DAI FLOPPY DISK DRIVE MANUAL

D A I FLOPPY DISK DRIVE CONTENTS

D A I FLOPPY DISK DRIVE MANUAL

1. INTRODUCTION

This manual describes the start procedure and the DOS commands to help the user to use the BASIC and DOS operating system with the **DAI** Computer.

The **DAI** Computer communicates with the Floppy Disk System via a parallel bus corresponding to the DCE-BUS, via a flat cable.

The Disk System includes a power supply, an intelligent controller, two mini-floppy disk drives (80K bytes capacity each) and all necessary read/write electronics.

Also, a disk operating system is build-in which can handle commands and errors. A system disk must be available for use.

At power on (or reset) the **DAI** Computer verifies the presence (or not) of the disk system, and subsequently initializes for disks or cassette. The disk initialization involves loading of a disk operating system from the master disk (in drive 0) into the DC RAM.

Once the disk operating system is loaded, any character from the keyboard will start normal operation of the DC (as for the cassette configuration).

The standard commands SAVE, LOAD, SAVEA, LOADA of BASIC and RW commands of the UTILITY, are all usable with disks exactly as for cassettes when drive 0 is in use.

2. POWER-ON SEQUENCE

Place the dual disk drive at the right side of the **DAI** computer : the left drive is drive 0 and the drive on the right is drive 1. It is important to identify them correctly.

I. Connect the disk system via the flat cable to the DCE-BUS connector at the back of the Computer.

II. Ensure that the doors of both diskette drives are open and turn on the disk system.

III. Insert the system diskette into drive 0 and close the door.

IV. Turn on power to the Computer (press RESET if necessary)

The following sequence of steps will then be executed automatically.

I. The bootstrap file $DKBOOTS on the system diskette in drive 0 will be loaded into the stack RAM memory from address F800H upwards. The bootstrap routine will occupy approximately 200 bytes. Control will then be passed to its entry point at address F800H.

This bootstrap file has a special format and contains pure binary data, without the usual file-definition parameters which specify the start, end and entry addresses. This bootstrap routine is used to load and execute the diskette file $MSTRDOS, which normally contains DOS.

II. The file $MSTRDOS on the system diskette in drive 0 will then be loaded and executed. This is a normal binary file which is approximately 5K bytes in length, and contains the full DOS. The binary data on this file is preceded by the three usual file-definition parameters indicating the start, end and entry addresses.

NOTE : This file can also be a user program in the correct binary format. In this case, it will be automatically loaded and executed on power-on or system reset.

III. The binary file $USER.BIN will then be searched on the system diskette.

If no such file is found, execution will proceed to step IV.

If the file is found, it will be opened for input and loaded into memory as specified in the file-definition parameters. Any loading error will cause execution to return to the normal BASIC initialisation sequence (i.e. wait for any key input, and then display the BASIC sign-on message and prompt character).

If the entry address specified in the file-definition parameters is equal to FFFFH (i.e. no entry address), execution will proceed to step IV.

If a valid entry address is specified in the file-definition parameters (i.e. any value except FFFFH), the loaded program will be called as a machine code subroutine with that entry address. Upon executing the Return instruction code at the end of the routine, the contents of the zero flag in the CPU flag register will be tested. If this flag is set (=1) execution will proceed to step IV; if it is clear (=0) execution will return to the normal BASIC initialisation sequence (i.e. wait for any key input, and then display the BASIC sign-on message and prompt character). The zero flag should therefore be set or cleared by the program stored in $USER.BIN to indicate the presence or absence of a BASIC program for automatic loading followed by running (step IV).

IV. The BASIC program file $USER.BAS will then be searched on the system diskette.

If no such file is found, execution will return to the normal BASIC initialisation sequence (wait for any key input etc.)

If the file is found, it will be opened for input and loaded into memory as a BASIC program (semi-compiled code) plus symbol table, and then executed (i.e. RUN).

3. FILE REFERENCES

Each file is known to DOS by its "file reference". The file reference consists of three items of information which serve to identify a file uniquely.

They are :

* Filename
* Extension
* Diskunit

The standard format of a complete file reference is :

Filename. extension : diskunit

3.1 Filename

The **filename** is a name that is given to identify the file and must always be included in the file reference. It is made by 1 to 8 following characters :

ABCDEFGHIJKLMNOPQRSTUVWXYZ

0123456789

! " // $ % & ' () +- * / ; < = > ?

DOS command words (e.g. BACKUP, COPY, IDISK, etc) must not be used as file names.

3.2 Extension

The **extension** is an optional group of three of the above characters which serves to identify the type of information contained in the file. It immediately follows the filename, being separated from the latter only by a single point. No other characters or spaces are allowed in between filename and extension. If an extension is not specified, DOS assumes three space characters as the extension.

Although all data in files consist of some sequence of binary digits, these bits are interpreted in different ways according to the type and usage of the file.

DOS does not interprete the extension in a filereference in any way. It simply uses the extension as an integral part of the filename, to identify the file. The user is therefore free to use extensions of his own choice.

3.3 DISKUNIT

The **diskunit** is also optional, and specifies on which of the two drives the file is located. It consists of a single digit, 0 or 1, preceeded by a colon. It immediately follows the extension, if one exists, or otherwise the filename. No other characters or spaces may precede the diskunit. If a diskunit is not specified, DOS assumes zero and the file is assumed to be on the diskette in drive 0.

3.4 EXAMPLES

Examples of filereferences are given below :

FILE	refers to a file with the filename FILE. DOS assumes an extension of three spaces, and drive 0.
FILE.	)
FILE.:0	) Functionally identical to filereference in
FILE:0	) previous example
FILES-1.BAS	refers to a file with filename FILES-1 and extension BAS, containing semi-compiled code. It is assumed to be in drive 0
FILES-1.BAS:1	Same file as before, but on the diskette in drive 1

3.5 BASIC FILE

Basic and utility commands which write files to diskette automatically generate implicit extension names.

3.5.1 BASIC FILE

FILENAME.BAS : A Basic program file (containing semi-compiled code), with the program code followed by symbol table
Generated by the BASIC command SAVE; can be read back via the BASIC command LOAD

Example:

*SAVE "EXAM:0" Save the basic program currently in memory onto the diskette in drive 0 as file EXAM.BAS

*LOAD "EXAM:0" This basic command will load the BASIC program file EXAM.BAS from drive 0 into system memory

3.5.2 BINARY FILE

FILENAME.BIN : A binary file preceded by file definition parameters (see DSAVE command)
Each file definition parameter is a 16-bit address in low-high order. The binary file will be generated by Utility command W and can be read back via the Utility command R. If the execution address in the file- definition parameters is equal to the FFFFH, this will be interepreted as no-entry. The utility command W will produce execution address = FFFFH; if a valid entry address is desired, the DOS command DSAVE can be used. The utility command R will read such a file and automatically enter it at the execution address if it is not equal to FFFFH (only if no address offset is specified in the R command).
(See sections 7.3.11 and 7.3.12 in DAI Computer Manual).

3.5.3 FLOATING-POINT ARRAY FILE

FILENAME.FPT

Example : * SAVEA A! "EXAMP"
Save the floating point array A! on disk as file EXAMP.FPT

* LOADA A! "EXAMP"
Load the file EXAMP.FPT in the floating point array A!

3.5.4 INTEGER ARRAY FILE

FILENAME.INT

Example : * SAVEA A% "TEST"
Save the integer array A% on disk as file TEST.INT

* LOADA A% "TEST"
Load the file TEST.INT in the integer array A%

3.5.5 STRING ARRAY FILE

FILENAME.STR

Example : * SAVEA A$ "CODE"
Save the string array A$ on disk as file CODE.STR

* LOADA A$ "CODE"
Load the file CODE.STR in the string array A$

Since the above extension names are automatically generated by the relevant Utility or BASIC commands, file references specified in such commands must not include these extensions. Subsequent Utility or BASIC commands used to load such files must also use file refences excluding the extensions.

However, subsequent DOS commands accessing such files must include the correct extension name in the file reference.

Example :

SAVE "PROGRAM:1" This BASIC command will copy the BASIC program currently in memory onto the diskette in drive 1, as file PROGRAM.BAS.

LOAD "PROGRAM:1" This BASIC command will load the BASIC program file PROGRAM.BAS from the diskette in drive 1 into system memory.

If, for example, this file is to be deleted via the DOS command DELETE, the file reference must include the extension, as shown below :

```
DELETE PROGRAM.BAS:1
```

4. THE DISK OPERATING SYSTEM (DOS)

DOS performs all diskette data and file management tasks for the user, and takes care of the physical positioning of data on the diskettes.

The DOS resides on the system diskette, and is loaded for execution on the bottom of system RAM memory (see memory map section 7).

The Commands

RESETD	**COPY**	**LOCAL**	**CREATE**	**IDISK**	**RENAME**
DIR	**BACKUP**	**REMOTE**	**DELETE**	**RECOVER**	**VERIFIY**
TINYDOS	**COMPACT**	**AUTO**	**VERIFY**	**PROTECT**	
SAVE	**LOAD**	**OPENI**	**ASSIGN DISK**		
SAVEA	**LOADA**		**ASSIGN CASSETTE**		
DSAVE	**DLOAD**		**ASSIGN SCREEN**		
			ASSIGN PRINTER		
			ASSIGN KEYBOARD		
			ASSIGN RS232		

DOS commands can be entered via the keyboard, in response to the usual BASIC prompt. They cannot be entered from the utility or from the Editor.

Suc a DOS command must be the first one on that line. Anything that follows a DOS command and its paramaters on the same line will be ignored.

4.1 DOS COMMANDS (V 2.0)

4.1.1 ASSIGN - Assign Peripherals

FORMATS

*** ASSIGN DISK** Assign diskette for read and write operations associated with LOAD,LOADA,SAVE,SAVEA,R, W commands

Once a file is created with a certain file length it is not possible to extend its length. However, extending the length of an existing file can be indirectly achieved by creating a new file with a larger file length, and copying the first file to it.

Examples:

*** CREATE FILEY:1 8**	Create a new file, 8 sectors in length
*** COPY FILEX:1 FILEY:1**	Copy FILEX to FILEY
*** DELETE FILEX:1**	Delete FILEX
*** RENAME FILEY:1 FILEX:1**	Rename FILEY as FILEX

Important Note : The diskette containing the specified file must have its write-protect notch uncovered.

4.1.7 DELETE - Delete Files

FORMAT : *** DELETE filereference filereference2...**

The DELETE, command removes the specified files "filereference", "filereference2",... from the corresponding diskette directory files. The deleted filename can then be used in the creation of new files. When a file is deleted, only the directory entry is flagged as deleted and the data in the file continues to remain on the diskette. The sectors used by the deleted files will not be released for re-allocations to new files until all the accessible files on the diskette are compacted into one block via the COMPACT command. Until a diskette is compacted, all deleted files can be restored by using the RECOVER command with the correct file reference.

Files that have been write-protected via the PROTECT command cannot be deleted. Attempting to delete such a write-protected file will produce a DOS error message with code 07 indicating that the file is write-protected.

One or more filereferences can be specified in the DELETE command.

Up to 15 file references can be specified in a single DELETE command, provided the total number of characters in the command (including spaces) does not exceed the input buffer length of 80 characters.

Examples :

*** DELETE TEST.SRC:1 PROG TEST.OBJ:1**

Delete files TEST.SRC and TEST.OBJ from diskette in drive 1, and delete file PROG from diskette in drive 0

Important Note : The diskettes containing the specified files must have their write-protect notches uncovered

4.1.8 DIR - Display Diskette Directory:

FORMAT : *** DIR Diskunit**

Each diskette has a directory file containing information about all files on that diskette. The directory entry for each file contains the filename, extension, protect mode and the length of the file. The DIR command displays on the console directory information relating to all files present on the diskette loaded into drive "diskunit" (0 or 1). The number of free sectors on the diskette available for allocation to new files is also displayed.

File extensions are listed under heading EXT. Files created without extension names are given extensions of three spaces. The protect mode for each file is given under the heading ATR. A 'W' in this column indicates that the file has been write and delete protected via the PROTECT command. Write protection of all the files on the diskette by the covering of the write-protect notch will not be indicated in the directory listing.

A 'N' in the ATR column indicates a non-protected file. All 18 sectors on track 0 are reserved for the directory file $DKDIR. User files start from track 1 sector 1. The number of sectors allocated to each file is given as a decimal number in the last column. The number of free sectors available on the diskette is displayed as a decimal number at the end.

Example: ***DIR1** Display the directory of diskette in drive 1

4.1.9 DLOAD - Load a Binary File into Memory :

FORMAT : *** DLOAD Filereference**

This command loads the file 'filereference' containing binary data written via the DSAVE command into memory. The file is loaded starting at the address that was specified when the data was written to the file via the DSAVE command. The file 'filereference' will be opened and closed automatically by the DLOAD command.

Example :

The file TESTPROG.BIN:1 contains a binary program written to it by the command DSAVE TESTPRG.BIN:1 100 1C00 A10 (start-address = 100H; end-address = 1C00H; entry-address = A10H).

This command DLOAD TESTPRSG.BIN:1 will load the program file TESTPRG.BIN:1 into memory starting at address 100H, and extending up to address 1C00H.

4.1.10 DSAVE - Save a Binary File on Diskette :

FORMAT : *** DSAVE filereference startadr endadr entryadr**

This command can be used to save a binary program resident in memory starting at low address 'startadr' and ending at high address 'endadr', onto an existing or non-existing diskette file 'filereference'.

If the program entry-point address 'entryadr' is omitted, it will be assumed to be equal to 'startadr'. The three address parameters 'startadr', 'endadr' and 'entryadr' must be given in hexadecimal. They will be written to the file at the start of it, to be used subsequently when re-loading the file into memory and executed.

The file 'filereference' will be opened and closed automatically. If the file 'filereference' has been created using the CREATE command with the file length unspecified i.e. equal to all free sectors on that diskette, it will be closed and its length adjusted to the number of sectors actually used.

Before writing the data to the file 'filereference' the DSAVE command writes to it the parameter 'startadr' followed by 'endadr' and 'entryadr'. Each parameter is written as a 16-bit binary value, and occupies 2 bytes in the file. Data is written to the file following the 6-byte block containing these three parameters.

Example :

CREATE FILE1.BIN:1

*** DSAVE FILE1.BIN:1 100 7FF 200**

(saves the binary program in memory between address 100H and 7FFH having the program entry address at 200H, on the file FILE1.BIN:1 and adjusts the file length to 15 sectors).

4.1.11 IDISK - Initialise Diskette

FORMAT : ***IDISK Diskunit**

The IDISK command initialise a diskette in drive 'diskunit' (0 or 1) by creating the directory file, writing sector identification labels on it, and testing each sector for correct read ano write operation. New diskettes must be formatted via the IDISK command before they can be used.

Diskettes used in the DAI diskette system are "softsectored". This means that each sector is preceded by a label block of data containing identification information for that sector.

During read and write operations DOS identifies the correct sector on the selected track by reading these identification labels.

The IDISK command does not verify the diskette (can be done subsequently via the VERIFY command).

For example : a new diskette in drive 1 can be initialized as follows :

***IDISK 1**

The absence of error messages indicates that all sectors on the above diskette can be written to and read back correctly.

Important Notes

1) The diskette being initialized must have its write-protect notch uncovered.

2) The original contents of the diskette will be lost.

4.1.12 OPENI - Open File for Input

FORMAT : ***OPENI filereference**

The OPENI command opens the diskette file 'filereference' in input mode. Input mode specifies that data will be read from the file.

Opening a file in input mode enters its name in the active file table maintained by DOS, initializes a file pointer to the start of that file, and allocates one of the buffers on the diskette controller card to the file. After a file has been opened in input mode, its contents can be read serially starting at the beginning.

4.1.13 PROTECT - Write-Protect File

FORMAT : *** PROTECT Filereference**

A write-protect file cannot be over-written, renamed or deleted. Attempts to do so will produce a DOS error message indicating that the file is write-protected.

However, certain commands such as IDISK and BACKUP will overwrite all the original contents of a diskette without recognising the write-protect attributes given to files on it.

The PROTECT command enables selected files on a diskette to be write-protected, whereas covering the write-protect notch of the diskette will protect the entire diskette. Write-protect attributes given to files via this command will be indicated on the diskette directory listing with a 'W'.

Example : *** PROTECT DATA1:1**

Important Note : The diskette containing the specified file must have its protect notch uncovered.

4.1.14 RECOVER - Recover a Deleted File or Unprotect a Protected File

FORMAT : *** RECOVER Filereference**

The RECOVER command enables a previously deleted 'filereference' to be recovered and made accessible again. A deleted file cannot be recovered if the data on the diskette has subsequently been compacted via the COMPACT command.

When a file is deleted, oly the directory entry is flagged as deleted and the data in the file remains on the diskette.

The sectors used by the deleted files will not released for re-allocation to new files, until all the accessible files on the diskette are compacted into one block via the COMPACT command. Until a diskette is compacted, all deleted files on it can be restored by using the RECOVER command with the correct filereferences.

The RECOVER command also unprotects the protected file 'filereference'.

Example : *** RECOVER DATA:0**

Important Note : The diskette containing the specified file must have its write-protect notch uncovered.

4.1.15 REMOTE :

FORMAT : *** REMOTE**

Opens cassette motor control contacts and enables automatic motor control.

4.1.16 LOCAL

FORMAT : *** LOCAL**

Closes cassette motor control contacts to enable use of cassette unit in local mode, and disables automatic motor control.

4.1.17 RENAME - Rename a File:

FORMAT : *** RENAME oldfilereference newfilereference**

This command can be used to rename an existing file 'oldfilereference' with the new name 'newfilereference'. Only the directory entry is changed.

The file 'newfilereference' should not exist already.

The file 'oldfilereference' should exist.

If a non-existant file is specified, a DOS error will be produced.

The file 'oldfilereference' may have a writeprotect attribute given to it via the PROTECT command.

The RENAME command assumes that the diskunit in 'newfilereference' is the same as that in 'oldfilereference'. If a different diskunit is specified in the 'newfilereference' it will be ignored.

Example: *** RENAME TEST.BIN:1 PROG:1**

Important Note :The diskette containing the specified file must have its write-protect notch uncovered.

4.1.18 RESETD - Reset Diskette Controller

FORMAT : *** RESETD**

The RESETD command will reset the diskette controller card, clear the active file table maintained by DOS, and make standard peripheral assignments to the logical I/O channels.

4.1.19 TINYDOS :

FORMAT : *** TINYDOS**

See the section "RAM memory Map".

4.1.20 VERIFY - Verify Data on Diskette:

FORMAT : *** VERIFY FILEREFERENCE**

The verify command reads the contents of the file 'filereference' to check for read errors. The original contents of the specified file remain unchanged. The file will be opened and closed automatically.

*** VERIFY DISK n** — Verify contents of the diskette in drive (n = 0 or 1). In case read errors are detected, several error messages will be displayed, and the checking will continue.

*** VERIFY DISKS** — Similar to above command, except that the contents of diskettes in both drives will be verified.

4.3 SPECIAL DOS FEATURE (V 2.0)

An additional feature is provided by DOS to enable a BASIC program file to be loaded and executed when its filename is stored in a BASIC string variable.

This is supplementary to the normal LOAD "file" command which only allows a string constant as the filereference.

This feature is called up by the sequence :

CALLM #300, A$ where A$ can be any array variable.

Example : Write program and save it.

```
* 10 FOR X= 1 TO 10 : PRINT X
* 15 NEXT
* 20 END

* SAVE "TEST:0"
save program as file TEST.BAS.

: Load program and execute it.
* NEW
* 10 A$ = "TEST:0"
* 20 CALLM #300, A$
* 30 END
* RUN
```

On the screen :

```
BASIC V1.1
 1.0
 2.0
 3.0
 4.0
 5.0
 6.0
 7.0
 8.0
 9.0
10.0
 END PROGRAM
```

The file reference must be exactly in the format of the example. Do not include an extent name, and do not expect automatic default to disk unit 0.

5. INCLUDING DOS COMMANDS IN BASIC PROGRAMS

Any DOS command and its parameters can be included in a BASIC program by containing it in a PRINT statement.

Any single or any group of such DOS commands contained in PRINT statements must be preceded by the BASIC command (see also section 11.2).

```
POKE # 131,3
```

and ended by the BASIC command

```
POKE # 131,0
```

The total number of characters generated by such a DOS command contained in a PRINT statement (upto the terminating carriage-return) must be less than or equal to 48.

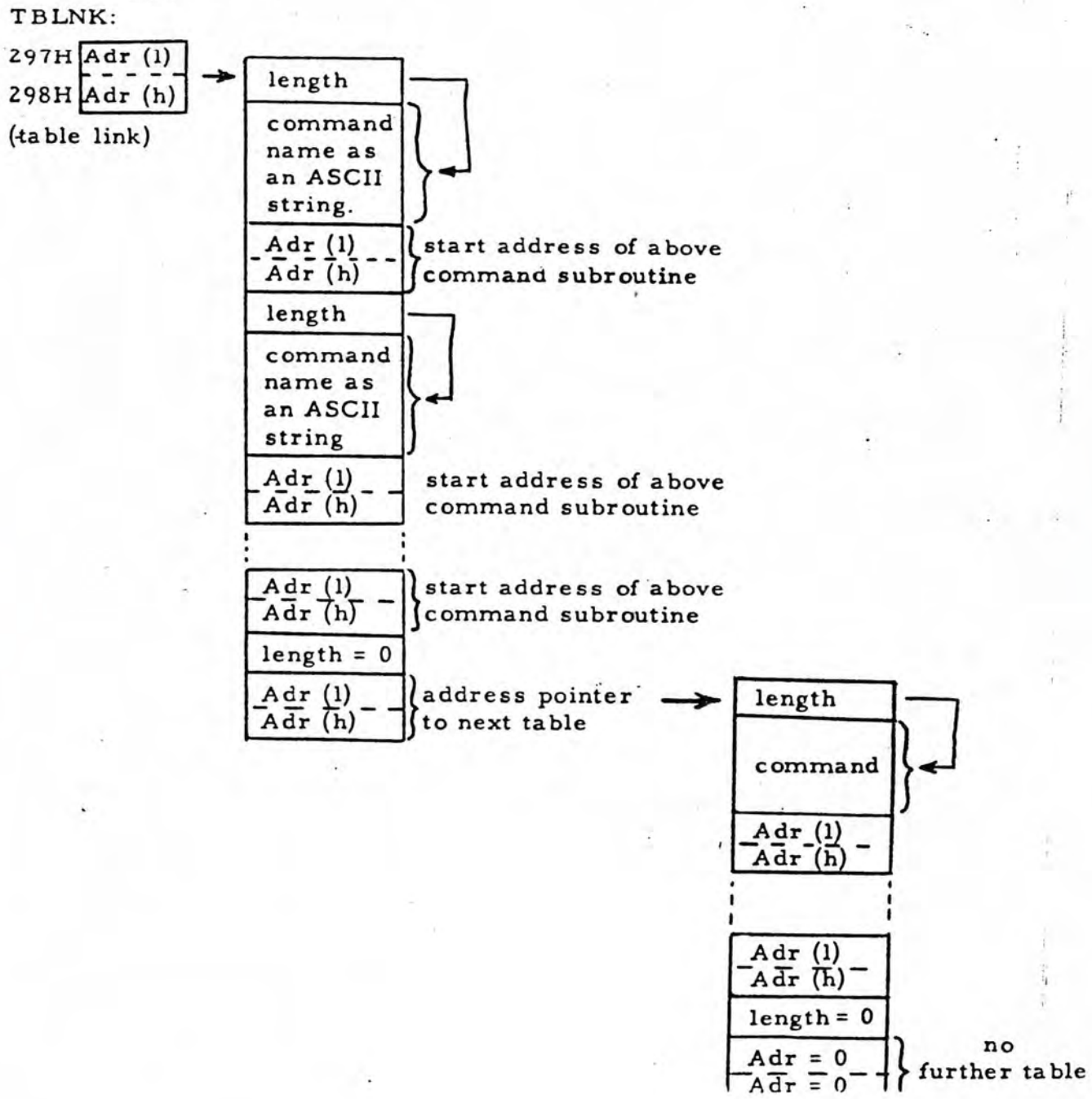
The command "POKE #131,3" will channel all subsequent characters output via the BASIC statement PRINT to the DOS command handler. They will be re-channeled to the printer in execution of the BASIC command "POKE #131,0".

Example :

```
INPUT "FILENAME=";N$
POKE #131,3
PRINT "CREATE";N$;":1"
PRINT "COPY SOURCE:1";N$;":1"
PRINT "PROTECT";N$;":1"
POKE #131,0
```

6. DOS COMMAND TABLE STRUCTURE

The structure of the DOS command table need not be known for normal use of DOS commands. However, an understanding of this table structure enables the user to add machine code sequences which can be executed simply as new DOS-type command codes (after full DOS has been loaded). Whenever a DOS command is encountered (via keyboard or via the PRINT statement) control is passed to the DOS command handler, which will scan the DOS command table for the presence of that command.



The scheme above illustrates visually the structure of the DOS command table. The address locations 297H and 298H contain the pointer address (low, highorder) to the start of the command table. The first entry in the command table contains a binary number specifying the number of ASCII characters which follow immediately and constitute the first command code name. The last character of the command code name is followed by the start address of a machine-code subroutine which must be called to execute that command. This address will be followed by the length of the next command code etc. If no more codes follow, the length will be set to zero.

The two bytes following the last length parameter (zero) indicates the presence or absence of an address to the next table (low, high order).

New user-defined commands can be linked into the DOS command table in the following way :

1. Set up the user command table in correct format.
   The last length parameter which is zero must be followed by an address pointer to the start of the normal DOS command table.

   Example :

   Set the label corresponding to the start of the user-defined command table be NEWTB, and let the label corresponding to the address pointer at the end of it (pointing to the DOS command table) be CONTA

   The user table can be linked to the DOS command table in the following way :

```
INIT :

        PUSH    H
        LHLD    TBLNK
        SHLD    CONTA
        LXI     H,NEWTB
        SHLD    TBLNK
        POP     H
        RET
```

   Such a sequence must be executed once only.
   If not, the link to the DOS command table will be lost.

*** ASSIGN CASSETTE**

Assign cassette for read and write operations associated with LOAD, LOADA, SAVE, SAVEA,R, W commands. Normal default is to cassette unit 0 after reset; if this has been changed subsequently, the above command will reinstate the last selection.

*** ASSIGN CASSETTE n**

Assign cassette unit n (n=0 or 1) for read and write operations associated with LOAD, LOADA, SAVE, SAVEA, R, W commands.

*** ASSIGN TO DISK n**

DISK n (n = 0 or 1)
Similar to above ASSIGN commands, but for write operations only.

*** ASSIGN FROM DISK n**

DISK n (n = 0 or 1
Similar to above ASSIGN commands, but for read operations only.

*** ASSIGN OUTPUT TO SCREEN**
PRINTER
DISK

Assigns the system data outputs to the SCREEN, SCREEN + PRINTER, or diskette, respectively. If the DISK option is specified the system output will be written to a currently open ASCII diskette file.

*** ASSIGN INPUT FROM KEYBOARD**
RS232
DISK

Reads the system input data from the keyboard, RS232 channel, or diskette respectively. The first two options will generate automatic echo of the data read onto the screen. If the DISK option is specified, the data will be read from a currently open ASCII diskette file

4.1.2 BACKUP = Produces a backup diskette copy

FORMAT : *** BACKUP ORIGINDISK**

The BACKUP command completely duplicates all of the data on the diskette loaded on drive "origindisk" (0 or 1) to an initialised or uninitialised diskette loaded into the destination disk drive. The BACKUP command automatically formats the destination diskette. The original contents of the diskette in drive "origindisk" remain unchanged. Write-protect attributes, given to files via the PROTECT command, will be copied correctly.

Example : *** BACKUP 0** Make an exact copy of the total contents of the diskette in drive 0 onto the diskette in drive 1

In case a read or write error is encountered during the execution of the BACKUP command, an error message will be displayed on the console.

Important Note : The destination diskette must have its write-protect notch uncovered.

4.1.3 AUTO - Automatic Line Numbering

FORMATS :

AUTO nnn Enables automatic line numbering for keyboard entry of BASIC programs, starting at line number "nnn" (must be non-zero) with steps of 10

AUTO Carriag-return
Disables the above automatic line numbering and resumes manual line numbering. Before the command is typed, the line number which automatically appears at the start of the line must be deleted by using char. delete.

4.1.4 COMPACT - Compact Data on Diskette

FORMAT : *** COMPACT Origindisk**

When a diskette file is deleted the sectors that were utilized for that file are not automatically released for re-allocation to new files. As more files are added the number of free sectors remaining on the diskette can decrease to zero, even though some files have been deleted. The COMPACT command rearranges the data on the diskette and releases all sectors used by deleted files for re-allocation to new files. All of the sectors used by non-deleted files are made to occupy one contiguous block from the start of the diskette. All remaining free sectors will then be contiguous at the end of the diskette, and become available for allocation to new files.

The COMPACT command reads all the non-deleted files on the diskette in drive "origindisk" (0 or 1), and writes them in sequence onto an initialized or unitialized diskette in the other drive . It automatically formats the destination diskette. Write-protect attributes given to files via the PROTECT command will be copied correctly.

Examples :

* COMPACT 1 Compacts the data on diskette in drive 1 onto the diskette in drive 0

* COMPACT 0 Compacts the data on diskette in drive 0 onto the diskette in drive 1.

In case a read or write error is detected during the execution of the COMPACT command an error message will be displayed on the console.

Important Note : The destination diskette must have its write-protect notch uncovered

4.1.5 COPY - Copy a File

FORMATS :

* **COPY Filereference** (with diskunit = 0, or omitted)
  Copy file from diskette in drive 0 to the diskette in drive 1

* **COPY Filereference** (with diskunit 1)
  Copy file from diskette in drive 1 to the diskette in drive 0

* **COPY Sourcefileref:ø destinationfileref:1**
  Copy the specified source file to the specified destination file

Note : All three formats of the COPY command automatically open and close the destination file. If the destination file does not already exist, it is created to be 1 sector longer in length than the source file. This feature can easily be used to extend the length of an existing file.
If the same file length is desired, the destination file should first be created via the CREATE command.

Important Note :

The diskette containing the specified destination file must have its write protect notch uncovered.

4.1.6 CREATE - Create a New File

FORMAT : *** CREATE filereference filelenght**
CREATE filereference

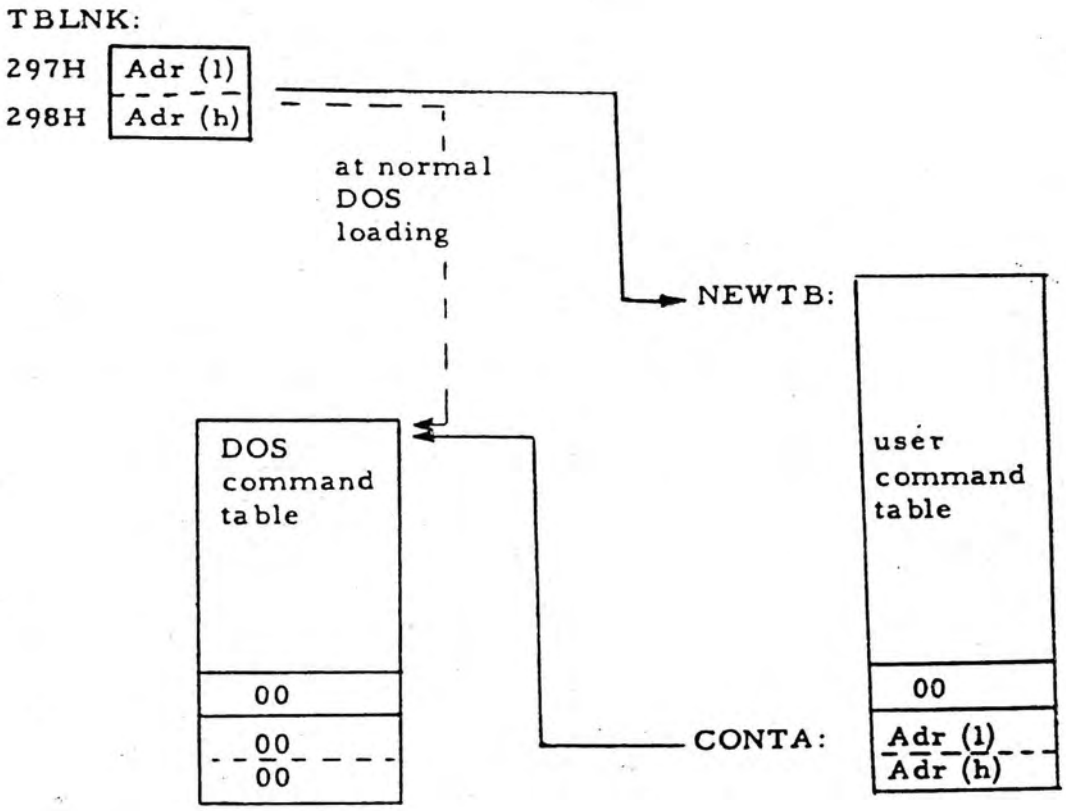
The CREATE command creates a new file "filereference". The "filelength" is a decimal number specifying the number of sectors to be allocated to the new file. The CREATE command enters the filename in the diskette directory file, and allocates the specified number of sectors to the created file. Space is allocated to a file in complete sectors, even if the last sector in a file is only partially used As files are created DOS allocates sectors sequentially, starting from the first sector following the end of the last file (accessible or deleted) physically present on the diskette.

The final length requirement of an output is sometimes not known until after all data is written to it. In such cases it is uesful to be able to allocate all the free sectors currently available on a diskette to that file. This is achieved by omitting the "filelength" parameter in the CREATE command. Only one such file is permitted on each diskette. After all the data has been written to such a file, it will be closed to adjust its length to the number of sectors actually used and to release all the unused sectors.

Examples :

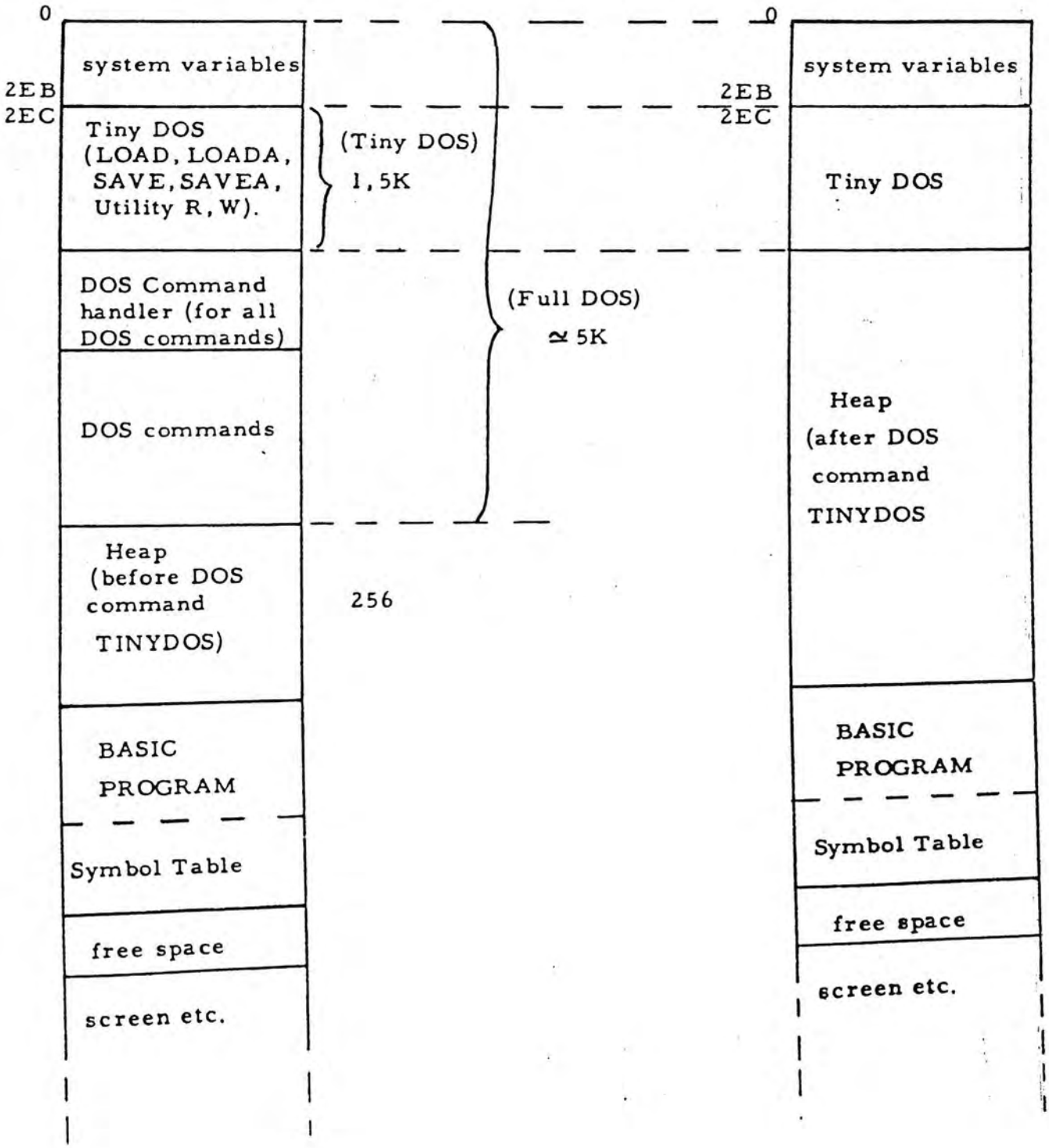
*** CREATE GEN1 1**	Create file GEN1 of length 1 sector, on diskette in drive 0
*** CREATE GEN1:1 1**	Idem Disk 1
*** CREATE GEN2**	Create file GEN2 on diskette in drive 0 and allocate all free sectors to it.

The locations TBLNK and TBLNK+1 will be initialized to the start of the DOS command table when DOS is loaded into memory.



7. RAM MEMORY MAP

RAM Memory Map



The power-on reset bootstrap sequence loads the full DOS (about 5K) into memory. At any subsequent time, this full DOS can be reduced to tiny DOS via the DOS command TINYDOS. Tiny DOS represents a cassette replacement, and only supports the BASIC commands LOAD, LOADA, SAVE, SAVEA and the Utility commands R,W.

All DOS commands are processed via the DOS command handler. The DOS command TINYDOS will delete the DOS command area and ther DOS command handler, by re-allocating these areas to the heap. The heap can then be reduced and the BASIC program area corresponding increased via the BASIC command CLEAR.

For example : The binary program file $USER.BIN loaded during the power-on reset bootstrap procedure can call the DOS command routine TINYDOS to reduce the full DOS top TINYDOS.

8. ASCII INPUT AND OUTPUT FILES

In order to provide the user a more flexible file format outside the constraints of the array SAVE and LOAD commands (LOADA, SAVEA), sequential ASCII files can be used.

Theser operate, and are interfaced by the user as follows :

8.1 ASCII INPUT FILES

The file must first be opened by using the 'OPENI' command. The format of this is :

*** OPENI FIRELEREFENCE**

This command automatically closes any file previously opened for ASCII input.

After the execution of the command 'ASSIGN INPUT FROM DISK' any time that the system would normally access the keyboard for character input these will come, in sequence, from the opened file. Thus, immediate commands, program entry, utility commands may all be prepared as are responses to 'INPUT' or 'GETC' statements. When the end-of-file error is reported, the input stream automatically return to the keyboard, thus preventing any serious system crashes that may otherwise be caused.

An example of using the ASCII input mode follows :

```
 10   Clear 1000 :POKE #131,3:PRINT "OPENI $DKDIR"
 15   PRINT "ASSIGN INPUT FROM DISK":PRINT "ASSIGN OUTPUT TO
      SCREEN"
 20   FOR I=1 to 18:REM 18 SECTORS IN THE DIRECTORY
 30   FOR J=1 to  6:REM  6 ENTRIES PER SECTOR
 40   A$=""
 50   FOR K=1 to 21:REM 21 BYTES PER ENTRTY
 60   A$=A$+CHR$(GETC):REM NEXT CHARACTER FROM DISK
 70   NEXT K
 80   PRINT LEFT$(A$,11):REM FILENAME + EXTENSION
 90   NEXT J
100   K=GETC+GETC:REM PURGE 2 PAD CHARACTERS
110   NEXT I
120   POKE #131,3:PRINT "ASSIGN INPUT FROM KEYBOARD"
130   POKE #131,1
140   END
```

8.2 ASCII OUTPUT FILES

A system of ASCII control characters has been defined to direct the command character stream ('PRINTED' after 'POKE #131,3') to either the DOS command handler, the ASCII file manager, or the currently selected ASCII output file. These control characters, and their functions are as follows :

CHR$(1) Close current output file (with adjust) switches character stream to file manager (normally used to precede filereference). CHR$(1) means SOH.

Special Case :

SOH + ETX, closes file but does not open a new file

CHR$(2) Used to select the ASCII file to receive characters. When STX follows the filereference (preceded by SOH), the file is created if necessary, then opened for output. CHR$(2) means STX.

CHR$(3) Returns character stream back to the DOS command handler. This is the only character which can suspend the character stream going to the output file. Hence, it is also the only exception to the otherwise totally transparent nature of the ASCII input/output mode. CHR$(3) means ETX.

These characters, and the filereference and data stream are sent by normal PRINT statements whilst the output has been assigned to disk.

I.e. POKE # 131,3

Normal output to screen or printer may be freely interspersed with disk data. DOS commands may also be issued but caution must be taken in avoiding the use of any DOS command which accesses the diskettes. These can cause the file to be closed prematurely and without adjust.

Thus the final sector of the dataset may not be written onto the diskette. This cautions also applies to the ASCII input mode.

An example of ASCII output :

```
  1   CLEAR 1000:SOH$=CHR$(1):STX$=CHR$(2):ETX$=CHR$(3)
 10   POKE #131,3:PRINT "RESETD"
 20   PRINT SOH$+"FILENAME"+STX$+ETX$
 30   PRINT "ASSIGN OUTPUT TO SCREEN"
 40   PRINT "THIS TEXT IS ON THE SCREEN"
 50   POKE #131,3:PRINT STX$+THIS TEXT IS TO THE DISK FILE
      "+ETX$
 60   PRINT STX$+"THIS IS SOME MORE TEXT"
 70   PRINT:PRINT:PRINT "THEN SOME BLANK LINES"+ETX$
 80   PRINT SOH$="ANOTHER"+STX$+"DATA TO ANOTHER FILE"+ETX$
 90   PRINT SOH$+ETX$:REM CLOSE LAST FILE
100   PRINT "ASSIGN OUTPUT TO SCREEN"
110   PRINT:PRINT "BYE-BYE"
120   STOP
```

9. MERGING OF PROGRAMS

9.1 Introduction

Large programs can be devided into smaller modules, and executed.

9.2 Example :

STEPS TO BE DONE :

1) Write program 1

2) Save on Disk

3) Write second program 2 with different line numbers

4) When ready : to merge both programs :

   a) Clear enough place for the program on disk, for instance CLEAR 10000

   b) EDIT (go to editor)

   c) BREAK BREAK (2 times BREAK !!!)

   d) Load program 1 from disk

   e) Poke #135,2 (fetch program 2 from EDIT buffer)

5) LIST

Both programs are merged together now.

10. MASTERDOS ROUTINES

10.1 Different Routines

Under utility control or with a machine code routine, the user has access to all the disk operating system features.

The subjoined table gives a summary of these features.

FUNCTIONS	Accu	B,L	H,L	D,E	ADDRESS(HEX)
CREATE A FILE	Diskunit in ASCII	File Length	Pointer to filename	Length of filename	501
DELETE A FILE	Diskunit in ASCII		Pointer to filename	Lenght of filename	A35
OPEN A FILE	Diskunit in ASCII	B='O';Output ='I';Input ='R';Directory	Pointer to filename	Length of filename	447
CLOSE A FILE	A='I';Input ='O';Output ='D';Directory				512
WRITE A MEMORY AREA IO DISK	'I'for Output		Pointer to first byte to send	Pointer to last byte to send	A7E
READ TO A MEMORY AREA FROM DISK	'I'for Input		Pointer to start of memory area	Pointer to end of memory area	A84
WRITE A BLOCK OF DATA TO DISK	'O'for Output		Pointer to start of block	Length of block	58F
READ A BLOCK OF DATA FROM DISK	'I'for Input		Pointer to start of same area	Pointer to end of same area	569
WRITE A SECTOR	Diskunit in ASCII	B=tracknumber C=Sectornumber	Pointer to sector buffer		AAD
READ A SECTOR	Diskunit in ASCII	B=tracknumber C=Sectornumber	Pointer to sector buffer.		A97

10.2 Example in Utility Mode

A typical sequence is :

```
Z3          ; reset buffer
```

Set registers

G (address)

On return the A register may contain an error code.

Create File

A = Disk Unit in ASCII

B,C = File Length

H,L = Pointer to file name

D,E = Length of file name

Entry address 501 Hex

EXAMPLE :

```
Z3
S2000 XX - 45 XX - 58 XX - 41 XX - 40
                              set EXAM at location
                              2000 H
XA XX - 31                  select unit 1
XB XX - 00 XX - 10  create 10 sectors
XD XX - 00 XX - 04  length of EXAM
XH XX - 20 XX - 00  points to EXAM
                              (file name)

G501                          execute
XA                            fetch error code
```

11. SPECIAL RAM LOCATIONS

11.1 Branch VECTOR

To change from cassette to disk operation or back, a series of branch locations in RAM must be changed.

For example this allows programs to be loaded from cassette and saved on disk.

Address in RAM (HEX)	Casette Contents (HEX)	Disk Contents (HEX)	Routine
2C5	C3 B8 D2	C3 A7 06	Write open (open disk file for output)
2C8	C3 F1 D2	C3 D7 06	Write block (to an opened disk)
2CB	C3 27 D4	C3 51 07	Write close (close and output file)
2CE	C3 25 D3	C3 65 08	Read open (open a file for input)
2D1	C3 40 D3	C3 79 08	Read block from disk
2D4	C3 45 D4	C3 63 09	Read close (close current input file)
2D7	C3 A2 D3	C3 A2 D3	Match block
2DA	C9 00 00	C9 00 00	Reset
2DD	C9 00 00	C3 CF 0A	Character out
2E0	C3 B4 DD	C3 46 17	Character in
2E3	C9 00 00	C9 00 00	Spare

Example

To load programs from cassettes to a disk based computer change the branches at locations 2CE, 2D1 and 2D4.

```
* UT

- s2CE   C3 - C3  65 - 25  08 - D3

- s2D1   C3 - C3  79 - 40  08 - D3

- s2D4   C3 - C3  63 - 45  09 - D4

- B

* LOAD "EXAM"    : REM Load the file EXAM.BAS from cassette

* SAVE "EXAM"    : REM Save the file EXAM. BAS on diskette 0
```

11.2 Print Output

Because the disk controller recognises commands in ASCII code the print statement may be used to issue some commands.

Issue a POKE to location # 131 as below to direct output for various purposes.

Code	Result Output
0	Screen + RS232 (serial)
1	Screen only
2	Edit buffer
3	Disk (DOUTC)

Example : see section **5.1**

11.3 INPUT

Issue a poke to location #135 as below to read data from device.

Code	Result Input
0	Keyboard
1	RS232
2	EDIT BUFFER

13. DOS ERRORS

Any errors encountered by master or slave DOS when attempting to interpret or execute a DOS command will normally be signalled by error messages on the console device.

They contain an identifying error code, usually followed by an explanatory error message. After the error indication, DOS will send the prompt character '*' to the console and wait for the next command.

Error messages generated during DOS command identification, parameter recognition or subsequent execution will contain an error code which identifies the cause of the error.

This error code can be used to deduce the corrective action to be taken before the command can be reattemped.

Read/write errors detected during disk access while executing certain DOS commands (such as VERIFY DISK n) will produce error messages containing an identification code.

After the first word in the command has been correctly identified, BASIC will continue to read the following characters as necessary to find the parameters associated with that command. Any error discovered at this stage will simply produce the error message "SYNTAX ERROR IN DOS COMMAND". (no action has yet been taken by BASIC). For example, if the user attemps to list a diskette directory via the command DIR, but fails to specify which diskunit, a Syntax error will be produced as follows :

```
*DIR                    (instead of DIR 0 or DIR 1)
SYNTAX ERROR IN DOS COMMAND
*
```

Once all the required parameters following the command word have been correctly read, BASIC will initiate execution of the specified task. Any characters found between the space or comma which follows the last parameter required by a DOS command, and the carriage-return which terminates the command, will be ignored by master DOS.

This feature can be used to include comments within a DOS command, without causing any syntax errors.

13.1 DOS Error Codes and Messages

Master and slave DOS contain a large number of resident routines which perform the necessary tasks during the execution of the DOS commands.

At the end of its task, a routine wil return a binary error code (in the Accumulator) to master DOS, or to the calling program, which indicates whether or not any errors were detected during the execution of that task. If no error was detected, a code of 00 will be returned. Any non-zero error code indicates the occurence of an error, and its value identifies the type of error. This error code will then be sent by master DOS to an error analysis routine, which will display a suitable error message on the console.

If the master DOS commands are invoked by a user program via calls to the relevant DOS routines, the returned error codes have to be interpreted by that program.

If master DOS is not used at all, and communication with the slave DOS is done by a user program, only the error codes 01H to 11H, and 81H to 8AH, from slave DOS will be received by that program.

Such a program has to include an error analysis routine for dealing with those errors.

13.1.1 Command Syntax Errors

During the command identification stage, BASIC reads the first word typed by the Operator following the prompt character, to determine what action has to be performed.

This command word should correspond to one of the standard BASIC or DOS commands. If it does not match one of the command words, an error message "SYNTAX ERROR" will be produced.

13.1.2 Error Code Generation

After basic has correctly read a command word and the necessary parameters via the console input channel, it will execute the command or transmit the data which is necessary for a DOS command to master DOS.

Master DOS will then transmit the command to slave DOS via one or mor File Control Blocks (FCBs).

Any errors detected by master DOS during the command identification or parameter recognition stage will generate an error code in the range 21H to 24H.

If master DOS is not used at all, and communication with slave DOS is done by a user program, the above error codes will not be encountered. In such case, the FCBs for communicating with slave DOS will have to be generated by the user program.

If slave DOS receives an FCB with errors, or if the operation specified in the FCBs is invalid, it will send back an identifying error code in the range 01H to 10H. When the FCBs are created by master DOS they should be error free, unless master DOS has been corrupted. Such errors can usually be corrected by reloading master DOS from a good system diskette.

The slave DOS uses the information contained in the FCBs received via the DCE-BUS, to drive the hardware on the diskette controller card for executing the specified task.

If any hardware errors are detected during the command execution, slave DOS will send back an identifying error code in the range 81H to 8AH.

13.1.3 Error Codes Generated by Slave DOS

A list of error codes from slave DOS is given below.

These can be due to errors detected in the FCBs received by slave DOS, or due to invalid operations specified in them. Some of these error codes are given in hexadecimal.

Error Code	Error Message	Cause of Error
01	-	Invalid command code specified in the FCB received by slave DOS.
02	-	Incomplete or erroneous FCB received by slave DOS
03	Diskunit can only be 0 or 1	An invalid number has been specified for the diskunit. The only valid values are 0 and 1 (in ASCII).
04	File to be created already exists	Attempt to create a file which already exists on that diskette
05	File too long for remaining disk space	Attempt to create a file longer than the space available on that diskette
06	-	Invalid access mode specified in teh FCB received by slave DOS. Valid access modes are I (for input) and O (for output), in ASCII
07	File is write-protected use recover	Attempt to delete, rename or write a file which has been given an attribute 'W' via the PROTECT command
08	File not found check spelling of filename	Attempt to open a file which is not entered in the diskette directory. This error can also be caused by incorrect DOS command names
09	File is not opened for access	Attempt to read from or write to a file which has not been opened in the corresponding I/O mode
0A	Directory is full	Diskette directory full. Occurs when a diskette directory contains 108 file entries (including directory file), and an attempt is made to create a new file on it.
0B	-	FCB received by slave DOS too long. Occurs when too many parameter bytes are sent to slave DOS. The maximum number of parameter bytes that can be sent after the command code is 23. (If the FCB contains more characters that what is required by the command code, they will be ignored by slave DOS)

0C	-	No paramaters in the FCB received by slave DOS. Indicates that only the command code has been sent without the necessary paramaters
0D	File was already open Pointer is not reset	Attempt to open a file is already open in that I/O mode. Can be caused by DOS commands which open parameter files automatically
0E	End of file	Occurs if the physical end-of-file is detected when attempting to read from or write to a file. Input files with a software end-of-file marker (Control-Z) will not produce this error
10H		Occurs when attempting to open a new file when the Active File Table (AFT) is full due to five files already being open. No more than five diskette files can be open at any time
11H		Occurs when attempting to read from or write to a diskette file in direct access mode, with a record number higher than the file length

13.1.4 Error Codes Generated by Master DOS

A list of error codes generated by master DOS is given below.

If master DOS is not used at all, and communication with slave DOS is done by a user program, these error codes will not be encountered. The error codes are given in hexadecimal.

Error Code	Error Message	Cause of Error
21	SYNTAX ERROR IN DOS COMMAND	Indicates a syntax error in the command input (e.g. missing paramaters, non-hexadecimal address parameter etc.). The error code will not be included in the message and the currently open files will not be indicated.
22	-	Invalid peripheral assignment. Occurs when attempting to assign a logical channel to a physical peripheral of the incorrect I/O type, or when specifying invalid channel or peripheral codes, via the DOS command ASSIGN.
(23)	(not an error)	This error code provides a warning that no extension has been specified in a filereference. It is ingored by the error analysis routine in master DOS, and causes no error.

13.1.5 Hardware Error Codes Sent Via Slave DOS

The slave DOS uses the information contained in the FCBs to drive the hardware on the diskette controller card for executing the specified task.

If an error is detected during such an operation, slave DOS will send one of the error codes given below. All these errors codes start with the hexadecimal digit 8.

Error Code	Error Message	Cause of Error
81	-	Invalid disk command
82	-	Invalid disk paramater
83	-	Select fault
84	SECTOR NOT FOUND	Failed to find sector. Can be due to an incorrectly inserted, corrupted or uninitialized diskette
85	FOUND WRONG TRACK RELOAD DISK AND TRY AGAIN	Failed to find track
86	CRC ERROR NOTICED IN IO	CRC error in sector identification block. Can be due to a corrupted diskette
87	CRC ERROR NOTICED DURING DATA READ	CRC error in data. Usually due to disk corruption or damage
88	ILLEGAL DATA MARK	Illegal data mark
89	DATA MARK DELETED	Deleted datamark
8A	DISK ERROR DURING WRITING	Write fault

Except from the above hardware errors, codes 84 and 87 can be caused by a diskette being incorrectly inserted into the drive, or the data being on it being corrupted due to a scratch on its surface etc. All of the errors except 84 and 87 are very rare. If they occur, the following corrective action is recommended before trying the command again.

* Check if diskette is correctly inserted into the drive

* Ensure that the diskette has already been formatted (via the IDISK command)

* Replace the diskette producing the error

* Reload DOS from a good system diskette.

13.2 LOADING ERRORS (BASIC)

Loading errors may be detected when accessing diskette files via the BASIC commands LOAD, LOADA or the Utility command R.

LOADING ERROR 0 : File not found.

1 : Block too large (i.e. not enough memory). Data will be loaded till the end of the available memory block.
For example, LOAD allocates from top of heap to bottom of screen area for loading programs

2 : Bad syntax, or invalid file type

3 : Any other disk error (PEEK ## "erradr" will give the errorcode)

In all cases, the actual DOS error code will be stored at the memory address "erradr" (to be specified later), which can be read and processed.

13.3 OTHER ERRORS

All other DOS errors will be displayed on the TV screen as :

+++ DISK ERROR nn +++

where 'nn' is the 2-digit hexadecimal error code specified before.

13.4 ERROR HANDLING

Any DOS error can be handled (for example, within a BASIC program) in the following way :

ERROR % = PEEK ## A32

The result will be zero (if no error), or it will give the detected error code.

Example :

```
140   ERROR%=PEEK(##A32):REM ERROR BYTE
150   POKE ##A32,0:REM SET ERROR BYTE TO 0
160   IF ERROR%=14  THEN 210
170   IF ERROR%=19  THEN 210
180   IF KEY$(Y%-1)="UUUUUUUUUUUU" THEN 210
190   IF ERROR%=0  THEN 110
200   PRINT "ERROR NR.  ";ERROR%;:END
```

14. STANDARD SYSTEM DISKETTE : DSK-PCD V2.0 (+ AZERTY Format)

This diskette provides 4 system programs to the user.

1) $DKBOOTS Bootstrap routine. See power-up details

2) $MSTRDOS DAI Computer DOS V 2.0

3) $USER.BAS DAI "AUTO-MENU" program executed on power-up

4) $AZERTY.BIN Modified Keyboard map to transform the keyboard into an AZERTY (European) format. Must be renamedUSER.BIN before use. The keycaps must be re-arranged into the following layout :

Keyboard Layout