Addendum to
ATE User's Manual

Disk/ATE

A Disk Operating System, System Monitor, Assembler, and Text Editor

Table of Contents

Thinker Toys™

DISK/ATE

DISK COMMANDS

DISK COMMAND SYNTAX

File Names

May be up to eight characters long, and may use any printing characters except comma, colon, leading double-quote, or leading @.

The file will be assumed to be on the current drive (see the CD command) unless another drive is indicated by placing a colon immediately after the file name, followed by an upper case letter A to H. In standard Disk/ATE, drives A to D refer to DISCUS drives one to four, drives E to G refer to North Star drives one to three, and drive H is undefined (and will give an error when invoked). To recap,

Drives A to D	=	DISCUS drives 1 to 4
Drives E to G	=	North Star drives 1 to 3
Drive H	=	undefined.

These references can be changed quite easily (see below). Drive A is current at power-up unless you change that (see below).

Whenever a file name can appear, a double quote " or ditto can be used instead to refer to the previously-referenced file. The ditto refers to a name-drive combination; the drive cannot be re-specified with a colon. See the Save and ? commands for more information.

Memory Address Specification

Standard ATE argument format is used here. See below for examples. Whenever a disk command wants a memory address or interval, one must be given. To protect against errors, the missing-argument convention does not apply to disk commands.

Multiple Arguments

Must be separated by blanks, not commas, as with all ATE commands. (Commas separate commands.)

DISK COMMANDS

<u>S</u> <u>filename</u> <u>interval</u>   "Save"

Save a disk file with the given name, containing the given interval of memory. The file will be placed on the current drive unless another drive is specified (see 'filenames' above). If the named file already existed, then its old contents are lost. If it did not already exist, it is created. If there is not a large enough space for it on the disk, but if there are enough free blocks scattered around the disk, then the used file space will be compacted to make room.

Type assignment is automatic (Ø = source file from the source area, 1 = other file), as is "memory address" (called "go address" by North Star) assignment. See also the W command.

Error messages: DISK TOO FULL ON DRIVE __ if there are not enough free blocks, and DIRECTORY FULL ON DRIVE __ if you are trying to create a new file and there are already 64 directory entries.

Examples: S DOS:E 2ØØØH..29FFH saves North Star DOS on drive E. S SYMTAB <T> saves the current symbol table in a file called SYMTAB on the current drive. S". saves the current memory file in the most recently referenced disk file. (Caution - see the REF command.)

<u>L</u> <u>filename</u> <u>optional address</u>   "Load"

Loads the given file. If an address is given, the file is loaded there with no further ado. If no address is given, then the type is consulted. If type = 1, it is loaded at the recorded memory address. If type = Ø ("source"), then it is inserted in the source area at the position of the entry pointer. In this latter case, it is stripped of any leading or trailing zeros (it is not necessary to save these delimiters in the first place) and made part of whatever file it was placed in (which need not be the current file). Thus, files can be concatenated or merged. If you want the disk file to occupy a separate memory file, precede the L with an N or an O (q.v.). Caution: make sure ↑ is within <S>.

Errors: CAN'T FIND _____ ON DRIVE __.

Example: L ATETBL <T> will overlay the current symbol table with the table of internal Disk/ATE references supplied on the disk.

GO filename

For type Ø files, this is equivalent to L filename, D<R>.
For type ≠ Ø files, this is equivalent to L filename, X<R>.
(Note that <R> gives the beginning and ending addresses of the most recently loaded record.) It is usually a good idea to do an O or N before GOing an ATE command file. Note that a new load address cannot be specifield with a GO.

I optional drive letter A to H "Identify"

Lists the directory on the spcified drive, or on the current drive if none is specified.

FS optional drive letter " Free Space"

Gives the number of free blocks on the disk, and the number of free directory entries.

U filename "Unsave"

Deletes the named file from the directory.

T oldfile newfile "Transfer"

Transfers the contents (and directory attribues) from oldfile to newfile, on the same or on different drives. This is similar to Save, except that the information to be saved comes from another disk file instead of from memory.

TD olddrive newdrive "Transfer disk"

This is a selective "disk copy" function. It does the equivalent of T file:olddrive file:newdrive for each file on the "old drive". So, files that are unique to the old drive are created anew on the new drive. Files that are common to both are updated from the old drive. Files that are unique to the new drive are left untouched. If the disk on the new drive is initially blank, then this is similar to a "copy disk" function.

Errors: If the disk on the new drive runs out of room, then the process terminates with the appropriate message (either DIRECTORY FULL... or DISK FULL...). This is not really an error -- the system has transferred as much information as possible given the size of the disk on the new drive. There is nothing wrong with doing a TD command from a full sized floppy to a mini-floppy, for example. In no case is a directory entry created unless there is room to save the entire file.

<u>RN</u> <u>oldname</u> <u>newname</u> "Rename"

Renames the specified file. Any drive specification on rename is ignored.

<u>W</u> <u>filename</u> <u>address</u> "Write address"

Change the memory address (to which the file may be written at load time) of the specified file to the given value.

<u>CD</u> <u>new drive letter A to H</u> "Current Drive" or "Change Directory"

Changes the default drive to the one given. At power-up, the default drive is A.

<u>?"</u> "What is ditto?"

Prints the value of ", the filename and drive number remembered from the last command.

DISK FILE STRUCTURE

Disk/ATE's disk file and directory structure are logically compatible with North Star's. Disk/ATE assumes that a disk consists of 256-byte blocks numbered from Ø up to some maximum value. (The physical configuration of the disk may be quite different. See below.) The disk directory occupies blocks 0 to 3 and consists of 64 sixteen-byte entries. Each entry is structured as follows:

Directory entry bytes

0-7 file name. Any ASCII characters > blank except comma or colon. The first character cannot be a double-quote or an @. The entry is considered empty if the first character is blank.

8-9 disk address (low byte, high byte). Block number where file begins.

10-11 length (low, high). Number of consecutive blocks in file. Disk/ATE 1.0 ignores the high byte, limiting files to 64K.

12 type. Only two types are significant to Disk/ATE: Ø and non-zero. For loading, type Ø files are considered "source" files and are loaded into the source area unless otherwise directed. Type non-zero files are loaded at the recorded memory address unless otherwise directed. New files are given types Ø and 1.

13-14 memory address (low, high). The address from which the file was originally saved, unless changed by a W command.

15 Unused. Set to zero for new files, preserved in old files. The I command lists the contents of a disk directory. If you want more information than this provides, you can do the following:

Load the ATE symbol table (L ATETBL <T>) and set X = DISKBUF. (See "Internal Disk/ATE references" below.) Then do an I command. This leaves a copy of the directory at DISKBUF. (Disk/ATE does not use this copy for subsequent disk references.) Now type: R64,"X..X+7,#X+8..X+15,X=X+16

This will show you every byte in the directory.

CHANGES IN THE EDITOR

Argument evaluation remains unchanged except for the following symbols, and for the effects of the REF command.

[ ] Square brackets replace parentheses as the ASCII string delimiters. On most terminals these can be typed without the shift key (as opposed to parentheses). Also, square brackets occur less frequently in text than parentheses, and this lets you edit a single parenthesis without unbalancing your delimiters.

% replaces * as the line-expander, and has the same meaning. 2 string % returns the line containing the 2nd occurrance of string.

| replaces / as the search-restrictor, and has the same meaning. pattern|pattern will search first for pattern 1, and then search for pattern 2 within pattern 1. The reason for these last two changes is to free * and / for their traditional meanings.

* and / now mean multiplication and division. As with + and -, all arithmetic is done mod 2^{16} (so -1 = ØFFFFH). * and / have the same priority as + and -, and evaluation proceeds from left-to-right. (Unary minus has highest priority, however.) So, for example: 5-2*4 gives 12, and 1-2*-1 gives 1. Parentheses are not understood.

! is a new operation. 123! returns the 123rd line of the current file. This is similar to 123←%, but it is much faster (since the latter invokes the general pattern-search mechanism to find the 123rd carriage return). Also, 123! is absolute, in that M123!..139! will move lines 123 thru 139, while to do the same thing with pattern-searching requires M123←%..139-123← . Also, 123! gives the 123rd line no matter what the current reference string is, i.e., ! is not limited to searching within the reference string, as 123← is. (See the REF command.)

X! will give the Xth line, as will X←%. But ! is not as general as %. For example, ↑% gives the line containing the entry pointer, while ↑! as an argument would give nonsense (it would give the (address of ↑)th line). So in general, use ! with absolute line numbers, and use % otherwise.

Finally, if the file only has 123 lines, then using any larger number with ! will give the zero-byte at the end of the file. Also, ! does not search backwards. So -1! will be the end of the file (while -1←% is the last line, -2←% is the next to the last line, etc.). The command ↑-1! will set the entry pointer to the end of the file, in a position to append text to the file.

! alone or Ø! is equivalent to 1!.

? as an argument still has its old meaning, with the following addition:

if ATE is doing output from a " or P command, and you abort the output with an Esc, a ? will be left on the screen at the spot where output was terminated. Now, using ? as an argument will give the address within the file where output was terminated. So "?.. will continue quoting the file from the point where you left off, or K?% will kill the line containing the ?, etc. Caution: if the ? is left in place of a line number (during a P command), then it will <u>not</u> address the current file. You can single-step the output (with the S key) to avoid this and put the ? wherever you want it.

T^ and S^ stand for the memory address one greater than the top of <T> and <S>. This is useful for memory space allocation in assembly language programs, for appending one table to another, etc.

Editor commands are unchanged except:

COM   no argument   "Command"

When used in an ATE edit macro, this will type a prompt and accept and execute an arbitrary command line before returning to its command string. (Warning: don't use this in a "command line" since the new commands will overwrite the old one in the command buffer.) Note that the new command line could be a multi-line enter, could contain a Do, etc. An error or an Esc will force a return to the terminal (or to the calling program if applicable.)

ATE   no argument

Same as COM, but will continue to accept command lines, in spite of errors and Esc's, until you force a return with the BYE command. (Mainly useful with RST Ø from outside ATE. See below.)

BYE   no argument

Forces ATE to return to its caller. If you say BYE to the initial power-up version of ATE, it re-loads from the disk. BYE is inefficient with COM, which return anyway.

Y   no argument   "wipe"

Executes the user-provided terminal-initialization routine. For many users, this will blank the screen. You might also want to use this after an IO command.

IO value

Changes the device number passed in reg A to the user's IO routines. ATE does not use this number -- it is for the convenience of the user in switching between devices. You might type IO1, A DISKFILE IOØ to route an assembly listing to your printer and then return output to the CRT. IO prints a carriage return (on the new device) when invoked.

RENT   no argument   "Re-enter"

Re-enters ATE at the "ground level" without re-initializing. You might use this in case you have instances of COM or ATE stacked up, or to re-enter after an error in a user command (see below).

PAUSE   no argument

Causes ATE to enter the "panic state", as if you typed a panic stop. Everything is frozen while you change disks, think, or whatever. Then typing Esc will return control to the terminal, 's' will single-step the output, and anything else will continue as if nothing had happened. "Pause" is printed at the terminal to signal this command.

REF [any ATE argument, enclosed by square brackets]
After this is typed, the given argument (the one inside the square brackets) replaces <F> as the initial reference interval for pattern searches.

For example, REF [↑%] will force all subsequent command arguments to do their pattern searches within the line then containing the entry pointer. As the entry pointer moves from line to line, the intial reference interval moves with it. (ATE in effect prefixes all subsequent command arguments with ↑%| , except that the previous-argument conventions still hold: a missing argument is equivalent to repeating the previous one, arguments beginning with | use the previous argument as a reference interval, and < and > refer to the left and right endpoints of the previous argument, until modified by the current argument.)

Warning: REF changes the meaning of dot. An easy and disasterous mistake would be to edit a file line-by-line using REF [↑%] , and then type S"., saving only one line instead of the entire file <F>. Remember that . (or ...) alone as an argument refers to the reference interval which may or may not be all of <F>.

To see the current reference string, type REF with no argument. To wipe out the reference string, type REF [ ]. This returns ... to meaning <F>.

Reference strings have no effect on F, D, or commands, which continue to use <S>. Reference strings are limited to twenty-four characters.

DEF [any command string, enclosed in square brackets] .
Sets a DEFault command string to be executed whenever an empty command line (carriage return only) is typed. Example: REF [↑%] , DEF [↑_+1,".] is a useful combination. It lets you edit a file line-by-line, stepping to the next line by typing 'return'.

Default strings are limited to twenty-four characters (but may contain Do commands). DEF, and DEF [ ] behave as in REF.

E "Enter"
This is the same old enter function, but remember that it now uses [ ] as ASCII delimiters. Also, it is now much faster -- it will keep up with the fastest typist even on large source files.

O "Origin"
This behaves as before, except that when used without an argument, it collapses the source area to an empty file.

V "Evaluate"
Simply evaluates the argument. This is mainly useful with machine language programs, since it returns the beginning and ending addresses of the argument in HL and DE. See "Machine Language Interface." Nothing is printed at the terminal unless an error occurs.

G "Goto"
Replaces > as the goto command.

NEW ASSEMBLER FEATURES

A   file1   file2   file3 ...   "Assemble"
will do both passes over the concatenation of the named disk files. Any number of file names, separated by blanks, can be given. In this case, the files are buffered -- they are not loaded. Object code is still placed in memory in the usual way, however.

Including @ as a file name will suspend the assembly while you changes disks (or whatever). "Pause" will be printed at the terminal, after which you are in a panic'stop state: any key will continue the assembly except Esc, which will abort it. (If the assembler is producing a listing, then 'S' will single step the output. If it is not producing a listing (see J and Q below). Then 'S' will assemble one line at a time without producing any visible indication of that fact.)

If no file names are given with the A command, then the current memory file is used.

A1 and A2 are the same as A, except that they do only pass 1 and pass 2, respectively. Both can be used with or without file names.

Q and J Q tells the assembler that from now on, it should suppress listings ("quite"). This can be used as a prefix to A (QA) or to A2 (QA2), or it can be separted from A as a distinct command. J tells the assembler that from now on, it should produce listings ("jabber") (sorry). It's use is identical to Q.

INTE can be used either as a command or a pseudo-op. It tells the assembler that from now on, the source code will be in the Intel assembly language format: initial blanks are unnecessary, labels are terminated by a colon, and comment lines begin with a semi-colon (although * will also work). This format is a bit more economical of memory, since usually fewer than half the lines have labels.

PROS is used like INTE, and tells the assembler to expect Processor Technology format. This is the power-up state, unless you change it. Note that the Pring command will not respond to PROS or INTE when these are used as pseudo-ops.

IF   expression, label   (note the comma)
This is an assembler pseudo-opcode that allows conditional assembly. If the expression evaluates to Ø, assembly will skip ahead to the statement labeled label. Assembly continues normally if the expression is non-zero.

TAB

With no arguments, this will print the current "tab" stops. At power-up, these are 8 15 2Ø 29 21. This means that a Print command will begin the labels in column 8, the opcodes in 15, the arguments in 2Ø and the comments in 29. The left-most column is column Ø. Finally, the 21 is the assembly source listing offset, the new "column zero" relative to which other tab stops are treated when listing an assembly language program to the right of its object code.

These tab stops can be changed by following the TAB command with up to five arguments (separated by blanks). These will change the appropriate stops.

WID

With no arguments, prints the current terminal width. With an argument, resets the width to the given value. If your terminal produces an automatic newline on line overflow, then you should set the width to one less than the actual value.

New operations

* and / mean multiplication and division, see "argument evaluation" above. Evaluation is from left to right; parentheses are not understood. (Since all arithmetic is mod 2^{16}, and 8080 memory addressing is also mod 2^{16}, there is no need to worry about overflow.)

New operands

T^ and S^ refer to top-of-symbol-table + 1 and top-of-source-area + 1. These can be used with AORG and SORG pseudo-ops to aid in memory space allocation. Caution: do not use SORG T^ if you also use $ as an assembly argument, since this will have a different value during pass2 than during pass1.

PERSONALIZING DISK/ATE

Power up, and then use any of the following commands to bring ATE to the desired state: IO, CD, B, O, M<T>, INTE, PROS, &, $, J, Q, TAB WID, REF, DEF. Finally, install a user command table, if desired (see below). Then type L ATETBL <T> followed by S ATE BEGIN..END

From this point on, whenever you power up ATE, it will be in the state created above. If you type BYE from the "ground level" of ATE, or if you type X3 from anywhere, ATE will be reloaded in this same configuration.

User Command Table

See the ATE User's Manual for the format of a command table.

There is a small amount of space within ATE itself for a user command table. This is located between USRCT and ROMEND. (L ATETBL <T> to get these addresses.) Remember that this table must end with a zero byte, ATE expects the address of the user command table to be stored at UCTAD. This is initially set to USRCT, and is brought into memory from the disk each time ATE is loaded. You can change UCTAD before resaving ATE as described above. A logical place for a large user command table would be in the IO file. Remember that ATE checks the user command table before its own command tables, so that your table must be in place before you UCTAD, or before you power up ATE with a changed UCTAD.

DISK/ATE ORGANIZATION

Disk/ATE is stored on the disk in two separate files, called ATE and IO.

ATE is the IO-independent portion of Disk/ATE. It may be located anywhere in memory, depending on what version you are using. The standard system disk provides two versions: one in low memory, and one at the top of 32K. At power-up, ATE writes the vector JMP ATECOMS at address Ø. This lets other programs execute any Disk/ATE command (see below).

IO contains the terminal and disk drivers, and a loader (using one of the disk drivers) for the file ATE. Again two versions are provided, but the source code is also provided so that IO can be changed and relocated anywhere. ATE's only link to IO is the vector JMP IO written at address 3 (by the loader; see below). ATE expects IO to begin with a jump table. (See the accompanying IO source listing for a complete example.) The jump table is configured as follows:

```
IO      JMP   LOADER
IO+3    JMP   character output routine
IO+6    JMP   character input routine
IO+9    JMP   terminal initialization routine
IO+12   JMP   panic detect routine
IO+15   MVI   C, some drive number
        JMP   driver for drive A
IO+20   MVI   C, ____
        JMP   driver for drive B
IO+25   MVI   C, _____
        JMP   driver for drive C
IO+30   MVI   C, _____
        JMP   driver for drive D
IO+35   MVI   C, ____
        JMP   driver for drive E
IO+40   MVI   C, _____
        JMP   driver for drive F
IO+45   MVI   C, _____
        JMP   driver for drive G
IO+50   MVI   C, _____
        JMP   driver for drive H
```

The loader writes the vector JMP IO at address 3, loads ATE from one of the disks, and jumps to the first byte of ATE. Since IO begins with JMO LOADER, the entire system can be bootstrapped by loading and executing IO. Thereafter, executing address 3 will restart ATE without altering IO. Thus programs that overwrite ATE can end with a jump to address 3. ATE itself never executes address 3, but it does pick up the address of the IO jump table from bytes 4 and 5.

The loader provided in the standard version of IO assumes only that ATE is the second entry in the directory on drive A. It reads this directory entry to find ATE's disk address and memory address. For detailed comments on the loader, see the IO source listing.

The terminal IO routine can use the corresponding North Star IO routines, i.e., the vectors in the IO jump table can be aimed directly at the corresponding vectors in North Star DOS. But the requirements that ATE makes of these routines are more relaxed.

In all cases, only SP need be preserved. Also in each case, a "device number" is passed to the routine through reg A. This is provided for the user's convenience; it has no significance to ATE. It is set to Ø at power-up and can be changed by the IO command.

The character output routine sends the byte in B to device A. (If you have only one device, you can of course ignore this device number.)

The character input routine should get a byte from device A and return it in A.

The terminal initialization routine should initialize device A. This is called at power-up, when ATE is re-initialized, and in response to a Y command.

The panic detect routine should determine whether or not a "panic stop" is being requested, and return with the Z-flag on if so. North Star's "control-C" detect will work, but it is more convenient to have any key force a panic stop. Again, a device number is provided in A. This is called during output, assemblies, and ATE programs.

There are eight (8) disk drive vectors corresponding to drives A to H. Each vector is preceded by a MVI C instruction that can be used to supply a drive number to the driver routine. This number has no significance to ATE itself. If once of the drives does not exist on your system, you should replace the two-byte MVI C instruction by STC, RET. On return from any disk call, carry = 1 always indicates a disk error to ATE.

The disk command is passed in register B:
B = 3 means that the drive and/or driver should be initialized. If this is not necessary, simply return. ATE issues this command when a drive is first accessed after power-up or after a disk error.

B = 2 requests that the disk size be returned to BC. This is the total number of 256-byte blocks[1] on the disk, including the directory blocks. (For the North Star, this is 350. For the DISCUS, this is 1000, leaving 1 block for an on-disk bootstrap inaccessible to ATE.) This command should not perform any disk access. The maximum disk size that ATE can accomodate is an assembly variable; it is given as MAXBS in ATETBL. Returning with a larger size will cause a 'disk error'.

B = 1 requests a read, B - Ø requests a write. In this case, A contains the number of blocks to transfer[1], DE contains the beginning memory address, and HL contains the beginning disk address (i.e., a block number from Ø to disk size -1). Again, a return with carry = 1 indicates an error.

[1]Although ATE assumes a logical disk configuration of contiguous 256-byte blocks, the driver routine can translate this into a much different physical configuration. For instance, the DISCUS is actually layed out in 128-byte sectors with a 5-sector skew.

MACHINE LANGUAGE INTERFACE TO ATE

Any machine language program can execute any string of ATE commands. Simply load HL with the address of the string (the string must be terminated by a zero byte) and do a RST Ø. Note that one of the commands could be 'ATE', which would continue to read and execute command lines until 'BYE' forced a return to the calling program. In this case, ATE would use the caller's stack. To be absolutely safe, 100 bytes should be sufficient.

Writing your own ATE commands

Any machine language program can be an ATE command. There are four main considerations here: how to invoke the program, how to interface the program to ATE (if necessary), how to pass arguments, and how to return to ATE.

How to invoke the command

Suppose you have written a program called MEMTST. The simplest way to invoke it would be XMEMTST (as long as MEMTST is the first byte of the program and is in the symbol table, either after assembling it or by typing MEMTST = _____). Or, MEMTST could be stored on the disk, and could be invoked by GO MEMTST.

Finally, you could enter MEMTST in the user command table and invoke it by name. See the ATE User's Manual for the command table format. See "Personalizing Disk/ATE" for locating the user command table.

How to return to ATE

Any command invoked as above can be part of any ATE command string as long as it returns normally (with a RET, RZ, etc.). But what should you do if your program detects an error and you want to tell ATE to discontinue whatever command string it is in? The grossest way is to jump to address 3: this will reload ATE (not IO) and start from scratch. A better way would be to point HL at a string containing the letters RENT followed by a zero-byte, ground level. The best way would be to jump to the internal ATE routine WHAT:

Interfacing programs to ATE

A symbol table containing internal references to ATE is provided on the system disk (see Internal ATE references below). The simplest way to use this is to load the table (L ATETBL <T>) before assembling your program. Then your program can reference the internal ATE routines listed in this table.

If you are concerned with writing a program that will work without reassembly with <u>any</u> version of ATE, then your program will have to link itself to ATE. This can be done through the JMP ATECOMS vector at address 0. For example, suppose that you want the address of WHAT. Your program could contain the following code.

```
                LXI     H,COM1
                RST     0
                LXI     H,COM2
                RST     Ø
                 '
                 '
                 '          At this point, HL will contain the
                            address of WHAT.
     COM1       ASC-    LATETBL- T
                DB      Ø
     COM2       ASC     VWHAT
                DB      Ø
```

Notes: The V command simply evaluates its argument and returns the beginning and ending addresses in HL and DE. After a RST Ø, carry = 1 indicates an error. There is no way to suppress printing of error messages (except by redirecting the IO with the IO command).

Argument Passing

Passing arguments from the command level down to machine language programs is easy. The process of using repeated calls to VCHK and CVALS is described in the ATE User's Manual. But vice-versa is a bit harder. Suppose that your program has calculated the beginning and ending adresses of a file that it would like to save on the disk. One tedious way would be to construct the ASCII digits and pass S FILENAME 1234...5678 through RST Ø. A better way would be to store the desired addresses at P1 and P2, and then use S FILENAME <.>

INTERNAL DISK/ATE REFERENCES

The symbol table ATETBL contains the standard 8080 register symbols, followed by these internal ATE references:

BEGIN	The first byte of ATE. Entering here will give a minor reinitialization. SP is initialized, a few internal variables are set, the command string O, Z, Y is executed, and the initialization message is printed.
RENT	The re-entry point to ATE that avoids reinitialization.
ATECOMS	Executes an arbitrary ATE command string beginning at the address in HL. See "Machine language interface" for details.
WHAT	This is the error exit for command routines. See "Machine language interface" for details.
READ	Reads a line from the terminal into a buffer of length B beginning at address HL. Uses the prompt in A, or no prompt if A = Ø.
VCHK	Returns with the Z flag off if there is an argument after your command (or after the previous argument).
CVALS	Evalutes an argument, returning with the beginning and ending values in HL and DE. In case of an error, this does not return -- it exits thru WHAT back to the routine that initiated the current command string, printing a ? at the terminal.
VALUS	Same as CVALS, but returns with the Z flag off in case of an error. Also, you must provide the initial search interval in HL...DE. (If no search is required, this can be ignored.)
OUT	Prints the character in A at the terminal. Preserves all flags and registers. Takes care of providing a line feed after a carriage return, and updates the internal print head counter, providing a cr-lf when this reaches the terminal width.
INECO	Gets a character from the terminal and echos it, doing the housekeeping mentioned above. Preserves all flags and registers except A, in which the character is returned.
PHLSB	Prints the value in HL in the current base, including any leading zeros, using four digits for base 16, or 3 digits per byte for bases less than 16 with a colon between the bytes.
PHLDC	Prints the value in HL in base 1Ø, with leading zeros suppressed.

USRCT ROMEND	Storage is provided from here to ROMEND for a user command table (although this is consulted only if UCTAD = USRCT).
UCTAD	Storage for the address of the current user command table.
ASPC	See ATE User's Manual.
STCTR	"
BOSAP	"
SYMTB	"
END	The end of the part of ATE that is stored on the disk.
EOSAP	See ATE User's Manual.
BOFP	"
EOFP	"
TABA	"
CHPTR	"
P1	"
P2	"
RECAD	"
RECND	"
ERSAV	"
PHD	"
MAXBS	The maximum disk size (in blocks) that this version of ATE can handle.
DISKBUF	ATE's disk buffer. After a disk command, a copy of the disk directory is left here (but is not subsequently used by ATE). The first 1024 bytes are used only during disk command, but the rest is shared by some other ATE buffers.
RAMEND	End of ATE's internal RAM. Memory above this point is free.