UMEÅ UNIVERSITET
Fysiska institutionen
Sverker Johansson

1984-06-14

HÅKAN JÖELSON
871130

Elektronik

MIKRODATORTEKNIK

Häfte 10: The FLEX Operating System

FLEX 2.0 (6800)
FLEX 9.0 (6809)

INNEHÅLL

		sid
10.1	Introduction	10-1
10.2	System requirements	10-2
10.3	Getting the system started	10-2
10.4	Disk files and their names	10-3
10.5	Entering commands	10-5
10.6	Command descriptions	10-7
	10.6.1 Memory resident commands	10-7
	10.6.2 Utility command set	10-8
10.7	General system information	10-30
10.8	System and memory map	10-32
10.9	User-callable system I/O-subroutines	10-33
10.10	Command and subroutine summary	10-36

L84130

10 The FLEX OPERATING SYSTEM

10.1 INTRODUCTION

The FLEX™ Operating System is a very versatile and flexible operating system. It provides the user with a powerful set of system commands to control all disk operations directly from the user's terminal. The systems programmer will be delighted with the wide variety of disk access and file management routines available for personal use. Overall, FLEX is one of the most powerful operating systems available today.

The FLEX Operating System is comprised of three parts, the File Management System (FMS), the Disk Operating System (DOS), and the Utility Command Set (UCS). Part of the power of the overall system lies in the fact that the system can be greatly expanded by simply adding additional utility commands. The user should expect to see many more utilities available for FLEX in the future. Some of the other important features include: fully dynamic file space allocation, the automatic "removal" of defective sectors from the disk, automatic space compression and expansion on all text files, complete user environment control using the TTYSET utility command, and uniform disk wear due to the high performance dynamic space allocator.

The UCS currently contains many very useful commands. These programs reside on the system disk and are only loaded into memory when needed. This means that the set of commands can be easily extended at any time, without the necessity of replacing the entire operating system. The utilities provided with FLEX perform such tasks as the saving, loading, copying, renaming, deleting, appending, and listing of disk files. There is an extensive CATalog command for examining the disk's file directory. Several environment control commands are also provided. Overall, FLEX provides all of the necessary tools for the user's interaction with the disk.

---

* FLEX is a registered trademark of Technical Systems Consultants, Inc.

vk. 871201 ti 2

Repet fr igår 1 se häfte 17

FLEX operativsystem

3 delar

RMS File Management System
sköter filhantering

DOS Disk Operating System

UCS Utility Command Set

CAT listar katalogen

DIR sa, c mer uppg.r

COPY

DELETE

LIST

NEWDISK 2

PO

RENAME

Systemsubrutiner

s 10-32 .. 35

Lista s 10-36

10.2 SYSTEM REQUIREMENTS

FLEX 2.0 (6800) requires random access memory, RAM, from location $0000 through location $2FFF (12k). RAM is also required from $A000 through $BFFF, where the actual operating system resides.

If FLEX 9.0 (6809) is used, all addresses in this manual are valid if $AXXX is changed to $CXXX and $BXXX is changed to $DXXX.

The systems also assumes at least 2 disk drives are connected to the controller and that they are configured as drives #0 and #1. FLEX interfaces with the disk controller through a set of driver routines and with the operator console or terminal through a set of terminal I/O routines.

10.3 GETTING THE SYSTEM STARTED

Each FLEX system diskette contains a binary loader for loading the operating system into RAM. There needs to be some way of getting the loader off of the disk so it can do its work. This can be done by either ~~hand~~ entering the bootstrap loader provided with the disk system, or by using the boot provided in ROM if appropriate to FLEX.

System Ericsson USE 6800/6809 has a EPROM based bootstrap called EBUG (SBUG). A "boot" program on the 'system' drive (drive #0) is loaded into RAM by a 'E'-command of the EBUG-monitor. The boot-program, once loaded, then loads FLEX from the disk.

If you wish to run FLEX, make the following steps:

1 Power on the video terminal (and the printer, if any) and make sure the EBUG (SBUG) promt (*) is present on the terminal. (>)

2 Be sure the doors on both drives are open and power on the Ericsson-system.

3 Insert the system diskette into drive #0 and the working diskette into drive #1. Close the doors on the drives.

4 Type 'E' on the terminal (↵)

After a few seconds the following should be displayed on the terminal:

```
FLEX X.X
DATE (MM,DD,YY)?
```

The name FLEX identifies the operating system and the X.X will be the version number of the operating system. At this time the current date should be entered, such as 14,6,84. The FLEX prompt is the three plus signs (+++), and will always be present when the system is ready to accept an operator command. The '+++' should become a familiar sight and signifies that FLEX is ready to work for you!

871205 fre [4]

FLEX OPERATIVSYSTEM

FMS
DOS (s 10-1)
LCS

SBUG	$F000
I/O	$E000
FLEX	
	$C000

SBUG h en minnesarea S krockar c DiskDriven när man trycker RESET

Disk Drivers	$DE00 - $DFFF
FMS	$D400
DOS	$CC00
System-filer	$C980
System FCB × 320kg	$C840
Scheduler & Printer Spooler	$C700
Utility Command Area	$C100
Input Buffer	$C080
[System] Stack	$C000
Användar-RAM	$FFFF

ex CAT, DIR, ...

× Skapa vid sth av inläsn av filer t RAM. ~~Bufferminnesområden~~ En sektor i taget (+ lite fil.info) läggs här [i bokstl.] på väg t RAM.

Memory Resident Commands

MON → man går tbaka t monitorn (i vårt fall SBUG)

GET hämtar binärfil

Disk Resident Commands

ligger på systemdisken & h extension .CMD

Samlade i LCS

= [vanliga] program s man ropar in & startar

Genomgång av kommando-listan s 10-39

GET hämta in fil

JUMP starta inhämtat progr (om man vet startadr) bl.a.

LIST

FLEX

SE

ASM B

10.4 DISK FILES AND THEIR NAMES

All disk files are stored in the form of 'sectors' on the disk and in this version, each sector contains 256 'bytes' of information. Each byte can contain one character of text or one byte of binary machine information. A maximum of 340(*) user-accessible sectors will fit on a single-sided mini disk or 1140 sectors on a single-sided full size floppy. Double-sided disks would hold exactly twice that number of sectors. Double-density systems will hold more still. The user, however, need not keep count, for the system does this automatically. A file will always be at least one sector long and can have as many as the maximum number of sectors on the disk. The user should not be concerned with the actual placement of the files on the disk since this is done by the operating system. File deletion is also supported and all previously used sectors become immediately available again after a file has been deleted.

All files on the disk have a name. Names such as the following are typical:

```
PAYROLL
INVNTORY
TEST1234
APRIL-78
WKLY-PAY
```

Anytime a file is created, referenced, or deleted, its name must be used. Names can be most anything but must begin with a letter (not numbers or symbols) and be followed by at most 7 additional characters, called 'name characters'. These 'name characters' can be any combination of the letters 'A' through 'Z' or 'a' through 'z', any digit '0' through '9', or one of the two special characters, the hyphen (-) or the underscore '_'.

File names must also contain an 'extension'. The file extension further defines the file and usually indicates the type of information contained therein. Examples of extensions are: TXT for text type files, BIN for machine readable binary encoded files, CMD for utility command files, and BAS for BASIC source programs. Extensions may contain up to 3 'name characters' with the first character being a letter. Most of the FLEX commands assume a default extension on the file name and the user need not be concerned with the actual extension on the file. The user may at anytime assign new extensions, overiding the default value, and treat the extension as just part of the file name. Some examples of file names with their extensions follow:

```
DELETE.CMD
LEDGER.BAS
TEST.BIN
```

Note that the extension is always separated from the name by a period '.'. The period is the name 'field separator'. It tells FLEX to treat the following characters as a new field in the name specification.

(*) 340 user-accessible sectors in the ADA-system
790 user-accessible sectors in the Ericsson-system.

Filangivelse

[<drive>.]<namn>[.<extension>]

+++ Ø.LIST.CMD

+++ LIST, 1.ADD.TXT

W <CR> händer följ:
tx strängen ovan går in i
INPUT BUFFER

Hämtar Ø.LIST.CMD
laddar in d i RAM
startar d
Nu h LIST kontrollen ö MPU
LIST hämtar 1.ADD.TXT
ska ut på skärmen
[redacted]
LIST lämnar [redacted] tbaka kontrollen
+ FLEX.

A file name can be further refined. The name and extension uniquely define a file on a particular drive, but the same name may exist on several drives simultaneously. To designate a particular drive a 'drive number' is added to the file specification. It consists of a single digit (0-1) and is separated from the name by the field separator '.'. The drive number may appear either before the name or after it (after the extension if it is given). If the drive is not specified, the system will default to either the 'system' drive or the 'working' drive. These terms will be described a little later.

Some examples of file specifications with drive numbers follow:

```
0.BASIC
MONDAY.1
1.TEST.BIN
LIST.CMD.1
```

In summary, a file specification may contain up to three fields separated by the field separator. These fields are; 'drive', 'name', and 'extension'. The rules for the file specification can be stated quite concisely using the following notation:

```
[<drive>.]<name>[.<extension>]
  or
<name>[.<extension>][.<drive>]
```

The '<>' enclose a field and do not actually appear in the specification, and the '[]' surround optional items of the specification. The following are all syntactically correct:

```
0.NAME.EXT
NAME.EXT.0
NAME.EXT
0.NAME
NAME.0
NAME
```

Note that the only required field is the actual 'name' itself and the other values will usually default to predetermined values. Studying the above examples will clarify the notation used. The same notation will occur regularly throughout the manual.

10.5 ENTERING COMMANDS

When FLEX is displaying '+++', the system is ready to accept a command line. A command line is usually a name followed by certain parameters depending on the command being executed. There is no 'RUN' command in FLEX. The first file name on a command line is always loaded into memory and execution is attempted. If no extension is given with the file name, 'CMD' is the default. If an extension is specified, the one entered is the one used. Some examples of commands and how they would look on the terminal follow:

```
+++TTYSET
+++TTYSET.CMD
+++LOOKUP.BIN
```

The first two lines are identical to FLEX since the first would default to an extension of CMD. The third line would load the binary file 'LOOKUP.BIN' into memory and, assuming the file contained a transfer address, the program would be executed. A transfer address tells the program loader where to start the program executing after it has been loaded. If you try to load and execute a program in the above manner and no transfer address is present, the message, 'NO LINK' will be output to the terminal, where 'link' refers to the transfer address. Some other error messages which can occur are 'WHAT?' if an illegal file specification has been typed as the first part of a command line, and 'NOT THERE' if the file typed does not exist on the disk.

During the typing of a command line, the system simply accepts all characters until a 'RETURN' key is typed. Any time before typing the RETURN key, the user may use one of two special characters to correct any mistyped characters. One of these characters is the 'back space' and allows deletion of the previously typed character. Typing two back spaces will delete the previous two characters. (The back space is initially defined to be a 'control H' but may be redefined by the user using the TTYSET utility command.) The second special character is the line 'delete' character. Typing this character will effectively delete all of the characters which have been typed on the current line. A new prompt will be output to the terminal, but instead of the usual '+++' prompt, to show the action of the delete character, the prompt will be '???'. Any time the delete character is used, the new prompt will be '???', and signifies that the last line typed did not get entered into the computer. (The delete character is initially a 'control X' but may also be redefined using TTYSET.)

As mentioned earlier, the first name on a command line is always interpreted as a command. Following the command is an optional list of names and parameters, depending on the particular command being entered. The fields of a command line must be separated by either a space or a comma. The general format of a command line is:

```
<command>[,<list of names and parameters>]
```

A comma is shown, but a space may be used. FLEX also allows several commands to be entered on one command line by use of the 'end of line' character. (This character is initially a colon (':'), but may be user defined with the TTYSET utility.) By ending a command with the end of line character, it is possible to follow it immediately with another command. FLEX will execute all commands on the line before returning with the '+++' prompt. An error in any of the command entries will cause the system to terminate operation of that command line and return with the prompt. Some examples of valid command lines follow:

```
+++CAT 1
+++CAT 1:ASN S=1
+++LIST LIBRARY:CAT 1:CAT 0
```

As many commands may be typed in one command line as desired, but the total number of characters typed must not exceed 128. Any excess characters will be ignored by FLEX.

One last system feature to be described is the idea of 'system' and 'working' drives. As stated earlier, if a file specification does not specifically designate a drive number, it will assume a default value. This default value will either be the current 'system' drive assignment or the current 'working' drive assignment. The system drive is the default for all command names, or in other words, all file names which are typed first on a command line. Any other file name on the command line will default to the working drive. This version of FLEX also supports automatic drive searching. When in the auto search mode if no drive numbers are specified, the operating system will first search drive 0 for the file. If the file is not found, drive 1 will be searched and so on. When the system is initialized (booted up), all drive defaults will be to drive 0. It is sometimes convenient to assign drive 1 as the working drive in which case all file references, except commands, will automatically look on drive 1. It is then convenient to have a diskette in drive 0 with all the system utility commands on it (the 'system drive'), and a disk with the files being worked on in drive 1 (the 'working drive'). If the system drive is 0 and the working drive is 1, and the command line was:

```
+++LIST TEXTFILE
```

FLEX would go to drive 0 for the command LIST and to drive 1 for the file TEXTFILE. The actual assignment of drives is performed by the ASN utility. See its description for details. (Page 10-9)

10.6 COMMAND DESCRIPTIONS

There are two types of commands in FLEX, memory resident (those which actually are part of the operating system) and disk utility commands (those commands which reside on the disk and are part of the UCS). There are only two resident commands, GET and MON. They will be described on this page while the UCS is described in the following section.

10.6.1 MEMORY RESIDENT COMMANDS

GET

The GET command is used to load a binary file into memory. It is a special purpose command and has the following syntax:

```
GET[,<file name list>]

where <file name list> is: <file spec>[,<file spec>] etc.
```

Again the '[]' surround optional items. 'File spec' denotes a file name as described earlier. The action of the GET command is to load the file or files specified in the list into memory for later use. If no extension is provided in the file spec, BIN is assumed, in other words, BIN is the default extension. Examples:

```
GET,TEST
GET,1.TEST,TEST2.0
```

where the first example will load the file named 'TEST.BIN' from the assigned working drive, and the second example will load TEST.BIN from drive 1 and TEST2.BIN from drive 0.

MON

MON is used to exit FLEX and return to the hardware monitor system such as EBUG. The syntax for this command is simply MON followed by the 'RETURN' key. The EBUG-prompt '*' will then be present on the terminal.

NOTE: to re-enter FLEX after using the MON command, you should jump to location AD03 hex. (Type: G AD03)

10.6.2 UTILITY COMMAND SET

The following pages describe the utility commands currently included in the UCS (Utility Command Set).

COMMON ERROR MESSAGES

Several error messages are common to many of the FLEX utility commands. These error messages and their meanings include the following:

NO SUCH FILE. This message indicates that a file referenced in a particular command was not found on the disk specified. Usually the wrong drive was specified (or defaulted), or a misspelling of the name was made.

ILLEGAL FILE NAME. This can happen if the name or extension did not start with a letter, or the name or extension field was too long (limited to 8 and 3 respectively). This message may also mean that the command being executed expected a file name to follow and one was not provided.

FILE EXISTS. This message will be output if you try to create a file with a name the same as one which currently exists on the same disk. Two different files with the same name are not allowed to exist on the same disk.

SYNTAX ERROR. This means that the command line just typed does not follow the rules stated for the particular command used. Refer to the individual command descriptions for syntax rules.

GENERAL SYSTEM FEATURES

Any time one of the utility commands is sending output to the terminal, it may be temporarily halted by typing the 'escape' character (see TTYSET for the definition of this character). Once the output is stopped, the user has two choices: typing the 'escape' character again or typing 'RETURN'. If the 'escape' character is typed again, the output will resume. If the 'RETURN' is typed, control will return to FLEX and the command will be terminated. All other characters are ignored while output is stopped.

ASN

The ASN command is used for 1) assigning the 'system' drive and the 'working' drive or to 2) select automatic drive searching. The system drive is used by FLEX as the default for command names or, in general, the first name on a command line. The working drive is used by FLEX as the default on all other file specifications within a command line. Upon initialization, FLEX assigns drive #0 as both the system and working drive. An example will show how the system defaults to these values:

```
COPY,FILE1,FILE2
```

If the system is assigned to be #0 and the working drive is assigned to drive #1, the above example will perform the following operation: get the COPY command from drive #0 (the system drive), then copy FILE1 from drive #1 (the working drive) to FILE2 on drive #1. As can be seen, the system drive was the default for COPY where the working drive was the default for all other file specs listed.

Automatic drive searching causes FLEX to automatically scan the ready drives for the file specified. Hardware limitations prevent most mini-floppy versions from searching for "ready" drives. For this reason, FLEX has been setup to ALWAYS assume drives 0 and 1 are ready. Thus if a mini-floppy version of FLEX attempts to search a drive which does not have a disk inserted, it will hang up until a disk is inserted and the door closed. Alternatively, the system reset could be hit and a warm start executed (a jump to $AD03). The full size floppy version CAN detect a ready condition and will not check drives which are out of the ready state during automatic drive searching. S 10-30-D

Automatic drive searching causes FLEX to first check drive #0 for the file specified. If not there (or if not ready in the full size versions), FLEX skips to drive #1. If the file is not found on drive #1 in the mini-floppy version, FLEX gives up and a file not found error results. In the full size versions, FLEX continues to search on drives #2 and #3 before reporting an error.

DESCRIPTION

The general syntax for the ASN command is as follows:

```
ASN[,W=<drive>][,S=<drive>]
```

where <drive> is a single digit drive number or the letter A. If just ASN is typed followed by a 'RETURN', no values will be changed, but the system will output a message which tells the current assignments of the system and working drives, for example:

```
+++ASN
SYSTEM DRIVE IS #0
WORK DRIVE IS #1
```

(Turn over)

ASN (continued)

Some examples of using the ASN command are:

```
ASN,W=1
ASN,S=1,W=0
```

where the first line would set the working drive to 1 and leave the system drive assigned to its previous value. The second example sets the system drive to 1 and the working drive to 0. Careful use of drive assignments can allow the operator to avoid the use of drive numbers on file specifications most of the time!

If auto drive searching is desired, then the letter A for automatic, should be used in place of the drive number.

```
Example:
ASN W=A
ASN S=A, W=1
ASN S=A, W=A
```

Skabslösa 87120

CAT

The CATalog command is used to display the FLEX disk file names in the directory on each disk. The user may display selected files on one or multiple drives if desired.

DESCRIPTION

The general syntax of the CAT command is:

```
CAT[,<drive list>][,<match list>]
```

where <drive list> can be one or more drive numbers seperated by commas, and <match list> is a set of name and extension characters to be matched against names in the directory. For example, if only file names which started with the characters 'VE' were to be cataloged, then VE would be in the match list. If only files whose extensions were 'TXT' were to be cataloged, then .TXT should appear in the match list. A few specific examples will help clarify the syntax:

```
+++CAT
+++CAT,1,A.T,DR
+++CAT,PR
+++CAT,0,1
+++CAT,0,1,.CMD,.SYS
```

The first example will catalog all file names on the working drive or on all drives if auto drive searching is selected. The second example will catalog only those files on drive 1 whose names begin with 'A' and whose extensions begin with 'T', and also all files on drive 1 whose names start with 'DR'. The next example will catalog all files on the working drive (or on all drive if auto drive searching is selected) whose names start with 'PR'. The next line causes all files on both drive 0 and drive 1 to be cataloged. Finally, the last example will catalog the files on drive 0 and 1 whose extensions are CMD or SYS.

During the catalog operation, before each drive's files are displayed, a header message stating the drive number is output to the terminal. The actual directory entries are listed in the following form:

```
NAME.EXTENSION    SIZE PROTECTION CODE
```

where size is the number of sectors that file occupies on the disk. If more than one set of matching characters was specified on the command line, each set of names will be grouped according to the characters they match. For example, if all .TXT and .CMD files were cataloged, the TXT types would be listed together, followed by the CMD types.

In summary, if the CAT command is not parameterized, then all files on the assigned working drive will be displayed. If a working drive is not assigned (auto drive searching mode) the CAT command will display files

(Turn over)

CAT (continued)

on all on line drives. If it is parameterized by only a drive number, then all files on that drive will be displayed. If the CAT command is parameterized by only an extension, then only files with that extension will be displayed. If only the name is used, then only files which start with that name will be displayed. If the CAT command is parameterized by only name and extension, then only files of that root name and root extension (on the working drive) will be displayed. Learn to use the CAT command and all of its features and your work with the disk will become a little easier.

The current protection code options that can be displayed are as follows:

D	File is delete protected (delete or rename prohibited)
W	File is write protected (delete, rename and write prohibited)
(blank)	No special protection

COPY

The COPY command is used for making copies of files on a disk. 1) Individual files may be copied, 2) groups of name-similar files may be copied, or 3) entire disks may be copied. The COPY command also re-groups the sectors of a file in case they were spread all over the old disk. This regrouping can make file access times much faster. It should be noted that before copying files to a new disk, the disk must be formatted first.

DESCRIPTION

The general syntax of the COPY command has three forms:

a. COPY,<file spec>,<file spec>
b. COPY,<file spec>,<drive>
c. COPY,<drive>,<drive>[,<match list>]

where <match list> is the same as that described in the CAT command and all rules apply to matching names and extensions. When copying files, if the destination disk already contains a file with the same name as the one being copied, the file name and the message, "FILE EXISTS DELETE ORIGINAL?" will be output to the terminal. Typing Y will cause the file on the destination disk to be deleted and the file from the source disk will be copied to the destination disk. Typing N will direct FLEX not to copy the file in question.

The first type of COPY allows copying a single file into another. The output file may be on a different drive but if on the same drive the file names must be different. It is always necessary to specify the extension of the input file but the output file's extension will default to that of the input's if none is specified. An example of this form of COPY is:

```
+++COPY,0.TEST.TXT,1.TEST25
```

This command line would cause the file TEST.TXT on drive 0 to be copied into a file called TEST25.TXT on drive 1. Note how the second file's extension defaulted to TXT, the extension of the input file.

The second type of COPY allows copying a file from one drive to another drive with the file keeping its original name. An example of this is:

```
+++COPY,0.LIST.CMD,1
```

Here the file named LIST.CMD on drive 0 would be copied to drive 1. It is again necessary to specify the file's extension in the file specification. This form of the command is more convenient than the previous form if the file is to retain its original name after the copying process.

(Turn over)

COPY (Continued)

The final form of COPY is the most versatile and the most powerful. It is possible to copy all files from one drive to another, or to copy only those files which match the match list characters given. Some examples will clarify its use:

```
+++COPY,0,1
+++COPY,1,0,.CMD,.SYS
+++COPY,0,1,A,B,CA.T
```

The first example will copy all files from drive 0 to drive 1 keeping the same names in the process. The second example will copy only those files on drive 1 whose extensions are CMD and SYS to drive 0. No other files will be copied. The last example will copy the files from drive 0 whose names start with 'A' or 'B' regardless of extension, and those files whose names start with the letters 'CA' and whose extensions start with 'T'.,to the output drive which is drive 1. The last form of copy is the most versatile because it will allow putting just the command (CMD) files on a new disk, or just the SYS files, etc., with a single command entry. During the COPY process, the name of the file which is currently being copied will be output to the terminal, as well as the drive to which it is being copied.

DATE

The DATE command is used to display or change an internal FLEX date register. This date register may be used by future programs and FLEX utilities.

DESCRIPTION

The general syntax of the DATE command is:

```
DATE[,<month,day,year>]
```

where 'month' is the numerical month, 'day' is the numerical day and 'year' is the last two digits of the year.

```
+++DATE 5,2,78  Sets the date register to May 2, 1978
```

Typing DATE followed by a carriage return will return the last entered date.

Example:

```
+++DATE
May 2, 1978
```

DELETE

The DELETE command is used to delete a file from the disk. Its name will be removed from the directory and its sector space will be returned to the free space on the disk.

DESCRIPTION

The general syntax of the DELETE command is:

```
DELETE,<file spec>[,<file list>]
```

where <file list> can be an optional list of file specifications. It is necessary to include the extension on each file specified. As the DELETE command is executing it will prompt you with:

```
DELETE "FILE NAME"?
```

The entire file specification will be displayed, including the drive number. If you decide the file should be deleted, type 'Y'; otherwise, any other response will cause that file to remain on the disk. If a 'Y' was typed, the message 'ARE YOU SURE?' will be displayed on the terminal. If you are absolutely sure you want the file deleted from the disk, type another 'Y' and it will be gone. Any other character will leave the file intact. ONCE A FILE HAS BEEN DELETED, THERE IS NO WAY TO GET IT BACK! Be absolutely sure you have the right file before answering the prompt questions with Y's. Once the file is deleted, the space it had occupied on the disk is returned back to the list of free space for future use by other files. Few examples follow:

```
+++DELETE,MATHPACK.BIN
+++DELETE,1.TEST.TXT,0.AUGUST.TXT
```

The first example will DELETE the file named MATHPACK.BIN from the working drive. If auto drive searching is selected, the file will be deleted from the first drive it is found on. The second line will DELETE the file TEST.TXT from drive 1, and AUGUST.TXT from drive 0.

There are several restrictions on the DELETE command. First, a file that is delete or write protected may not be deleted without first removing the protection. Also a file which is currently in the print queue (waiting list) can not be deleted using the DELETE command.

LIST

The LIST command is used to LIST the contents of text or BASIC files on the terminal. It is often desirable to examine a files without having to use an editor or other such program. The LIST utility allows examining entire files, or selected lines of the file. Line numbers may also be optionally printed with each line.

DESCRIPTION

The general syntax of the LIST command is:

```
LIST,<file spec>[,<line range>][,+(options)]
```

where the <file spec> designates the file to be LISTed (with a default extension of TXT),and <line range> is the first and last line number of the file which you wish to be displayed. All lines are output if no range specification is given. The LIST command supports two additional options. If a +N option is given, line numbers will be displayed with the listed file. If a +P option is given, the output will be formatted in pages and LIST will prompt for "TITLE" at which time a title for the output may be entered. The TITLE may be up to 40 characters long. This feature is useful for obtaining output on a printer for documentation purposes (see P command). Each page will consist of the title, date, page number, 54 lines of output and a hex 0C formfeed character. Entering a +NP will select both options. A few examples will clarify the syntax used:

```
+++LIST,RECEIPTS
+++LIST,CHAPTER1,30-200,+NP
+++LIST,LETTER,100
```

The first example will list the file named 'RECEIPTS.TXT' without line numbers. All lines will be output unless the 'escape character' is used as described in the Utility Command Set introduction. The second example will LIST the 30th line through the 200th line of the file named 'CHAPTER1.TXT' on the terminal. The hyphen ('-') is required as the range number separator. Line numbering and page formatting will be output because of the '+NP' option. The last example shows a special feature of the range specification. If only one number is stated, it will be interpretted as the first line to be displayed. All lines following that line will also be LISTed. The last example will LIST the lines from line 100 to the end of the file. No line numbers will be output since the 'N' was omitted.

O

The O (not zero) command can be used to route all displayed output from a utility to an output file instead of the terminal. The function of O is similar to P (the printer command) except that output is stored in a file rather than being printed on the terminal or printer. Other SWTPC and TSC software may support this utility. Check the supplied software instructions for more details.

DESCRIPTION

The general syntax of the O command is:

```
O,<file spec>,<command>
```

where <command> can be any standard utility command line and <file spec> is the name of the desired output file. The default extension on <file spec> is .OUT. If O is used with multiple commands per line (using the 'end of line' character ':') it will only have affect on the command it immediately precedes. Some examples will clarify its use.

```
+++O,CAT,CAT
```
writes a listing of the current disk directory into a file called CAT.OUT

```
+++O,BAS,ASMB,BASIC.TXT
```
writes the assembled source listing of the text source file 'BASIC.TXT' into a file called 'BAS.OUT' when using the assembler

Vad går då g P, dir,1

P

P

P is the system print routine and will allow the output of any command to be routed to the printer. This is very useful for getting printed copies of the CATalog or used with the LIST command will allow the printing of FLEX text files. The P command simply switches an output flag such that any output which would normally have gone to the console will go to a printer. In order to know how to talk to the printer (it may be any type of printer), the P command reads a file from the disk called "PRINT.SYS" which should contain the necessary driver routines for the particular printer and configuration in use.

DESCRIPTION

The general syntax of the P command is:

```
P,<command>
```

where <command> can be any standard utility command line. If P is used with multiple commands per line (using the 'end of line' character), it will only have affect on the command it immediately preceeds. Some examples will clarify its use:

```
+++P,CAT
+++P,LIST,MONDAY:CAT,1
```

The first example would print a CATalog of the directory of the working drive on the printer. The second example will print a LISTing of the text file MONDAY.TXT and then display on the terminal a CATalog of drive 1 (this assumes the 'end of line' character is a ':'). Note how the P did not cause the 'CAT,1' to go to the printer. Consult the 'Advanced Programmer's Guide' for details concerning adaption of the P command to various printers through the writing of new printer driver routines (the PRINT.SYS file).

The P command tries to load a file named PRINT.SYS from the same disk which P itself was retrieved. The PRINT.SYS file which is supplied with the system diskette contains the necessary routines to operate a printer connected through a parallel Centronics type interface on PORT 7 of a SWTPc computer (PIA located at $801C). If you wish to use a different printer configuration, consult the 'Advanced Programmer's Guide' for details on writing your own printer driver routines to replace the PRINT.SYS file. The Centronics drivers, however, are compatible with many parallel interfaced printers presently on the market.

PROT

PROT

The PROT command is used to change a protection code associated with each file. When a file is first saved, it has no protection associated with it thereby allowing the user to write to, rename, or delete the file. Delete or write protection can be added to a file by using the PROT command.

DESCRIPTION

The general syntax of the PROT command is:

PROT,<file spec>[,(option list)]

where the <file spec> designates the file to be protected and (option list) is any combination of the following options.

D A 'D' will delete protect a file. A delete protected file cannot be affected by using the DELETE or RENAME Commands, or by the delete functions of SAVE,

W A 'W' will write protect a file. A write protected file cannot be deleted, renamed or have any additional information written to it. Therefore a write protected file is automatically delete protected as well.

C A 'C' will Catalog protect a file. Any files with a C protection code will function as before but will not be displayed when a CAT command is issued.

X An 'X' will remove all protection options on a specific file.

Examples:

+++PROT CAT.CMD,XW	Remove any previous protection on the CAT.CMD Utility and write protect it.
+++PROT CAT.CMD,X	Remove all protection from the CAT.CMD utility.
+++PROT INFO.SYS,C	Prohibit INFO.SYS from being displayed in a catalog listing.

RENAME

The RENAME command is used to give an existing file a new name in the directory. It is useful for changing the actual name as well as changing the extension type.

DESCRIPTION

The general syntax of the RENAME command is:

```
RENAME,<file spec 1>,<file spec 2>
```

where <file spec 1> is the name of the file you wish to RENAME and <file spec 2> is the new name you are assigning to it. The default extension for file spec 1 is TXT and the default drive is the working drive. If no extension is given on <file spec 2>, it defaults to that of <file spec 1>. No drive is requird on the second file name, and if one is given it is ignored. Some examples follow:

```
+++RENAME,TEST1.BIN,TEST2
+++RENAME,1.LETTER,REPLY
+++RENAME,0.FIND.BIN,FIND.CMD
```

The first example will RENAME TEST1.BIN to TEST2.BIN. The next example RENAMEs the file LETTER.TXT on drive 1 to REPLY.TXT. The last line would cause the file FIND.BIN on drive 0 to be renamed FIND.CMD. This is useful for making binary files created by an assembler into command files (changing the extension from BIN to CMD). If you try to give a file a name which already exists in the directory, the message:

```
FILE EXISTS
```

will be displayed on the terminal. Keep in mind that RENAME only changes the file's name and in no way changes the actual file's contents.

One last note of interest. Since utility commands are just like any other file, it is possible to rename them also. If you would prefer some of the command names to be shorter, or different all together, simply use RENAME and assign them the names you desire.

SAVE

SAVE

The SAVE command is used for saving a section of memory on the disk. Its primary use is for saving programs which have been loaded into memory from tape or by hand.

DESCRIPTION

The general syntax of the SAVE command is:

SAVE,<file spec>,<begin adr>,<end adr>[,<transfer adr>]

where <file spec> is the name to be assigned to the file. The default extension is BIN and the default drive is the working drive. The address fields define the beginning and ending addresses of the section of memory to be written on the disk. The addresses should be expressed as hex numbers. The optional <transfer address> would be included if the program is to be loaded and executed by FLEX. This address tells FLEX where execution should begin. Some examples will clarify the use of SAVE:

```
+++SAVE,DATA,100,1FF
+++SAVE,1.GAME,0,1680,100
```

The first line would SAVE the memory locations 100 to 1FF hex on the disk in a file called DATA.BIN. The file would be put on the working drive and no transfer address would be assigned. The second example would cause the contents of memory locations 0 through 1680 to be SAVEd on the disk in file GAME.BIN on drive 1. Since a transfer address of 100 was specified as a parameter, typing 'GAME.BIN' in response to the FLEX prompt after saving would cause the file to be loaded back into memory and execution started at location 100.

If an attempt is made to save a program under a file name that already exists, the prompt "MAY THE EXISTING FILE BE DELETED?" will be displayed. A Y response will replace the file with the new data to be saved while a N response will terminate the save operation.

Sometimes it is desirable to save noncontiguous segments of memory. To do this it would be necessary to first SAVE each segment as a separate file and then use the APPEND command to combine them into one file. If the final file is to have a transfer address, you should assign it to one of the segments as it is being saved. After the APPEND operation, the final file will retain that transfer address.

SAVE.LOW

SAVE.LOW

There is another form of the SAVE command resident in the UCS. It is called SAVE.LOW and loads in a lower section of memory than the standard SAVE command. Its use is for saving programs in the Utility Command Space where SAVE.CMD is loaded. Those interested in creating their own utility commands should consult the 'Advanced Programmer's Guide' for further details.

STARTUP

STARTUP is not a utility command but is a feature of FLEX. It is often desirable to have the operating system do some special action or actions upon initialization of the system (during the bootstrap loading process). As an example, the user may always want to use BASIC immediately following the boot process. STARTUP will allow for this without the necessity of calling the BASIC interpreter each time.

DESCRIPTION

FLEX always checks the disk's directory immediately following the system initialization for a file called STARTUP.TXT. If none is found, the three plus sign prompt is output and the system is ready to accept user's commands. If a STARTUP file is present, it is read and interpreted as a single command line and the appropriate actions are performed. As an example, suppose we wanted FLEX to execute BASIC each time the system was booted. First it is necessary to create the STARTUP file:

```
+++EDIT,STARTUP
NEW FILE:
    1.ØØ=BASIC
    2.ØØ=#
    1.ØØ=BASIC
    #S
    +++
```

The above procedure using the EDIT command will create the desired file. Note that the file consisted of one line (which is all FLEX reads from the STARTUP file anyway). This line will tell FLEX to load and execute BASIC. Now each time this disk is used to boot the operating system, BASIC will also be loaded and run. Note that this example assumes two things. First, the disk must contain DOS.SYS and must have been LINKed in order for the boot to work properly. Second, it is assumed that a file called BASIC.CMD actually exists on the disk.

Another example of the use of STARTUP is to set system environment paramters such as TTYSET parameters or the assigning of a system and working drive. If the STARTUP command consisted of the following line:

```
TTYSET,DP=16,WD=60:ASN,W=1:ASN:CAT,0
```

each time the system was booted the following actions would occur. First, TTYSET would set the 'depth' to 16 and the 'width' to 60. Next, assuming the 'end of line' character is the ':', the ASN command would assign the working drive to drive 1. Next ASN would display the assigned system and working drives on the terminal. Finally, a CATalog of the files on drive 0 would be displayed. For details of the actions of the individual commands, refer to their descriptions elsewhere in this manual.

(Turn over)

STARTUP (continued)

As it stands, it looks as if the STARTUP feature is limited to the execution of a single command line. This is true but there is a way around the restriction, the EX command. If a longer list of operations is desired than will fit on one line, simply create a command file containing all of the commands desired. Then create the STARTUP file placing the single line:

EXEC,<file name>

where <file name> would be replaced by the name assigned to the command file created. A little imagination and experience will show many uses for the STARTUP feature.

By directing STARTUP to a file that does not have a return to DOS command it is possible to lockout access to DOS. You can correct the problem by hitting the RESET button and beginning execution at address $AD03. The STARTUP file may then be deleted and if desired, modified. Directing execution to AD03, the DOS warm start address, bypasses the DOS STARTUP function.

TTYSET

The TTYSET utility command is provided so the user may control the characteristics of the terminal. With this command, the action of the terminal on input and the display format on output may be controlled.

DESCRIPTION

The general syntax of the TTYSET command is:

```
TTYSET[,<parameter list>]
```

where <parameter list> is a list of 2 letter parameter names, each followed by an equals sign ('='), and then by the value being assigned. Each parameter should be separated by a comma or a space. If no parameters are given, the values of all of the TTYSET parameters will be displayed on the terminal.

The default number base for numerical values is the base most appropriate to the parameter. In the descriptions that follow, 'hh' is used for parameters whose default base is hex; 'dd' is used for those whose default base is decimal. Values which should be expressed in hex are displayed in the TTYSET parameter listing preceded by a '$'. Some examples follow:

```
+++TTYSET
+++TTYSET,DP=16,WD=63
+++TTYSET,BS=8,ES=3
```

The first example simply lists the current values of all TTYSET parameters on the terminal. The next line sets the depth 'DP' to 16 lines and the terminal width, 'WD' to 63 columns. The last example sets the backspace character to the value of hex 8, and the escape character to hex 3.

The following fully describes all of the TTYSET parameters available to the user. Their initial values are defined, as well as any special characteristics they may possess.

BS=hh BackSpace character

This sets the 'backspace' charcter to the character having the ASCII hex value of hh. This character is initially a 'control H' (hex 08), but may be defined to any ASCII character. The action of the backspace character is to delete the last character typed from the terminal. If two backspace characters are typed, the last two characters will be deleted, etc. Setting BS=0 will disable the backspace feature.

(Turn over)

BE=hh Backspace Echo character

This defines the character to be sent to the terminal after a 'backspace' character is received. The character printed will have the ASCII hex value of hh. This character is initially set to a null but can be set to any ASCII character.

The BE command also has a very special use that will be of interest to some terminal owners, such as SWTPC CT-64.

If a hex 08 is specified as the echo character, FLEX will output a space (20) then another 08. This feature is very uesful for terminals which decode a hex 08 as a cursor left but which do not erase characters as the cursor is moved.

Example: Say that you mis-typed the word cat as shown below:
+++CAY

typing in one CTRL-H (hex 08) would position the cursor on top of the Y and delete the Y from the DOS input buffer. FLEX would then send out a space ($20) to erase the Y and another 08 (cursor left) to re-position the cursor.

DL=hh DeLete character

This sets the 'delete current line' character to the hex value hh. This character is initially a 'control X' (hex 18). The action of the delete character is to 'erase' the current input line before it is accepted into the computer for execution. Setting DL=0 will disable the line delete feature.

EL=hh End of Line character

This character is the one used by FLEX to separate multiple commands on one input line. It is initially set to a colon (':'), a hex value of 3A. Setting this character to 0 will disable the multiple command per line capability of FLEX. The parameter 'EL=hh' will set the end of line character to the character having the ASCII hex value of hh. This character must be set to a printable character (control characters not allowed).

DP=dd DePth count

This parameter specifies that a page consists of dd (decimal) physical lines of output. A page may be considered to be the number of lines between the fold if using fan folded paper on a hard copy terminal, or a page may be defined to be the number of lines which can be displayed at any one time on a CRT type terminal. Setting DP=0 will disable the paging (this is the initial value). See EJ and PS below for more details of depth.

(Turn over)

WD=dd WiDth

The WD parameter specifies the (decimal) number of characters to be displayed on a physical line at the terminal (the number of columns). Lines of text longer than the value of width will be 'folded' at every multiple of WD characters. For example, if WD is 50 and a line of 125 characters is to be displayed, the first 50 characters are displayed on a physical line at the terminal, the next 50 characters are displayed on the next physical line, and the last 25 characters are displayed on the third physical line. If WD is set to 0, the width feature will be disabled, and any number of characters will be permitted on a physical line.

NL=dd NuLl count

This parameter sets the (decimal) number of non-printing (Null) 'pad' characters to be sent to the terminal at the end of each line. These pad characters are used so the terminal carriage has enough time to return to the left margin before the next printable characters are sent. The initial value is 4. Users using CRT type terminals may want to set NL=0 since no pad characters are usually required on this type of terminal.

TB=hh TaB character

The tab character is not used by FLEX but some of the utilities may require one (such as the Text Editing System). This parameter will set the tab character to the character having the ASCII hex value hh. This character should be a printable character.

EJ=dd EJect count

This parameter is used to specify the (decimal) number of 'eject lines' to be sent to the terminal at the bottom of each page. If Pause is 'on', the 'eject sequence' is sent to the terminal after the pause is terminated. If the value dd is zero (which it is by default), no 'eject lines' are issued. An eject line is simply a blank line (line feed) sent to the terminal. This feature is especially useful for terminals with fan fold paper to skip over the fold (see Depth). It may also be useful for certain CRT terminals to be able to erase the previous screen contents at the end of each page.

(Turn over)

TTYSET (continued)

PS=Y or PS=N PauSe control

This parameter enables (PS=Y) or disables (PS=N) the end-of-page pause feature. If Pause is on and depth is set to some nonzero value, the output display is automatically suspended at the end of each page. The output may be restarted by typing the 'escape' character (see ES description). If pause is disabled, there will be no end-of-page pausing. This feature is useful for those using high-speed CRT terminals to suspend output long enough to read the page of text.

ES=hh EScape character

The character whose ASCII hex value is hh is defined to be the 'escape character'. Its initial value is $1B, the ASCII ESC character. The escape character is used to stop output from being displayed, and once it is stopped, restart it again. It is also used to restart output after Pause has stopped it. As an example, suppose you are LISTing a long text file on the terminal and you wish to temporarily halt the output. Typing the 'escape character' will do this (this feature is not supported on computers using a Control Port for terminal communications). At this time (output halted), typing another 'escape character' will resume output, while typing a RETURN key will cause control to return to FLEX and the three plus sign prompt will be output to the terminal. It should be noted that line output stopping always happens at the end of a line.

VERSION

The VERSION utility is used to display the version number of a utility command. If problems or updates ever occur in any of the utilities, they may be replaced with updated versions. The VERSION command will allow you to determine which version of a particular utility you have.

DESCRIPTION

The general syntax of the VERSION command is:

```
VERSION,<file spec>
```

where <file spec> is the name of the utility you wish to check. The default extension is CMD and the drive defaults to the working drive. As an example:

```
+++VERSION,0.CAT
```

would display the version number of the CAT command (from drive 0) on the terminal.

In order for a version number to be valid, the utility must have been written in a certain, defined manner. That is to say, the version number should be the third byte of the utility (the first two would normally be a branch instruction to bypass the version number). If the utility was not written in this manner, the version number reported will be meaningless. If you receive a version number from some command that is greater than 20 or 30, that command was probably not written to include a version number and the number you have received is meaningless. All the utility commands provided by TSC have valid version numbers.

10.7 GENERAL SYSTEM INFORMATION

I. DISK CAPACITY

Each sector of a FLEX disk contains 252 characters or bytes of user data (4 bytes of each 256 byte sector are used by the system). Thus a single-sided mini disk has 340*sectors or 85,680 characters or bytes of user information. A single-sided full size disk has 1140 sectors or 287,280 bytes of user data. Double-sided disks would contain exactly twice these amounts.

II. WRITE PROTECT

Floppy disks can usually be physically write protected to prevent FLEX from performing a write operation. Any attempt to write to such a disk will cause an error message to be issued. It is good practice to write protect disks which have important files on them. A mini disk can be write protected by placing a piece of opaque tape over the small rectangular cutout on the edge of the disk. Full size floppys are just the opposite. In order to write protect a full size disk, you must remove the tape from the cutout. In other words, the notch must be exposed to write protect the disk. Some full size disks do not have this cutout and therefore cannot be write protected.

III. THE 'RESET' BUTTON

The RESET button on the front panel of your computer should NEVER BE PRESSED DURING A DISK OPERATION. There should never be a need to 'reset' the machine while in FLEX. If the machine is 'reset' and the system is writing data on the disk, it is possible that the entire disk will become damaged. Again, never press 'reset' while the disk is operating! Refer to the 'escape' character in TTYSET for ways of stopping FLEX.

IV. NOTES ON THE P COMMAND

The P command tries to load a printer driver file named PRINT.SYS from the same disk which P itself was retrieved. For the requirements of this file and on writing your own custom PRINT.SYS file, see the section on such later in this manual or consult the 'Advanced Programmer's Guide'.

V. ACCESSING DRIVES NOT CONTAINING A DISKETTE

If an attempt is made to access a minifloppy drive not containing a diskette, the system will hang up attempting to read until a disk is inserted and the door closed. Alternatively, you could reset the machine and begin execution at the warm start location $AD03. This is not a problem on full-size drives.

* 340 sectors in the ADA-system
790 sectors in the Ericsson-system

VI. SYSTEM ERROR NUMBERS

Any time that FLEX detects an error during an operation, an appropriate error message will be displayed on the terminal. FLEX internally translates a derived error number into a plain language statement using a look-up table called ERROR.SYS. If you have forgotten to copy this .SYS file onto a disk that you are using, FLEX will report a corresponding number as shown below:

DISK ERROR #xx

where 'xx' is a decimal error number. The table below is a list of these numbers and what error they represent.

ERROR #	MEANING
1	ILLEGAL FMA FUNCTION CODE ENCOUNTERED
2	THE REQUESTED FILE IS IN USE
3	THE FILE SPECIFIED ALREADY EXISTS
4	THE SPECIFIED FILE COULD NOT BE FOUND
5	SYSTEM DIRECTORY ERROR-REBOOT SYSTEM
6	THE SYSTEM DIRECTORY IS FULL
7	ALL AVAILABLE DISK SPACE HAS BEEN USED
8	READ PAST END OF FILE
9	DISK FILE READ ERROR
10	DISK FILE WRITE ERROR
11	THE FILE OR DISK IS WRITE PROTECTED
12	THE FILE IS PROTECTED-FILE NOT DELETED
13	ILLEGAL FILE CONTROL BLOCK SPECIFIED
14	ILLEGAL DISK ADDRESS ENCOUNTERED
15	AN ILLEGAL DRIVE NUMBER WAS SPECIFIED
16	DRIVE NOT READY
17	THE FILE IS PROTECTED-ACCESS DENIED
18	SYSTEM FILE STATUS ERROR
19	FMS DATA INDEX RANGE ERROR
20	FMS INACTIVE-REBOOT SYSTEM
21	ILLEGAL FILE SPECIFICATION
22	SYSTEM FILE CLOSE ERROR
23	SECTOR MAP OVERFLOW-DISK TOO SEGMENTED
24	NON-EXISTENT RECORD NUMBER SPECIFIED
25	RECORD NUMBER MATCH ERROR-FILE DAMAGED
26	COMMAND SYNTAX ERROR-RE-TYPE COMMAND
27	THAT COMMAND IS NOT ALLOWED WHILE PRINTING
28	WRONG HARDWARE CONFIGURATION

HÅKAN JÖELSON
871130

Mikrodatorkonstruktion 10p
/UJ

Skillnader mellan FLEX 2.0 (6800) och FLEX 2.8:4 (6809)

När operativsystemet FLEX introducerades 1978 så var det i första hand tänkt för en 6800 baserad maskin som hette SWTPC (South West Technical ProduCts).Det låg på en 8" disk (DMAF1).
När sedan MOTOROLA introducerade mikroprocessorn M6809 (1979) Så var SWTPC en av de första tillverkarna som beställde en större kvantitet kretsar från MOTOROLA.
För att snabbt komma ut på marknaden så använde man samma operativsystem som för 6800-maskinen FLEX.
Den enda skillnaden är att FLEX för 09 ligger mellan $C000 - $DFFF
OBS! objektkoden för 6800 är inte kompatibel med 6809!
Man har alltså i princip tagit källkoden till FLEX (6800) och assemblerat om den med en 6809 assembler.(ORG $C100)
Om man vill använda subrutiner i FLEX för 6809 så är den enda skillnaden att man i anropsadresserna byter A mot C och B mot D.

```
ex:         PUTCHR      EQU     $CD18       (6800 PUTCHR $AD18)
            LDA         #'A                 skriv ett 'A'
            JSR         PUTCHR
```

för att starta FLEX från monitorn:

```
>^P  PC=XXXX  CD03              varmstart för FLEX (6800 $AD03)
>G
 +++
```

10.8 SYSTEM AND DOS MEMORY MAP

The Disk Operating System (DOS) forms the communication link between the user (via a computer terminal) and the File Management System. All commands are accepted through DOS. Functions such as file specification parsing, command argument parsing, terminal I/O, and error reporting are all handled by DOS.

The following memory map shows the location of user RAM and several major sections of the FLEX operating system. All addresses are in hexadecimal.

ADDRESS	DESCRIPTION
0000 - 7FFF	User RAM (Some of the lower end of this area is used by certain utilities such as COPY.)
A000 - A07F	Stack Area (SP is initialized to A07F)
A080 - A0FF	Input Buffer
A100 - A6FF	Utility Command Area
A700 - A83F	Scheduler & Printer Spooler
A840 - A97F	System FCB
A980 - ABFF	System Files Area
AC00 - B3FF	DOS
B400 - BE7F	FMS
BE80 - BFFF	Disk Drivers

FLEX 9.0, change $AXXX to $CXXX

The following is a description of those memory locations within the DOS portion of FLEX which contain information of interest to the programmer.

- $A080-$A0FF - Line Buffer
  The line buffer is a 128 byte area into which characters typed at the keyboard are placed by the routine INBUF. All characters entered from the keyboard are placed in this buffer with the exception of control characters. Characters which have been deleted by entering the backspace character do not appear in the buffer, nor does the backspace character itself appear. The carriage return signaling the end of the keyboard input is, however, put in the buffer. This buffer is also used to hold the STARTUP file during a coldstart (boot) operation.

- $AC0E-$AC10 - System Date Registers
  These three bytes are used to store the system date. It is stored in binary form with the (1) month in the first byte, followed by the (2) day, then the (3) year. The year byte contains only the tens and ones digits.

- $AC14-$AC15 - Line Buffer Pointer
  These locations contain the address of the next character in the Line Buffer to be processed.

- $AC16-$AC17 - Escape Return Register
  These locations contain the address to which to jump if a RETURN is typed while output has been stopped by an Escape Character. See TTYSET (page 10-25) for information on Escape processing. See also the documentation for the routine PCRLF in the next section (page 10-34)

10.9 USER-CALLABLE SYSTEM I/O SUBROUTINES

Unless specifically documented otherwise, the content of all registers should be presumed destroyed by calls to these routines. All routines, unless otherwise indicated, should be called with a JSR instruction.

$AD15 (GETCHR) Get Character

This routine will look for one character from the control terminal (I/O port #1) and store it in the A accumulator. Once called, the input routine will loop within itself until a character has been input. Anytime input is desired, the call JSR GETCHR or JSR $AD15 should be used.

GETCHR automatically sets the 8th bit to 0 and does not check for parity. A call to this subroutine affects the processor's registers as follows:

ACC A	loaded with the character input from the terminal (ASCII form)
ACC B	not affected
IXR	not affected

$AD18 (PUTCHR) Put Character

This subroutine is used to output one character from the computer to the control port (I/O port #1 in the SWTPc system).

To use PUTCHR, the character to be output should be placed in the A accumulator in its ASCII form. For example, to output the letter 'A' on the control terminal, the following program should be used:

```
LDA A  #$41
JSR    $AD18
```

The processor's registers are affected as follows:

ACC A	changed internally
ACC B	not affected
IXR	not affected

$AD1E (PSTRNG) Print String

PSTRNG is a subroutine used to output a string of text on the control terminal. When address $AD1E is called, a carriage return and line feed will automatically be generated and data output will begin at the location pointed to by the index register. Output will continue until a hex 04 is seen. The same rules for using the ESCAPE and RETURN keys for stopping output apply as described earlier.

The accumulator and register status after using PSTRNG are as follows:

ACC A	Changed during the operation
ACC B	UNCHANGED
IXR	Contains the memory location of the last character read from the string (usually the 04 unless stopped by the ESC key)

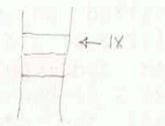
NOTE: The ability of using backspace and line delete characters is a function of your user program and not of the FLEX I/O routines described above.

$AD1B (INBUFF) Input into Line Buffer

This routine inputs a line from the keyboard into the Line Buffer. The TTYSET Backspace and Delete characters are checked and processed if encountered. All other control characters except RETURN and LINE FEED, are ignored. The RETURN is placed in the buffer at the end of the line. A LINE FEED is entered into the buffer as a space character but is echoed back to the terminal as a Carriage Return and Line Feed pair for continuation of the text on a new line. At most, 128 characters may be entered on the line, including the final RETURN. If more are entered, only the first 127 are kept, the RETURN being the 128th. On exit, the Line Buffer Pointer is pointing to the first character in the Line Buffer. Caution: The command line entered from the keyboard is kept in the Line Buffer. Calling INBUF from a user program will destroy the command line, including all unprocessed commands on the same line. Using INBUF and the Line Buffer for other than DOS commands may result in unpredictable side-effects.

$AD24 (PCRLF) Print Carriage Return and Line Feed

In addition to printing a carriage return and line feed, this routine checks and honors several TTYSET conditions. On entry, this routine checks for a TTYSET Escape Character having been entered while the previous line was being printed. If so, the routine waits for another TTYSET Escape Character or a RETURN to be typed. If a RETURN was entered, the routine clears the Last Terminator location so as to ignore any commands remaining in the command line, and then jumps to the address contained in the Escape Return Register locations. Unless changed by the user's program, this address is that of the Warmstart Entry Point. If, instead of a RETURN, another TTYSET Escape Character was typed, or it wasn't necessary to wait for one, the Current Line Number is checked. If the last line of the page has been printed and the TTYSET Pause feature is enabled, the routine waits for a RETURN or a TTYSET Escape Character, as above. Note that all pausing is done before the carriage return and line feed are printed. The carriage return and line feed are now printed, followed by the number of nulls specified by the TTYSET Null Count. If the end of the page was encountered on entry to this routine, an "eject" is performed by issuing additional carriage return, line feeds, and nulls until the total number of blank lines is that specified in the TTYSET Eject Count. The X register is preserved.



$AD45 (OUTADR) Output Hexadecimal Address

On entry, the X register contains the address of the most significant byte of a 2 byte hex value. The bytes to which the X register points are printed as 4 hexadecimal digits.

$AD39 (OUTDEC) Output Decimal Number

On entry, the X-register contains the address of the most significant byte of a 16-bit (2 byte), unsigned, binary number. The B-register, on entry, should contain a space suppression flag. The number will be printed as a decimal number with leading zeroes suppressed. If the B-register was non-zero on entry, spaces will be substituted for the leading zeroes. If the B-register is zero on entry, printing of the number will start with the first non-zero digit.

$AD3C (OUTHEX) Output Hexadecimal Number

On entry, the X-register contains the address of a single binary byte. The byte to which the X-register points is printed as 2 hexadecimal digits. The B and X registers are preserved.

$AD42 (GETHEX) Get Hexadecimal Number

This routine gets a hexadecimal number from the Line Buffer. On entry, the Line Buffer Pointer must point to the first character of the number in the Line Buffer. On exit, the carry bit is cleared if a valid number was found, the B-register is set non-zero, and the X-register contains the value of the number. The Line Buffer Pointer is left pointing to the character immediately following the separator character, unless that character is a carriage return or End of Line. If the first character examined in the Line Buffer is a separator character (such as a comma), the carry bit is still cleared, but the B-register is set to zero indicating that no actual number was found. In this case, the value returned in the X-register is zero. If a non-hexadecimal character is found while processing the number, characters in the Line Buffer are skipped until a separator character is found, then the routine returns to the caller with the carry bit set. The number in the Line Buffer may be of any length, but the value is truncated to between 0 and $FFFF, inclusive.

$AD48 (INDEC) Input Decimal Number

This routine gets an unsigned decimal number from the Line Buffer. On entry, the Line Buffer Pointer must point to the first character of the number in the Line Buffer. On exit, the carry bit is cleared if a valid number was found, the B-register is set non-zero, and the X-register contains the binary value of the number. The Line Buffer Pointer is left pointing as described in the routine GETHEX. If the first character examined in the buffer is a separator character (such as a comma), the carry bit is still cleared, but the B-register is set to zero indicating that no actual number was found. In this case, the number returned in X is zero. The number in the Line Buffer may be of any length but the result is truncated to 16 bit precision.

10.10 COMMAND AND SUBROUTINE SUMMARY

		Page
ASN[,W=<drive>][,S=<drive>]	ASN	10-9
CAT[,<drive list>][,<match list>]	CAT	10-11
COPY,<file spec>,<file spec> COPY,<file spec>,<drive> COPY,<drive>,<drive>[,<match list>]	COPY	10-13
DATE[,<mm,dd,yy>]	DATE	10-15
DELETE,<file spec>[,<file list>]	DELETE	10-16
GET,<file spec>[,<file list>] Default extension: .BIN	GET	10-7
LIST,<file spec>[,<line range>][,N] Default extension: .TXT	LIST	10-17
MON	MON	10-7
O,<file spec>,<command> Default extension: .OUT	O	10-18
P,<command>	P	10-19
PROT,<file spec>[,(options)]	PROT	10-20
RENAME,<file spec 1>,<file spec 2> Default extension: .TXT	RENAME	10-21
SAVE,<file spec>,<begin adr>,<end adr>[,<transfer adr>] Default extension: .BIN	SAVE	10-22
SAVE.LOW	SAVE.LOW	10-22
STARTUP	STARTUP	10-23
TTYSET[,<parameter list>]	TTYSET	10-25
VERSION,<file spec> Default extension: .CMD	VERSION	10-29

FLEX		FLEX 2.0	Page	$CD03
GETCHR	Get Character	$AD15	10-33	FLEX 9.0, change
PUTCHR	Put Character	$AD18	10-33	$AXXX to $CXXX
PSTRNG	Print String	$AD1E	10-33	
INBUFF	Input into Line Buffer	$AD1B	10-34	
PCRLF	Print Carriage Return and Line Feed	$AD24	10-34	
OUTADR	Output Hexadecimal Address	$AD45	10-34	
OUTDEC	Output Decimal Number	$AD39	10-35	
OUTHEX	Output Hexadecimal Number	$AD3C	10-35	
GETHEX	Get Hexadecimal Number	$AD42	10-35	
INDEC	Input Decimal Number	$AD48	10-35	
Line Buffer		$A080-$A0FF	10-32	
System Date Registers		$AC0E-$AC10	10-32	
Line Buffer Pointer		$AC14-$AC15	10-32	
Escape Return Register		$AC16-$AC17	10-32	

DIRECTORY OF DRIVE NUMBER 0
DISK: FLEX9 #8 CREATED: 19-JAN-87

FILE#	NAME	TYPE	R	BEGIN	END	SIZE	DATE	PRT
1	FLEX_DD	.SYS		01-01	01-1A	26	25-OCT-84	
2	STARTUP	.TXT		01-1B	01-1B	1	2-AUG-85	
3	MEMEND	.BIN		01-1C	01-1C	1	24-MAY-84	
4	PRINTSET	.CMD		01-1D	01-1D	1	30-OCT-84	
5	PRINT	.SYS		01-1E	01-1E	1	26-AUG-84	
6	ERRORS	.SYS	R	01-1F	02-03	9	30-JAN-80	
7	NEWDISK2	.CMD		02-04	02-0C	9	22-AUG-85	
8	RA	.CMD		02-0D	04-12	78	0-JAN- 0	
9	SE	.CMD		04-13	06-0B	65	25-OCT-84	
10	UVPROG	.CMD		06-0C	06-11	6	28-JAN-85	
11	UVPROG	.HLP		06-12	06-13	2	25-SEP-84	
12	APPEND	.CMD		06-14	07-05	22	5-NOV-81	
13	AR	.CMD		07-06	07-0E	9	10-MAR-81	
14	ASN	.CMD		07-0F	07-0F	1	30-JAN-80	
15	BUILD	.CMD		07-10	07-10	1	30-JAN-80	
16	PC	.CMD		07-11	0A-20	124	15-JAN-85	
17	CAT	.CMD		0A-21	0A-23	3	8-MAR-81	
18	DB	.CMD		0A-24	0C-1B	64	30-OCT-84	
19	COPY	.CMD		0C-1C	0C-20	5	9-MAR-81	
20	DATE	.CMD		0C-21	0C-22	2	30-JAN-80	
21	DELETE	.CMD		0C-23	0C-24	2	30-JAN-80	
22	ECHO	.CMD		0D-01	0D-01	1	30-JAN-80	
23	EXEC	.CMD		0D-02	0D-02	1	30-JAN-80	
24	FIND	.CMD		0D-03	0D-1E	28	29-JUL-81	
25	FIX	.CMD		0D-1F	0E-01	7	30-JAN-80	
26	I	.CMD		0E-02	0E-03	2	8-OCT-80	
27	JUMP	.CMD		0E-04	0E-04	1	30-JAN-80	
28	L	.CMD		0E-05	0E-19	21	29-JUL-81	
29	LINK	.CMD		0E-1A	0E-1A	1	30-JAN-80	
30	LIST	.CMD		0E-1B	0E-1D	3	30-JAN-80	
31	MIRROR	.CMD		0E-1E	0E-24	7	7-MAR-81	
32	MV	.CMD		0F-01	0F-04	4	26-MAR-80	
33	N	.CMD		0F-05	0F-05	1	30-JAN-80	
34	DBOV	.BIN		0F-06	10-09	40	30-OCT-84	
35	NF	.CMD		10-0A	10-0B	2	20-MAY-81	
36	O	.CMD		10-0C	10-0E	3	8-OCT-80	
37	LC	.CMD		10-0F	11-12	40	30-OCT-84	
38	PROT	.CMD		11-13	11-13	1	30-JAN-80	
39	LB	.CMD		11-14	12-03	20	30-OCT-84	
40	PUTBOOT	.CMD		12-04	12-09	6	7-NOV-80	
41	PROMPT	.CMD		12-0A	12-0B	2	8-MAR-81	
42	Q	.CMD		12-0C	12-10	5	29-SEP-81	
43	RE	.CMD		12-11	12-11	1	30-OCT-84	
44	CHAIN	.CMD		12-12	12-13	2	30-OCT-84	
45	RENAME	.CMD		12-14	12-14	1	30-JAN-80	
46	RM	.CMD		12-15	12-16	2	1-JUL-80	
47	S	.CMD		12-17	12-19	3	1-JUL-80	
48	SAVE	.CMD		12-1A	12-1B	2	30-JAN-80	
49	SBOX	.CMD		12-1C	12-1F	4	24-OCT-80	
50	SP	.CMD		12-20	12-23	4	1-JUL-80	
51	SUM	.CMD		12-24	13-01	2	20-AUG-80	
52	T	.CMD		13-02	13-03	2	6-JUL-81	
53	TESTMEM	.CMD		13-04	13-09	6	9-APR-84	
54	RL	.RO		13-0A	15-10	79	30-OCT-84	
55	TOUCH	.CMD		15-11	15-12	2	10-MAR-81	
56	TTYSET	.CMD		15-13	15-14	2	30-JAN-80	
57	UNITERM	.CMD		15-15	15-17	3	22-JAN-82	
58	VER	.CMD		15-18	15-1A	3	6-MAR-81	
59	VERIFY	.CMD		15-1B	15-1B	1	30-JAN-80	
60	XOUT	.CMD		15-1C	15-1D	2	30-JAN-80	
61	Y	.CMD		15-1E	15-1E	1	30-JAN-80	
62	SAVE	.LOW		15-1F	15-20	2	30-JAN-80	
63	FLEXLIB	.TXT		15-21	16-10	20	16-DEC-81	
64	DIR	.CMD		16-11	16-15	5	16-JUN-89	
65	FRAME	.CMD		16-16	16-22	13	19-AUG-82	
66	EDIT	.CMD		16-23	17-1A	28	16-JUN-89	
67	ASMB	.CMD		17-1B	19-01	47	16-JUN-89	
68	DEBUG	.CMD		19-02	1A-0B	46	16-JUN-84	
69	PO	.CMD		1A-0C	1A-0C	1	26-AUG-84	
70	MAP	.CMD		1A-0D	1A-0D	1	16-JUN-89	
71	SGEN	.CMD		1A-0E	1B-02	25	16-AUG-82	
72	DUMP	.CMD		1B-03	1B-03	1	25-SEP-84	
73	MEMDUMP	.CMD		1B-04	1B-04	1	25-SEP-84	
74	LL	.CMD		1B-05	1C-13	51	0-JAN- 0	
75	LOAD	.CMD		1C-14	1C-14	1	25-JAN-85	
76	USD1	.RO		1C-15	1C-17	3	28-JAN-85	
77	HINIT	.RO		1C-18	1C-18	1	26-JAN-85	
78	RESET	.RO		1C-19	1C-19	1	25-JAN-85	
79	R	.CMD		1C-1A	1C-1A	1	14-FEB-85	
80	SEND	.CMD		1C-1B	1C-1C	2	26-FEB-85	
81	FYSPROG	.CMD		1C-1D	1C-24	8	18-APR-85	
82	FYSPROG	.HLP		1D-01	1D-03	3	18-APR-85	
83	P	.COR	R	1D-04	1D-08	5	10-MAR-81	
84	SERIAL	.CMD		1D-09	1D-09	1	2-AUG-85	
85	PDEL	.CMD		1D-0A	1D-0C	3	19-AUG-82	

FILES=85, SECTORS=1020, LARGEST=124, FREE=1824

```
.
FLEX      PUBLIC

LINBUF    EQU     $C080           LINE BUFFER
CMDADR    EQU     $C100           UTILITY COMMAND SPACE (1.5 K)
CMDEND    EQU     $C700           UTILITY COMMAND SPACE END
SYSFCB    EQU     $C840           SYSTEM FCB ADDRESS
*
.   GLOBAL VALUES SPECIFIED BY TTYSET AND ASN
.
BSPCHR    EQU     $CC00           BACKSPACE CHARACTER
DELCHR    EQU     $CC01           DELETE CHARACTER
EOLCHR    EQU     $CC02           END OF LINE CHARACTER
DEPTH     EQU     $CC03           DEPTH COUNT
WIDTH     EQU     $CC04           WIDTH COUNT
NULLS     EQU     $CC05           NULL COUNT
TABCHR    EQU     $CC06           TAB CHARACTER
BSECHR    EQU     $CC07           BACKSPACE ECHO CHARACTER
PAUSE     EQU     $CC09           PAUSE CONTROL BYTE
ESCCHR    EQU     $CC0A           ESCAPE CHARACTER
S_DRN     EQU     $CC0B           SYSTEM DRIVE NUMBER
W_DRN     EQU     $CC0C           WORKING DRIVE NUMBER
*
.   FLEX SYSTEM GLOBAL VARIABLES
.
SYSFLG    EQU     $CC0D           USE SYSTEM DRIVE FLAG
SYSDATE   EQU     $CC0E           DATE REGISTERS
LSTTRM    EQU     $CC11           LAST TERMINATOR CHARACTER
CBUFPT    EQU     $CC14           LINE BUFFER POINTER
ESCRET    EQU     $CC16           ESCAPE RETURN REGISTER
CURCHR    EQU     $CC18           CURRENT NXTCH CHARACTER
PREVCH    EQU     $CC19           PREVIOUS NXTCH CHARACTER
CURLCT    EQU     $CC1A           CURRENT LINE COUNT
LOADAO    EQU     $CC1B           LOADER ADDRESS OFFSET DATA
XFRFLG    EQU     $CC1D           TRANSFER ADDRESS FLAG
XFRADR    EQU     $CC1E           TRANSFER ADDRESS OF LOADED FILE
OUTSWT    EQU     $CC22           OUTPUT SWITCH
INSWT     EQU     $CC23           INPUT SWITCH
DOCMDF    EQU     $CC28           DOCMD ENTRY FLAG
CURCOL    EQU     $CC29           CURRENT OUTPUT COLUMN
MEMEND    EQU     $CC2B           END OF MEMORY ADDRESS
FMSBUSY   EQU     $CC30           FLEX ALREADY BUSY FLAG
CPUTYPE   EQU     $CC33           CPU TYPE FLAG
RETADR    EQU     $CC43           DOCMD RETURN ADDRESS
ULCFLAG   EQU     $CC49           UPPER/LOWER CASE MAP FLAG
PROMPT    EQU     $CC4E           POINTER TO PROMPT STRING
*
.   CPU TYPE FLAG BIT DEFINITIONS
.
CPU_2MHZ  EQU     %10000000       1 => 2 MHz CPU CLOCK RATE
CPU_SLOW  EQU     %01000000       1 => MEMORY STRETCH ACTIVE
CPU_50HZ  EQU     %00100000       1 => 50 Hz POWER LINE FREQUENCY
CPU_RAMF  EQU     %00010000       1 => CPU RAM IS AVAILABLE
CPU_RTCK  EQU     %00001000       1 => 6819 REAL TIME CLOCK AVAILABLE
CPU_IOBX  EQU     %00000100       1 => I/O SET UP LIKE OLD BOX
CPU_TIME  EQU     %00000010       1 => 6840 TIMER AVAILABLE
CPU_XMEM  EQU     %00000001       1 => EXTENDED MEMORY IS USED
*
.   PRINTER DRIVER INTERFACE ADDRESSES
.
PT_RAP    EQU     $CC35           PRINTER RESERVED AREA POINTER
PT_LEN    EQU     $CC37           PRINTER RESERVED AREA LENGTH
PT_DEV    EQU     $CC39           PRINTER DEVICE ADDRESS
PINIT     EQU     $CCC0           PRINTER INITIALIZE VECTOR
PTERM     EQU     $CCD0           PRINTER CLOSE VECTOR
PCHK      EQU     $CCD8           PRINTER READY CHECK VECTOR
POUT      EQU     $CCE4           PRINTER CHARACTER OUTPUT ROUTINE
PRCFLG    EQU     $CCFC           ACTIVE SPOOLING PROCESS FLAG
*
.   FLEX SYSTEM DEFINED ENTRY VECTORS
.
COLDS     EQU     $CD00           FLEX COLD START ADDRESS
WARMS     EQU     $CD03           FLEX WARM START ADDRESS
RENTER    EQU     $CD06           RE-ENTER FLEX PROCESSING
INCH      EQU     $CD09           INPUT CHARACTER (LOW LEVEL)
OUTCH     EQU     $CD0F           OUTPUT CHARACTER (LOW LEVEL)
GETCHR    EQU     $CD15           INPUT CHARACTER ROUTINE
PUTCHR    EQU     $CD18           OUTPUT CHARACTER ROUTINE
INBUFF    EQU     $CD1B           INPUT LINE BUFFER
PSTRNG    EQU     $CD1E           PRINT STRING
CLASS     EQU     $CD21           CLASSIFY CHARACTER
PCRLF     EQU     $CD24           PRINT CR/LF SEQUENCE
NXTCH     EQU     $CD27           GET NEXT CHARACTER FROM INPUT BUFFER
GETFIL    EQU     $CD2D           SCAN FILE SPEC ADDRESS
LOAD      EQU     $CD30           LOAD FILE ENTRY POINT
SETEXT    EQU     $CD33           SET UP FILE EXTENSION
OUTDEC    EQU     $CD39           OUTPUT DECIMAL NUMBER
OUTHEX    EQU     $CD3C           OUTPUT HEXADECIMAL NUMBER
RPTERR    EQU     $CD3F           I/O ERROR ABORT ROUTINE
GETHEX    EQU     $CD42           GET HEXIDECIMAL SPECIFICATION
OUTADR    EQU     $CD45           OUTPUT HEXADECIMAL ADDRESS
INDEC     EQU     $CD48           GET DECIMAL NUMBER
DOCMD     EQU     $CD4B           DOCMD ENTRY ADDRESS
STATUS    EQU     $CD4E           CHECK TERMINAL INPUT STATUS
*
.   LOW LEVEL TERMINAL AND INTERRUPT CONTROL ADDRESSES
.
INTAP     EQU     $D3DE           VECTOR FOR INPUT TAP ROUTINE
DUMMY     EQU     $D3E0           DUMMY RTS INSTRUCTION USED BY RM
SETIRQ    EQU     $D3E1           SET IRQ PROCESS VECTOR
CLRIRQ    EQU     $D3E3           CLEAR IRQ PROCESS VECTOR
TINCH     EQU     $D3E5           LOW-LEVEL TERM INPUT WITHOUT ECHO
T_OFF     EQU     $D3ED           TIMER OFF ROUTINE ADDRESS
T_ON      EQU     $D3EF           TIMER ON ROUTINE ADDRESS
T_INIT    EQU     $D3F1           TIMER INITIALIZE ROUTINE ADDRESS
TINIT     EQU     $D3F5           LOW-LEVEL TERMINAL INITIALIZE
TCHECK    EQU     $D3F7           LOW-LEVEL TERMINAL CHECK ADDRESS
TOUTCH    EQU     $D3F9           LOW-LEVEL TERMINAL OUTPUT ADDRESS
TINCHE    EQU     $D3FB           LOW-LEVEL TERMINAL INPUT WITH ECHO
*
.   FILE MANAGEMENT SYSTEM ENTRY POINTS
.
FMSCLS    EQU     $D403           CLOSE UP ALL FILES ENTRY
FMS       EQU     $D406           FILE MANAGER EXEC CALL
FCBASE    EQU     $D409           FILE CONTROL BLOCK BASE
VERIFY    EQU     $D435           FMS VERIFY FLAG
SURTAB    EQU     $D436           FMS SURNAME TABLE

FCBLEN    EQU     256+64          FILE CONTROL BLOCK LENGTH
*
.   DISK DRIVER ENTRY POINTS
.
DREAD     EQU     $DE00           READ SECTOR ROUTINE
DWRITE    EQU     $DE03           WRITE SECTOR ROUTINE
DVERFY    EQU     $DE06           VERIFY ROUTINE
DREST     EQU     $DE09           DRIVE RESTORE ROUTINE
DRIVE     EQU     $DE0C           DRIVE SELECT ROUTINE
DCHECK    EQU     $DE0F           CHECK DRIVE READY
DQUICK    EQU     $DE12           QUICK CHECK DRIVE READY
DSEEK     EQU     $DE1B           DRIVE SEEK-TO-SECTOR ROUTINE

          END
```

Tabell för omvandling mellan decimala och hexadecimala tal

DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX
0	00	50	32	100	64	150	96	200	C8
1	01	51	33	101	65	151	97	201	C9
2	02	52	34	102	66	152	98	202	CA
3	03	53	35	103	67	153	99	203	CB
4	04	54	36	104	68	154	9A	204	CC
5	05	55	37	105	69	155	9B	205	CD
6	06	56	38	106	6A	156	9C	206	CE
7	07	57	39	107	6B	157	9D	207	CF
8	08	58	3A	108	6C	158	9E	208	D0
9	09	59	3B	109	6D	159	9F	209	D1
10	0A	60	3C	110	6E	160	A0	210	D2
11	0B	61	3D	111	6F	161	A1	211	D3
12	0C	62	3E	112	70	162	A2	212	D4
13	0D	63	3F	113	71	163	A3	213	D5
14	0E	64	40	114	72	164	A4	214	D6
15	0F	65	41	115	73	165	A5	215	D7
16	10	66	42	116	74	166	A6	216	D8
17	11	67	43	117	75	167	A7	217	D9
18	12	68	44	118	76	168	A8	218	DA
19	13	69	45	119	77	169	A9	219	DB
20	14	70	46	120	78	170	AA	220	DC
21	15	71	47	121	79	171	AB	221	DD
22	16	72	48	122	7A	172	AC	222	DE
23	17	73	49	123	7B	173	AD	223	DF
24	18	74	4A	124	7C	174	AE	224	E0
25	19	75	4B	125	7D	175	AF	225	E1
26	1A	76	4C	126	7E	176	B0	226	E2
27	1B	77	4D	127	7F	177	B1	227	E3
28	1C	78	4E	128	80	178	B2	228	E4
29	1D	79	4F	129	81	179	B3	229	E5
30	1E	80	50	130	82	180	B4	230	E6
31	1F	81	51	131	83	181	B5	231	E7
32	20	82	52	132	84	182	B6	232	E8
33	21	83	53	133	85	183	B7	233	E9
34	22	84	54	134	86	184	B8	234	EA
35	23	85	55	135	87	185	B9	235	EB
36	24	86	56	136	88	186	BA	236	EC
37	25	87	57	137	89	187	BB	237	ED
38	26	88	58	138	8A	188	BC	238	EE
39	27	89	59	139	8B	189	BD	239	EF
40	28	90	5A	140	8C	190	BE	240	F0
41	29	91	5B	141	8D	191	BF	241	F1
42	2A	92	5C	142	8E	192	C0	242	F2
43	2B	93	5D	143	8F	193	C1	243	F3
44	2C	94	5E	144	90	194	C2	244	F4
45	2D	95	5F	145	91	195	C3	245	F5
46	2E	96	60	146	92	196	C4	246	F6
47	2F	97	61	147	93	197	C5	247	F7
48	30	98	62	148	94	198	C6	248	F8
49	31	99	63	149	95	199	C7	249	F9

Tabell över hexadecimala tal angivna i 2-komplementform

DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX
-128	80	- 80	B0	- 32	E0	16	10	64	40
-127	81	- 79	B1	- 31	E1	17	11	65	41
-126	82	- 78	B2	- 30	E2	18	12	66	42
-125	83	- 77	B3	- 29	E3	19	13	67	43
-124	84	- 76	B4	- 28	E4	20	14	68	44
-123	85	- 75	B5	- 27	E5	21	15	69	45
-122	86	- 74	B6	- 26	E6	22	16	70	46
-121	87	- 73	B7	- 25	E7	23	17	71	47
-120	88	- 72	B8	- 24	E8	24	18	72	48
-119	89	- 71	B9	- 23	E9	25	19	73	49
-118	8A	- 70	BA	- 22	EA	26	1A	74	4A
-117	8B	- 69	BB	- 21	EB	27	1B	75	4B
-116	8C	- 68	BC	- 20	EC	28	1C	76	4C
-115	8D	- 67	BD	- 19	ED	29	1D	77	4D
-114	8E	- 66	BE	- 18	EE	30	1E	78	4E
-113	8F	- 65	BF	- 17	EF	31	1F	79	4F
-112	90	- 64	C0	- 16	F0	32	20	80	50
-111	91	- 63	C1	- 15	F1	33	21	81	51
-110	92	- 62	C2	- 14	F2	34	22	82	52
-109	93	- 61	C3	- 13	F3	35	23	83	53
-108	94	- 60	C4	- 12	F4	36	24	84	54
-107	95	- 59	C5	- 11	F5	37	25	85	55
-106	96	- 58	C6	- 10	F6	38	26	86	56
-105	97	- 57	C7	- 9	F7	39	27	87	57
-104	98	- 56	C8	- 8	F8	40	28	88	58
-103	99	- 55	C9	- 7	F9	41	29	89	59
-102	9A	- 54	CA	- 6	FA	42	2A	90	5A
-101	9B	- 53	CB	- 5	FB	43	2B	91	5B
-100	9C	- 52	CC	- 4	FC	44	2C	92	5C
- 99	9D	- 51	CD	- 3	FD	45	2D	93	5D
- 98	9E	- 50	CE	- 2	FE	46	2E	94	5E
- 97	9F	- 49	CF	- 1	FF	47	2F	95	5F
- 96	A0	- 48	D0	0	00	48	30	96	60
- 95	A1	- 47	D1	1	01	49	31	97	61
- 94	A2	- 46	D2	2	02	50	32	98	62
- 93	A3	- 45	D3	3	03	51	33	99	63
- 92	A4	- 44	D4	4	04	52	34	100	64
- 91	A5	- 43	D5	5	05	53	35	101	65
- 90	A6	- 42	D6	6	06	54	36	102	66
- 89	A7	- 41	D7	7	07	55	37	103	67
- 88	A8	- 40	D8	8	08	56	38	104	68
- 87	A9	- 39	D9	9	09	57	39	105	69
- 86	AA	- 38	DA	10	0A	58	3A	106	6A
- 85	AB	- 37	DB	11	0B	59	3B	107	6B
- 84	AC	- 36	DC	12	0C	60	3C	108	6C
- 83	AD	- 35	DD	13	0D	61	3D	109	6D
- 82	AE	- 34	DE	14	0E	62	3E	110	6E
- 81	AF	- 33	DF	15	0F	63	3F	111	6F

% ⇔ binära tal @ ⇔ oktala tal $ ⇔ hexadecimala tal

Bild 2.8 Omvandlingstabell hexadecimalt–decimalt

HEXADECIMAL COLUMNS											
6		5		4		3		2		1	
HEX	= DEC	HEX	= DEC	HEX	= DEC	HEX	= DEC	HEX	= DEC	HEX	= DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
0123		4567		0123		4567		0123		4567	
BYTE				BYTE				BYTE			

16^n	n	16^{-n}
1	0	1.0
16	1	0.062 5
256	2	0 003 906 25
4 096	3	0.000 244 140 625
65 536	4	0.000 015 258 789 062 5
1 048 576	5	0 000 000 953 674 316 406 25
16 777 216	6	0 000 000 059 604 644 775 390 625
268 435 456	7	0.000 000 003 725 290 298 461 914 062 5
4 294 967 296	8	0.000 000 000 232 830 643 653 869 628 906 25
68 719 476 736	9	0.000 000 000 014 551 915 228 366 851 806 640 625
1 099 511 627 776	10	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

imala tal angivna i 2-komplementform

HEX	DEC	HEX	DEC	HEX	DEC	HEX
B0	- 32	E0	16	10	64	40
B1	- 31	E1	17	11	65	41
B2	- 30	E2	18	12	66	42
B3	- 29	E3	19	13	67	43
B4	- 28	E4	20	14	68	44
B5	- 27	E5	21	15	69	45
B6	- 26	E6	22	16	70	46
B7	- 25	E7	23	17	71	47
B8	- 24	E8	24	18	72	48
B9	- 23	E9	25	19	73	49
BA	- 22	EA	26	1A	74	4A
BB	- 21	EB	27	1B	75	4B
BC	- 20	EC	28	1C	76	4C
BD	- 19	ED	29	1D	77	4D
BE	- 18	EE	30	1E	78	4E
BF	- 17	EF	31	1F	79	4F
C0	- 16	F0	32	20	80	50
C1	- 15	F1	33	21	81	51
C2	- 14	F2	34	22	82	52
C3	- 13	F3	35	23	83	53
C4	- 12	F4	36	24	84	54
C5	- 11	F5	37	25	85	55
C6	- 10	F6	38	26	86	56
C7	- 9	F7	39	27	87	57
C8	- 8	F8	40	28	88	58
C9	- 7	F9	41	29	89	59
CA	- 6	FA	42	2A	90	5A
CB	- 5	FB	43	2B	91	5B
CC	- 4	FC	44	2C	92	5C
CD	- 3	FD	45	2D	93	5D
CE	- 2	FE	46	2E	94	5E
CF	- 1	FF	47	2F	95	5F
D0	0	00	48	30	96	60
D1	1	01	49	31	97	61
D2	2	02	50	32	98	62
D3	3	03	51	33	99	63
D4	4	04	52	34	100	64
D5	5	05	53	35	101	65
D6	6	06	54	36	102	66
D7	7	07	55	37	103	67
D8	8	08	56	38	104	68
D9	9	09	57	39	105	69
DA	10	0A	58	3A	106	6A
DB	11	0B	59	3B	107	6B
DC	12	0C	60	3C	108	6C
DD	13	0D	61	3D	109	6D
DE	14	0E	62	3E	110	6E
DF	15	0F	63	3F	111	6F

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25

APPENDIX A

HEXADECIMAL CONVERSION TABLE

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

5		4		3		2		1		0	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

APPENDIX B

ASCII CONVERSION TABLE

HEX	MSD	0	1	2	3	4	5	6	7
LSD	BITS	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SPACE	0	@	P	‵	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	¦
D	1101	CR	GS	–	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	←	o	DEL

THE ASCII SYMBOLS

NUL — Null
SOH — Start of Heading
STX — Start of Text
ETX — End of Text
EOT — End of Transmission
ENQ — Enquiry
ACK — Acknowledge
BEL — Bell
BS — Backspace
HT — Horizontal Tabulation
LF — Line Feed
VT — Vertical Tabulation
FF — Form Feed
CR — Carriage Return
SO — Shift Out
SI — Shift In
DLE — Data Link Escape
DC — Device Control
NAK — Negative Acknowledge
SYN — Synchronous Idle
ETB — End of Transmission Block
CAN — Cancel
EM — End of Medium
SUB — Substitute
ESC — Escape
FS — File Separator
GS — Group Separator
RS — Record Separator
US — Unit Separator
SP — Space (Blank)
DEL — Delete

Table 3-9. New 6809 Instructions (Without 6800 Equivalents)

Instruction Mnemonic	
ABX	(Also implemented on 6801 microprocessor)
BRN	
LBRN	
LEA	
SEX	
SYNC	(but similar to 6800 WAI)

Table 3-10. 6809 Implementations of Missing 6800 Instructions

6800 Instruction	6809 Equivalent
ABA	PSHS B; ADDA ,S+
CBA	PSHS B; CMPA ,S+
CLC	ANDCC #%11111110
CLI	ANDCC #%11101111
CLV	ANDCC #%11111101
DES	LEAS −1,S
DEX	LEAX −1,X
INS	LEAS 1,S
INX	LEAX 1,X
PSHA	PSHS A *
PSHB	PSHS B *
PULA	PULS A *
PULB	PULS B *
SBA	PSHS B; SUBA ,S+
SEC	ORCC #%00000001
SEI	ORCC #%00010000
SEV	ORCC #%00000010
TAB	TFR A,B; TSTA
TAP	TFR A,CC
TBA	TFR B,A; TSTA
TPA	TFR CC,A
TSX	TFR S,X
TXS	TFR X,S
WAI	CWAI #$FF or CWAI #$EF to enable regular interrupt (replaces CLI, WAI)

* 6809 Stack Pointer manipulation differs from that of the 6800. See the text for further information.

6801/6809 COMPATIBILITY

The 6801 microprocessor is a slightly improved version of the 6800 microprocessor that is manufactured by some of the same companies. The 6801 instruction set is almost the same as the 6800's except that the 6801 has a multiplication instruction and a ABX instruction (as on the 6809), as well as 16-bit shifts for the double accumulator that are not implemented on either the 6800 or the 6809. The 6801, like the 6809, does set the flags properly after CMPX (CPX). The only added difference is that the 6801 multiply instruction (MUL) does not affect the Zero flag, whereas the 6809 MUL instruction does.

HÅKAN JOELSON 871130

APPEND,<file spec>[,<file list>],<file spec>
Default Extension: .TXT
Description Page: A.1

AR,<drive>,<drive>
Description Page: A.3 ____Updatera filer.

ASN[,W=<drive>][,S=<drive>]
Description Page: A.2

BUILD,<file spec> ___Skriva en textfil
Default Extension: .TXT
Description Page: B.1

(*) CAT[,<drive list>][,<match list>
Description Page: C.1

(*) DIR[,<drive list>][,<match list>]___directory med spår,sect.

CLEAN,<drive number>
Description Page: C.6.1 ____Rengöra diskdrive

(*) COPY,<file spec>,<file spec>
COPY,<file spec>,<drive>
COPY,<drive>,<drive>[,<match list>] cf MV
Description Page: C.2

DATE[,<mm,dd,yy>]
Description Page: D.1

(*) DELETE,<file spec>[,<file list>]
Description Page: D.2

ECHO,<string>
Description Page: E.1 ___Skicka text till skärm

EXEC,<file spec>
Default Extension: .TXT
Description Page: E.2 ____Länka kommandon

FIX,<file spec>[,<file spec>] (Ändra opkod på filen)
Default Extension: .BIN
Description Page: F.1 ___Editera binära filer på skiva.

GET,<file spec>[,<file list>] ___ Hämtar fil
Default Extension: .BIN
Description Page: 1.7

I,<file spec>,<command>
Default Extension: .TXT ____Indata från en fil
Description Page: I.1 Man k lägga in data på en fil via

JUMP,<hex address> ____Hoppa till adress
Description Page: J.1

LINK,<file spec>
Default Extension: .SYS
Description Page: L.1

(*) LIST,<file spec>[,<line range>][,+<options>]
Default Extension: .TXT
Description Page: L.2

MIRROR,<drive>,<drive>[,+<options>]
Description Page: M.3 ____Exakt kopia av skiva

MON
Description Page: 1.7

MV,<file spec>,<file spec>
Default Extension: .TXT ____Flytta en fil till
Description Page: M.2 annan skiva. cf COPY

N,<command>
Description Page: N.2 ____Svara "nej" i EXEC

(*) NEWDISK[,<drive>]
Description Page: N.1 ____Formattera en skiva
OBS! för DS DD NEWDISK2!

O,<file spec>,<command>
Default Extension: .OUT
Description Page: O.1 ____Utdata till fil

P,<command> Gamla
Description Page: P.1 ____Nytt "P" (fungerar ej!)

P.COR
Description Page: P.2 ____Binär version

(*) PO,<command> Nytt
Description Page: P.3 ____Gamla "P" (fungerar!)

PROT,<file spec>[,<options>]
Description Page: P.5 ____Skyddar en fil

PSP[,<file spec>][,<print command>]
Default Extension: .OUT
.CMD
Description Page: P.4 ____Printer spooler går ej!

PUTBOOT,<drive>
PUTBOOT,<file spec> ____Sriver bootblocket på angiven plats
Default Extension: .SYS
Description Page: P.6

(*) MAP, <file spec>

Q,<command>
Description Page: Q.1 ___Driver för QUME printer

QCHECK
Description Page: Q.2 ____Kolla print kön.

(*) RENAME,<file spec 1>,<file spec 2>
Default Extension: .TXT
Description Page: R.1

RM[,<size>]
Description Page: R.2 ____Reserverar minnesarea för stora printerdrive

READPROM,<file spec>
Default Extension: .BIN
Description Page: R.3 ____Läs av en EPROM (går ej)

S,<command>
Description Page: S.1 ____Seriell printer driver

SAVE,<file spec>,<begin adr>,<end adr>[,<transfer adr
Default Extension: .BIN
Description Page: S.2

SAVE.LOW
Description Page: S.2.2 ____Samma som SAVE men lägger sig lågt i minnet

SBOX[,<parameter list>]
Description Page: S.3 ____Konfig FLEX

SUM,<file spec>
Description Page: S.6 ____32 bit checksumma på fil

SP,<command>
Description Page: S.4 ____IBM skrivare driver

STARTUP
Description Page: S.5

TIME,<command>
Description Page: T.3 ____Visar tiden (går ej!)

TOUCH,<file spec>
Default Extension: .TXT ____Ändra datum på en fil
Description Page: T.2

TTYSET[,<parameter list>] Sätta spec f olika saker
Description Page: T.1

UCAL
Description Page: U.2 ____Hårddisk drive (går ej!)

(*) Vanligast, mest använda

SCREEN EDITOR COMMANDS

Line syntax : "." = current, "$" = last, digits = offset or relative line number, + and - allowed.

Buffered Commands ('E' prompt)

Edit <filename>	Clear buffer and read file.
New <filename>	Clear buffer and set filename.
Get <filename>	Get file past cursor.
Save [<filename>]	Save entire file.
Save and Quit	Save entire file and quit.
Save Partial <line>,<line> <filename>	Save part of file.
Move <line>[,<line>]	will move lines to after cursor position.
Copy <line>[,<line>]	will copy lines to after cursor position.
DElete <line>[,<line>]	will delete lines.
Position <line>	positions cursor to line.
Tabs Repetitive <incr>	Sets tabs at columns 0, incr, 2 * incr, 3 * incr, etc.
Tabs Non-repetitive (<column>)	Sets tab at entered columns. (Columns separated by spaces).
Tabs	Clear all tabs.
T+	Enable display of tab markers.
T-	Disable display of tab markers.
?	Display version, filename, and line and memory status. Enter any key to exit.

When loading a new file or exiting the editor you will be prompted to save the current buffer if it has been modified since last saved. Any response other than an 'N' or 'n' will save the old buffer. A response of 'C' or 'c' will allow you to change the file name before saving.

Non-buffered commands (see page 19-9)

		original version
Quit	Exit editor.	
4	move cursor left one character.	J
6	move cursor right one character.	L
8	move cursor up one line.	I
2	move cursor down one line.	,
5	alternate between left and right of line.	K
0	move cursor 1/2 screen up	O (letter)
.	move cursor 1/2 screen down	
space	move cursor to next identifier.	
HT or cntrl T	move cursor to next tab.	
cntrl R	move cursor backwards to previous tab.	
cntrl S	single character delete. deleting the carriage return will merge lines, with multiple leading spaces compressed into one space.	
cntrl X	delete current line.	

		original version
9	Overwrite one character.	1 (digit one)
3	overwrite characters until escape.	V
1	enter append mode.	A
7	enter insert mode.	;

H-19

Text search commands

Find _____ will position cursor, response is
- CR = stop command
- Space = continue
- M = multiple occurrences on line

Replace _____ will position cursor, response is
- CR = stop command
- Space = continue
- R = Replace and continue
- M = multiple occurrences on line
- E = entire buffer

Note : When the 'F' prompt appears enter the find pattern - if you enter just a carriage return then this will terminate the command. When the 'R' prompt appears enter the Replace pattern. A '~' is used as a don't care in the find and Replace patterns. The entire buffer option will enable the replacement throughout the rest of the buffer. The multiple occurrences option will allow a match on each occurence of the find pattern on any one line, else only the first match on a line is recognized. The search will start on the current line, continue to the end of the buffer, start back at the top, and stop on the current line. The cursor will be positioned at the last match when the command terminates.

Insert or append mode

Character entry until an escape is entered. Insert mode will insert characters before the current cursor position. If a carriage return is entered during insert mode then append mode will be entered with the last part of the line carried over until append mode is exited. Append mode will start new lines past the current line with no partial line carried over. In addition the following special characters are recognized :

BS or DEL	backspaces over character.
HT or cntrl T	forward tab.
cntrl R	reverse tab.
cntrl A	copies remainder of last line entered.
cntrl X	removes all characters entered so far.

tor

DS

ist, digits = offset or

buffer and read file.
buffer and set filename.
le past cursor.
ntire file.
ntire file and quit.

art of file.
ove lines to after
cursor position.
opy lines to after
cursor position.
elete lines.
ons cursor to line.
abs at columns 0, incr,
cr, 3 * incr, etc.
ab at entered columns.
ns separated by spaces).
all tabs.
display of tab markers.
e display of tab markers.
y version, filename, and
nd memory status. Enter
y to exit.

the editor you will be
t has been modified since
'N' or 'n' will save the
l allow you to change the

original version

aracter. J
character. L
. I
ne. ,
and right of line. K
up 0 (letter)
down
ntifier.
.
o previous tab.
. deleting the carriage
s, with multiple leading
one space.
original version
. 1 (digit one)
til escape. V
A
;

ponse is
= stop command
ce = continue
multiple
occurrences on line

ponse is
= stop command
ce = continue
= Replace and continue
= multiple
occurrences on line
= entire buffer

nter the find pattern - if
n this will terminate the
enter the Replace pattern.
find and Replace patterns.
the replacement throughout
occurrences option will
he find pattern on any one
line is recognized. The
continue to the end of the
p on the current line. The
t match when the command

ntered. Insert mode will
nt cursor position. If a
sert mode then append mode
he line carried over until
ll start new lines past the
ried over. In addition the
nized :

cter.

ast line entered.
s entered so far.

TABLE 4 — 8-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
ADCA, ADCB	Add memory to accumulator with carry
ADDA, ADDB	Add memory to accumulator
ANDA, ANDB	And memory with accumulator
ASL, ASLA, ASLB	Arithmetic shift of accumulator or memory left
ASR, ASRA, ASRB	Arithmetic shift of accumulator or memory right
BITA, BITB	Bit test memory with accumulator
CLR, CLRA, CLRB	Clear accumulator or memory location
CMPA, CMPB	Compare memory from accumulator
COM, COMA, COMB	Complement accumulator or memory location
DAA	Decimal adjust A accumulator
DEC, DECA, DECB	Decrement accumulator or memory location
EORA, EORB	Exclusive or memory with accumulator
EXG R1, R2	Exchange R1 with R2 (R1, R2 = A, B, CC, DP)
INC, INCA, INCB	Increment accumulator or memory location
LDA, LDB	Load accumulator from memory
LSL, LSLA, LSLB	Logical shift left accumulator or memory location
LSR, LSRA, LSRB	Logical shift right accumulator or memory location
MUL	Unsigned multiply (A x B → D)
NEG, NEGA, NEGB	Negate accumulator or memory
ORA, ORB	Or memory with accumulator
ROL, ROLA, ROLB	Rotate accumulator or memory left
ROR, RORA, RORB	Rotate accumulator or memory right
SBCA, SBCB	Subtract memory from accumulator with borrow
STA, STB	Store accumulator to memory
SUBA, SUBB	Subtract memory from accumulator
TST, TSTA, TSTB	Test accumulator or memory location
TFR R1, R2	Transfer R1 to R2 (R1, R2 = A, B, CC, DP)

NOTE: A, B, CC, or DP may be pushed to (pulled from) stack with either PSHS, PSHU (PULS, PULU) instructions.

TABLE 5 — 16-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

Mnemonic(s)	Operation
ADDD	Add memory to D accumulator
CMPD	Compare memory from D accumulator
EXG D, R	Exchange D with X, Y, S, U, or PC
LDD	Load D accumulator from memory
SEX	Sign Extend B accumulator into A accumulator
STD	Store D accumulator to memory
SUBD	Subtract memory from D accumulator
TFR D, R	Transfer D to X, Y, S, U, or PC
TFR R, D	Transfer X, Y, S, U, or PC to D

NOTE: D may be pushed (pulled) to stack with either PSHS, PSHU (PULS, PULU) instructions.

TABLE 6 — INDEX REGISTER/STACK POINTER INSTRUCTIONS

Instruction	Description
CMPS, CMPU	Compare memory from stack pointer
CMPX, CMPY	Compare memory from index register
EXG R1, R2	Exchange D, X, Y, S, U, or PC with D, X, Y, S, U, or PC
LEAS, LEAU	Load effective address into stack pointer
LEAX, LEAY	Load effective address into index register
LDS, LDU	Load stack pointer from memory
LDX, LDY	Load index register from memory
PSHS	Push A, B, CC, DP, D, X, Y, U, or PC onto hardware stack
PSHU	Push A, B, CC, DP, D, X, Y, S, or PC onto user stack
PULS	Pull A, B, CC, DP, D, X, Y, U, or PC from hardware stack
PULU	Pull A, B, CC, DP, D, X, Y, S, or PC from hardware stack
STS, STU	Store stack pointer to memory
STX, STY	Store index register to memory
TFR R1, R2	Transfer D, X, Y, S, U or PC to D, X, Y, S, U, or PC
ABX	Add B accumulator to X (unsigned)

TABLE 7 — BRANCH INSTRUCTIONS

Instruction	Description
SIMPLE BRANCHES	
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BMI, LBMI	Branch if minus
BPL, LBPL	Branch if plus
BCS, LBCS	Branch if carry set
BCC, LBCC	Branch if carry clear
BVS, LBVS	Branch if overflow set
BVC, LBVC	Branch if overflow clear
SIGNED BRANCHES	
BGT, LBGT	Branch if greater (signed)
BVS, LBVS	Branch if invalid 2s complement result
BGE, LBGE	Branch if greater than or equal (signed)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLE, LBLE	Branch if less than or equal (signed)
BVC, LBVC	Branch if valid 2s complement result
BLT, LBLT	Branch if less than (signed)
UNSIGNED BRANCHES	
BHI, LBHI	Branch if higher (unsigned)
BCC, LBCC	Branch if higher or same (unsigned)
BHS, LBHS	Branch if higher or same (unsigned)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLS, LBLS	Branch if lower or same (unsigned)
BCS, LBCS	Branch if lower (unsigned)
BLO, LBLO	Branch if lower (unsigned)
OTHER BRANCHES	
BSR, LBSR	Branch to subroutine
BRA, LBRA	Branch always
BRN, LBRN	Branch never

TABLE 8 — MISCELLANEOUS INSTRUCTIONS

Instruction	Description
ANDCC	AND condition code register
CWAI	AND condition code register, then wait for interrupt
NOP	No operation
ORCC	OR condition code register
JMP	Jump
JSR	Jump to subroutine
RTI	Return from interrupt
RTS	Return from subroutine
SWI, SWI2, SWI3	Software interrupt (absolute indirect)
SYNC	Synchronize with interrupt line

FIGURE 5 — CONDITION CODE REGISTER FORMAT

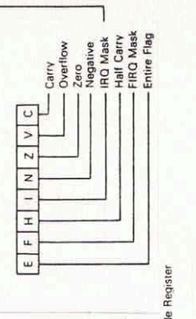


FIGURE 4 — PROGRAMMING MODEL OF THE MICROPROCESSING UNIT

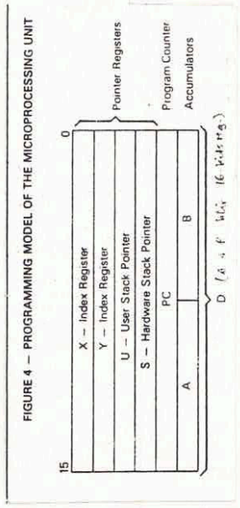


TABLE 3 — LEA EXAMPLES

Instruction	Operation	Comment
LEAX	10, X	X + 10 → X	Adds 5-Bit Constant 10 to X
LEAX	500, X	X + 500 → X	Adds 16-Bit Constant 500 to X
LEAY	A, Y	Y + A → Y	Adds 8-Bit A Accumulator to Y
LEAY	D, Y	Y + D → Y	Adds 16-Bit D Accumulator to Y
LEAU	–10, U	U – 10 → U	Subtracts 10 from U
LEAS	–10, S	S – 10 → S	Used to Reserve Area on Stack
LEAS	10, S	S + 10 → S	Used to 'Clean Up' Stack
LEAX	5, S	S + 5 → X	Transfers As Well As Adds

Stacking Order
Pull Order
↓
CC
A
B
DP
X Hi
X Lo
Y Hi
Y Lo
U/S Hi
U/S Lo
PC Hi
PC Lo
↑
Push Order

Increasing Memory ↓

Push/Pull Postbyte

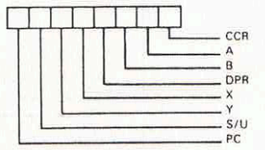


TABLE 9 — HEXADECIMAL VALUES OF MACHINE CODES

OP	Mnem	Mode	~	#
00	NEG	Direct	6	2
01	*			
02	*			
03	COM		6	2
04	LSR		6	2
05	*			
06	ROR		6	2
07	ASR		6	2
08	ASL, LSL		6	2
09	ROL		6	2
0A	DEC		6	2
0B	*			
0C	INC		6	2
0D	TST		6	2
0E	JMP		3	2
0F	CLR		6	2
10	Page 2	–	–	–
11	Page 3	–	–	–
12	NOP	Inherent	2	1
13	SYNC	Inherent	≥4	1
14	*			
15	*			
16	LBRA	Relative	5	3
17	LBSR	Relative	9	3
18	*			
19	DAA	Inherent	2	1
1A	ORCC	Immed	3	2
1B	*			
1C	ANDCC	Immed	3	2
1D	SEX	Inherent	2	1
1E	EXG	Immed	8	2
1F	TFR	Immed	6	2
20	BRA	Relative	3	2
21	BRN		3	2
22	BHI		3	2
23	BLS		3	2
24	BHS, BCC		3	2
25	BLO, BCS		3	2
26	BNE		3	2
27	BEQ		3	2
28	BVC		3	2
29	BVS		3	2
2A	BPL		3	2
2B	BMI		3	2
2C	BGE		3	2
2D	BLT		3	2
2E	BGT		3	2
2F	BLE	Relative	3	2
30	LEAX	Indexed	4+	2+
31	LEAY		4+	2+
32	LEAS		4+	2+
33	LEAU	Indexed	4+	2+
34	PSHS	Immed	5+	2
35	PULS	Immed	5+	2
36	PSHU	Immed	5+	2
37	PULU	Immed	5+	2
38	*			
39	RTS	Inherent	5	1
3A	ABX		3	1
3B	RTI		6/15	1
3C	CWAI		≥20	2
3D	MUL	Inherent	11	1
3E	*			
3F	SWI	Inherent	19	1
40	NEGA	Inherent	2	1
41	*			
42	*			
43	COMA		2	1
44	LSRA		2	1
45	*			
46	RORA		2	1
47	ASRA		2	1
48	ASLA, LSLA		2	1
49	ROLA		2	1
4A	DECA		2	1
4B	*			
4C	INCA		2	1
4D	TSTA		2	1
4E	*			
4F	CLRA	Inherent	2	1
50	NEGB	Inherent	2	1
51	*			
52	*			
53	COMB		2	1
54	LSRB		2	1
55	*			
56	RORB		2	1
57	ASRB		2	1
58	ASLB, LSLB		2	1
59	ROLB		2	1
5A	DECB		2	1
5B	*			
5C	INCB		2	1
5D	TSTB		2	1
5E	*			
5F	CLRB	Inherent	2	1
60	NEG	Indexed	6+	2+
61	*			
62	*			
63	COM		6+	2+
64	LSR		6+	2+
65	*			
66	ROR		6+	2+
67	ASR		6+	2+
68	ASL, LSL		6+	2+
69	ROL		6+	2+
6A	DEC		6+	2+
6B	*			
6C	INC		6+	2+
6D	TST		6+	2+
6E	JMP		3+	2+
6F	CLR	Indexed	6+	2+
70	NEG	Extended	7	3
71	*			
72	*			
73	COM		7	3
74	LSR		7	3
75	*			
76	ROR		7	3
77	ASR		7	3
78	ASL, LSL		7	3
79	ROL		7	3
7A	DEC		7	3
7B	*			
7C	INC		7	3
7D	TST		7	3
7E	JMP		4	3
7F	CLR	Extended	7	3
80	SUBA	Immed	2	2
81	CMPA		2	2
82	SBCA		2	2
83	SUBD		4	3
84	ANDA		2	2
85	BITA		2	2
86	LDA		2	2
87	*			
88	EORA		2	2
89	ADCA		2	2
8A	ORA		2	2
8B	ADDA		2	2
8C	CMPX	Immed	4	3
8D	BSR	Relative	7	2
8E	LDX	Immed	3	3
8F	*			
90	SUBA	Direct	4	2
91	CMPA		4	2
92	SBCA		4	2
93	SUBD		6	2
94	ANDA		4	2
95	BITA		4	2
96	LDA		4	2
97	STA		4	2
98	EORA		4	2
99	ADCA		4	2
9A	ORA		4	2
9B	ADDA		4	2
9C	CMPX		6	2
9D	JSR		7	2
9E	LDX		5	2
9F	STX	Direct	5	2
A0	SUBA	Indexed	4+	2+
A1	CMPA		4+	2+
A2	SBCA		4+	2+
A3	SUBD		6+	2+
A4	ANDA		4+	2+
A5	BITA		4+	2+
A6	LDA		4+	2+
A7	STA		4+	2+
A8	EORA		4+	2+
A9	ADCA		4+	2+
AA	ORA		4+	2+
AB	ADDA		4+	2+
AC	CMPX		6+	2+
AD	JSR		7+	2+
AE	LDX		5+	2+
AF	STX	Indexed	5+	2+
B0	SUBA	Extended	5	3
B1	CMPA		5	3
B2	SBCA		5	3
B3	SUBD		7	3
B4	ANDA		5	3
B5	BITA		5	3
B6	LDA		5	3
B7	STA		5	3
B8	EORA		5	3
B9	ADCA		5	3
BA	ORA		5	3
BB	ADDA		5	3
BC	CMPX		7	3
BD	JSR		8	3
BE	LDX		6	3
BF	STX	Extended	6	3
C0	SUBB	Immed	2	2
C1	CMPB		2	2
C2	SBCB		2	2
C3	ADDD		4	3
C4	ANDB		2	2
C5	BITB	Immed	2	2
C6	LDB	Immed	2	2
C7	*			
C8	EORB		2	2
C9	ADCB		2	2
CA	ORB		2	2
CB	ADDB		2	2
CC	LDD		3	3
CD	*			
CE	LDU	Immed	3	3
CF	*			
D0	SUBB	Direct	4	2
D1	CMPB		4	2
D2	SBCB		4	2
D3	ADDD		6	2
D4	ANDB		4	2
D5	BITB		4	2
D6	LDB		4	2
D7	STB		4	2
D8	EORB		4	2
D9	ADCB		4	2
DA	ORB		4	2
DB	ADDB		4	2
DC	LDD		5	2
DD	STD		5	2
DE	LDU		5	2
DF	STU	Direct	5	2
E0	SUBB	Indexed	4+	2+
E1	CMPB		4+	2+
E2	SBCB		4+	2+
E3	ADDD		6+	2+
E4	ANDB		4+	2+
E5	BITB		4+	2+
E6	LDB		4+	2+
E7	STB		4+	2+
E8	EORB		4+	2+
E9	ADCB		4+	2+
EA	ORB		4+	2+
EB	ADDB		4+	2+
EC	LDD		5+	2+
ED	STD		5+	2+
EE	LDU		5+	2+
EF	STU	Indexed	5+	2+
F0	SUBB	Extended	5	3
F1	CMPB		5	3
F2	SBCB		5	3
F3	ADDD		7	3
F4	ANDB		5	3
F5	BITB		5	3
F6	LDB		5	3
F7	STB		5	3
F8	EORB		5	3
F9	ADCB		5	3
FA	ORB		5	3
FB	ADDB		5	3
FC	LDD		6	3
FD	STD		6	3
FE	LDU		6	3
FF	STU	Extended	6	3

NOTE All unused opcodes are both undefined and illegal

Page 2 and 3 Machine Codes

1021	LBRN	Relative	5	4
1022	LBHI		5(6)	4
1023	LBLS		5(6)	4
1024	LBHS, LBCC		5(6)	4
1025	LBCS, LBLO		5(6)	4
1026	LBNE		5(6)	4
1027	LBEQ		5(6)	4
1028	LBVC		5(6)	4
1029	LBVS		5(6)	4
102A	LBPL		5(6)	4
102B	LBMI		5(6)	4
102C	LBGE		5(6)	4
102D	LBLT		5(6)	4
102E	LBGT		5(6)	4
102F	LBLE	Relative	5(6)	4
103F	SWI2	Inherent	20	2
1083	CMPD	Immed	5	4
108C	CMPY		5	4
108E	LDY	Immed	4	3
1093	CMPD	Direct	7	3
109C	CMPY		7	3
109E	LDY	Direct	6	3
109F	STY		6	3
10A3	CMPD	Indexed	7+	3
10AC	CMPY		7+	3
10AE	LDY		6+	3
10AF	STY	Indexed	6+	3
10B3	CMPD	Extended	8	4
10BC	CMPY		8	4
10BE	LDY		7	4
10BF	STY	Extended	7	4
10CE	LDS	Immed	4	3
10DE	LDS	Direct	6	3
10DF	STS	Direct	6	3
10EE	LDS	Indexed	6+	3
10EF	STS	Indexed	6+	3
10FE	LDS	Extended	7	4
10FF	STS	Extended	7	4
113F	SWI3	Inherent	20	2
1183	CMPU	Immed	5	4
118C	CMPS	Immed	5	4
1193	CMPU	Direct	7	3
119C	CMPS	Direct	7	3
11A3	CMPU	Indexed	7+	3
11AC	CMPS	Indexed	7+	3
11B3	CMPU	Extended	8	4
11BC	CMPS	Extended	8	4

LEGEND:
~ Number of MPU cycles (less possible push pull or indexed-mode cycles)
Number of program bytes
* Denotes unused opcode

EXADECIMAL VALUES OF MACHINE CODES

Mnem	Mode	~	#	OP	Mnem	Mode	~	#
LEAX	Indexed	4+	2+	60	NEG	Indexed	6+	2+
LEAY	↕	4+	2+	61	•	↕		
LEAS	↕	4+	2+	62	•	↕		
LEAU	Indexed	4+	2+	63	COM	↕	6+	2+
PSHS	Immed	5+	2	64	LSR	↕	6+	2+
PULS	Immed	5+	2	65	•	↕		
PSHU	Immed	5+	2	66	ROR	↕	6+	2+
PULU	Immed	5+	2	67	ASR	↕	6+	2+
•	–			68	ASL, LSL	↕	6+	2+
RTS	Inherent	5	1	69	ROL	↕	6+	2+
ABX	↕	3	1	6A	DEC	↕	6+	2+
RTI	↕	6/15	1	6B	•	↕		
CWAI	↕	≥20	2	6C	INC	↕	6+	2+
MUL	Inherent	11	1	6D	TST	↕	6+	2+
•	–			6E	JMP	↕	3+	2+
SWI	Inherent	19	1	6F	CLR	Indexed	6+	2+
NEGA	Inherent	2	1	70	NEG	Extended	7	3
•	↕			71	•	↕		
•	↕			72	•	↕		
COMA	↕	2	1	73	COM	↕	7	3
LSRA	↕	2	1	74	LSR	↕	7	3
•	↕			75	•	↕		
RORA	↕	2	1	76	ROR	↕	7	3
ASRA	↕	2	1	77	ASR	↕	7	3
ASLA, LSLA	↕	2	1	78	ASL, LSL	↕	7	3
ROLA	↕	2	1	79	ROL	↕	7	3
DECA	↕	2	1	7A	DEC	↕	7	3
•	↕			7B	•	↕		
INCA	↕	2	1	7C	INC	↕	7	3
TSTA	↕	2	1	7D	TST	↕	7	3
•	↕			7E	JMP	↕	4	3
CLRA	Inherent	2	1	7F	CLR	Extended	7	3
NEGB	Inherent	2	1	80	SUBA	Immed	2	2
•	↕			81	CMPA	↕	2	2
•	↕			82	SBCA	↕	2	2
COMB	↕	2	1	83	SUBD	↕	4	3
LSRB	↕	2	1	84	ANDA	↕	2	2
•	↕			85	BITA	↕	2	2
RORB	↕	2	1	86	LDA	↕	2	2
ASRB	↕	2	1	87	•	↕		
ASLB, LSLB	↕	2	1	88	EORA	↕	2	2
ROLB	↕	2	1	89	ADCA	↕	2	2
DECB	↕	2	1	8A	ORA	↕	2	2
•	↕			8B	ADDA	↕	2	2
INCB	↕	2	1	8C	CMPX	Immed	4	3
TSTB	↕	2	1	8D	BSR	Relative	7	2
•	↕			8E	LDX	Immed	3	3
CLRB	Inherent	2	1	8F	•			

Mnem	Mode	~	#
SUBB	Immed	2	2
CMPB	↕	2	2
SBCB	↕	2	2
ADDD	↕	4	3
ANDB	↕	2	2
BITB	Immed	2	2
LDB	Immed	2	2
•	↕		
EORB	↕	2	2
ADCB	↕	2	2
ORB	↕	2	2
ADDB	↕	2	2
LDD	↕	3	3
•	↕		
LDU	Immed	3	3
•			
SUBB	Direct	4	2
CMPB	↕	4	2
SBCB	↕	4	2
ADDD	↕	6	2
ANDB	↕	4	2
BITB	↕	4	2
LDB	↕	4	2
STB	↕	4	2
EORB	↕	4	2
ADCB	↕	4	2
ORB	↕	4	2
ADDB	↕	4	2
LDD	↕	5	2
STD	↕	5	2
LDU	↕	5	2
STU	Direct	5	2
SUBB	Indexed	4+	2+
CMPB	↕	4+	2+
SBCB	↕	4+	2+
ADDD	↕	6+	2+
ANDB	↕	4+	2+
BITB	↕	4+	2+
LDB	↕	4+	2+
STB	↕	4+	2+
EORB	↕	4+	2+
ADCB	↕	4+	2+
ORB	↕	4+	2+
ADDB	↕	4+	2+
LDD	↕	5+	2+
STD	↕	5+	2+
LDU	↕	5+	2+
STU	Indexed	5+	2+
SUBB	Extended	5	3
CMPB	↕	5	3
SBCB	↕	5	3
ADDD	↕	7	3
ANDB	↕	5	3
BITB	↕	5	3
LDB	↕	5	3
STB	↕	5	3
EORB	↕	5	3
ADCB	↕	5	3
ORB	↕	5	3
ADDB	Extended	5	3
LDD	Extended	6	3
STD	↕	6	3
LDU	↕	6	3
STU	Extended	6	3

Page 2 and 3 Machine Codes

OP	Mnem	Mode	~	#
1021	LBRN	Relative	5	4
1022	LBHI	↕	5(6)	4
1023	LBLS	↕	5(6)	4
1024	LBHS, LBCC	↕	5(6)	4
1025	LBCS, LBLO	↕	5(6)	4
1026	LBNE	↕	5(6)	4
1027	LBEQ	↕	5(6)	4
1028	LBVC	↕	5(6)	4
1029	LBVS	↕	5(6)	4
102A	LBPL	↕	5(6)	4
102B	LBMI	↕	5(6)	4
102C	LBGE	↕	5(6)	4
102D	LBLT	↕	5(6)	4
102E	LBGT	↕	5(6)	4
102F	LBLE	Relative	5(6)	4
103F	SWI2	Inherent	20	2
1083	CMPD	Immed	5	4
108C	CMPY	↕	5	4
108E	LDY	Immed	4	4
1093	CMPD	Direct	7	3
109C	CMPY	↕	7	3
109E	LDY	↕	6	3
109F	STY	Direct	6	3
10A3	CMPD	Indexed	7+	3+
10AC	CMPY	↕	7+	3+
10AE	LDY	↕	6+	3+
10AF	STY	Indexed	6+	3+
10B3	CMPD	Extended	8	4
10BC	CMPY	↕	8	4
10BE	LDY	↕	7	4
10BF	STY	Extended	7	4
10CE	LDS	Immed	4	4
10DE	LDS	Direct	6	3
10DF	STS	Direct	6	3
10EE	LDS	Indexed	6+	3+
10EF	STS	Indexed	6+	3+
10FE	LDS	Extended	7	4
10FF	STS	Extended	7	4
113F	SWI3	Inherent	20	2
1183	CMPU	Immed	5	4
118C	CMPS	Immed	5	4
1193	CMPU	Direct	7	3
119C	CMPS	Direct	7	3
11A3	CMPU	Indexed	7+	3+
11AC	CMPS	Indexed	7+	3+
11B3	CMPU	Extended	8	4
11BC	CMPS	Extended	8	4

pull or indexed-mode cycles)

FIGURE 19 — PROGRAMMING AID

Instruction	Forms	Immediate Op	~	#	Direct Op	~	#	Indexed Op	~	#	Extended Op	~	#	Inherent Op	~	#	Description	5 H	3 N	2 Z	1 V	0 C
ABX														3A	3	1	B + X → X (Unsigned)	•	•	•	•	•
ADC	ADCA	89	2	2	99	4	2	A9	4+	2+	B9	5	3				A + M + C → A	↕	↕	↕	↕	↕
	ADCB	C9	2	2	D9	4	2	E9	4+	2+	F9	5	3				B + M + C → B	↕	↕	↕	↕	↕
ADD	ADDA	8B	2	2	9B	4	2	AB	4+	2+	BB	5	3				A + M → A	↕	↕	↕	↕	↕
	ADDB	CB	2	2	DB	4	2	EB	4+	2+	FB	5	3				B + M → B	↕	↕	↕	↕	↕
	ADDD	C3	4	3	D3	6	2	E3	6+	2+	F3	7	3				D + M:M + 1 → D	•	↕	↕	↕	↕
AND	ANDA	84	2	2	94	4	2	A4	4+	2+	B4	5	3				A ∧ M → A	•	↕	↕	0	•
	ANDB	C4	2	2	D4	4	2	E4	4+	2+	F4	5	3				B ∧ M → B	•	↕	↕	0	•
	ANDCC	1C	3	2													CC ∧ IMM → CC					7
ASL	ASLA													48	2	1	A, B, M: C ← b7 … b0 ← 0	8	↕	↕	↕	↕
	ASLB													58	2	1		8	↕	↕	↕	↕
	ASL				08	6	2	68	6+	2+	78	7	3					8	↕	↕	↕	↕
ASR	ASRA													47	2	1	A, B, M: b7 → b7 … b0 → C	8	↕	↕	•	↕
	ASRB													57	2	1		8	↕	↕	•	↕
	ASR				07	6	2	67	6+	2+	77	7	3					8	↕	↕	•	↕
BIT	BITA	85	2	2	95	4	2	A5	4+	2+	B5	5	3				Bit Test A (M ∧ A)	•	↕	↕	0	•
	BITB	C5	2	2	D5	4	2	E5	4+	2+	F5	5	3				Bit Test B (M ∧ B)	•	↕	↕	0	•
CLR	CLRA													4F	2	1	0 → A	•	0	1	0	0
	CLRB													5F	2	1	0 → B	•	0	1	0	0
	CLR				0F	6	2	6F	6+	2+	7F	7	3				0 → M	•	0	1	0	0
CMP	CMPA	81	2	2	91	4	2	A1	4+	2+	B1	5	3				Compare M from A	8	↕	↕	↕	↕
	CMPB	C1	2	2	D1	4	2	E1	4+	2+	F1	5	3				Compare M from B	8	↕	↕	↕	↕
	CMPD	10 83	5	4	10 93	7	3	10 A3	7+	3+	10 B3	8	4				Compare M:M + 1 from D	•	↕	↕	↕	↕
	CMPS	11 8C	5	4	11 9C	7	3	11 AC	7+	3+	11 BC	8	4				Compare M:M + 1 from S	•	↕	↕	↕	↕
	CMPU	11 83	5	4	11 93	7	3	11 A3	7+	3+	11 B3	8	4				Compare M:M + 1 from U	•	↕	↕	↕	↕
	CMPX	8C	4	3	9C	6	2	AC	6+	2+	BC	7	3				Compare M:M + 1 from X	•	↕	↕	↕	↕
	CMPY	10 8C	5	4	10 9C	7	3	10 AC	7+	3+	10 BC	8	4				Compare M:M + 1 from Y	•	↕	↕	↕	↕
COM	COMA													43	2	1	Ā → A	•	↕	↕	0	1
	COMB													53	2	1	B̄ → B	•	↕	↕	0	1
	COM				03	6	2	63	6+	2+	73	7	3				M̄ → M	•	↕	↕	0	1
CWAI		3C	≥20	2													CC ∧ IMM → CC Wait for Interrupt					7
DAA														19	2	1	Decimal Adjust A	•	↕	↕	0	↕
DEC	DECA													4A	2	1	A − 1 → A	•	↕	↕	↕	•
	DECB													5A	2	1	B − 1 → B	•	↕	↕	↕	•
	DEC				0A	6	2	6A	6+	2+	7A	7	3				M − 1 → M	•	↕	↕	↕	•
EOR	EORA	88	2	2	98	4	2	A8	4+	2+	B8	5	3				A ⊻ M → A	•	↕	↕	0	•
	EORB	C8	2	2	D8	4	2	E8	4+	2+	F8	5	3				B ⊻ M → B	•	↕	↕	0	•
EXG	R1, R2	1E	8	2													R1 ↔ R2[2]	•	•	•	•	•
INC	INCA													4C	2	1	A + 1 → A	•	↕	↕	↕	•
	INCB													5C	2	1	B + 1 → B	•	↕	↕	↕	•
	INC				0C	6	2	6C	6+	2+	7C	7	3				M + 1 → M	•	↕	↕	↕	•
JMP					0E	3	2	6E	3+	2+	7E	4	3				EA[3] → PC	•	•	•	•	•
JSR					9D	7	2	AD	7+	2+	BD	8	3				Jump to Subroutine	•	•	•	•	•
LD	LDA	86	2	2	96	4	2	A6	4+	2+	B6	5	3				M → A	•	↕	↕	0	•
	LDB	C6	2	2	D6	4	2	E6	4+	2+	F6	5	3				M → B	•	↕	↕	0	•
	LDD	CC	3	3	DC	5	2	EC	5+	2+	FC	6	3				M:M + 1 → D	•	↕	↕	0	•
	LDS	10 CE	4	4	10 DE	6	3	10 EE	6+	3+	10 FE	7	4				M:M + 1 → S	•	↕	↕	0	•
	LDU	CE	3	3	DE	5	2	EE	5+	2+	FE	6	3				M:M + 1 → U	•	↕	↕	0	•
	LDX	8E	3	3	9E	5	2	AE	5+	2+	BE	6	3				M:M + 1 → X	•	↕	↕	0	•
	LDY	10 8E	4	4	10 9E	6	3	10 AE	6+	3+	10 BE	7	4				M:M + 1 → Y	•	↕	↕	0	•
LEA	LEAS							32	4+	2+							EA[3] → S	•	•	•	•	•
	LEAU							33	4+	2+							EA[3] → U	•	•	•	•	•
	LEAX							30	4+	2+							EA[3] → X	•	•	↕	•	•
	LEAY							31	4+	2+							EA[3] → Y	•	•	↕	•	•
LSL	LSLA													48	2	1	A, B, M: C ← b7 … b0 ← 0	•	↕	↕	↕	↕
	LSLB													58	2	1		•	↕	↕	↕	↕
	LSL				08	6	2	68	6+	2+	78	7	3					•	↕	↕	↕	↕
LSR	LSRA													44	2	1	A, B, M: 0 → b7 … b0 → C	•	0	↕	•	↕
	LSRB													54	2	1		•	0	↕	•	↕
	LSR				04	6	2	64	6+	2+	74	7	3					•	0	↕	•	↕
MUL														3D	11	1	A × B → D (Unsigned)	•	•	↕	•	9
NEG	NEGA													40	2	1	Ā + 1 → A	8	↕	↕	↕	↕
	NEGB													50	2	1	B̄ + 1 → B	8	↕	↕	↕	↕
	NEG				00	6	2	60	6+	2+	70	7	3				M̄ + 1 → M	8	↕	↕	↕	↕
NOP														12	2	1	No Operation	•	•	•	•	•
OR	ORA	8A	2	2	9A	4	2	AA	4+	2+	BA	5	3				A ∨ M → A	•	↕	↕	0	•
	ORB	CA	2	2	DA	4	2	EA	4+	2+	FA	5	3				B ∨ M → B	•	↕	↕	0	•
	ORCC	1A	3	2													CC ∨ IMM → CC					7
PSH	PSHS	34	5+[4]	2													Push Registers on S Stack	•	•	•	•	•
	PSHU	36	5+[4]	2													Push Registers on U Stack	•	•	•	•	•
PUL	PULS	35	5+[4]	2													Pull Registers from S Stack	•	•	•	•	•
	PULU	37	5+[4]	2													Pull Registers from U Stack	•	•	•	•	•
ROL	ROLA													49	2	1	A, B, M: C ← b7 … b0 ← C	•	↕	↕	↕	↕
	ROLB													59	2	1		•	↕	↕	↕	↕
	ROL				09	6	2	69	6+	2+	79	7	3					•	↕	↕	↕	↕
ROR	RORA													46	2	1	A, B, M: C → b7 … b0 → C	•	↕	↕	•	↕
	RORB													56	2	1		•	↕	↕	•	↕
	ROR				06	6	2	66	6+	2+	76	7	3					•	↕	↕	•	↕
RTI														3B	6/15	1	Return From Interrupt					7
RTS														39	5	1	Return from Subroutine	•	•	•	•	•
SBC	SBCA	82	2	2	92	4	2	A2	4+	2+	B2	5	3				A − M − C → A	8	↕	↕	↕	↕
	SBCB	C2	2	2	D2	4	2	E2	4+	2+	F2	5	3				B − M − C → B	8	↕	↕	↕	↕
SEX														1D	2	1	Sign Extend B into A	•	↕	↕	0	•
ST	STA				97	4	2	A7	4+	2+	B7	5	3				A → M	•	↕	↕	0	•
	STB				D7	4	2	E7	4+	2+	F7	5	3				B → M	•	↕	↕	0	•
	STD				DD	5	2	ED	5+	2+	FD	6	3				D → M:M + 1	•	↕	↕	0	•
	STS				10 DF	6	3	10 EF	6+	3+	10 FF	7	4				S → M:M + 1	•	↕	↕	0	•
	STU				DF	5	2	EF	5+	2+	FF	6	3				U → M:M + 1	•	↕	↕	0	•
	STX				9F	5	2	AF	5+	2+	BF	6	3				X → M:M + 1	•	↕	↕	0	•
	STY				10 9F	6	3	10 AF	6+	3+	10 BF	7	4				Y → M:M + 1	•	↕	↕	0	•
SUB	SUBA	80	2	2	90	4	2	A0	4+	2+	B0	5	3				A − M → A	8	↕	↕	↕	↕
	SUBB	C0	2	2	D0	4	2	E0	4+	2+	F0	5	3				B − M → B	8	↕	↕	↕	↕
	SUBD	83	4	3	93	6	2	A3	6+	2+	B3	7	3				D − M:M + 1 → D	•	↕	↕	↕	↕
SWI	SWI[6]													3F	19	1	Software Interrupt 1	•	•	•	•	•
	SWI2[6]													10 3F	20	2	Software Interrupt 2	•	•	•	•	•
	SWI3[6]													11 3F	20	1	Software Interrupt 3	•	•	•	•	•
SYNC														13	≥4	1	Synchronize to Interrupt	•	•	•	•	•
TFR	R1, R2	1F	6	2													R1 → R2[2]	•	•	•	•	•
TST	TSTA													4D	2	1	Test A	•	↕	↕	0	•
	TSTB													5D	2	1	Test B	•	↕	↕	0	•
	TST				0D	6	2	6D	6+	2+	7D	7	3				Test M	•	↕	↕	0	•

NOTES:
1. This column gives a base cycle and byte count. To obtain total count, add the values obtained from the INDEXED ADDRESSING MODE table, Table 2.
2. R1 and R2 may be any pair of 8 bit or any pair of 16 bit registers.
   The 8 bit registers are: A, B, CC, DP
   The 16 bit registers are: X, Y, U, S, D, PC
3. EA is the effective address.
4. The PSH and PUL instructions require 5 cycles plus 1 cycle for each byte pushed or pulled.
5. 5(6) means: 5 cycles if branch not taken, 6 cycles if taken (Branch instructions).
6. SWI sets I and F bits. SWI2 and SWI3 do not affect I and F.
7. Conditions Codes set as a direct result of the instruction.
8. Vaue of half-carry flag is undefined.
9. Special Case — Carry set if b7 is SET

URE 19 — PROGRAMMING AID

Indexed ~	#	Extended Op	~	#	Inherent Op	~	#	Description	H	N	Z	V	C
					3A	3	1	B + X → X (Unsigned)	•	•	•	•	•
4+	2+	B9	5	3				A + M + C → A	↕	↕	↕	↕	↕
4+	2+	F9	5	3				B + M + C → B	↕	↕	↕	↕	↕
4+	2+	BB	5	3				A + M → A	↕	↕	↕	↕	↕
4+	2+	FB	5	3				B + M → B	↕	↕	↕	↕	↕
6+	2+	F3	7	3				D + M:M + 1 → D	•	↕	↕	↕	↕
4+	2+	B4	5	3				A ∧ M → A	•	↕	↕	0	•
4+	2+	F4	5	3				B ∧ M → B	•	↕	↕	0	•
								CC ∧ IMM → CC					7
					48	2	1	A, B, M: C ← b7 … b0 ← 0	8	↕	↕	↕	↕
					58	2	1		8	↕	↕	↕	↕
6+	2+	78	7	3					8	↕	↕	↕	↕
					47	2	1	A, B, M: b7 → b7 … b0 → C	8	↕	↕	•	↕
					57	2	1		8	↕	↕	•	↕
6+	2+	77	7	3					8	↕	↕	•	↕
4+	2+	B5	5	3				Bit Test A (M ∧ A)	•	↕	↕	0	•
4+	2+	F5	5	3				Bit Test B (M ∧ B)	•	↕	↕	0	•
					4F	2	1	0 → A	•	0	1	0	0
					5F	2	1	0 → B	•	0	1	0	0
6+	2+	7F	7	3				0 → M	•	0	1	0	0
4+	2+	B1	5	3				Compare M from A	8	↕	↕	↕	↕
4+	2+	F1	5	3				Compare M from B	8	↕	↕	↕	↕
7+	3+	10 B3	8	4				Compare M:M + 1 from D	•	↕	↕	↕	↕
7+	3+	11 BC	8	4				Compare M:M + 1 from S	•	↕	↕	↕	↕
7+	3+	11 B3	8	4				Compare M:M + 1 from U	•	↕	↕	↕	↕
6+	2+	BC	7	3				Compare M:M + 1 from X	•	↕	↕	↕	↕
7+	3+	10 BC	8	4				Compare M:M + 1 from Y	•	↕	↕	↕	↕
					43	2	1	Ā → A	•	↕	↕	0	1
					53	2	1	B̄ → B	•	↕	↕	0	1
6+	2+	73	7	3				M̄ → M	•	↕	↕	0	1
								CC ∧ IMM → CC Wait for Interrupt					7
					19	2	1	Decimal Adjust A	•	↕	↕	0	↕
					4A	2	1	A − 1 → A	•	↕	↕	↕	•
					5A	2	1	B − 1 → B	•	↕	↕	↕	•
6+	2+	7A	7	3				M − 1 → M	•	↕	↕	↕	•
4+	2+	B8	5	3				A ⊻ M → A	•	↕	↕	0	•
4+	2+	F8	5	3				B ⊻ M → B	•	↕	↕	0	•
								R1 ↔ R2[2]	•	•	•	•	•
					4C	2	1	A + 1 → A	•	↕	↕	↕	•
					5C	2	1	B + 1 → B	•	↕	↕	↕	•
6+	2+	7C	7	3				M + 1 → M	•	↕	↕	↕	•
3+	2+	7E	4	3				EA[3] → PC	•	•	•	•	•
7+	2+	BD	8	3				Jump to Subroutine	•	•	•	•	•
4+	2+	B6	5	3				M → A	•	↕	↕	0	•
4+	2+	F6	5	3				M → B	•	↕	↕	0	•
5+	2+	FC	6	3				M:M + 1 → D	•	↕	↕	0	•
6+	3+	10 FE	7	4				M:M + 1 → S	•	↕	↕	0	•
5+	2+	FE	6	3				M:M + 1 → U	•	↕	↕	0	•
5+	2+	BE	6	3				M:M + 1 → X	•	↕	↕	0	•
6+	3+	10 BE	7	4				M:M + 1 → Y	•	↕	↕	0	•
4+	2+							EA[3] → S	•	•	•	•	•
4+	2+							EA[3] → U	•	•	•	•	•
4+	2+							EA[3] → X	•	•	↕	•	•
4+	2+							EA[3] → Y	•	•	↕	•	•
					48	2	1	A, B, M: C ← b7 … b0 ← 0	•	↕	↕	↕	↕
					58	2	1		•	↕	↕	↕	↕
6+	2+	78	7	3					•	↕	↕	↕	↕
					44	2	1	A, B, M: 0 → b7 … b0 → C	•	0	↕	•	↕
					54	2	1		•	0	↕	•	↕
6+	2+	74	7	3					•	0	↕	•	↕
					3D	11	1	A × B → D (Unsigned)	•	•	↕	•	9
					40	2	1	Ā + 1 → A	8	↕	↕	↕	↕
					50	2	1	B̄ + 1 → B	8	↕	↕	↕	↕
6+	2+	70	7	3				M̄ + 1 → M	8	↕	↕	↕	↕
					12	2	1	No Operation	•	•	•	•	•
4+	2+	BA	5	3				A ∨ M → A	•	↕	↕	0	•
4+	2+	FA	5	3				B ∨ M → B	•	↕	↕	0	•
								CC ∨ IMM → CC					7
								Push Registers on S Stack	•	•	•	•	•
								Push Registers on U Stack	•	•	•	•	•
								Pull Registers from S Stack	•	•	•	•	•
								Pull Registers from U Stack	•	•	•	•	•
					49	2	1	A, B, M: C ← b7 … b0 ← C	•	↕	↕	↕	↕
					59	2	1		•	↕	↕	↕	↕
6+	2+	79	7	3					•	↕	↕	↕	↕
					46	2	1	A, B, M: C → b7 … b0 → C	•	↕	↕	•	↕
					56	2	1		•	↕	↕	•	↕
6+	2+	76	7	3					•	↕	↕	•	↕
					3B	6/15	1	Return From Interrupt					7
					39	5	1	Return from Subroutine	•	•	•	•	•
4+	2+	B2	5	3				A − M − C → A	8	↕	↕	↕	↕
4+	2+	F2	5	3				B − M − C → B	8	↕	↕	↕	↕
					1D	2	1	Sign Extend B into A	•	↕	↕	0	•
4+	2+	B7	5	3				A → M	•	↕	↕	0	•
4+	2+	F7	5	3				B → M	•	↕	↕	0	•
5+	2+	FD	6	3				D → M:M + 1	•	↕	↕	0	•
6+	3+	10 FF	7	4				S → M:M + 1	•	↕	↕	0	•
5+	2+	FF	6	3				U → M:M + 1	•	↕	↕	0	•
5+	2+	BF	6	3				X → M:M + 1	•	↕	↕	0	•
6+	3+	10 BF	7	4				Y → M:M + 1	•	↕	↕	0	•
4+	2+	B0	5	3				A − M → A	8	↕	↕	↕	↕
4+	2+	F0	5	3				B − M → B	8	↕	↕	↕	↕
6+	2+	B3	7	3				D − M:M + 1 → D	•	↕	↕	↕	↕
					3F	19	1	Software Interrupt 1	•	•	•	•	•
					10 3F	20	2	Software Interrupt 2	•	•	•	•	•
					11 3F	20	1	Software Interrupt 3	•	•	•	•	•
					13	≥4	1	Synchronize to Interrupt	•	•	•	•	•
								R1 → R2[2]	•	•	•	•	•
					4D	2	1	Test A	•	↕	↕	0	•
					5D	2	1	Test B	•	↕	↕	0	•
6+	2+	7D	7	3				Test M	•	↕	↕	0	•

o obtain total count, add the values obtained from the INDEXED ADDRESSING MODE table.
of 16 bit registers.
s plus 1 cycle for each byte pushed or pulled.
cles if taken (Branch instructions).
affect I and F.
nstruction.

FIGURE 19 — PROGRAMMING AID (CONTINUED)

Branch Instructions

Instruction	Forms	OP	~[5]	#	Description	H	N	Z	V	C
BCC	BCC	24	3	2	Branch C = 0	•	•	•	•	•
	LBCC	10 24	5(6)	4	Long Branch C = 0	•	•	•	•	•
BCS	BCS	25	3	2	Branch C = 1	•	•	•	•	•
	LBCS	10 25	5(6)	4	Long Branch C = 1	•	•	•	•	•
BEQ	BEQ	27	3	2	Branch Z = 1	•	•	•	•	•
	LBEQ	10 27	5(6)	4	Long Branch Z = 0	•	•	•	•	•
BGE	BGE	2C	3	2	Branch ≥ Zero	•	•	•	•	•
	LBGE	10 2C	5(6)	4	Long Branch ≥ Zero	•	•	•	•	•
BGT	BGT	2E	3	2	Branch > Zero	•	•	•	•	•
	LBGT	10 2E	5(6)	4	Long Branch > Zero	•	•	•	•	•
BHI	BHI	22	3	2	Branch Higher	•	•	•	•	•
	LBHI	10 22	5(6)	4	Long Branch Higher	•	•	•	•	•
BHS	BHS	24	3	2	Branch Higher or Same	•	•	•	•	•
	LBHS	10 24	5(6)	4	Long Branch Higher or Same	•	•	•	•	•
BLE	BLE	2F	3	2	Branch ≤ Zero	•	•	•	•	•
	LBLE	10 2F	5(6)	4	Long Branch ≤ Zero	•	•	•	•	•
BLO	BLO	25	3	2	Branch lower	•	•	•	•	•
	LBLO	10 25	5(6)	4	Long Branch Lower	•	•	•	•	•
BLS	BLS	23	3	2	Branch Lower or Same	•	•	•	•	•
	LBLS	10 23	5(6)	4	Long Branch Lower or Same	•	•	•	•	•
BLT	BLT	2D	3	2	Branch < Zero	•	•	•	•	•
	LBLT	10 2D	5(6)	4	Long Branch < Zero	•	•	•	•	•
BMI	BMI	2B	3	2	Branch Minus	•	•	•	•	•
	LBMI	10 2B	5(6)	4	Long Branch Minus	•	•	•	•	•
BNE	BNE	26	3	2	Branch Z = 0	•	•	•	•	•
	LBNE	10 26	5(6)	4	Long Branch Z ≠ 0	•	•	•	•	•
BPL	BPL	2A	3	2	Branch Plus	•	•	•	•	•
	LBPL	10 2A	5(6)	4	Long Branch Plus	•	•	•	•	•
BRA	BRA	20	3	2	Branch Always	•	•	•	•	•
	LBRA	16	5	3	Long Branch Always	•	•	•	•	•
BRN	BRN	21	3	2	Branch Never	•	•	•	•	•
	LBRN	10 21	5	4	Long Branch Never	•	•	•	•	•
BSR	BSR	8D	7	2	Branch to Subroutine	•	•	•	•	•
	LBSR	17	9	3	Long Branch to Subroutine	•	•	•	•	•
BVC	BVC	28	3	2	Branch V = 0	•	•	•	•	•
	LBVC	10 28	5(6)	4	Long Branch V = 0	•	•	•	•	•
BVS	BVS	29	3	2	Branch V = 1	•	•	•	•	•
	LBVS	10 29	5(6)	4	Long Branch V = 1	•	•	•	•	•

SIMPLE BRANCHES

	OP	~	#
BRA	20	3	2
LBRA	16	5	3
BRN	21	3	2
LBRN	1021	5	4
BSR	8D	7	2
LBSR	17	9	3

SIMPLE CONDITIONAL BRANCHES (Notes 1-4)

Test	True	OP	False	OP
N = 1	BMI	2B	BPL	2A
Z = 1	BEQ	27	BNE	26
V = 1	BVS	29	BVC	28
C = 1	BCS	25	BCC	24

SIGNED CONDITIONAL BRANCHES (Notes 1-4)

Test	True	OP	False	OP
r > m	BGT	2E	BLE	2F
r ≥ m	BGE	2C	BLT	2D
r = m	BEQ	27	BNE	26
r ≤ m	BLE	2F	BGT	2E
r < m	BLT	2D	BGE	2C

UNSIGNED CONDITIONAL BRANCHES (Notes 1-4)

Test	True	OP	False	OP
r > m	BHI	22	BLS	23
r ≥ m	BHS	24	BLO	25
r = m	BEQ	27	BNE	26
r ≤ m	BLS	23	BHI	22
r < m	BLO	25	BHS	24

NOTES:
1. All conditional branches have both short and long variations.
2. All short branches are two bytes and require three cycles.
3. All conditional long branches are formed by prefixing the short branch opcode with $10 and using a 16-bit destination offset.
4. All conditional long branches require four bytes and six cycles if the branch is taken or five cycles if the branch is not taken.

TABLE 2 — INDEXED ADDRESSING MODE

Type	Forms	Non Indirect Assembler Form	Postbyte Opcode	+~	+#	Indirect Assembler Form	Postbyte Opcode	+~	+#
Constant Offset From R	No Offset	,R	1RR00100	0	0	[,R]	1RR10100	3	0
(2s Complement Offsets)	5-Bit Offset	n, R	0RRnnnnn	1	0	defaults to 8-bit			
	8-Bit Offset	n, R	1RR01000	1	1	[n, R]	1RR11000	4	1
	16-Bit Offset	n, R	1RR01001	4	2	[n, R]	1RR11001	7	2
Accumulator Offset From R	A Register Offset	A, R	1RR00110	1	0	[A, R]	1RR10110	4	0
(2s Complement Offsets)	B Register Offset	B, R	1RR00101	1	0	[B, R]	1RR10101	4	0
	D Register Offset	D, R	1RR01011	4	0	[D, R]	1RR11011	7	0
Auto Increment/Decrement R	Increment By 1	,R+	1RR00000	2	0	not allowed			
	Increment By 2	,R++	1RR00001	3	0	[,R++]	1RR10001	6	0
	Decrement By 1	,−R	1RR00010	2	0	not allowed			
	Decrement By 2	,−−R	1RR00011	3	0	[,−−R]	1RR10011	6	0
Constant Offset From PC	8-Bit Offset	n, PCR	1xx01100	1	1	[n, PCR]	1xx11100	4	1
(2s Complement Offsets)	16-Bit Offset	n, PCR	1xx01101	5	2	[n, PCR]	1xx11101	8	2
Extended Indirect	16-Bit Address	—	—	—	—	[n]	10011111	5	2

R = X, Y, U, or S   RR:
x = Don't Care   00 = X
01 = Y
10 = U
11 = S

$\overset{+}{\sim}$ and $\overset{+}{\#}$ indicate the number of additional cycles and bytes for the particular variation.

LEGEND:
- OP Operation Code (Hexadecimal)
- ~ Number of MPU Cycles
- # Number of Program Bytes
- \+ Arithmetic Plus
- − Arithmetic Minus
- • Multiply
- M̄ Complement of M
- → Transfer Into
- H Half-carry (from bit 3)
- N Negative (sign bit)
- Z Zero result
- V Overflow, 2's complement
- C Carry from ALU
- ↕ Test and set if true, cleared otherwise
- • Not Affected
- CC Condition Code Register
- : Concatenation
- V Logical or
- Λ Logical and
- ⊻ Logical Exclusive or

ASSEMBLER DIRECTIVES AND COMMAND SUMMARY

DIRECTIVES

END	end of source program	12-12
EQU	equate a symbol to operand	12-12
FCB	form constant byte	12-13
FDB	form double constant byte	12-14
FCC	form constant characters	12-13
LIB	process library file	12-14
NAM	specify name or title	12-14
ORG	assign origin of program counter	12-15
PAG	skip to next page	12-16
RMB	reserve memory bytes	12-16
SPC	insert spaces of output listing	12-16
TTL	specify name or title	12-14
OPT	assembler control options	12-15
	PAG enable page formatting	

COMMAND

[P,]ASMB,<filename1>[,<filename2>[,+<optionlist>]

optionlist

- **B** no binary file
- **D** no date on list
- **G** print only one line for FCB, FDB and FCC
- **L** no list
- **S** no symbol table
- **Y** delete binary file

DEBUG COMMAND SUMMARY

General system control

CALC	Calc	13-16
DELAY=<value>	DELay	13-17
DEPTH	DEPth	13-17
EXIT	Exit	13-17
FLAG[=<adress>]	FLag	13-18
IND=ON or OFF	IND	13-18
MACH	MAch	13-18
MODE=1 or 0	MOde	13-18
REG	Reg	13-19
RESET	RESet	13-19
RET	RET	13-19
SET,<register list>	Set	13-20
STACK[,<value>]	STACK	13-20
STATES	STATes	13-20
TRAIL	TRAIL	13-20
X,<FLEX command>	X	13-21

Memory commands

ASM[,<adress>]	Asm	13-21
DIS,<start address>,<stop adress>	Dis	13-22
DUMP,<address>	DUmp	13-22
FILL,<start address>,<stop adress>[,<byte>]	FILl	13-22
FIND,<start adress>,<stop adress>,<string>	FINd	13-22
MEM,<adress>	Mem	13-23

Simulation control

GO	Go	13-24
JUMP,<address>	Jump	13-24
PAST[,<value>]	PAst	13-24
SIM[,<count>]	SIM	13-24
START,<address>	STart	13-25
STEP[<count>]	STEP	13-25
TRACE=<value>	Trace	13-25
TSM[,<count>]	TSm	13-26
'Ctrl'C	operator break	13-26
Escape	halt on EOL	13-26
'Ctrl'F	faster	13-26
'Ctrl'S	slower	13-26

Breakpoints

B,<actions>â<address>[,<modifier><count>][,IF<condition>]

B	B	13-27

Bâ<address>[,<modifier><count>][,IF<condition>]

- R...Print register contents
- Z...Zero the states counter
- T...Trace mode on
- U...Trace mode off (untrace)
- H...Histogram counter
- M...Print message
- J...Jump to new address
- S...Stop simulation

BP[,<address>[-<address>]]	BP	13-29
CLB[,<address>[-<address>]]	CLB	13-29
CLH[,<address>[-<address>]]	CLH	13-29
CLM	CLM	13-29
HIST[,<address>[->address>]]	Hist	13-30
RT[,<address>]	RT	13-30

Memory protection

BOUNDS[,<types>]	BOunds	13-31
CLP[,<type>]	CLP	13-31
PROT,<address>[-<address>],<type>	Prot	13-32

Execution traps

INST=<count>	INST	13-32
ITRAP=ON or OFF	ITtrap	13-33
NEST=<value>	Nest	13-33
STOP=<address>	STOP	13-33
XFR=ON or OFF	XFR	13-33

Interrupt control

IRQ=<count>	IRQ	13-34
NMI=<count>	NMI	13-34
'Ctrl'I	'Ctrl'I	13-34
'Ctrl'N	'Ctrl'N	13-34

FLEX COMMAND AND SUBROUTINE SUMMARY

command	cmd	page
ASN[,W=<drive>][,S=<drive>]	ASN	10-9
CAT[,<drive list>][,<match list>]	CAT	10-11
COPY,<filespec>,<file spec>	COPY	10-13
COPY,<file spec>,<drive>		
COPY,<drive>,<drive>[,<match list>]		
DATE[,mm.dd.yy]	DATE	10-15
DELETE,<file spec>[,<file list>]	DELETE	10-16
GET,<file spec>[,<file list>]	GET	10-7
default extension : .BIN		
LIST,<file spec>[,<line range>][,N]	LIST	10-17
MON	MON	10-7
O,<file spec>,<command>	O	10-18
default extension : .CMD		
P,<command>	P	10-19
PROT,<file spec>[,<options>]	PROT	10-20
RENAME,<file spec 1>,<file spec 2>	RENAME	10-21
default extension : .TXT		
SAVE,<file spec>,<begin adr>,<end adr>[,<transfer adr>]		
default extension : .BIN	SAVE	10-22
SAVELOW	SAVELOW	10-22
STARTUP	STARTUP	10-23
TTYSET[,<parameter list>]	TTYSET	10-25
VERSION,<file spec>	VERSION	10-29
default extension : .CMD		

FLEX

GETCHR	Get character	$AD15	10-33
PUTCHR	Put character	$AD18	10-33
PSTRNG	Print string	$AD1E	10-33
INBUFF	Input to line buffer	$AD1B	10-34
PCRLF	Print CR + LF	$AD24	10-34
OUTADR	Output hexadecimal address	$AD45	10-34
OUTDEC	Output decimal number	$AD39	10-35
OUTHEX	Output hexadecimal number	$AD3C	10-35
GETHEX	Get hexadecimal number	$AD42	10-35
INDEC	Input decimal number	$AD48	10-35
Line buffer		$A080 - $A0FF	10-32
System date registers		$AC0E - $AC10	10-32
Line buffer pointer		$AC14 - $AC15	10-32
Escape return register		$AC16 - $AC17	10-32