0B89 | FA950B |  | JM 0B95 | If A<1 or A>6 Then goto fixme. |
| 0B8C | FE07 |  | CPI 07 |  |
| 0B8E | D2950B |  | JNC 0B95 |  |
| 0B91 | 3C |  | INR A |  |
| 0B92 | 47 |  | MOV B,A |  |
| 0B93 | 3E01 |  | MVI A,01 | A=1, indicating scientific notation. |
| fixme. |
| 0B95 | 3D |  | DCR A |  |
| 0B96 | E1 |  | POP H | HL=output buffer |
| 0B97 | F5 |  | PUSH PSW | Preserve decimal exponent adjustment (and preserve zero flag used to indicate scientific notation wanted). |
| 0B98 | 110F0C |  | LXI D,[DECIMAL\_POWERS](http://altairbasic.org/math_dis_10.htm#DECIMAL_POWERS) |  |
| NextDigit. This is the outer loop of printing, where each ASCII digit is calculated in turn. We start by writing out the decimal point, but we only advance HL to keep it if B==0, which means (obviously) that the decimal point has been reached. |
| 0B9B | 05 | NextDigit | DCR B |  |
| 0B9C | 362E |  | MVI M,'.' |  |
| 0B9E | CC270A |  | CZ [IncHL+Return](http://altairbasic.org/math_dis_6.htm#IncHL+Return) | 0A27 just happens to inc HL and RET. |
| 0BA1 | C5 |  | PUSH B |  |
| 0BA2 | E5 |  | PUSH H |  |
| 0BA3 | D5 |  | PUSH D | DE=>decimal power |
| 0BA4 | CD1D0A |  | CALL [FCopyToBCDE](http://altairbasic.org/math_dis_6.htm#FCopyToBCDE) | Store BCDE to FACCUM. |
| 0BA7 | E1 |  | POP H | HL=>decimal power. |
| 0BA8 | 062F |  | MVI B,'0'-1 |  |
| Work out the digit corresponding to the current decimal power. We do this by subtracting the decimal power (eg 100) from CDE until it overflows, and incrementing the ASCII digit value in B each time. When it overflows, we have our digit. And when it overflows, we call FAddMantissas to undo the last subtraction which was one step too far. |
| 0BAA | 04 | DigitLoop | INR B |  |
| 0BAB | 7B |  | MOV A,E |  |
| 0BAC | 96 |  | SUB M |  |
| 0BAD | 5F |  | MOV E,A |  |
| 0BAE | 23 |  | INX H |  |
| 0BAF | 7A |  | MOV A,D |  |
| 0BB0 | 9E |  | SBB M |  |
| 0BB1 | 57 |  | MOV D,A |  |
| 0BB2 | 23 |  | INX H |  |
| 0BB3 | 79 |  | MOV A,C |  |
| 0BB4 | 9E |  | SBB M |  |
| 0BB5 | 4F |  | MOV C,A |  |
| 0BB6 | 2B |  | DCX H |  |
| 0BB7 | 2B |  | DCX H |  |
| 0BB8 | D2AA0B |  | JNC [DigitLoop](http://altairbasic.org/math_dis_10.htm#DigitLoop) |  |
| 0BBB | CDA908 |  | CALL [FAddMantissas](http://altairbasic.org/math_dis_3.htm#FAddMantissas) |  |
| 0BBE | 23 |  | INX H | ??? |
| 0BBF | CD120A |  | CALL [FLoadFromBCDE](http://altairbasic.org/math_dis_6.htm#FLoadFromBCDE) |  |
| Write out the digit. If we still have digits to do then loop back. |
| 0BC2 | EB |  | XCHG |  |
| 0BC3 | E1 |  | POP H | HL=output buffer |
| 0BC4 | 70 |  | MOV M,B |  |
| 0BC5 | 23 |  | INX H |  |
| 0BC6 | C1 |  | POP B | B=decimal point place |
| 0BC7 | 0D |  | DCR C | C=digits remaining, minus one. |
| 0BC8 | C29B0B |  | JNZ [NextDigit](http://altairbasic.org/math_dis_10.htm#NextDigit) |  |
| 0BCB | 05 |  | DCR B |  |
| 0BCC | CADB0B |  | JZ 0BDB |  |
| Move HL one byte behind the first trailing zero. |
| 0BCF | 2B |  | DCX H |  |
| 0BD0 | 7E |  | MOV A,M |  |
| 0BD1 | FE30 |  | CPI '0' |  |
| 0BD3 | CACF0B |  | JZ 0BCF |  |
| If we've no decimal point, then increment HL so it's |
| 0BD6 | FE2E |  | CPI '.' |  |
| 0BD8 | C4270A |  | CNZ [IncHL+Return](http://altairbasic.org/math_dis_6.htm#IncHL+Return) |  |
| 0BDB | F1 |  | POP PSW |  |
| 0BDC | CAFA0B |  | JZ [NullTerm](http://altairbasic.org/math_dis_10.htm#NullTerm) |  |
| Write exponent part of scientific format. |
| 0BDF | 3645 |  | MVI M,'E' | Write 'E' |
| 0BE1 | 23 |  | INX H |  |
| 0BE2 | 362B |  | MVI M,'+' | Write '+' or '-' |
| 0BE4 | F2EB0B |  | JP 0BEB |  |
| 0BE7 | 362D |  | MVI M,'-' | Write '-' if it's negative, also |
| 0BE9 | 2F |  | CMA | two's complement the decimal exponent |
| 0BEA | 3C |  | INR A | so printing it will work. |
| 0BEB | 062F |  | MVI B,'0'-1 |  |
| Work out the first digit of exponent in B. Done by usual method of repeatedly subtracting 10 until it overflows. |
| 0BED | 04 | ExpDigitLoop | INR B |  |
| 0BEE | D60A |  | SUI 0A |  |
| 0BF0 | D2ED0B |  | JNC [ExpDigitLoop](http://altairbasic.org/math_dis_10.htm#ExpDigitLoop) |  |
| 0BF3 | C63A |  | ADI 3A | Adding '0'+10 gives us the 2nd digit |
| 0BF5 | 23 |  | INX H | of the exponent. |
| 0BF6 | 70 |  | MOV M,B | Write first digit. |
| 0BF7 | 23 |  | INX H |  |
| 0BF8 | 77 |  | MOV M,A | Write second digit of exponent. |
| 0BF9 | 23 |  | INX H |  |
| 0BFA | 71 | NullTerm | MOV M,C | Null byte terminator. |
| 0BFB | E1 |  | POP H |  |
| 0BFC | C9 |  | RET |  |

### ToUnder1,000,000

Divides FACCUM by ten until it's less than 1,000,000. This function is semi-recursive... if it needs to recurse (ie

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0BFD | 017494 | ToUnder1,000,000 | LXI B,9474 | BCDE=(float) 1,000,000 |
| 0C00 | 11F723 |  | LXI D,23F7 |  |
| 0C03 | CD4C0A |  | CALL [FCompare](http://altairbasic.org/math_dis_7.htm#FCompare) |  |
| 0C06 | E1 |  | POP H |  |
| 0C07 | E2710B |  | JPO 0B71 |  |
| 0C0A | E9 |  | PCHL |  |

### ONE\_HALF

Constant value 0.5, used by FRoundUp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0C0B | 00000080 | ONE\_HALF | DD 0.5 |  |

### DECIMAL\_POWERS

Table of powers of ten.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0C0F | A08601 | DECIMAL\_POWERS | DT 100000 |  |
| 0C13 | 102700 |  | DT 10000 |  |
| 0C17 | E80300 |  | DT 1000 |  |
| 0C1B | 640000 |  | DT 100 |  |
| 0C1F | 0A0000 |  | DT 10 |  |
| 0C1F | 010000 |  | DT 1 |  |

## 2.11 SQR - Optional Function #1

As mentioned elsewhere, during initialisation the user was asked if he wanted the SIN, RND, and SQR functions to be available. The reason for this was to give the user a chance to save a bit more memory for their programs, by having Basic reclaim the memory used by those functions. That is why these functions are at the very top of the program code.

### Sqr

We use Newton's method to get a close approximation to the square root.

|  |
| --- |
| If FACCUM is negative then error out with invalid Function Call (FC) error. |
| 0C21 | EF | Sqr | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |
| 0C22 | FA9804 |  | JM [FunctionCallError](http://altairbasic.org/int_dis_10.htm#FunctionCallError) |  |
| Return immediately if FACCUM is zero, since SQR(0)==0. |
| 0C25 | C8 |  | RZ |  |
| Get exponent div 2, plus 0x40 (from the bias of 0x80) and keep this value on the stack. Also preserve ptr to exponent byte. |
| 0C26 | 217201 |  | LXI H,[FACCUM](http://altairbasic.org/int_dis_3.htm#FACCUM)+3 |  |
| 0C29 | 7E |  | MOV A,M |  |
| 0C2A | 1F |  | RAR |  |
| 0C2B | F5 |  | PUSH PSW |  |
| 0C2C | E5 |  | PUSH H |  |
| Set exponent to 0. |
| 0C2D | 3E40 |  | MVI A,40 |  |
| 0C2F | 17 |  | RAL |  |
| 0C30 | 77 |  | MOV M,A |  |
| Create a copy of FACCUM in FBUFFER. |
| 0C31 | 217401 |  | LXI H,[FBUFFER](http://altairbasic.org/int_dis_3.htm#FBUFFER) |  |
| 0C34 | CD290A |  | CALL [FCopyToMem](http://altairbasic.org/math_dis_6.htm#FCopyToMem) |  |
| Loop through Newton's method 4 times. |
| 0C37 | 3E04 |  | MVI A,04 |  |
| 0C39 | F5 | SqrLoop | PUSH PSW |  |
| FACCUM = ( (Original FACCUM / FACCUM) + FACCUM) \* 0.5. This is the essence of newton's method. |
| 0C3A | CD020A |  | CALL [FPush](http://altairbasic.org/math_dis_6.htm#FPush) |  |
| 0C3D | 217401 |  | LXI H,[FBUFFER](http://altairbasic.org/int_dis_3.htm#FBUFFER) |  |
| 0C40 | CD200A |  | CALL [FLoadBCDEfromMem](http://altairbasic.org/math_dis_6.htm#FLoadBCDEfromMem) |  |
| 0C43 | CD3109 |  | CALL [FDiv](http://altairbasic.org/math_dis_4.htm#FDiv)+2 |  |
| 0C46 | C1 |  | POP B |  |
| 0C47 | D1 |  | POP D |  |
| 0C48 | CD1208 |  | CALL [FAdd](http://altairbasic.org/math_dis_2.htm#FAdd)+2 |  |
| 0C4B | 010080 |  | LXI B,8000 |  |
| 0C4E | 51 |  | MOV D,C |  |
| 0C4F | 59 |  | MOV E,C |  |
| 0C50 | CDE508 |  | CALL [FMul](http://altairbasic.org/math_dis_4.htm#FMul) |  |
| Loop until done 4 iterations |
| 0C53 | F1 |  | POP PSW |  |
| 0C54 | 3D |  | DCR A |  |
| 0C55 | C2390C |  | JNZ [SqrLoop](http://altairbasic.org/math_dis_11.htm#SqrLoop) |  |
| Restore ptr to exponent byte, and restore the original exponent-div-2-plus-0x40 to A. |
| 0C58 | E1 |  | POP H |  |
| 0C59 | F1 |  | POP PSW |  |
| Correct the bias, so now we have original exponent-div-2 in A. This is then added to the existing exponent and stored. |
| 0C5A | C6C0 |  | ADI C0 |  |
| 0C5C | 86 |  | ADD M |  |
| 0C5D | 77 |  | MOV M,A |  |
| 0C5E | C9 |  | RET |  |

## 2.12 RND - Optional Function #2

### Rnd

Generates a random number. This is a bit odd... like all inline functions it takes a numeric argument, but in RNDs case this argument is mostly ignored. If it's a negative number it skips a couple of stages of reseeding RND\_SEED.

|  |
| --- |
| If tghe argument in FACCUM is negative, then skip over the |
| 0C5F | EF | Rnd | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |
| 0C60 | FA7C0C |  | JM 0C7C |  |
| Load the seed into FACCUM. |
| 0C63 | 21910C |  | LXI H,[RND\_SEED](http://altairbasic.org/math_dis_12.htm#RND_SEED) |  |
| 0C66 | CD0F0A |  | CALL [FLoadFromMem](http://altairbasic.org/math_dis_6.htm#FLoadFromMem) |  |
| 0C69 | C8 |  | RZ |  |
| Multiply seed by 11,879,546. |
| 0C6A | 013598 |  | LXI B,9835 |  |
| 0C6D | 117A44 |  | LXI D,447A |  |
| 0C70 | CDE508 |  | CALL [FMul](http://altairbasic.org/math_dis_4.htm#FMul)+2 |  |
| Add 0.00000003927678 |
| 0C73 | 012868 |  | LXI B,6828 |  |
| 0C76 | 1146B1 |  | LXI D,B146 |  |
| 0C79 | CD1208 |  | CALL [FAdd](http://altairbasic.org/math_dis_2.htm#FAdd)+2 |  |
| Swap first and third mantissa bytes. |
| 0C7C | CD1D0A |  | CALL [FCopyToBCDE](http://altairbasic.org/math_dis_6.htm#FCopyToBCDE) |  |
| 0C7F | 7B |  | MOV A,E |  |
| 0C80 | 59 |  | MOV E,C |  |
| 0C81 | 4F |  | MOV C,A |  |
| Set FTEMP\_SIGN to 0x80 to indicate to FNormalise that it doesn't need to change the sign. Also set FACCUM's exponent to 0 so the result, when normalised, will be less than 1. |
| 0C82 | 3680 |  | MVI M,80 |  |
| 0C84 | 2B |  | DCX H |  |
| 0C85 | 46 |  | MOV B,M |  |
| 0C86 | 3680 |  | MVI M,80 |  |
| 0C88 | CD5E08 |  | CALL [FNormalise](http://altairbasic.org/math_dis_3.htm#FNormalise)+3 |  |
| Exit via a copy of the result to RND\_SEED so it can be used for the next time RND is called. |
| 0C8B | 21910C |  | LXI H,[RND\_SEED](http://altairbasic.org/math_dis_12.htm#RND_SEED) |  |
| 0C8E | C3290A |  | JMP [FCopyToMem](http://altairbasic.org/math_dis_6.htm#FCopyToMem) |  |
| Seed for random number generation. |
| 0C91 | 52C74F80 | RND\_SEED | 0.811635 |  |

## 2.13 SIN - Optional Function #3

Returns the sine of the radians argument, which we'll call x. This is the most mathematically-intense function, and I have to thank Ian Griffiths again for doing the hard work behind the analysis presented here.

***The MacLaurin Series***

Altair BASIC's calculation of the sine function is based upon the MacLaurin series. A MacLaurin series is an expansion of functions whose derivatives are continuous.

http://altairbasic.org/sinm1.png

Now, how do we use this series for the sine function? It's going to look like this :

http://altairbasic.org/sinm3.png

Well, we know that the derivative of sin(x) is cos(x), and the derivative of cos(x) is -sin(x). We also know that sin(0)=0, and cos(x)=1. Therefore the series can be reduced to :

http://altairbasic.org/sinm4.png

Simplifying a bit further, and limiting the number of terms to a reasonable approximation, we get down to :

http://altairbasic.org/sin1.png

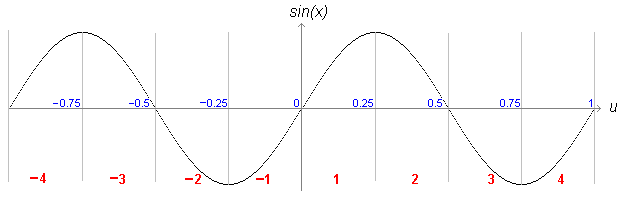
***Scaling***

To make the code in this and the next section smaller, our first step is to scale our radians argument x into a new variable u that is in the range ***-1  u 1***. To do this we first divide x by 2 and then lose the integer part of that number.

|  |
| --- |
| Divide x (in FACCUM) by 2 to get *u*. |
| 0C95 | CD020A | Sin | CALL [FPush](http://altairbasic.org/math_dis_6.htm#FPush) | Push x |
| 0C98 | 014983 |  | LXI B,8349 | BCDE=2 |
| 0C9B | 11DB0F |  | LXI D,0FDB |  |
| 0C9E | CD120A |  | CALL [FLoadFromBCDE](http://altairbasic.org/math_dis_6.htm#FLoadFromBCDE) | rhs = 2 |
| 0CA1 | C1 |  | POP B | lhs = x |
| 0CA2 | D1 |  | POP D |  |
| 0CA3 | CD3109 |  | CALL [FDiv](http://altairbasic.org/math_dis_4.htm#FDiv)+2 | u=x/2 |
| Lose the integer part of u. |
| 0CA6 | CD020A |  | CALL [FPush](http://altairbasic.org/math_dis_6.htm#FPush) |  |
| 0CA9 | CDA20A |  | CALL [Int](http://altairbasic.org/math_dis_8.htm#Int) | rhs = INT(u) |
| 0CAC | C1 |  | POP B | lhs = u |
| 0CAD | D1 |  | POP D |  |
| 0CAE | CD0C08 |  | CALL [FSub](http://altairbasic.org/math_dis_2.htm#FSub)+2 | u=u-INT(u) |

***Quadrantization***

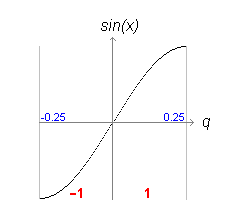
The second part of the algorithm is an optimisation to reduce *u* to a 'quadrantized' value, *q*, so called because it lies within the two quadrants either side of the origin, ie ***-0.25<q<0.25***. Here's a little graph showing sin(x) against u, with the quadrant numbers shown in red.



To get from *u* to *q*, we don't need to scale again but we do take advantage of two trignometric identities : *sin(-x) = -sin(x)*, and *sin(x)=sin(pi-x)*.

<fixme: insert Monte's explanation here>

So we have our value *q*. Here's another graph showing *sin(x)* against the quadrantised value *q*.



**fixme: the comments inlined below are train-of-thought and should not be used. I haven't quite worked this out.**

|  |
| --- |
| Firstly we subtract from 0.25 to get x from -0.75<=x<1.25 |
| 0CB1 | 01007F |  | LXI B,7F00 | BCDE=0.25 |
| 0CB4 | 51 |  | MOV D,C |  |
| 0CB5 | 59 |  | MOV E,C |  |
| 0CB6 | CD0C08 |  | CALL [FSub](http://altairbasic.org/math_dis_2.htm#FSub)+2 |  |
| If x is +ve then skip ahead having set the carry flag to indicate we do not need to negate |
| 0CB9 | EF |  | RST [FTestSign](http://altairbasic.org/int_dis_1.htm#FTestSign) |  |
| 0CBA | 37 |  | STC | Set carry (ie no later negate) |
| 0CBB | F2C30C |  | JP [NegateIfPositive](http://altairbasic.org/math_dis_13.htm#NegateIfPositive) |  |
| x is between -0.75 and 0. Here we add 0.5 to get x between -0.25 and 0.25 and signal that negation is required |
| 0CBE | CD0108 |  | CALL [FAddOneHalf](http://altairbasic.org/math_dis_1.htm#FAddOneHalf) |  |
| 0CC1 | EF |  | RST 5 |  |
| 0CC2 | B7 |  | ORA A | Resets carry (ie later negate) |
| Preserve carry flag and negate x if it's +ve. |
| 0CC3 | F5 | NegateIfPositive | PUSH PSW |  |
| 0CC4 | F4FA09 |  | CP [FNegate](http://altairbasic.org/math_dis_5.htm#FNegate) |  |
| Adding 0.25 |
| 0CC7 | 01007F |  | LXI B,7F00 | BCDE=0.25 |
| 0CCA | 51 |  | MOV D,C |  |
| 0CCB | 59 |  | MOV E,C |  |
| 0CCC | CD1208 |  | CALL [FAdd](http://altairbasic.org/math_dis_2.htm#FAdd)+2 |  |
| Final negate (depends on above). |
| 0CCF | F1 |  | POP PSW |  |
| 0CD0 | D4FA09 |  | CNC [FNegate](http://altairbasic.org/math_dis_5.htm#FNegate) |  |

***Progression Calculation***

<fixme>

|  |
| --- |
| Push x. |
| 0CD3 | CD020A |  | CALL [FPush](http://altairbasic.org/math_dis_6.htm#FPush) |  |
| Push x^2 |
| 0CD6 | CD1D0A |  | CALL FAccToBCDE |  |
| 0CD9 | CDE508 |  | CALL [FMul](http://altairbasic.org/math_dis_4.htm#FMul)+2 | x = x\*x |
| 0CDC | CD020A |  | CALL [FPush](http://altairbasic.org/math_dis_6.htm#FPush) | Push x\*x |
| Let q be the first term of the Taylor series. |
| 0CDF | 21030D |  | LXI H,[TAYLOR\_SERIES](http://altairbasic.org/math_dis_13.htm#TAYLOR_SERIES) |  |
| 0CE2 | CD0F0A |  | CALL [FLoadFromMem](http://altairbasic.org/math_dis_6.htm#FLoadFromMem) |  |
| Restore x^2 |
| 0CE5 | C1 |  | POP B |  |
| 0CE6 | D1 |  | POP D |  |
| 0CE7 | 3E04 |  | MVI A,04 |  |
| 0CE9 | F5 | TaylorLoop | PUSH PSW | Push #terms remaining |
| Push x^2 |
| 0CEA | D5 |  | PUSH D | Push BCDE |
| 0CEB | C5 |  | PUSH B |  |
| q = (q \* x^2) + next term |
| 0CEC | E5 |  | PUSH H |  |
| 0CED | CDE508 |  | CALL [FMul](http://altairbasic.org/math_dis_4.htm#FMul)+2 |  |
| 0CF0 | E1 |  | POP H |  |
| 0CF1 | CD200A |  | CALL FLoadBCDE |  |
| 0CF4 | E5 |  | PUSH H |  |
| 0CF5 | CD1208 |  | CALL [FAdd](http://altairbasic.org/math_dis_2.htm#FAdd)+2 |  |
| 0CF8 | E1 |  | POP H |  |
| Restore x^2 to BCDE. |
| 0CF9 | C1 |  | POP B |  |
| 0CFA | D1 |  | POP D |  |
| 0CFB | F1 |  | POP PSW | Pop #terms remaining into A. |
| 0CFC | 3D |  | DCR A | Decrement #terms and loop back if not |
| 0CFD | C2E90C |  | JNZ [TaylorLoop](http://altairbasic.org/math_dis_13.htm#TaylorLoop) | done all 4 of them. |
| Finally multiply q by x. |
| 0D00 | C3E308 |  | JMP [FMul](http://altairbasic.org/math_dis_4.htm#FMul) |  |

The modified Taylor series used by SIN.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0D03 | BAD71E86 | TAYLOR\_SERIES | DD 39.710670 |  |
| 0D07 | 64269987 |  | DD -76.574982 |  |
| 0D0B | 58342387 |  | DD 81.602234 |  |
| 0D0F | E05DA586 |  | DD -41.341675 |  |
| 0D13 | DA0F4983 |  | DD 6.283185 |  |

## 3.1 Initialisation Part 1

The initalisation code runs before BASIC really begins, and once BASIC has begun the memory that this init code occupies is reclaimed for program storage. It's first job is to do detect what ports the teletype and terminal are hooked in to. Quite how and why this works, I have absolutely no idea. Has anyone got an Altair manual?? After that, code resumes in the second part of initialisation (see next page) where the user is given a chance to specify some configuration options.

|  |
| --- |
| Set stack pointer and STACK\_TOP to some bytes on from the top of this Initiliasation section. |
| 0D21 | 211A0F | Init | LXI H,0F1A |  |
| 0D24 | F9 |  | SPHL |  |
| 0D25 | 226301 |  | SHLD [STACK\_TOP](http://altairbasic.org/int_dis_3.htm#STACK_TOP) |  |
| 0D28 | DB01 |  | IN 01 |  |
| 0D2A | 0EFF |  | MVI C,FF |  |
| Set return address to ConfigIOcode, which is in the second part of Initialisation. |
| 0D2C | 118E0D |  | LXI D,[ConfigIOcode](http://altairbasic.org/init_dis_2.htm#ConfigIOcode) |  |
| 0D2F | D5 |  | PUSH D |  |
| 0D30 | 3AFF0F |  | LDA 0FFF |  |
| 0D33 | 47 |  | MOV B,A |  |
| 0D34 | DBFF |  | IN FF |  |
| 0D36 | 1F |  | RAR |  |
| 0D37 | DA410D |  | JC 0D41 |  |
| 0D3A | E60C |  | ANI 0C |  |
| 0D3C | CA420D |  | JZ 0D42 |  |
| 0D3F | 0610 |  | MVI B,10 |  |
| 0D41 | 78 |  | MOV A,B |  |
| 0D42 | 328C0D |  | STA 0D8C |  |
| 0D45 | DBFF |  | IN FF |  |
| 0D47 | 17 |  | RAL |  |
| 0D48 | 17 |  | RAL |  |
| 0D49 | 0620 |  | MVI B,20 |  |
| 0D4B | 1102CA |  | LXI D,CA02 |  |
| 0D4E | D8 |  | RC |  |
| 0D4F | 17 |  | RAL |  |
| 0D50 | 43 |  | MOV B,E |  |
| 0D51 | 1D |  | DCR E |  |
| 0D52 | D8 |  | RC |  |
| 0D53 | 17 |  | RAL |  |
| 0D54 | DA6F0D |  | JC 0D6F |  |
| 0D57 | 43 |  | MOV B,E |  |
| 0D58 | 1180C2 |  | LXI D,C280 |  |
| 0D5B | 17 |  | RAL |  |
| 0D5C | D0 |  | RNC |  |
| 0D5D | 17 |  | RAL |  |
| 0D5E | 3E03 |  | MVI A,03 |  |
| 0D60 | CD8B0D |  | CALL 0D8B |  |
| 0D63 | 3D |  | DCR A |  |
| 0D64 | 8F |  | ADC A |  |
| 0D65 | 87 |  | ADD A |  |
| 0D66 | 87 |  | ADD A |  |
| 0D67 | 3C |  | INR A |  |
| 0D68 | CD8B0D |  | CALL 0D8B |  |
| 0D6B | 37 |  | STC |  |
| 0D6C | C34B0D |  | JMP 0D4B |  |
| 0D6F | AF |  | XRA A |  |
| 0D70 | CD8B0D |  | CALL 0D8B |  |
| 0D73 | CD870D |  | CALL 0D87 |  |
| 0D76 | CD870D |  | CALL 0D87 |  |
| 0D79 | 4B |  | MOV C,E |  |
| 0D7A | 2F |  | CMA |  |
| 0D7B | CD870D |  | CALL 0D87 |  |
| 0D7E | 3E04 |  | MVI A,04 |  |
| 0D80 | 35 |  | DCR M |  |
| 0D81 | CD8B0D |  | CALL 0D8B |  |
| 0D84 | 35 |  | DCR M |  |
| 0D85 | 35 |  | DCR M |  |
| 0D86 | 35 |  | DCR M |  |
| 0D87 | 218C0D |  | LXI H,0D8C |  |
| 0D8A | 34 |  | INR M |  |
| 0D8B | D300 |  | OUT 00 |  |
| 0D8D | C9 |  | RET |  |

## 3.2 Initialisation Part 2

The results of IO diagnostics are in various registers. In this first block we configure the IO code with these results.

|  |
| --- |
| Configure InputChar's device-ready bit and conditional jump. |
| 0D8E | 62 | ConfigIOcode | MOV H,D |  |
| 0D8F | 68 |  | MOV L,B |  |
| 0D90 | 228503 |  | SHLD [InputChar](http://altairbasic.org/int_dis_6.htm#InputChar)+3 |  |
| Configure TestBreakKey's device-ready bit. |
| 0D93 | 7C |  | MOV A,H |  |
| 0D94 | E6C8 |  | ANI C8 |  |
| 0D96 | 67 |  | MOV H,A |  |
| 0D97 | 227604 |  | SHLD 0476 |  |
| 0D9A | EB |  | XCHG |  |
| 0D9B | 227A03 |  | SHLD 037A |  |
| 0D9E | 3A8C0D |  | LDA 0D8C |  |
| 0DA1 | 328303 |  | STA 0383 |  |
| 0DA4 | 327404 |  | STA 0474 |  |
| 0DA7 | 3C |  | INR A |  |
| 0DA8 | 328A03 |  | STA 038A |  |
| 0DAB | 81 |  | ADD C |  |
| 0DAC | 327803 |  | STA 0378 |  |
| 0DAF | 3C |  | INR A |  |
| 0DB0 | 328003 |  | STA 0380 |  |

***"Memory Size?"***

In this block we ask the user how much memory they have, in bytes.

|  |
| --- |
| Set CURRENT\_LINE to -1 to indicate direct mode. |
| 0DB3 | 21FFFF |  | LXI H,FFFF |  |
| 0DB6 | 226101 |  | SHLD [CURRENT\_LINE](http://altairbasic.org/int_dis_3.htm#CURRENT_LINE) |  |
| Print a new line, followed by the "MEMORY SIZE?" prompt. |
| 0DB9 | CD8A05 |  | CALL [NewLine](http://altairbasic.org/int_dis_14.htm#NewLine) |  |
| 0DBC | 21F00E |  | LXI H,[szMemorySize](http://altairbasic.org/init_dis_2.htm#szMemorySize) |  |
| 0DBF | CDA305 |  | CALL [PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |
| 0DC2 | CDC202 |  | CALL [InputLineWith'?'](http://altairbasic.org/int_dis_5.htm#InputLineWithxQx) |  |
| 0DC5 | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0DC6 | B7 |  | ORA A |  |
| 0DC7 | C2DE0D |  | JNZ 0DDE |  |
| No answer given to the request for memory size, therefore we find the top of memory ourselves. This is done by writing alternating 0x37 and 0x36's to progressively higher addresses and reading the values back from memory. When the value read is not the value written, we know we have written past the top of memory. |
| 0DCA | 21FC0E |  | LXI H,UnusedMemory |  |
| 0DCD | 23 | FindMemTopLoop | INX H |  |
| 0DCE | 3E37 |  | MVI A,37 |  |
| 0DD0 | 77 |  | MOV M,A |  |
| 0DD1 | BE |  | CMP M |  |
| 0DD2 | C2EA0D |  | JNZ [DoneMemSize](http://altairbasic.org/init_dis_2.htm#DoneMemSize) |  |
| 0DD5 | 3D |  | DCR A |  |
| 0DD6 | 77 |  | MOV M,A |  |
| 0DD7 | BE |  | CMP M |  |
| 0DD8 | CACD0D |  | JZ [FindMemTopLoop](http://altairbasic.org/init_dis_2.htm#FindMemTopLoop) |  |
| 0DDB | C3EA0D |  | JMP [DoneMemSize](http://altairbasic.org/init_dis_2.htm#DoneMemSize) |  |
| Memory size has been given in bytes. Here we convert the string input to an integer value and error out if it's 0 or a non-numeric input. |
| 0DDE | 211301 |  | LXI H,[LINE\_BUFFER](http://altairbasic.org/int_dis_3.htm#LINE_BUFFER) |  |
| 0DE1 | CD9D04 |  | CALL [LineNumberFromStr](http://altairbasic.org/int_dis_11.htm#LineNumberFromStr) |  |
| 0DE4 | B7 |  | ORA A |  |
| 0DE5 | C2D001 |  | JNZ [SyntaxError](http://altairbasic.org/int_dis_4.htm#SyntaxError) |  |
| 0DE8 | EB |  | XCHG |  |
| 0DE9 | 2B |  | DCX H |  |
| Put the address of the last word of RAM on the stack. |
| 0DEA | 2B | DoneMemSize | DCX H |  |
| 0DEB | E5 |  | PUSH H |  |

***"Terminal Width?"***

Ask the user how many columns wide their terminal is. This defaults to 72 if empty input is given.

|  |
| --- |
| Print the "TERMINAL WIDTH" prompt and get user input. |
| 0DEC | 21B40E | GetTerminalWidth | LXI H,[szTerminalWidth](http://altairbasic.org/init_dis_2.htm#szTerminalWidth) |  |
| 0DEF | CDA305 |  | CALL [PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |
| 0DF2 | CDC202 |  | CALL [InputLineWith'?'](http://altairbasic.org/int_dis_5.htm#InputLineWithxQx) |  |
| If no input given then we'll use the hard-coded default (72) and can jump straight to the next section. |
| 0DF5 | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0DF6 | B7 |  | ORA A |  |
| 0DF7 | CA1B0E |  | JZ [DoOptionalFns](http://altairbasic.org/init_dis_2.htm#DoOptionalFns) |  |
| User has given an input for terminal wdith, so convert that to an integer in DE. |
| 0DFA | 211301 |  | LXI H,[LINE\_BUFFER](http://altairbasic.org/int_dis_3.htm#LINE_BUFFER) |  |
| 0DFD | CD9D04 |  | CALL [LineNumberFromStr](http://altairbasic.org/int_dis_11.htm#LineNumberFromStr) |  |
| If user-supplied terminal width is >=256 or <16 then that's out of range so jump back to ask again. |
| 0E00 | 7A |  | MOV A,D |  |
| 0E01 | B7 |  | ORA A |  |
| 0E02 | C2EC0D |  | JNZ [GetTerminalWidth](http://altairbasic.org/init_dis_2.htm#GetTerminalWidth) |  |
| 0E05 | 7B |  | MOV A,E |  |
| 0E06 | FE10 |  | CPI 10 |  |
| 0E08 | DAEC0D |  | JC [GetTerminalWidth](http://altairbasic.org/init_dis_2.htm#GetTerminalWidth) |  |
| Config printing code with the user-supplied terminal width. |
| 0E0B | 326F03 |  | STA [OutChar\_tail](http://altairbasic.org/int_dis_6.htm#OutChar_tail)+1 |  |
| Calculate the column of the last tab-break and write this number to the right place in the ToNextTabBreak function. The tab-break size is 14, so the last tab-break is calculated as (width - ((width % 14)+14). So for 72, the last tab brk is at column 56. |
| 0E0E | D60E | CalcTabBrkSize | SUI 0E |  |
| 0E10 | D20E0E |  | JNC [CalcTabBrkSize](http://altairbasic.org/init_dis_2.htm#CalcTabBrkSize) |  |
| 0E13 | C61C |  | ADI 1C |  |
| 0E15 | 2F |  | CMA |  |
| 0E16 | 3C |  | INR A |  |
| 0E17 | 83 |  | ADD E |  |
| 0E18 | 32B705 |  | STA [ToNextTabBreak](http://altairbasic.org/int_dis_14.htm#ToNextTabBreak)+4 |  |

***"Sin? Rnd? Sqr?"***

Now we ask the user whether which optional inline functions they want support for. If they answer Y(es) to any, they do not get an option to turn off support for the later ones, ie if you say yes to SIN you have implicitly accepted RND and SQR too. If functions are turned off, the memory they occupy is reclaimed for program space (look for where PROGRAM\_BASE gets set and how it is calculated).

|  |
| --- |
| Initialise HL to point to the first optional function descriptor (which is for SIN). |
| 0E1B | 21850E | DoOptionalFns | LXI H,[OPT\_FN\_DESCS](http://altairbasic.org/init_dis_2.htm#OPT_FN_DESCS) |  |
| Push the first word of the descriptor onto the stack. This is where program storage can begin should the function be accepted. |
| 0E1E | F7 | OptionalFnsLoop | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) |  |
| Have we gone past the end of the descriptor table? If so, jump down a bit and on to InitProgramBase. |
| 0E1F | 11990E |  | LXI D,[szWantSin](http://altairbasic.org/init_dis_2.htm#szWantSin) |  |
| 0E22 | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) |  |
| 0E23 | CA320E |  | JZ 0E32 |  |
| Get the address of the string prompt into HL, print the prompt, get the input, get the first character of that input into A and restore HL. |
| 0E26 | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) |  |
| 0E27 | E3 |  | XTHL |  |
| 0E28 | CDA305 |  | CALL [PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |
| 0E2B | CDC202 |  | CALL [InputLineWith'?'](http://altairbasic.org/int_dis_5.htm#InputLineWithxQx) |  |
| 0E2E | D7 |  | RST [NextChar](http://altairbasic.org/int_dis_1.htm#NextChar) |  |
| 0E2F | E1 |  | POP H |  |
| If 'Y'es selected, ie keep this function, then pop the beginning of program storage into DE and jump down to InitProgramBase |
| 0E30 | FE59 |  | CPI 'Y' |  |
| 0E32 | D1 |  | POP D |  |
| 0E33 | CA470E |  | JZ [InitProgramBase](http://altairbasic.org/init_dis_2.htm#InitProgramBase) |  |
| If user has entered something other than 'N'o then jump back to the start of the optional functions section. |
| 0E36 | FE4E |  | CPI 'N' |  |
| 0E38 | C21B0E |  | JNZ [DoOptionalFns](http://altairbasic.org/init_dis_2.htm#DoOptionalFns) |  |
| User has selected No. Here we get the next word of the descriptor into HL, which is the optional function's entry in the KW\_INLINE\_FNS table. |
| 0E3B | F7 |  | RST [PushNextWord](http://altairbasic.org/int_dis_1.htm#PushNextWord) |  |
| 0E3C | E3 |  | XTHL |  |
| Write the address of FunctionCallError into the functions' entry in KW\_INLINE\_FNS. |
| 0E3D | 119804 |  | LXI D,[FunctionCallError](http://altairbasic.org/int_dis_10.htm#FunctionCallError) |  |
| 0E40 | 73 |  | MOV M,E |  |
| 0E41 | 23 |  | INX H |  |
| 0E42 | 72 |  | MOV M,D |  |
| Restore HL and jump back to deal with the next optional function. |
| 0E43 | E1 |  | POP H |  |
| 0E44 | C31E0E |  | JMP [OptionalFnsLoop](http://altairbasic.org/init_dis_2.htm#OptionalFnsLoop) |  |
| Got the bottom of program memory in DE. Here we write a null byte to that address and store the address in PROGRAM\_BASE. |
| 0E47 | EB | InitProgramBase | XCHG |  |
| 0E48 | 3600 |  | MVI M,00 |  |
| 0E4A | 23 |  | INX H |  |
| 0E4B | 226501 |  | SHLD [PROGRAM\_BASE](http://altairbasic.org/int_dis_3.htm#PROGRAM_BASE) |  |
| Get the address of the last byte of RAM into HL and push PROGRAM\_BASE on the stack. |
| 0E4E | E3 |  | XTHL |  |
| If the address of the last byte of RAM is less than 0x0F1A then Out of Memory (OM) error. |
| 0E4F | 111A0F |  | LXI D,0F1A |  |
| 0E52 | E7 |  | RST [CompareHLDE](http://altairbasic.org/int_dis_1.htm#CompareHLDE) |  |
| 0E53 | DACD01 |  | JC ErrorOutOfMem |  |
| Get PROGRAM\_BASE into DE and set the stack pointer and STACK\_TOP to the very top of memory. |
| 0E56 | D1 |  | POP D |  |
| 0E57 | F9 |  | SPHL |  |
| 0E58 | 226301 |  | SHLD [STACK\_TOP](http://altairbasic.org/int_dis_3.htm#STACK_TOP) |  |
| Get PROGRAM\_BASE into HL and STACK\_TOP into DE and check that PROGRAM\_BASE is far enough from the stack pointer (32 bytes away). |
| 0E5B | EB |  | XCHG |  |
| 0E5C | CDC301 |  | CALL [CheckEnoughMem](http://altairbasic.org/int_dis_4.htm#CheckEnoughMem) |  |
| Bytes Free (in HL) is calculated here as (STACK\_TOP - PROGRAM\_BASE) - 16. |
| 0E5F | 7B |  | MOV A,E |  |
| 0E60 | 95 |  | SUB L |  |
| 0E61 | 6F |  | MOV L,A |  |
| 0E62 | 7A |  | MOV A,D |  |
| 0E63 | 9C |  | SBB H |  |
| 0E64 | 67 |  | MOV H,A |  |
| 0E65 | 01F0FF |  | LXI B,FFF0 |  |
| 0E68 | 09 |  | DAD B |  |
| Print version information - a new line, followed by the number of Bytes Free, followed by the version info string. |
| 0E69 | CD8A05 |  | CALL [NewLine](http://altairbasic.org/int_dis_14.htm#NewLine) |  |
| 0E6C | CD370B |  | CALL PrintNumber |  |
| 0E6F | 21C30E |  | LXI H,[szVersionInfo](http://altairbasic.org/init_dis_2.htm#szVersionInfo) |  |
| 0E72 | CDA305 |  | CALL [PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |
| Change code at start of Main to call PrintString instead of restarting. |
| 0E75 | 21A305 |  | LXI H,[PrintString](http://altairbasic.org/int_dis_14.htm#PrintString) |  |
| 0E78 | 22FD01 |  | SHLD [Main](http://altairbasic.org/int_dis_5.htm#Main)+4 |  |
| Call NEW handler to initalise BASIC interpreter variables, setup a blank program, etc. |
| 0E7B | CD9602 |  | CALL [New](http://altairbasic.org/int_dis_5.htm#New)+1 |  |
| Set the JMP address at the start of BASIC to jump to Main instead of Init. |
| 0E7E | 21F901 |  | LXI H,[Main](http://altairbasic.org/int_dis_5.htm#Main) |  |
| 0E81 | 220200 |  | SHLD [Start](http://altairbasic.org/int_dis_1.htm#Start)+2 |  |
| Jump to Main |
| 0E84 | E9 |  | PCHL |  |

### Optional Function Descriptors

blah

|  |
| --- |
| Optional function descriptors. The first one is for the SIN function. |
| 0E85 | 170D | OPT\_FN\_DESCS | DW 0D17 |  |
|  | 990E |  | DW [szWantSin](http://altairbasic.org/init_dis_2.htm#szWantSin) |  |
|  | 4900 |  | DW [KW\_INLINE\_FNS](http://altairbasic.org/int_dis_2.htm#KW_INLINE_FNS)+12 |  |
| RND function. |
| 0E8B | 950C |  | DW 0C95 |  |
|  | A20E |  | DW [szWantRnd](http://altairbasic.org/init_dis_2.htm#szWantRnd) |  |
|  | 4700 |  | DW [KW\_INLINE\_FNS](http://altairbasic.org/int_dis_2.htm#KW_INLINE_FNS)+10 |  |
| SQR function. |
| 0E91 | 5F0C |  | DW 0C5F |  |
|  | AB0E |  | DW [szWantSqr](http://altairbasic.org/init_dis_2.htm#szWantSqr) |  |
|  | 4500 |  | DW [KW\_INLINE\_FNS](http://altairbasic.org/int_dis_2.htm#KW_INLINE_FNS)+8 |  |
| Finally, the lowest address for program space if all optional functions are switched off |
| 0E97 | 210C |  | DW 0C21 |  |

### Init Strings

|  |
| --- |
| String prompts for the optional inline functions. |
| 0E99 | 57414E54205349CE00 | szWantSin | DS "WANT SIN\0" |
| 0EA2 | 57414E5420524EC400 | szWantRnd | DS "WANT RND\0" |
| 0EAB | 57414E54205351D200 | szWantSqr | DS "WANT SQR\0" |
| Other string constants |
| 0EB4 | 5445524D494E414C2057494454C800 | szTerminalWidth | DS "TERMINAL WIDTH\0" |  |
| 0EC3 | 20425954455320465245C50D0D 42415349432056455253494F4E20332E B20D5B344B2056455253494F4EDD0D00 | szVersionInfo | DS " BYTES FREE\r\r" "BASIC VERSION 3." "2\r[4K VERSION]\r\0" |  |
| 0EF0 | 4D454D4F52592053495AC500 | szMemorySize | DS "MEMORY SIZE\0" |  |

EFC to FFF are unused.