WHAT IS FAST AND HOW DOES IT WORK?

What is FAST? It is a new transient program for the CP/M (*) disk operating system. FAST is used in conjunction with other transients to speed up their execution. Of course, nothing but a hardware change can increase the rate at which your CPU executes instructions. But, most transients are "disk bound", that is, they spend a large percentage of their total execution time just waiting for the information they need from the disk. FAST uses this fact to increase execution speed by reducing the time spent waiting for the disks. This is accomplished by using two different types of buffering. Firstly, portions of the disk which are frequently accessed (the directory) are held in a buffer, eliminating the need to seek to read them. Secondly, disk accesses are grouped together in time to reduce the amount of time lost to rotational latency and head load delay.

HOW TO USE FAST

The normal CP/M command to assemble a file with the name MUMBLE.ASM is:

```
    A>ASM MUMBLE
```

To perform the same function under FAST, simply type:

```
    A>FAST ASM MUMBLE
```

Thus, in its simplest form, FAST can be used as a prefix to any normal CP/M command. This will load FAST which will link itself to your operating system and allocate its disk buffers. Then the transient ASM.COM is loaded and given control. The fact that FAST is in the system is totally transparent to the transient, except for the reduced memory size. Messages will be printed just before and after the transient is executed to indicate that FAST is in control. After execution, you should see something like this on your terminal:

```
    A>FAST ASM MUMBLE
    Beginning execution under FAST N.NN
    CP/M ASSEMBLER - VER 1.4
    1234
    056H USE FACTOR
    END OF ASSEMBLY
    Execution under FAST now complete
    A>
```

Where N.NN is the version number of FAST that you are using.

COMMAND LINE OPTIONS

The buffering action of FAST may be altered with the use of command line options. This is accomplished by following the FAST command with an option string enclosed in square brackets ("[" and "]"). A valid option string must consist of a drive specification followed by one or more buffering mode specifications. This group may be repeated for buffering of multiple drives. Examples of typical option strings will be given below.

Explicitly specifying the buffering mode is desirable to reach the best trade-off between speed of execution and memory usage. In general, as more buffering is used, execution speed will increase. However, more buffering means more memory usage. The type of buffering selected should be tailored to the disk usage patterns of the transient being executed. For instance, write buffering will offer no speed improvement when a transient only reads the disk.

One thing that FAST must know is which disk drive is to be buffered. This is communicated to FAST simply by using the single letter name of the desired drive (e.g. "A" for disk drive A). As a convenience, the current default drive may be specified with the commercial at sign ("@").

The second thing which FAST must know is which mode of buffering is to be used. FAST supports three different buffering modes. Each mode may be used individually or in combination with other modes to offer the fastest execution. The three modes available are:

Seek buffering
Read buffering
Write buffering
Yes (all of the above)

The type of buffering desired is communicated to FAST by a single letter in the option string. The letter is simply the first letter of the word which describes the type of buffering (capitalized above). The operation and application of each mode will be discussed below.

Thus, a typical FAST command line might look like this:

```
    B>FAST [ASWBSR] ASM MUMBLE.BAA
```

In this example, the .ASM source file is on drive B so read and seek buffering have been specified for that drive. The A drive, on the other hand, is to receive the result of the assembly (the .HEX and .PRN files), so it gets write and seek buffering.

BUFFERING MODE DESCRIPTIONS

Seek buffering causes the disk directory to be read into a buffer the first time a drive is accessed. From then on, all reads from and writes to the directory can be carried out without moving the disk head from its current position. Thus, transients which access the directory frequently will be sped up considerably by seek buffering. Transients which fall into this category are those which deal with many different files simultaneously, perform operations with temporary files and rename them, read or write large (more than 16K) files, and those which perform random disk I/O. In particular, ASM.COM, MAC.COM, PIP.COM, and ED.COM are examples of such transients.

Read buffering causes an entire track from the disk to be read into a buffer the first time any sector is read from that track. This increases execution speed because CP/M typically reads most of one track before going on to the next one. Additionally, the time required to read a whole track is a fairly small percentage increase over the time required to read a single sector. The net effect is that less time is spent waiting for the rotational latency of the disk. This mode of buffering is most beneficial to transients which read disk files sequentially. As a matter of fact, it may slow down transients which read files in a random access mode. Fortunatlly, the vast majority of CP/M transients read the disk sequentially. Examples of such transients are ASM.COM, MAC.COM, PIP.COM, and BASIC-E.COM (if the BASIC program doesn't do random I/O!!).

With write buffering, sectors that are to be written to disk are held in a buffer for a time, instead of writing them immediately. Sectors for any given track are held in the buffer until CP/M tries to write to a track that is not buffered. When it is time to change tracks, only those sectors which were changed are actually written before clearing the buffers to make room for the new data. This improves execution speed for the same reasons that read buffering does, i.e. a whole track can be written in about the time it takes to write a single sector. Transients which benifit from this type of buffering typically write to disk sequentially. Examples include ASM.COM, MAC.COM, and ED.COM.

DEFAULT PHILOSOPHY

Two of the qualities generally associated with a good program are versatility and ease of use. FAST has been written with these two qualities in mind as the primary design goals. Many times, however, these two goals can be contradictory. A versatile program is one which is capable of a wide variety of tasks or one that fits many different

applications. This usually means a program with many user selectable options. Forcing the user to type the same frequently used options every time a program is invoked or committing him to remember a long list of options is contradictory to the goal of ease of use. This conflict is resolved by the use of default mechanisms within FAST.

A default mechanism is simply a rule that can be used to make assumptions about the user's wishes when he has not stated them explicitly. Thus, versatility has been retained by allowing the user the ability to specify options explicitly, while the default mechanism frees the user from this tedium much of the time. In short, defaults allow versatility and ease of use to peacefully co-exist in the same program.

DEFAULT MECHANISMS IN FAST

There are two levels of defaults built into FAST. The first default mechanism is used when either the drive or buffering mode is omitted from an option string. If a drive name is not given, the default drive for CP/M is used (this is equivalent to "@"). If no buffering mode is specified, read and seek buffering are used. Thus, the following two commands would be equivalent.

```
    B>FAST [BRW] LOAD MUMBLE

    B>FAST [RW] LOAD MUMBLE
```

And so would the following two commands.

```
    A>FAST [BRS] PIP B:THIS=B:THAT
    A>FAST [B] PIP B:THIS=B:THAT
```

The second default mechanism comes into play when no option string is given on the command line. Instead of executing the transient with no buffering, FAST uses a default option string to specify the buffering used. This string is [@RS], giving the user read and seek buffering on the default drive when no option string is present on the command line. The user may, at his option, alter the default option to one which is more suitable for his typical uses. For instance, if you typically use FAST for assembling and don't have to worry about running out of memory, a good default string would be [@RWS]. Conversely, if you are running in a small memory system, a good choice would be [@R].

HOW TO CHANGE THE DEFAULT OPTION STRING

The default string is stored in FAST at address 0130H. Thus, to change the option string, use SID or DDT to change the existing string. Assuming you have SID this is very easy:

```
    A>SID FAST.COM
    SID VERS 1.4
    NEXT  PC  END
    0900 0100 94FF
    #S130
    0130 5B "[@RWS]
    0136 20 .
    #^C
    A>SAVE 8 FASTX.COM
```

If you do not have SID, you'll have to use DDT and figure out the hex for the ASCII string you wish to patch in. Then use the Substitute command to patch the file as above. In either case, test the patched file before killing the old FAST.COM and renaming the new file to FAST.COM.

WARNINGS

N-E-V-E-R CHANGE DISKS WHILE EXECUTING UNDER FAST!!!

Wait until you see the message "Execution under FAST now complete" before switching disks.

A rather nasty side effect of seek buffering is that it disables the disk change detection mechanism built into CP/M version 1.4. This means that you can't expect the operating system to give the warning message BDOS ERROR ON A: R/O if you forget and change disks without booting.

This also gives rise to a minor incompatibility with any transients which use the BDOS reset function (BDOS call 13). This function is used in transients which allow the user to change disks without rebooting. The only transient which we are currently aware of which fits this description is Microsoft's MBASIC.COM. MBASIC uses this BDOS function only when the RESET command is used. FAST is compatible with MBASIC as long as the RESET command is not used.

A good rule to remember is never to use FAST with any transient which requires user input. Any such transient, by waiting for your reply, would leave you with the opportunity to change disks.

COMPATIBILITY

FAST is fully compatible with the following transients and can be expected to provide a significant reduction in execution time.

ASM	LOAD	SUBMIT	DUMP	MAC
TEX	COMPARE	MODEM	FILES	PRINT
COMBINE	UNLOAD	FROMISIS	TOISIS	ICOPY
IDIR				

FAST is fully compatible with the following transients, but due to fact that they require user input, some caution should be exercised when they are used with FAST. As mentioned above, stopping for input gives the user the chance to inadvertently change disks which CP/M would not be able to detect. If the user is able to guard against this occurrence, he may use FAST with these transients and expect a significant reduction in execution time.

ED	PIP	DDT	SID	MBASIC
XYBASIC	BASIC-E	RUN WORDMASTER		

FAST is fully compatible with the following transients, but due to the way in which they access the disk, no speed improvement can be expected.

SYSGEN	MOVCPM	FAST

Due to an unfortunate memory dependency, STAT (version 1.4) does not correctly report the number of bytes remaining on the disk when it is executed under FAST. This is no great loss because FAST can do little to speed up the execution of STAT.

FAST MESSAGES

During the course of execution, FAST may print any of several different messages. Each of these messages is listed below along with conditions under which it is printed.

Beginning execution under FAST N.NN

This message is printed after the transient to be executed under fast has been successfully loaded, but just before control is transfered to it. This is simply an informatory message and does not signal an error condition. In this message, N.NN is the version number of FAST which you are using.

Execution under FAST now complete

This message is printed after the transient has finished execution and all disk buffers have been emptied. This too is simply an informatory message printed under normal conditons.

NO COM FILE

This message is printed when the transient filename given on the command line cannot be found in the directory. No recovery action can be taken, so the execution complete message is printed and FAST re-boots.

OUT OF MEMORY

This message is printed when the size of the TPA is exceeded while allocating buffers or while loading the transient. In either case, no recovery action can be taken, so the execution complete message is printed and FAST re-boots. There are several solutions to the out of memory problem: use less buffering, use smaller transients, or, of course, buy more memory.

DISK WRITE ERROR

This message is printed when FAST attempts to write data to disk and gets the unsuccessful completion flag back from the BIOS. The only recovery action taken is simply the printing of the message, then execution continues normally. This usually indicates a serious error (like a protected disk) and the user should boot and take corrective measures as soon as possible.

INVALID OPTION

This message is printed whenever an error is detected in the option string being scaned by FAST. This may be the option string supplied on the command line or the default option string in memory in the unlikely event that it has been incorrectly modified. Some examples of invalid options are the null option ([]), options containing invalid characters ([Q]), or options which specify buffering of the same type for the same drive twice ([ABA]).

MEMORY HIT

This message is printed whenever FAST attempts to write to a memory location and cannot read the data it has just written. The recovery action taken is to print the message and ignore the error. It is usually an indicator of hardware problems in your system and should be looked into immediately. If MEMORY HIT errors are persistent and you have a ROM monitor in your system, you can use FAST to help find the memory at fault. Create a special FAST.COM with a jump to your monitor patched into location 521H. This location is called when a memory hit is detected. The HL register contains the address of the bad byte and a bit in the A register is set for each bad bit in memory.

MEMORY ALLOCATON

When FAST is loaded, it automatically relocates itself to the top of the TPA (just like DDT and SID). This action overlays the CCP. A side effect of this is that transients which normally return to the CCP without booting will boot when executed under FAST. An example of such a transient command is LOAD.

As mentioned earlier, the only way a transient can tell that it is being executed under FAST is that the size of the TPA is decreased. The amount of memory taken out of the TPA by FAST is dependant on the number of buffers which are allocated. The formulas used to determine the amount of memory used are:

```
buffers = NTB * (3 + 131 * SPT) + NSB * (3 + 131 * SID)
SID = NDE / 4
mem used = 768 + [buffers]

Where:
    NTB  is the Number of Track Buffers allocated
    SPT  is the number of Sectors Per Track
    NSB  is the Number of Sector Buffers allocated
    SID  is the number of Sectors In the Directory
    NDE  is the maximum Number of Directory Entries
    [ ]  indicate rounding up the next highest
          multiple of 256
```

For a normal 8 inch IBM compatible version of CP/M, all these computations can be replaced by a simple estimating rule:

> Allow 768 bytes for FAST code overhead, 3409 bytes for each track buffer (R or W), and 2099 bytes for each seek buffer (S).

As an example, FAST executed without a command line option string will use 768 + 3409 + 2099 bytes. If this sum is rounded up to the next multiple of 256, the actual memory usage figure becomes 6400 bytes.