6809 FLEX™ Operating System

technical systems consultants, inc.

SWTPC MODIFICATION - APPLICATION NOTICE AN #125A

Product: FLEX™ version 2.7:3 and subsequent versions of FLEX 2.7
Date: Jan. 8, 1981 N.R.

Proper FLEX™ Disk Driver Operation

The disk drivers implemented in FLEX™ version 2.7 have been optimized for use with the SWTPC DMF2 and DC4 disk controllers attached to drives with 3 millisecond step times. These drivers may not function correctly with older controllers or drives, and must be modified if they are to be used on the older equipment. This note describes how to modify FLEX™ version 2.7 for use with DMF1, DC2, or DC3 disk controllers and Calcomp 143 or Shugart SA-400 disk drives. This version of FLEX™ should not be used with the SWTPC DC-1 disk controller.

When FLEX™ is initially booted from disk, special bootstrap disk drivers are brought in from the first few sectors on the disk. These drivers will function with any of the above configurations of equipment, thus allowing FLEX™ to be booted on older equipment. These drivers are discarded after the bootstrap process is complete. Hence, in order to properly run commands after booting, the drivers present in FLEX™ itself must be modified. In order to do this, a two-step process is necessary.

First, once FLEX™ is booted and the "+++" prompt is present, the MON command is used to return control to the ROM monitor. (If the message "-- Can't run STARTUP file." appears, it should be ignored.) The memory examine and change function of the monitor is then used to alter the copy of the FLEX™ disk drivers resident in memory.

Second, the FIX command is then used to alter the copy of the drivers in the FLEX.SYS file on disk. The altered copy of FLEX is then connected to the bootstrap with the LINK command. Once these changes are made, the altered FLEX™ may then booted and used normally.

To perform the first step of this alteration, boot the FLEX™ 2.7 disk and enter the date as requested. Then enter the the ROM monitor from FLEX™ using the MON command as follows:

```
+++MON

 - SP=C073  US=BFFC  DP=00  IX=1234  IY=5678
 - PC=D34C  A=00  B=00  CC: E - - - - - - -
>
```

Select the modifications appropriate for your hardware configuration from those described below. Make the modifications to the drivers already resident in memory by using the ROM monitor. Then return to FLEX™ from the monitor by typing "G".

At this point, the disk drivers that have been loaded into memory have been modified, but the copy on disk in the FLEX.SYS file is still

in its original condition. Use the FIX command to modify FLEX.SYS as follows:

```
>G
+++FIX FLEX.SYS

SWTPC Binary File Patch -- Version 2.5
:
```

The same modifications made to the drivers in memory must now be repeated to change the copy of the disk drivers in the FLEX.SYS file. When this has been done, exit from the fix command by typing "e". The modified FLEX.SYS file must be re-linked to the bootstrap process by using the LINK command:

```
:e
-- Fix complete.

+++LINK FLEX.SYS
+++
```

The following descriptions detail the changes that must be made to the FLEX™ disk drivers for each piece of equipment to be used. If a combination of equipment is used, the changes for BOTH types must be made. For example, if you wish to use a DC-3 disk controller with SA-400 disk drives, you must make both sets of changes to correctly modify the FLEX™ for normal operation.

NOTE: The addresses specified below are correct for FLEX™ 2.7:3. The same modifications may be made to subsequent versions of FLEX™ 2.7, but the addresses may not be correct. If that is the case, the object code in the vicinity of the specified addresses should be examined to locate the bytes that must be changed.

(1) The following change must be made to use Calcomp 143, 8-inch drives. (To use the Calcomp 143 drives with the DMF2 disk controller the hardware changes described in Modification/Application Notice 102 must be made.) The combination of the DMF1 and Calcomp 143 hardware does not require this change. This change doubles the amount of time allowed for a step to complete.

change the byte(s) at	DEEF	from	86 18	to	86 19
	DEF1		B7 F0 20		B7 F0 20
	CB0F		86 08		86 09
	CB11		B7 F0 20		B7 F0 20

(2) The following change must be made to use the DMF1 disk controller. This change disables the driver's use of the extended address hardware present on the DMF2 controller.

change the byte(s) at	DE65	from	8A 16	to	8A 16
	DE67		B7 F0 40		12 12 12

(3) The following change must be made to use the DC2 disk controller. This change causes the head load timer to be used for all disk operations. (The drivers normally use the head load timer only after seek operations.)

change the byte(s) at	DE2F	from	BA DE 1E	to	BA DE 1E
	DE32		BA DE 20		8A 04 12
	DE8B		BA DE 1E		BA DE 1E
	DE8E		BA DE 20		8A 04 12

(4) The following change must also be made to use the DC2 disk controller. This change disables the driver's use of the drive ready indication available on the DC3 and DC4 controllers.

change the byte(s) at	CB72	from	21 10	to	20 10
	CB74		8E 49 B3		8E 49 B3
	CBB2		21 D0		20 D0
	CBB4		8E 0B 0E		8E 0B 0E

(5) The following change must be made to use DC2 disk controllers or DC3 disk controllers. This change disables the driver's use of the double density hardware present on the DC4 controller.

change the byte(s) at	DF62	from	21 14	to	20 14
	DF64		A7 E2		A7 E2

(6) The following change must be made to use the Shugart SA-400, 5-inch drives. This change increases the amount of time allowed for a step to complete.

change the byte(s) at	DF39	from	86 18	to	86 1B
	DF3B		B7 E0 18		B7 E0 18
	CB15		86 08		86 0B
	CB17		B7 E0 18		B7 E0 18

FLEX™ is a trademark of Technical Systems Consultants

SWTPC MODIFICATION--APPLICATION NOTICE AN #113

Product: FLEX 2.6 DOS
Date: February 26, 1980

Configuring FLEX 2.6 for Computers with MP-B3 Motherboards (69A, 69K computers, not S/09 Computers)

FLEX 2.6 may incorrectly auto configure on computers with MP-B3 motherboards by indicating the presence of an internal interval timer. This can be checked by running the SBOX utility contained on the FLEX 2.6 disk. If the utility responds with:

-- Interval Timer = Yes

then the SBOX utility must be used to set the Interval Timer response to NO. This must be done even if the system has an optional MP-T interrupt timer plugged on to the system. The timer configurator of the SBOX utility is concerned with the presence of the 6840 type timer which is standard on S/09 computers rather than the optional MP-T timer board. S/09 computers are the only ones at the time of this writing that should respond with "Interval Timer = Yes" response.

To set the Interval Timer response to NO, enter the following:

```
SBOX,TIMER=NO
```

The SBOX command will change and confirm that the timer parameter has been properly set.

```
+++SBOX


SWTPC Configurator -- Version 2.1
-- Memory Size = __K
-- I/O Port Size = 16
-- CPU Clock Rate = 1 MHz
-- Power Line Frequency = __Hz
-- Extended Addressing = No
-- Interval Timer = No
-- Real Time Clock = No
-- Upper Case Only = Yes
```

If the Interval Timer parameter is not properly set as outlined above the P command and printer spooling will not function correctly.

The FLEX™ Disk Operating System

Technical Systems Consultants, Inc.

The FLEX™ Disk Operating System
System Documentation

CONTENTS

General Notes

Technical Systems Consultants, Inc.

GENERAL NOTES

This section contains suggestions on getting FLEX™ 9.0 up on your system and on compatibility with your existing hardware and software. This manual assumes you already have a working disk system and are familiar with the basics of floppy disk systems such as proper disk handling techniques, inserting and removing disks from the drives, etc.

One important point should be made in regard to getting FLEX "up and running". You receive only one disk and it is crucial that you protect this disk with your life. If you take the following steps, you might save yourself a lot of headaches and additional expense:

1) Write-protect the FLEX disk before you ever insert it into a drive. Consult your disk system hardware manual or the FLEX User's Manual for details on write-protecting a disk.

2) Boot up the FLEX system and once running copy all files from the original FLEX disk to a new disk. Next perform a LINK command to FLEX.SYS on this new disk.

3) Now remove the original FLEX disk and store it in a safe place. It should never be used again unless you wipe out all the new FLEX disks you make and need to repeat this procedure. Use the new FLEX disk you have made for all future disk work.

HARDWARE REQUIREMENTS

This section discusses the hardware requirements for running FLEX 9.0. This version is setup for the Southwest Technical Products Corporation's disk systems: the MF-68 or MF-69 5-inch minidiskette, the DMAF1 or DMAF2 8-inch diskette, and the CDS-1 Winchester disk unit.

Memory Requirements

The FLEX disk operating system itself resides in the range of $C000 to $DFFF. This means you will need 8K of memory starting at $C000. You should be certain your particular system can accept memory in this region.

You must also have "User Memory" (RAM) starting at location $0000 and running continuously up from there. The more user memory you have in your system the better off you will be. This is because you will be able to run larger programs and because software which works with files that are larger than memory can hold (such as the editor or sort/merge) will operate more efficiently and quickly. Although FLEX resides at $C000, certain of its commands utilize the lower end of this user RAM space. A minimum of 12K of RAM is required for such purposes.

Monitor ROM

As sold, this version of FLEX requires the S-BUG monitor ROM from SWTPc (or equivalent). FLEX 9.0 has its own internal terminal I/O routines, so S-BUG's are not used. These routines assume an ACIA at location $E004. S-BUG is required, however, for setting up interrupt vectors.

There are two exceptions to this ROM requirement. The first is that the interrupt vectors need not be set if no program will use interrupts. Note that many programs such as printer spooling, the SWTPc Editors, etc., do make use of interrupts. Thus if you did not require printer spooling or editing you would not require any monitor ROM at all except for booting the system up and to jump to when exiting FLEX. The second exception is to make use of the user adaptable version of FLEX which is supplied on disk along with the standard version. See 'Adapting FLEX to Custom Monitors' for details.

Printer Spooling

FLEX 9.0 Version 2.6 supports printer spooling which allows you to list a file (or files) on a line printer at the same time as you perform other FLEX operations such as editing, assembling, running BASIC, etc. In order to do this, FLEX requires an S/09 computer system, or an MP-T interrupt timer board on I/O port #5 for /09, 69/A and 69/K computer systems.

DISK COMPATIBILITY

Disks created under 6809 FLEX 9.0 are compatible with those created under 6800 FLEX 1.0 on the 8" drives or 6800 FLEX 2.0 on the 5" drives. The reverse is also true, meaning that FLEX 9.0 can read disks created by one of those 6800 FLEX systems. This means that transferring text files will require nothing more than copying with the COPY command. In fact it is not even necessary to put the files on a new disk. As long as a disk is being used for work files only (no disk command files) it may be used interchangeably.

The one place where the disks are different is in the bootstrap loader which the NEWDISK command places on track 0 when a disk is initialized. Obviously the loader must be different for 6800 and 6809. This simply means that a disk initialized with the 6809 NEWDISK command cannot be used to boot 6800 FLEX and vice versa.

The new double-density system is an exception to all the above. It cannot be used to read disks created by the original 6800 single-density system. Any disks, however, created as single-density with the new double-density version of NEWDISK (done by answering 'N' to the prompt 'Double-Sided Disk?') can be read on either a single or double density system. This is because the new double-density NEWDISK writes FF's in certain gap areas whereas the old single-density NEWDISK wrote 00's. The single-density controller board (which uses the Western Digital 1771) can read either type, but the double-density board (which uses the Western Digital 1791) can only read the type with FF's.

SOFTWARE COMPATIBILITY

6809 object code is NOT at all compatible with 6800 object code. This means you cannot run binary command files from a 6800 system on a 6809 system. Since 6809 FLEX can read a 6800 FLEX disk and vice versa, you must be careful not to execute a 6800 command in a 6809 system and again, vice versa.

Where the 6809 and 6800 ARE compatible is in the source code. Thus, if you have the source listing for a 6800 program on disk, it can be reassembled by the 6809 assembler to produce executable 6809 object code. Of course if the program calls any routines from FLEX, these addresses will have to be changed since 6809 FLEX resides at $C000 (6800 FLEX is at $A000). This is usually a matter of simply changing all occurrences of '$A' to '$C' and all '$B' to '$D' with the editor.

ADAPTING FLEX

The FLEX 9.0 disk supplied has two copies of the FLEX object code. One is called FLEX.SYS and is ready to boot up with SWTPc disk hardware. The second is called FLEX.COR which represents the CORe or main body of FLEX. It differs from the bootable form of FLEX in that it does not have any terminal or disk I/O routines built in. This allows the user to modify these I/O drivers, if desired, to produce a customized version of FLEX. Note that in order to produce this customized version you must have FLEX up and running so you will need the bootable version (FLEX.SYS). The customized terminal and disk I/O routines are supplied in two packages. We will discuss them separately and then examine how to add them onto FLEX.COR to produce a new, customized, bootable version of FLEX.

The CUSTOM I/O DRIVER PACKAGE

This package allows the user to alter the functioning of the terminal I/O and the functioning of printer spooling. Nine routines and two interrupt vectors are set up in this package. There is a space reserved for these routines beginning at location $D370 and ending at $D3E6. The address of these 11 items must be setup in a jump table found at locations $D3E7 thru $D3FB. A copy of the Custom I/O Driver Package used to produce FLEX.SYS is included at the end of the General Notes section. Use it as a guide for writing your own.

A description of each routine and vector follows.

INCH
The address of the input character routine should be placed at $D3FB. This routine should get one input character from the terminal and return it in 'A' with the parity bit cleared. It should also echo the character to the output device. Only 'A' and the condition codes may be modified.

OUTCH
The address of the output character should be placed at $D3F9. This routine should output the character found in 'A' to the output device. No registers should be modified except condition codes.

STAT
The address of the STAT routine should be placed at $D3F7. This routine checks the status of the input device. That is to say, it checks to see if a character has been typed on the keyboard. If so, a Not-Equal condition should be returned. If no character has been typed, an Equal to zero condition should be returned. No registers may be modified except condition codes.

TINIT
The address of the terminal initialization routine should be placed at $D3F5. This routine performs any necessary initialization for terminal I/O to take place. Any register may be modified except 'S'.

MONITR

This is the address to which execution will transfer when FLEX is exited. It is generally the reentry point of the system's monitor ROM. The address should be placed at $D3F3.

TMINT

The address of the timer initialization routine should be placed at $D3F1. This routine performs any necessary initialization for the interrupt timer used by the printer spooling process. Any register may be modified except 'S'.

TMON

The address of the timer on routine should be placed at $D3EF. This routines "turns the timer on" or in other words starts the interval IRQ interrupts. Any registers execpt 'S' may be modified.

TMOFF

The address of the timer off routine should be placed at $D3ED. This routine "turns the timer off" or in other words stops the interval IRQ interrupts. Any registers except 'S' may be modified.

IRQVEC

The IRQ vector is an address of a two byte location in RAM where FLEX can stuff the address of its IRQ interrupt handler routine. In other words, when an IRQ interrupt occurs control should be transferred to the address stored at the location specified by the IRQ vector. This IRQ vector location (address) should be placed at $D3EB.

SWIVEC

The SWI3 vector is an address of a two byte location in RAM where FLEX can stuff the address of its SWI3 interrupt handler routine. In other words, when an SWI3 interrupt occurs control should be transferred to the address stored at the location specified by the SWI3 vector. This SWI3 vector location (address) should be placed at $D3E9.

IHNDLR

The Interrupt Handler routine is the one which will be executed when an IRQ interrupt occurs. If using printer spooling, the routine should first clear the interrupt condtion and then jump to the 'change process' routine of the printer spooler at $C700. If not using printer spooling, this routine can be setup to do whatever the user desires. If it is desirable to do both printer spooling and have IRQ's from another device (besides the spooler clock), this routine would have to determine which device had caused the interrupt and handle it accordingly. The address of this routine should be placed at $D3E7.

The CUSTOM DISK DRIVER PACKAGE

This package supplies all the disk functions required by FLEX. There are eight routines in all:

READ	Reads a single sector
WRITE	Writes a single sector
VERIFY	Verifys a single sector
RESTORE	Restores the head to track 0
DRIVE	Selects the desired drive
CHECK	Checks a drive for a ready condition
QUICK	Same as CHECK but with no delay
INIT	Initializes any necessary values
WARM	Does any Warm Start initialization

These routines and what is required of them are decribed in the Advanced Programmer's Guide in the section titled 'DISK DRIVERS'. There is a jump table which contains the address of all these routines at $DE00. This table is as follows:

```
DE00    JMP  READ
DE03    JMP  WRITE
DE06    JMP  VERIFY
DE09    JMP  RESTOR
DE0C    JMP  DRIVE
DE0F    JMP  CHECK
DE12    JMP  QUICK
DE15    JMP  INIT
DE18    JMP  WARM
```

Immediately following this jump table there is a space for the disk driver routines. In the general case this space would start at $DE1B and run through $DFFF. In the SWTPc system with S-BUG installed, that entire space is not available due to the fact that S-BUG uses RAM in the area of $DFA0 to $DFFF for variables and stack. Thus the driver routine area is limited in this case to $DE18 through $DF9F.

The actual source listings for the SWTPc drivers are not included, but a skeletal Custom Disk Driver Package is included at the end of this section which should assist you in writing your own package.

PUTTING THE CUSTOM FLEX TOGETHER

Once you have written and assembled a Custom I/O and Custom Disk Driver packages, you are ready to append them to the core of FLEX (FLEX.COR) to produce a new, bootable version. This is done with the APPEND utility if FLEX, but before we get into that there is a very important point which must be covered.

*** IMPORTANT ***

The copy of FLEX on disk is much like any other standard binary file. IT MUST HAVE A TRANSFER ADDRESS IN ORDER TO WORK! It is also important to note that unlike other binary files FLEX can have ONLY ONE transfer address and it MUST BE THE LAST THING IN THE FILE! The simplest way of getting that transfer address into the file is by use of the END statement in the assembler. We recommend you put a transfer address on the END statement of the Custom I/O Driver Package and make sure it is the last thing in the final FLEX file.

Assuming you have put a transfer address on the Custom I/O Driver Package with an end statement of the form:

```
END  $CD00
```

you can now create a new version of FLEX by appending the custom disk drivers and custom I/O drivers onto FLEX.COR. You should use the APPEND command for this purpose as shown:

```
+++APPEND FLEX.COR DRVRS.BIN CUSTOMIO.BIN NEWFLEX.SYS
```

This command assumes the object file you created for the Custom Disk Drivers is called DRVRS.BIN and the Custom I/O Drivers are in a file called CUSTOMIO.BIN. The new, custom version of FLEX is called NEWFLEX.SYS. In order to boot up this NEWFLEX.SYS you must link it with the LINK command (see the FLEX User's and Advanced Progammer's Manuals). The command would be of the form:

```
+++LINK NEWFLEX.SYS
```

The disk containing your newly made and linked FLEX can now be booted with the normal boot procedure.

```
                        * SKELETAL 6809 DISK DRIVER PACKAGE

                        * TECHNICAL SYSTEMS CONSULTANTS, INC
                        * BOX 2574
                        * WEST LAFAYETTE, INDIANA  47906
                        *

                        *
                        * THE DRIVER ROUTINES PERFORM THE FOLLOWING
                        * 1. READ SINGLE SECTOR - DREAD
                        * 2. WRITE SINGLE SECTOR - DWRITE
                        * 3. VERIFY WRITE OPERATION - VERIFY
                        * 4. RESTORE HEAD TO TRACK 00 - RESTOR
                        * 5. DRIVE SELECTION - DRIVE
                        * 6. CHECK READY - DCHECK
                        * 7. QUICK CHECK READY - DQUICK
                        * 8. COLD START INITIALIZATION - DINIT
                        * 9. WARM START INITIALIZATION - DWARM
                        *

                        * SYSTEM CONSTANTS
                        *
                        * THIS SPACE IS WHERE ANY NECESSARY EQUATES MIGHT
                        * BE PLACED, SUCH AS DISK CONTROLLER REGISTER
                        * LOCATIONS, SECTOR LENGTH, ETC.
                        *


                        ************************************************************

DE00                            ORG     $DE00

                        * JUMP TABLE

DE00 7E   DE23          DREAD   JMP     READ
DE03 7E   DE28          DWRITE  JMP     WRITE
DE06 7E   DE2D          DVERFY  JMP     VERIFY
DE09 7E   DE31          RESTOR  JMP     RST
DE0C 7E   DE35          DRIVE   JMP     DRV
DE0F 7E   DE39          DCHECK  JMP     CHECK
DE12 7E   DE3F          DQUICK  JMP     QUICK
DE15 7E   DE1B          DINIT   JMP     INIT
DE18 7E   DE1F          DWARM   JMP     WARM

                        ************************************************************
```

```
                    ********************************************************

                    * VARIABLE STORAGE

                    * IF ANY VARIABLES ARE REQUIRED, THEY MIGHT BE PLACED
                    * HERE.  THIS MIGHT INCLUDE VARIABLES LIKE CURRENT
                    * DRIVE, CURRENT TRACK FOR EACH DRIVE, OR TEMPORARY
                    * STORAGE LOCATIONS.

                    ********************************************

                    * INIT
                    *
                    * INITIALIZES THE NECESSARY DRIVER VARIABLES.

DE1B 12             INIT    NOP             THIS ROUTINE IS CALLED
DE1C 12                     NOP             DURING FMS INITIALIZATION
DE1D 12                     NOP             AT COLD START
DE1E 39                     RTS

                    * WARM
                    *
                    * WARM START INITIALIZATION

DE1F 12             WARM    NOP             THIS ROUTINE IS CALLED
DE20 12                     NOP             DURING FMS INITIALIZATION
DE21 12                     NOP             AT WARM START
DE22 39                     RTS

                    * READ
                    *
                    * READ ONE SECTOR

DE23 12             READ    NOP             READS THE SECTOR POINTED
DE24 12                     NOP             TO BY TRACK IN 'A'
DE25 12                     NOP             AND SECTOR IN 'B'.
DE26 12                     NOP             'X' POINTS TO FCB.
DE27 39                     RTS

                    * WRITE
                    *
                    * WRITE ONE SECTOR

DE28 12             WRITE   NOP             WRITES THE SECTOR POINTED
DE29 12                     NOP             TO BY TRACK IN 'A'
DE2A 12                     NOP             AND SECTOR IN 'B'.
DE2B 12                     NOP             'X' POINTS TO FCB.
DE2C 39                     RTS
```

```
                        * VERIFY
                        *
                        * VERIFY LAST TRACK WRITTEN

DE2D 12         VERIFY  NOP             THE SECTOR JUST
DE2E 12                 NOP             WRITTEN IS VERIFIED.
DE2F 12                 NOP             NO PARAMETERS ARE SUPPLIED.
DE30 39                 RTS

                        * RST
                        *
                        * RST RESTORES THE HEAD TO 00

DE31 12         RST     NOP             HEAD RESTORED TO TRACK
DE32 12                 NOP             ZERO ON DRIVE POINTED
DE33 12                 NOP             TO BY FCB AT 'X'.
DE34 39                 RTS

                        * DRV
                        *
                        * DRV SELECTS THE DRIVE.

DE35 12         DRV     NOP             THE DRIVE NUMBER FOUND
DE36 12                 NOP             IN FCB POINTED TO BY 'X'
DE37 12                 NOP             IS SELECTED.
DE38 39                 RTS

                        * CHECK
                        *
                        * CHECK FOR DRIVE READY

DE39 12         CHECK   NOP             THE DRIVE POINTED TO
DE3A 12                 NOP             BY FCB AT 'X' IS CHECKED
DE3B 12                 NOP             FOR A READY STATE AFTER
DE3C 12                 NOP             DELAYING FOR DRIVES TO
DE3D 12                 NOP             COME UP TO SPEED.
DE3E 39                 RTS

                        * QUICK
                        *
                        * QUICK CHECK FOR READY

DE3F 12         QUICK   NOP             THE DRIVE POINTED TO
DE40 12                 NOP             BY FCB AT 'X' IS CHECKED
DE41 12                 NOP             FOR READY STATE WITHOUT
DE42 12                 NOP             DELAYING FOR DRIVES TO
DE43 12                 NOP             COME UP TO SPEED.
DE44 39                 RTS


                        END
```

```
                        * CUSTOM I/O DRIVER PACKAGE
                        *
                        * CONTAINS ALL TERMINAL I/O DRIVERS AND INTERRUPT
                        * HANDLING INFORMATION.

                        * SYSTEM EQUATES

                C700    CHPR    EQU     $C700       CHANGE PROCESS ROUTINE


                        ************************************************
                        *                                              *
                        * I/O ROUTINE VECTOR TABLE                     *
                        *                                              *
D3E7                            ORG     $D3E7       TABLE STARTS AT $D3E7   *
                        *                                              *
D3E7 D3CB               IHNDLR  FDB     IHND        IRQ INTERRUPT HANDLER   *
D3E9 DFC2               SWIVEC  FDB     $DFC2       SWI3 VECTOR LOCATION    *
D3EB DFC8               IRQVEC  FDB     $DFC8       IRQ VECTOR LOCATION     *
D3ED D3C4               TMOFF   FDB     TOFF        TIMER OFF ROUTINE       *
D3EF D3BD               TMON    FDB     TON         TIMER ON ROUTINE        *
D3F1 D3A7               TMINT   FDB     TINT        TIMER INITIALIZE ROUTINE *
D3F3 F814               MONITR  FDB     $F814       MONITOR RETURN ADDRESS  *
D3F5 D370               TINIT   FDB     INIT        TERMINAL INITIALIZATION *
D3F7 D39C               STAT    FDB     STATUS      CHECK TERMINAL STATUS   *
D3F9 D38B               OUTCH   FDB     OUTPUT      TERMINAL CHAR OUTPUT    *
D3FB D37D               INCH    FDB     INPUT       TERMINAL CHAR INPUT     *
                        *                                              *
                        ************************************************



                        * ACTUAL ROUTINES START HERE
                        ******************************

D370                            ORG     $D370

                        * TERMINAL INITIALIZE ROUTINE

D370 86   13            INIT    LDA     #$13        RESET ACIA
D372 A7   9F D3E5               STA     [ACIAC]
D376 86   11                    LDA     #$11        CONFIGURE ACIA
D378 A7   9F D3E5               STA     [ACIAC]
D37C 39                         RTS

                        * TERMINAL INPUT CHARACTER ROUTINE

D37D A6   9F D3E5       INPUT   LDA     [ACIAC]     GET STATUS
D381 84   01                    ANDA    #$01        CHARACTER PRESENT?
D383 27   F8                    BEQ     INPUT       LOOP IF NOT
D385 A6   9F D3E3               LDA     [ACIAD]     GET THE CHARACTER
D389 84   7F                    ANDA    #$7F        STRIP PARITY
```

```
                    * TERMINAL OUTPUT CHARACTER ROUTINE

D38B 34   02       OUTPUT  PSHS    A          SAVE CHARACTER
D38D A6   9F D3E5  OUTPU2  LDA     [ACIAC]    TRANSMIT BUFFER EMPTY?
D391 84   02               ANDA    #$02
D393 27   F8               BEQ     OUTPU2     WAIT IF NOT
D395 35   02               PULS    A          RESTORE CHARACTER
D397 A7   9F D3E3          STA     [ACIAD]    OUTPUT IT
D39B 39                    RTS

                    * TERMINAL STATUS CHECK (CHECK FOR CHARACTER HIT)

D39C 34   02       STATUS  PSHS    A          SAVE A REG.
D39E A6   9F D3E5          LDA     [ACIAC]    GET STATUS
D3A2 84   01               ANDA    #$01       CHECK FOR CHARACTER
D3A4 35   02               PULS    A          RESTORE A REG.
D3A6 39                    RTS


                    * TIMER INITIALIZE ROUTINE

D3A7 BE   D3E1     TINT    LDX     TMPIA      GET PIA ADDRESS
D3AA 86   FF               LDA     #$FF
D3AC A7   84               STA     0,X
D3AE 86   3C               LDA     #$3C
D3B0 A7   01               STA     1,X
D3B2 86   8F               LDA     #$8F
D3B4 A7   84               STA     0,X
D3B6 A6   84               LDA     0,X
D3B8 86   3D               LDA     #$3D
D3BA A7   01               STA     1,X
D3BC 39                    RTS

                    * TIMER ON ROUTINE

D3BD 86   04       TON     LDA     #$04       TURN ON TIMER
D3BF A7   9F D3E1          STA     [TMPIA]
D3C3 39                    RTS

                    * TIMER OFF ROUTINE

D3C4 86   8F       TOFF    LDA     #$8F       TURN OFF TIMER
D3C6 A7   9F D3E1          STA     [TMPIA]
D3CA 39                    RTS


                    * IRQ INTERRUPT HANDLER ROUTINE

D3CB A6   9F D3E1  IHND    LDA     [TMPIA]    RESET INTERRUPTS
D3CF 7E   C700             JMP     CHPR       GO TO SPOOLER
```

```
                    * ACIA AND PIA ADDRESSES FOR SUPPLIED ROUTINES

D3E1                        ORG     $D3E1

D3E1 E012           TMPIA   FDB     $E012       TIMER PIA ADDRESS
D3E3 E005           ACIAD   FDB     $E005       ACIA DATA REG. ADR.
D3E5 E004           ACIAC   FDB     $E004       ACIA CONTROL REG. ADR.

                    * END STATEMENT HAS FLEX TRANSFER ADDRESS!

                            END     $CD00
```

FLEX User's Manual

Technical Systems Consultants, Inc.

FLEX User's Manual

PREFACE

The purpose of this User's Guide is to provide the user of the FLEX Operating System with the information required to make effective use of the available system commands and utilities. This manual applies to FLEX 9.0 for full size and mini floppy disks. The user should keep this manual close at hand while becoming familiar with the system. It is organized to make it convenient as a quick reference guide, as well as a thorough reference manual.

TABLE OF CONTENTS

I. 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.

II. SYSTEM REQUIREMENTS

FLEX requires random access memory from location 0000 through location 2FFF hex (12K). Memory is also required from C000 (48K) through DFFF hex (56K), where the actual operating system resides. The system also assumes at least 2 disk drives are connected to the controller and that they are configured as drives #0 and #1. You should consult the disk drive instructions for this information. FLEX interfaces with the disk controller through a section of driver routines and with the operator console or terminal through a section of terminal I/O routines.

III. 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.

As a specific example, suppose the system we are using has SWTPc's S-BUG installed and we wish to run FLEX. The first step is to power on all equipment and make sure the S-BUG prompt is present (>). Next insert the system diskette into drive 0 (the boot must be performed with the disk in drive 0) and close the door on the drive. Type "D" on the terminal if using a full size floppy system or "U" if a minifloppy system. The disk motors should start, and after about 2 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 7,3,79. 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!

IV. 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 '_', (a left arrow on some terminals).

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:

```
APPEND.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.

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 (0-3) 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.2
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.

V. 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 first initialized the auto drive searching mode will be selected. At this time, 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.

VI. 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 here while the UCS is described in the following sections.

GET

The GET command is used to load a binary file into memory. It is a special purpose command and is not often used. It 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 S-BUG. The syntax for this command is simply MON followed by the 'RETURN' key.

NOTE: to re-enter FLEX after using the MON command, you should enter the program at location CD03 hex.

UTILITY COMMAND SET

The following pages describe all of the utility commands currently included in the UCS. You should note that the page numbers denote the first letter of the command name, as well as the number of the page for a particular command. For example, 'B.1.2' is the 2nd page of the description for the 1st utility name starting with the letter 'B'.

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.

APPEND

The APPEND command is used to append or concatenate files, creating a new file as the result. A specific line or line range of any origin file may be specified.

DESCRIPTION

The general syntax for the APPEND command is as follows:

APPEND, <origin file, [+line range]>, [origin file,[+line range] list], <destination file>

where <origin file> specifies the file(s) to be appended. A line number or line number range can follow any origin file name. If no line range is specied the entire file is used.

The <destination file> should be the last argument and should not exist on the disk. If the specified <destination file> exists, APPEND tells the user:

Destination file exists
Do you wish to continue?

A "Y" response will delete the contents of the <destination file>. A "N" response will terminate the execution of APPEND. All other files specified must exist since they are the ones to be appended together. If only two file names are given, the first file will be copied to the second file.

If no extension is given, the default is .TXT for all files. The exception to this is if the first filename given has an extension other than .TXT, all other filenames without an extension default to that first extension.

Any type of sequential file may be appended, but it only makes sense to append files to the same type in most cases. If appending binary files which have transfer addresses associated with them, the transfer address of the last file of the list will be the effective transfer address of the resultant file. All of the original files will be left intact. (Note: Random files may not be appended.)

Examples of APPEND

1. APPEND junk +1-12 file5 +80-100 super

will append lines 1 through 12 of JUNK.TXT and lines 80 through 100 of FILE5.TXT to form the file SUPER.TXT.
(Note: Line number ranges ONLY apply to the file immediately in front of them. Only one line number range is allowed per specied file.)

2. APPEND doc.asm pat +5-99 hosp

will append the file DOC.ASM and lines 5 through 99 of PAT.ASM to form HOSP.ASM. If PAT.ASM did not have 99 lines, all of PAT.ASM that existed would be copied into HOSP.ASM.

3. APPEND file1.txt file2.txt file12.txt

will append the file FILE1.TXT and the file FILE2.TXT to form the file FILE12.TXT.

4. APPEND junk.asm +100-* try.asm

will copy file JUNK.ASM starting at the 100th line into file TRY.ASM. The file JUNK.ASM will be copied until the end of file is reached. (The "*" character is useful when the exact number of lines in the file is not known.)

5. APPEND file1.txt +1-10 file2.txt +1-50 file1 +40-90 file9

will append lines 1 to 10 of FILE1.TXT, lines 1 to 50 of FILE2.TXT, and lines 40 to 90 of FILE1.TXT to form the file FILE9.TXT.

ASN

The ASN command is used for assigning the 'system' drive and the 'working' drive or to 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:

```
APPEND,FILE1,FILE2,FILE3
```

If the system drive is assigned to be #0 and the working drive is assigned to drive #1, the above example will perform the following operation: get the APPEND command from drive #0 (the system drive), then append FILE2 from drive #1 (the working drive) to FILE1 from drive #1 and put the result in FILE3 on drive #1. As can be seen, the system drive was the default for APPEND 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. On systems using the 5-inch drives with either a DC-1 or DC-2 controller, no ready signal is provided by the drives, thus flex always assumes drives zero and one are ready. If a drive is selected that does not have a disk inserted, the system will hang until a disk is placed in the drive or the RESET switch is pressed. Systems using 8-inch drives, or 5-inch drives with the DC-3 controller provide a ready signal and drives with no disks inserted are bypassed.

Automatic drive searching causes FLEX to first check drive #0 for the file specified. If not there or if not ready FLEX will skip to drive #1. If the file is not found on drive #1 in the 5-inch version, FLEX gives up and a file not found error results. In the 8-inch version 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
THE SYSTEM DRIVE IS #0
THE WORKING DRIVE IS #0
```

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
```

The archive command is used to update selected files on a disk by copying from one disk to another. This allows only those files that are obsolete to be replaced with newer copies from another disk.

DESCRIPTION

The general syntax of the AR command is:

AR,<input drive>,<output drive>

The selected files are copied from the input drive to the output drive.

The AR command may be thought of as a "copy only if there and more recent" command. The only files copied from the input drive to the output drive are those files that exist on both disks. Further, the creation date of the file to be copied must be more recent than the creation date of the file to be replaced.

The volume name and number of both the input and output drives is displayed and the operator is asked if he wishes to continue. An "N" answer will abort processing and return control to the operating system. To continue with the update a "Y" answer is required.

When a file is found which meets the selection criteria, the operator is asked if the file is to be copied. An "N" answer allows processing to continue without copying the file. A "Y" answer causes the file to be copied before continuing. Typing a Carrier Return will abort processing and return control to the operating system.

The prompting used by AR is designed for convenient use with the Y and N commands described elsewhere in this manual. In fact, a useful method for displaying a catalog of all files which need updating is to use the following command line.

+++N,AR,0,1

After answering "Y" to the first question, a list of all files which need updating will be displayed.

NOTE: Some early versions of the COPY command did not copy the file's creation date, resulting in "creation" dates reflecting the date of the preceding use of COPY rather than the actual creation date of the file. Attempts to use AR to copy files generated by such a version of COPY will not be particularly useful since AR depends on the creation date being an actual creation date. Any copies of these early versions of the COPY command should be replaced with the current version.

The user will find it useful to be familiar with the TOUCH and DATE commands when using the AR command.

The BUILD command is provided for those desiring to create small text files quickly (such as STARTUP files, see STARTUP) or not wishing to use the optionally available FLEX Text Editing System. The main purpose for BUILD is to generate short text files for use by either the EXEC command or the STARTUP facility provided in FLEX.

DESCRIPTION

The general syntax of the BUILD command is:

BUILD,<file spec>

where <file spec> is the name of the file you wish to be created. The default extension for the spec is TXT and the drive defaults to the working drive. If the output file already exists the question "MAY THE EXISTING FILE BE DELETED?" will be displayed. A Y response will delete the existing file and build a new file while a N response will terminate the BUILD command.

After you are in the 'BUILD' mode, the terminal will respond with an equals sign ('=') as the prompt character. This is similar to the Text Editing System's prompt for text input. To enter your text, simply type on the terminal the desired characters, keeping in mind that once the 'RETURN' is typed, the line is in the file and can not be changed. Any time before the 'RETURN' is typed, the backspace character may be used as well as the line delete character. If the delete character is used, the prompt will be '???' instead of the equals sign to show that the last line was deleted and not entered It should be noted that only printable characters (not control characters) may be entered into text files using the BUILD command.

To exit the BUILD mode, it is necessary to type a pound sign ('#') immediately following the prompt, then type 'RETURN'. The file will be finished and control returned back to FLEX where the three plus signs should again be output to the terminal. This exiting is similar to that of the Text Editing System.

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 name of the diskette as entered during the NEWDISK operation will also be displayed. 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

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

The COPY command is used for making copies of files on a disk. Individual files may be copied, groups of name-similar files may be copied, or entire disks may be copied. The copy command is a very versatile utility. 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. Refer to NEWDISK for instructions on this procedure.

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.

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.

Copied files will have the same last-altered date and protection mode as the original file. If it is necessary to change the last-altered date, the TOUCH command may be used. Similarly, protection may be changed by using the PROT command.

C4MAT

The C4MAT command is used to format the CalComp Marksman hard disk. It performs a surface verification and format function similar to the NEWDISK utility.

DESCRIPTION

The general syntax of the C4MAT command is:

```
C4MAT
```

This command will initialize the entire surface of the hard disk. It is important to format the disk initially to build directories and sector maps. Once initialized, the disk should never need to be reinitialized unless catastrophic damage occurs (like a wild program writing all over the disk). The initialization process destroys all data previously written on the disk and it is vital that disks with good data not be reformatted.

When the C4MAT program is run, it will ask you if you are sure you want to initialize the disk. If you do, type "Y". C4MAT will then ask for a volume name, which can be up to eight characters in length. The volume name is stored in the disk information sector and is displayed by the CAT command and others. Be absolutely sure the disk is not write protected when you run the C4MAT program. An initial directory capable of storing up to 750 files is built upon the disk. The directory will expand if more than 750 files are placed on the disk.

Due to the large size and high density of the disk, it is not unusual for several bad sectors to be found. The program will abort if it cannot find enough useable free sectors on the disk. All bad sectors are automatically bypassed.

The CLEAN command is used in conjunction with a Remex FD-08 cleaning kit to clean DMAF eight inch disk drive heads.

DESCRIPTION

The general syntax of the CLEAN command is:

CLEAN [,<drive number>]

where <drive number> is optional and defaults to drive zero. The CLEAN command will then prompt you to load the cleaning diskette into the specified drive. When the diskette is in place, the heads are loaded and stepped back and forth over the cleaning surface for thirty seconds. You should then remove the cleaning disk and inspect it for oxide deposits. If deposits are noted, follow the manufacturers recomendations for media replacement. Frequent oxide deposits could indicate defective disk heads.

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 5,2,79 Sets the date register to May 2, 1979

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

Example:

```
+++DATE
May 2, 1979
```

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 (see the PRINT command) can not be deleted using the DELETE command.

ECHO

The ECHO command is a utility that permits messages and control characters to be sent to the terminal device. It is particularly useful when used inside of EXEC files.

DESCRIPTION

The general syntax of the ECHO command is:

```
ECHO,<string>
```

where string is any string of printable characters or control escape sequences terminated by a carriage return or an end of line character. Some examples of the echo command are:

```
ECHO,THIS IS A MESSAGE
ECHO,\G THE FILE HAS BEEN DELETED!
```

The first example types "THIS IS A MESSAGE" on the terminal. The second example uses a control escape sequence to send a bell character (Control "G") to the terminal, followed by the message "THE FILE HAS BEEN DELETED!".

Control escape sequences provide a mechanism to send control characters to the terminal device, for example, a bell character may be sent to provide an audible alert signal. These sequences begin with a backslash character followed by an upper case letter or symbol. The 'control' value of the symbol is sent to the terminal. For example, the sequence "\G" sends a control-G character. Two other escape sequences available are "\Ø" which sends a null character (control-shift-P) and "\n" (Lower case "N") which sends a carriage return, line feed to the terminal.

The EXECute command is used to process a text file as a list of commands, just as if they had been typed from the keyboard. This is a very powerful feature of FLEX for it allows very complex procedures to be built up as a command file. When it is desirable to run this procedure, it is only necessary to type EXEC followed by the name of the command file. Essentially all EXEC does is to replace the FLEX keyboard entry routine with a routine which reads a line from the command file each time the keyboard routine would have been called. The FLEX utilities have no idea that the line of input is coming from a file instead of the terminal.

DESCRIPTION

The general syntax of the EX command is:

```
EXEC,<file spec>
```

where <file spec> is the name of the command file. The default extension is TXT. An example will give some ideas on how EXEC can be used. One set of commands which might be performed quite often is the set to make a new system diskette on drive 1 (see NEWDISK). Normally it is necessary to use NEWDISK and then copy all .CMD and all .SYS files to the new disk. Finally the LINK must be performed. Rather than having to type this set of commands each time it was desired to produce a new system diskette, we could create a command file called MAKEDISK.TXT which contained the necessary commands. The BUILD utility should be used to create this file. The creation of this file might go as follows:

```
+++BUILD,MAKEDISK
  =NEWDISK,1
  =COPY,0,1,.CMD,.OV,.LOW,.SYS
  =LINK,1.FLEX
  =#
+++
```

The first line of the example tells FLEX we wish to BUILD a file called MAKEDISK (with the default extension of .TXT). Next, the three necessary command lines are typed in just as they would be typed into FLEX. The COPY command will copy all files with CMD, OV, LOW, and SYS extensions from drive 0 to drive 1. Finally the LINK will be performed. Now when we want to create a system disk we only need to type the following:

```
+++EXEC,MAKEDISK
```

We are assuming here that MAKEDISK resides on the same disk which contains the system commands. EXEC can also be used to execute the STARTUP file (see STARTUP).

There are many applications for the EXEC command. The one shown is certainly useful but experience and imagination will lead you to other useful applications.

IMPORTANT NOTE: The EXEC utility is loaded into the upper end of user memory. This is done by first loading EXEC into the utility command space, then calculating the proper starting address so that it will reside right up against the end of the user memory space. Next EXEC is relocated to that location and a new end of memory pointer is set to just below EXEC. When the EXEC file is finished, if the user has not further changed the memory end location, EXEC will reset it to the original value.

The FIX command is used to modify binary files that are stored on the disk. Since the FIX command loads the file into an internal buffer, it is possible to modify binary files that have several segments or that load into system locations.

DESCRIPTION

The general syntax of the FIX command is:

FIX,<input file name> [,<output file name>]

where <input file name> is the name of the file you wish to modify, and <output file name>, if specified, is the name of the file into which the modified copy will be written. The default extension for FIX is .BIN and the drive defaults to the working drive. If the output file is not specified, the modified binary file will replace the input file.

When you run the FIX command, the computer will load the binary file into its internal buffer memory. If for some reason the file cannot be loaded into memory, an error message is produced and the file is left unmodified. After the file has been loaded, FIX will respond with a prompt character, ":", and will then accept one of the following single-letter commands:

B -- Add a new block of data to the file. This command requires a pair of addresses specifying the lower and upper bounds of the block to be added. The block is initially cleared to zeros. For example, to add a sixteen byte object code block at location $0700, type :B 0700-070F. Added blocks may be modified with the memory examine and change function.

E -- Exit. All data that was modified is written back to the output file on the disk. The resulting file has all of the object code blocks that were present in the original file, plus any new blocks that have been added. There may be exactly one transfer address in the file, and it will be the last block in the file. If no output file name was specified, the exit command will delete the old binary file and write a new file in its place.

L -- Display File Limits. This command will display the transfer address and the limit addresses of each contiguous block of object code in the file.

M -- Memory Examine and Change. This command is used to examine or modify a byte in the file. It cannot be used to extend the file by adding additional bytes. See the detailed description below.

N -- Next line. This command displays the next sixteen bytes of the binary file. It is normally used after the V or P commands (see below).

P -- Peek at the file. This command allows you specify a single address and have that address and a few surrounding bytes displayed in a format similar to that of the V command (see below).

T -- Specify the transfer address. For example to change the transfer address to $0100, type :T 0100.

U -- Remove the transfer address from the file. See the description of a binary file in the advanced programmer's guide.

V -- View a section of the file. This command expects a range of addresses and displays the contents of the binary file between these addresses. The binary code is displayed in both hexadecimal and ASCII. Any addresses in the range not contained within the binary file are displayed as a pair of hyphens.

X -- Exit (Give Up). No modified data is written back to the disk. (The file is left intact.) This command is useful if you have managed to screw up the file while making modifications.

FIX Memory Examine and Change

The memory examine and change function of fix permits the programmer to look at and change bytes in a binary file by referencing the address at which they will load. This function cannot be used to add additional bytes to the binary file. If you need to add data, use the "B" command to create the new bytes, and then use the memory command to change them.

1.) Enter the "M" command followed by the four digit address of the byte you wish to examine or change. The computer will display the address followed by the data found in the byte. If no such byte exists in the binary file, a question mark will be shown.

2.) At this point the user has the option of advancing, either forward or backward, to the next memory location, changing the data stored at the displayed address and advancing to the next location, or of exiting the examine and change function.

   a.) To display the next sequential address and data, type a space.

   b.) To display the previous address and data, type the up-arrow, "^".

   c.) To change the data stored at the displayed location simply enter the new data as two hexidecimal digits. If a non-hex value such as "3Q" is entered the data will remain unchanged and the memory change function will step to the next sequential address.

d.) If a delete character (control-X) or a backspace character (control-H) is typed, the current address and data is redisplayed.

e.) To exit the Memory Examine and Change function, type a carriage return.

The FIND command searches for a specified text pattern in one or more files and outputs the lines that contain the pattern.

DESCRIPTION

The general syntax of the FIND command is:

FIND [,+<option(s)>] ,pattern [,<filename(s)>]

where <options> designate the options to be used by FIND. The pattern is a collection of characters, some of which have special meanings. If no filename is given, standard input is assumed. By default, lower case letters in the pattern match only lower case letters in the file being searched, and upper case letters in the pattern match only upper case letters in the file.

Options Available:

y - all lower case letters in pattern will match both the upper and the lower case letters in the file being searched. (NOTE: This option does not affect upper case letters in the pattern; they only match upper case letters in the text being searched.)

Example: find +y "and" file1
will match "and", "And", "AND", and so on.

c - the output of matching lines will be suppressed. A count of the matching lines for each file searched will be output.

Example: find +c "and" file2.txt
would produce:
5 matching lines in file: file2.txt

Pattern:

A group of characters have been assigned special meanings, these characters are called metacharacters. An escape character (\) preceeding a metacharacter "turns off" the special meaning of that character. The pattern should be surrounded by quotes, i.e. "pattern", in order to match on imbedded spaces.

Metacharacters

\ - The escape character. When it is used before a metacharacter, it removes the special meaning from the character.

? - Matches any character except a newline.

< - Matches at the beginning of lines only. The "<" character is special only if it is the first character in the pattern string.

> - Matches at the end of lines only. The "<" character is special only if it is the last character in the pattern string.

& - Conjunction character. Matches if the subpattern before the "&" character and the subpattern after the "&" character are both present in the line being searched.

(str|str) - Alternation sequence. A match will be found if either string occurs in the line being searched. Note that these are simple character strings, metacharacters other than the "|" character will be treated as regular characters.

Explaination of Character Classes

Syntax for character class:
[list/range of characters]

Syntax for negation of character class:
[! list/range of characters]

A character class is a text pattern that matches on any single character from the bracketed list of characters. Character classes can consist of valid ranges such as "A-Z", "b-f", or "0-8". They also can consist of a list of characters, or they can be combination of both a range and a list. A character range such as "A-G" is internally expanded and will match on "ABCDEFG", in the line being searched. A negation of a character class matches on any character not contained in the character class.
Some examples:

[A-Z] will match on any upper case letter.
[a-z] will match on any lower case letter.
(If y flag is in use, it will match both upper and lower case letters.)
[0-9] will match on any digit.
[!a-f] will match on any character except a, b, c, d, e, or f.
[aeiou] will match on any lower case vowel.
[aAbB] will match on the characters: a, A, b, and B.

(NOTE: If the y flag is in use, the character class [a-z] will match upper and lower case letters, the character class [A-Z] will match only upper case letters (as always).)

Examples and Further Explanations of Metacharacters

1. find "?oad" file1 will search file1.txt for the pattern "?oad", where "?" can be any character but a newline. It would match "toad","load","road",... And it would produce output like:

```
File: file1
    5 The road was very wet.
```

2. find "<int" file1.asm will look for "int" at the begining of every line in file1.asm.

3. find "<single>" file1 will look for "single" on a line by itself in file1.txt.

4. find [A-Z][a-z] file2 will match a two letter sequence of an upper case letter and a lower case letter in file2.txt. (Such as the "Th" in "This is a test sentence.")

5. find un(happy|lucky) file3 will match the word "unhappy" or the word "unlucky" in file3.txt.

6. find is&it file3 will match lines containing both "is" and "it". The words can occur in any position in a line. It would match on all of the following lines in file3.txt:

```
This is the story of it.
it is the worst day of my life
His bird ate it.
The kites are his.
```

7. find "\(2\>3)" file6.asm will match a line containing "(2>3)" in file6.asm. The "(" and the ">" were escaped to make them regular characters. The ")" did not need to be escaped because it was not used in the context of an alternation.

8. find +y [A-Z][aeiou][a-z] file9 will match all three-character sequences in file9.txt with an upper case first character, with an upper case first character, an upper or lower case vowel as the second character, and an upper or lower case third character. Since the y flag was used, the second and third characters matched may be upper or lower case.

9. find " [!A-Z]??? " words.txt contains a negated character class. It will match a sequence of characters beginning with a blank, the second character cannot be an upper case letter (because the character class was negated), and following that it will match on the next three characters, except a newline (the ??? part of the pattern), and finally a blank character after those. It would match on the following strings: " that ", " a ", " ", " aA?& ", ...

Additional Notes about FIND

Multiple files can be searched for the same pattern by including the filenames on the command line. The lines in which the pattern is found will be output (preceded by a file name before the first line matched in each file.)

```
Example:    find "int" file1.txt file2.txt file3.txt

File:  file1.txt
    3  int j,k,count;
   10  printf("n");

File:  file2.txt
    5  int i,j;

File:  file3.txt
    8  int kk,ll;
```

For more complete details about the way most of this program works see SOFTWARE TOOLS by Brian W. Kernighan and P.J. Plauger, Addison-Wesely Publishing Co., 1976. Chapter 5: Text Patterns, outlines a find program that was the basis of this program.

I

The I command allows a utility to obtain input characters from a disk file rather than the terminal.

DESCRIPTION

The general syntax of the I command is:

```
I,<file spec>,<command>
```

where <file spec> is the name of the file containing the characters to be used as input and <command> is the FLEX utility command that will be executed and that will receive that input from <file spec>. The default extension on <file spec> is .TXT.

For example, say that on a startup you always wanted the file DATA.DAT deleted from the disk without having to answer the "ARE YOU SURE?" questions. This could be done in the following manner:

```
+++BUILD,YES
=YY
=#
```

The first Y will answer the "DELETE 0.DATA.DAT?" question while the second Y will answer the "ARE YOU SURE?" question.

```
+++BUILD,STARTUP
=I,YES,DELETE,DATA.DAT
=#
```

Upon booting the disk, FLEX will execute the STARTUP file and perform the following operation: delete the file DATA.DAT receiving all answers to any questions from the input file YES.TXT rather than from the terminal.

See the description of the STARTUP command for more information on STARTUP.

The JUMP command is provided for convenience. It is used to start execution of a program already stored in computer RAM memory.

DESCRIPTION

The general syntax of the JUMP command is:

JUMP,<hex address>

where <hex address> is a 1 to 4 digit hex number representing the address where program execution should begin. The primary reason for using JUMP is if there is a long program in memory already and you do not wish to load it off of the disk again. Some time can be saved but you must be sure the program really exists before JUMPing to it!

As an example, suppose we had a BASIC interpreter in memory and it had a 'warm start' address of 103 hex. To start its execution from FLEX we type the following:

+++JUMP,103

The BASIC interpreter would then be executed. Again, remember that you must be absolutely sure the program you are JUMPing to is actually present in memory.

The LINK command is used to tell the bootstrap loader where the FLEX operating system file resides on the disk. This is necessary each time a system disk is created using NEWDISK. The NEWDISK utility should be consulted for complete details on the use of LINK.

DESCRIPTION

The general syntax of the LINK command is:

LINK,<file spec>

where <file spec> is usually FLEX. The default extension is SYS. Some examples of the use of LINK follow:

```
+++LINK,FLEX
+++LINK,1.FLEX
```

The first line will LINK FLEX.SYS on the working drive, while the second example will LINK FLEX.SYS on drive 1. For more advanced details of the LINK utility, consult the "Advanced Programmers Guide".

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.

L

The L command is used to list a text file onto the terminal or any other display device.

DESCRIPTION

The general syntax of the L command is:

L,[+<option(s)>],<filename>,([+<option(s)>],<filename>)

where <options> designate the options to be used for the file(s) following. All options retain the conditions assigned to them until they are explicitly changed. The options should be specified in a string (one after another) with a + or - at the beginning of the string to denote an options string. Within the options string a + or - can be used to separate options where the syntax would be unclear otherwise. This is particularly useful for separating line number ranges from other options. <Filename> designates the file to be listed. If no extension is specified on the filename, .txt is the default. A space can also separate the filenames and options. If no filename is given, standard input is assumed.

The L utility allows the user to examine entire files or a selected range of lines of the file(s). Line numbers are printed by default.

Options Available

([num] - signifies use of a decimal integer)

p - Output will be formatted into pages. This option is especially useful when the output is directed to a hardcopy device.

h - Output will be formatted into pages with a heading. The L program will prompt the user to "ENTER PAGE HEADING: ", at which time a heading of up to 100 characters may be entered, with a carriage return indicating end of input. The sequence CANCL, RETURN will delete any part of the header already typed in, and make the header blank in the listing. Backspacing will remove unwanted characters from the header. (Note: if the folding option is used, the heading will be truncated to 50 characters.)

n - Line numbering will be suppressed.

t[num] - The output lines will be truncated after the specified number of characters. No truncation will occur if the t option is used without a number, or with zero.

w[num] - Folding: the w option specifies the width in characters of text to be listed. The width defaults to 80 characters if the w option is used without a number, or with zero. (Folding a line means that the specified number of characters are printed and if the input line is longer than that number, a newline character is output before the rest of the line is listed. This is particularly useful for listing files with long lines on a hard copy device.)

l[num] - Left pad/truncate: the l option can either pad the output line with blanks on the left margin, or truncate the input line on the left.
If the specified number is positive, then that many blanks will be put before the first character in the output line. (If line numbering is enabled, then the blanks are inserted before the line numbers in the output lines.)
If the specified number is negative, then that many characters will be cut from the beginning of the lines to be output. (Line numbers are output as usual, if line numbering is enabled.)

c[num] - Centronics: c is an option package design to be used when output is directed to a Centronics printer. The c option turns on paging format and heading.
A c1 sets the width at 70, useful on the Centronics 737.
A c2 sets the width at 120, useful on the Centronics 704.

Line Number Ranges

The line number range option is useful when only part of a file needs to be listed. It is available in two forms.

Type 1: [num] - starting line number: the files(s) specified will be listed starting at the line number specified.

Type 2: [num]-[num] - starting line number - ending line number: two numbers separated by a hyphen indicate a line number range. The specified file(s) are each listed starting at the first specified number and the listing is halted at the second specified number.

Examples and Further Explanations of Options

1. 1 +ht80 file1.txt

will prompt the user for a heading, and then list the FILE: file1.txt with all output lines truncated after the 80th character, in page format, and with line numbers.

2. 1 +nw8015 file5.asm

will list the FILE: file5.asm without line numbers, folding the lines after the 80th character, and padding the left margin with 5 blanks.

3. 1 +1-50 file4 +1-25 file7

will list lines 1 through 50 of FILE: file4.txt with line numbers. and lines 1 through 25 of FILE: file7.txt

4. 1 +cl file9.old

will list the FILE: file9.old with all special options that are associated with cl: line numbering, folding at 70, paging, and heading.

5. 1 +w40+10np file6

will list the FILE: file6.txt starting at line number 10, folding after the 40th character, with no line numbers, and in page format.

The MV command is a file move utility. MV will take any actions required to generate a file with the specified name on the specified drive. The actions taken may include renaming a file, deleting an existing file, or physically moving a file.

DESCRIPTION

The general syntax of the MV command is:

MV,<source file spec> [,<target file spec>]

where <source file spec> is the name of the file to be moved, and <target file spec> is the name to be associated with the file. The default extension for the source file specification is ".TXT". The default drive number for the source file specification is the working drive.

If the target file specification is incomplete, those items that are missing will be supplied from the source file specification. In the following example, the completed target file specification will be "0.DATAFILE.TXT":

+++MV,0.DATAFILE.BAK,.TXT

In this example, the intention was to change the name of the file "DATAFILE.BAK" on drive zero to "DATAFILE.TXT". Note that if a file having the name "DATAFILE.TXT" does not already exist on drive zero, the only action taken will be to rename the file "DATAFILE.BAK" to have the name "DATAFILE.TXT". If instead a file named "DATAFILE.TXT" already exists, the action taken is to first delete the file "DATAFILE.TXT" followed by renaming the file "DATAFILE.BAK". In either event, the end result is that "DATAFILE.BAK" is moved to "DATAFILE.TXT".

It should be noted that all actions taken by the MV program are taken WITHOUT informing the operator of the details necessary to perform the move function. This includes any required move operations or the deletion of existing files.

Consider the case where the file "DATAFILE.BAK" is protected against deletion (see the PROT command) in the following example. Since a file that is protected cannot be renamed, in order to obtain a file with the name "DATAFILE.TXT" it is necessary to physically copy the file "DATAFILE.BAK" into a new file named "DATAFILE.TXT" on drive zero:

```
+++PROT,DATAFILE.BAK,+D
+++MV,0.DATAFILE.BAK,.TXT
-- Source file not deleted.
```

Since the file "DATAFILE.BAK" is protected and hence cannot be deleted, the message "-- Source file not deleted." is displayed.

In the next example, the intention is to move the file "DATAFILE.TXT" from drive zero to drive one. This example illustrates another case where it is necessary for the MV command to copy the data in a file:

```
+++MV,0.DATAFILE,1
```

In this case, the completed source file specification is "0.DATAFILE.TXT" and the completed target file specification is "1.DATAFILE.TXT". In order to accomplish the specified move, the MV program must take the following actions:

1. If a file named "DATAFILE.TXT" already exists on drive one, it is deleted.

2. The file "DATAFILE.TXT" on drive zero is copied in its entirity to drive one. This process creates a duplicate of the original file, retaining the data and attributes of the original file.

3. Unless it is protected, the file "DATAFILE.TXT" on drive zero is deleted.

The end result of these actions is to place a file named "DATAFILE.TXT" on drive one which is identical to the file named "DATAFILE.TXT" which had previously existed on drive zero. That is, the file has been moved from drive zero to drive one.

In the final example, the working drive has been assigned to drive zero. The file "WORKFILE.TXT" exists on drive zero, but has been damaged due to operator oversight. A backup of this file, "WORKFILE.BAK" exists on drive one and will be used to replace the damaged file:

```
+++MV 1.WORKFILE.BAK,0..TXT
```

In this case, it is necessary to specify both the source file drive (which otherwise would default to drive zero) and the target file drive number. The completed target file specification will be "0.WORKFILE.TXT". The MV program will move the backup file by first deleting "WORKFILE.TXT" on drive zero, then copying the file "WORKFILE.BAK" from drive one to drive zero, and finally deleting the file on drive one. Again, all these actions take place without the intervention of the operator.

MIRROR

The MIRROR program is used to produce a mirror image copy of a disk. A double sided, double density eight inch floppy disk may be copied in approximately 90 seconds.

DESCRIPTION

The general syntax of the MIRROR command is:

```
MIRROR,<input drive>,<output drive> [,+V]
```

where <input drive> is the drive number of the disk to be duplicated, and <output drive> is the drive number of the disk to be overwritten. The +V option may be used to supress verification of the written data. All data and directory information (including free space, bootstrap sectors, etc.) is copied from the input drive to the output drive. The mirror program requests confirmation from the operator, since the entire contents of the output disk will be overwritten. An example of the use of mirror is as follows:

```
+++MIRROR,0,1
Formatted scratch disk in drive 1? Y
Are you sure? Y
-- Mirror Complete.
```

Copying is performed on a sector by sector basis with no regard for the contents of the disk. No reorginization is performed on files, free space, or on the directory. Unless supressed by the "+V" option, all data written to the output disk is verified, regardless of the state of the Flex VERIFY flag. After the entire contents of the disk has been copied, the volume name and number of the output disk are restored to their contents prior to the mirror operation.

If read or write errors are detected during the copy operation, they are retried several times. If the retries fail, the error is considered permanent and the mirror operation is aborted. Note that this implies that the mirror program cannot be used to copy from or to disks that have defective sectors.

Since no formatting operation is performed on the output disk, both disks must have previously been formatted (via the NEWDISK program) and both disks must have identical format (Single or Double Sided, Single or Double Density, etc.) It is not necessary to reformat an already formatted disk prior to running the mirror program as its entire contents will be overwritten.

NEWDISK

Newdisk is used to format a new diskette. Diskettes as purchased will not work with FLEX until certain formatting information has been put on them. The NEWDISK utility puts this information on the blank diskette, as well as checking for surface defects on the media.

DESCRIPTION

The general syntax of the NEWDISK command is:

NEWDISK,<drive>

where <drive> represents a single digit drive number and specifies the drive containing the diskette to be formatted. After entering the command, the system will ask if you are sure you want to format the diskette (Remember, the NEWDISK process will remove any information previously contained on the disk), and if diskette to be initialized is a scratch disk. Type 'Y' as the response to these questions if you are sure the NEWDISK command should continue.

Certain versions of NEWDISK will also ask you if you want to make double sided, double density, or extra density diskettes, and about how many tracks to format. These questions relate to the hardware configuration of the system in use and are summarized below.

NEWDISK will then prompt for a volume name and number. This gives you the ability to "name" the diskette for future reference. The volume name consists of eight characters, with an optional three character extension. The volume number should be in the range of 1 to 32000. Note that it is exceedingly poor practice to generate diskettes without volume identifiers.

The NEWDISK process takes several minutes to initialize a diskette, assuming there are no bad spots that must be accounted for. Defective sectors will make NEWDISK run even slower, depending on the number of bad sectors found. As bad sectors are detected, messages will be output to the terminal such as:

BAD SECTOR AT xxyy

where 'xx' is the diskette track number (in hex) and 'yy' is the sector number, also in hex. NEWDISK automatically removes bad sectors from the list of available sectors, so even if a diskette has several bad spots on it, it is still usable. When NEWDISK finishes, it will report the number of available sectors on the disk.

Sometimes during the NEWDISK process, a sector will be found defective in an area on the diskette which is required by the operating system. In such a case, NEWDISK will report:

FATAL ERROR - FORMATTING ABORTED

You should not immediately assume that the diskette is unusable if this occurs. You should remove the diskette from the drive and clean the heads in that drive using one of the available head cleaning kits. Then you should re-insert the diskette that was flagged as faulty, and try to format it again. If after several attempts the formatting process is still aborted, you can assume the diskette is unusable and discard it.

Diskette Formats:

The NEWDISK program provides several optional diskette formats. Certain formats may only be used with specific configurations of disk controllers. This table summarizes available formats and hardware constraints on their use:

1. Single Density -- supported by all controllers and drives. Diskettes that are to be shipped to other systems should be formatted in this manner if it is at all feasable.

2. Double Density -- supported by DC-4 and DMF-2 controllers with Qume, Remex and Tandon disk drives. Other configurations should use single density only.

3. Extra Density -- supported by DMF-2 controllers with Qume and Remex drives only. Note that the MIRROR command runs considerably slower with diskettes formatted extra density.

4. Quad Density -- supported by DC-4 controllers using Tandon or Qume quad density 5-inch drives. Note that diskettes should be certified for use with Quad Density.

5. Double Sided -- supported by DC-3 and DMF-1 (and higher) controllers using SOME Wangco and Calcomp drives, and ALL Qume, Remex, and Tandon drives. Note that diskettes certified for double sided operation must be used.

6. 40 Track - supported by DC-2 (and higher) controllers using Wangco, Tandon, or Qume 5 inch drives. Note that diskettes certified for 40 track operation must be used.

7. 77 Track - supported by DC-4 controllers using Tandon or Qume quad density drives with Verbatim Datalife quad density 5 inch diskettes.

8. 80 Track - supported by DC-4 controllers using Tandon or Qume quad density drives and Dysan or IBM quad density 5 inch diskettes certified for 80 track operation.

CREATING SYSTEM DISKETTES

A system diskette is one from which the operating system can be loaded. Normally the system diskette will also contain the Utility Command Set (UCS). The following procedure should be used when preparing system disks.

1. Initialize the diskette using NEWDISK as described above.

2. COPY all .SYS files desired to the new disk.

3. COPY all .CMD files to the new disk. It should be noted that steps 2 and 3 can be done with one command; "COPY,0,1,.SYS,.CM,.OV,.LOW", assuming that the new diskette in in drive 1 and that the operating system and all commands and their overlays are desired. (the .OV copies overlay files and .LOW copies the utility 'SAVE.LOW').

4. Finally LINK the file FLEX.SYS to the system using the LINK command.

It is not necessary to make every diskette a system diskette. It is possible to create 'working' diskettes, which do not have the operating system on them, for use with text files or BASIC files. Remember that a diskette can not be used for booting the system unless the operating system is contained on it and it has been linked.

The N command is used to automatically answer "N" to prompts produced by various FLEX utilities. This facility is especially useful when writing EXEC files.

DESCRIPTION

The general syntax of the N command is:

```
N,<command string>
```

where <command string> is a valid command line to be passed to FLEX. If the N command is used with multiple commands per line using the end of line character, it will only affect the command immediately following it. The N command is particularly useful when performing COPY operations that you do not wish to overwrite any existing files:

```
+++N,COPY,0,1,TEXT

  FILE    TEXT1   .TXT    TO DRIVE #1    COPIED.
  FILE    TEXT2   .TXT    TO DRIVE #1  FILE EXISTS
            DELETE ORIGINAL? N
  FILE    TEXT3   .TXT    TO DRIVE #1  FILE EXISTS
            DELETE ORIGINAL? N
  FILE    TEXT4   .TXT    TO DRIVE #1    COPIED.
```

The NF command is used to filter an output stream to remove form feed control characters. This is useful with certain printers that do not support form feeds or vertical tabs.

DESCRIPTION

The general syntax of the NF command is:

<print command>,NF,<command string>

where <print command> is one of the printer support commands (such as P, Q, etc.) and <command string> is a valid command line to be passed to FLEX. If the command line contains multiple commands, the NF command (and the print) will refer only to the first command. An example of the use of the NF command will clarify its use:

+++P,NF,CAT

This example will produce a catalog listing of the working drive on the printer selected by the P command. It is assumed that this printer does not have form feed capability (like the Centronix 737 printer).

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 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

The P command is used to direct the output of commands in the Utility Command Set to a parallel printer. It is normally used to produce hard copy output from text processors, assemblers, and other utility programs.

DESCRIPTION

The general syntax of the P command is:

P[#n],<command string>

where #n is optional and is the port number of the parallel interface connected to the printer and <command string> is a valid command line to be passed to FLEX.

If the port number is not specified, it will default to port seven, except on S/09 computers which will default to the MP-ID parallel port. If the P command is used with multiple commands per line using the end of line chracter, it will only affect the command immediately following it in the command line. Some examples will clarify the use of the P command:

```
+++P,CAT
+++P#3,LIST,TEXT
+++P#5-B,LIST,TXT
```

The first example will produce a printed listing of the catalog of the working drive. The port number is omitted and defaults to port seven or the MP-ID port, depending on the computer system.

The second example will produce a printed listing of the file TEXT.TXT on the printer connected to port number three. When using dual parallel ports, the port side may be specified by using an A or B suffix to the port number. The suffix is separated from the port number by a hyphen or slash character. If the suffix is omitted, the A side of the port is selected. In the third example, the port specification of 5-B will use the B side of port number five.

The P command initially loads into the utility command space at $C100. If sufficient memory has been reserved for the printer driver (see the RM command documentation) it will be relocated there, otherwise it will be relocated into the highest available locations in user memory and the end of user memory pointer will be updated. When the command to be printed has completed any user memory allocated to the printer driver will be released.

The parallel data and handshake conforms to the CENTRONIX® interface standard. This standard has been widely adopted for use with 8-bit ASCII printers and is available on nearly all parallel printers.

The P.COR file is a command file that can be used to construct customized printer driver commands. It contains the relocation and memory management routines required to use position independant printer drivers.

DESCRIPTION

P.COR is never used as a command by itself. Special internal checks are made to insure that a printer driver has been properly appended to the P.COR file. In order to use P.COR, you must write a position independant printer driver that begins at memory address $C300 and may extend up to location $C6FF, thus allowing the driver to be up to 1K in length. This is normally sufficient space for any reasonable printer driver. For more information on writing a position independant driver, see section X in chapter 3.

Once the printer driver has been written and assembled, you must use the append command to combine the P.COR file with the printer driver. To clarify this procedure, consider the following example. A serial printer driver (A listing of this driver can be found in Chapter 3 section XI) named SERIAL has been assembled. In order to make a print command out of this driver, the following command line must be entered:

```
APPEND,P.COR,SERIAL.BIN,SERIAL.CMD
```

The result is a print command that functions like the P command. To use the new SERIAL command, enter

```
SERIAL[#n],<command string>
```

where #n is the port number (you have set the default) and <command string> is any command string to be passed to Flex.

The PO command is the older version of the P command and has been included for compatability purposes.

DESCRIPTION

The general syntax of the PO command is:

PO,<command string>

where <command string> can be any standard utility command line. If PO 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:

```
+++PO,CAT
+++PO,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 PO did not cause the 'CAT,1' to go to the printer.

The PO command tries to load a file named PRINT.SYS from the same disk which PO itself was retrieved. The PRINT.SYS file which is supplied with the system diskette contains the necessary routines to operate a parallel printer connected to port seven of a /09 computer system. For complete details on these routines, refer to section XI of this publication.

The Print Spool Program provides FLEX with the ability to output stored data to a printer at the same time that it is performing other tasks. This method of printing is called Printer Spooling. This feature is especially useful when it is necessary to print a long listing without tying up the computer.

DESCRIPTION

The general syntax of the Print Spool Program is as follows:

PSP,[<spool file>] [,<print command>] [,+<repeat count>]

where <spool file> is the name of the file to be printed (normally produced by the "O" command), <print command> is the name of the print command that will be used to drive the printer, and <repeat count> is the number of additional copies of print you require. The default extension of the spool file is .OUT and the default extension for the print command is .CMD.

For example, say that your disk had a very large number of files on it and a printer catalog listing was desired. A file containing the output information should first be created by using the O command such as:

+++O,CAT.OUT,CAT.CMD or +++O,CAT,CAT
(see the description of the O command)

when printer output is desired the command

+++PSP,CAT.OUT,P.CMD or +++PSP,CAT,P

should be entered.

At this time the location of the file CAT.OUT is stored in a buffer called a print queue (waiting list). If another PSP command is issued before the first is finished, the second file will be in the next available location in the print queue.

After the file name to be printed has been stored in the print queue, the PSP program will load the specified printer driver command into the printer reserved area (See the RM command). If the reserved area is not big enough, a message is issued and the program terminates. Otherwise, the spooling process is started and control returns to the FLEX command interpreter. Note that any of the relocatable printer commands may be used, including the Q and SP commands.

Once printing is in progress, it is not possible to change printer drivers until the spooling process has completed. For example, if you have three files that you wish to print on a Qume printer, and two on a serial printer, you must first spool the three Qume files and allow them to complete printing. You can then spool the two files destined for the serial printer.

During printing, you can perform flex commands from the terminal, such as deleting files, copying disks, etc. While you are using FLEX, the Print Spool Program will be printing the desired file. PSP will automatically wait for the printer to become ready (power up) even after the file has been entered into the print queue. After printing the first file, the second file in the queue will be printed (if there is one), etc. The print queue may be examined or modified at any time by using the QCHECK utilty.

NOTE: There are several things that the user should be aware of when using the printer spooling:

1) Any file that is in the print queue may not be deleted, renamed, or changed in any way until it has been printed or removed by the QCHECK print queue manager utility.

2) Disks which contain the files in the print queue should not be removed while the files are still in the queue.

3) Non-Spooling print commands (like P) cannot be used while files are waiting in the print queue.

4) Any paper or cassette tape load or any other operation which requires that the computer accept data at precise time intervals should not be executed during a printer spooling operation.

5) In order for printer spooling to work in a non-S/09 computer system, an MP-T or MP-T2 interface must be installed in I/O port five and be strapped to provide IRQ interrupts.

6) The PSP command is not supported by FLEX9S.

7) The PSP command will function only with FLEX 9.0 Version 2.6 or above. If in doubt, check with "VER FLEX.SYS".

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, APPEND, etc.

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.

PUTBOOT

The PUTBOOT command is used to write the FLEX bootstrap loader onto a disk in FLEX format.

DESCRIPTION

The general syntax of the PUTBOOT command is:

```
PUTBOOT,<drive spec>     or
PUTBOOT,<file spec>
```

where <drive spec> is the drive number containing the disk to be written. If a file specification is provided, PUTBOOT will also perform the LINK function to the specified file. In either case, an explicit drive number MUST be specified. PUTBOOT will not default to the working drive. For example, in order to write a new bootstrap onto the disk in drive one, type:

```
+++PUTBOOT,1
```

This will cause a new bootstrap to be written to the disk. Note that this does not format the disk, nor does it change any files on the specified disk.

The bootstrap loader is normally written onto the FLEX disk when it is initially formatted with the NEWDISK program. The PUTBOOT program may be used to re-write the bootstrap in the event that it is damaged, or to update to a newer version of FLEX without having to copy the entire disk to a newly-formatted disk.

IMPORTANT NOTE

FLEX versions 2.6:8 or higher have a revised bootstrap loader. The new bootstrap must be present in order to be able to successfully boot the new versions of Flex. In order to upgrade a disk to these new versions, the following procedure should be followed:

1. The new release FLEX disk should be booted and the disk to be upgraded placed in drive one with write enabled.

2. The PUTBOOT program should be used to write a new bootstrap onto the old disk. This will not affect any data currently on the disk except the bootstrap itself.

3. The AR program should be used to replace any older programs that have been re-released. Consult the AR documentation for details.

4. Copy any new programs you wish to the upgraded disk. This completes the upgrade procedure.

The Q command is used to direct the output of commands in the Utility Command Set to a Qume Sprint-3™ printer attached to an MP-QP I/O interface. It is normally used to produce hard copy output from text processors, assemblers, and other utility programs.

DESCRIPTION

The general syntax of the Q command is:

Q[#<n>][/<format>],<command string>

where #<n> is optional and is the port number of the MP-QP interface connected to the printer, <format> is optional and is is one of the letters A, B, C, D, E, or F, and <command string> is a valid command line to be passed to FLEX. If the port number is not specified, it will default to port seven. If the Q command is used with multiple commands per line using the end of line chracter, it will only affect the command immediately following it in the command line.

The format letter is separated from the port number or Q command by a slash character, and specifies the printwheel pitch and print spacing to be used. If the format character is omitted, it will default to format "B". This format represents standard elite spacing and is used with the WP LETTER GOTHIC 12 printwheel (number 80956) supplied with the printers. The following formats are available:

A - 10 Characters/Inch Horizontal, 6 Lines/Inch Vertical
B - 12 Characters/Inch Horizontal, 6 Lines/Inch Vertical
C - 15 Characters/Inch Horizontal, 6 Lines/Inch Vertical
D - 10 Characters/Inch Horizontal, 8 Lines/Inch Vertical
E - 12 Characters/Inch Horizontal, 8 Lines/Inch Vertical
F - 15 Characters/Inch Horizontal, 8 Lines/Inch Vertical

Some examples will clarify the use of the Q command:

+++Q,CAT

This example will produce a printed listing of the catalog of the working drive. The port number is omitted and defaults to port seven. The format is also omitted, and defaults to "B", 12 characters per inch horizontal, and 6 lines per inch vertical.

+++Q#3,LIST,TEXT

This example will produce a printed listing of the file TEXT.TXT. The port has been specified as port number three. No format is specified and the default "B" will be assumed.

```
+++Q/F,ROFF,LETTER
```

In this example, no port number is given so the default of seven will be selected. Format "F" has been selected so the printed output will be spaced 15 characters per inch horizontal, and 8 lines per inch vertical. This format provides 88 lines per 11 inch page, and matches the Qume GOTHIC 15 printwheel (number 82090).

The Q command initially loads into the utility command space at $C100. If sufficient memory has been reserved for the printer driver (see the RM command documentation) it will be relocated there, otherwise it will be relocated into the highest available locations in user memory and the end of user memory pointer will be updated. When the command to be printed has completed any user memory allocated to the printer driver will be released.

The QCHECK utility can be used to examine the contents of the print queue and to modify it contents. QCHECK has no additional arguments with it. Simply type QCHECK. QCHECK will stop any printing that is taking place and then display the current contents of the print queue as follows:

```
+++QCHECK
      POS       NAME       TYPE      RPT
       1        TEST.      .OUT       2
       2        CHPTR.     .OUT       0
       3        CHPTR2.    .TXT       0
  COMMAND?
```

This output says that TEST.OUT is the next file to be printed (or that it is in the process of being printed) and that 3 copies (1 plus a repeat of 2) of this file will be printed. After these three copies have been printed, CHPTR.OUT will be printed and then CHPTR2.TXT. The COMMAND? prompt means QCHECK is waiting for one of the following commands:

COMMAND	FUNCTION
(carriage return)	Re-start printing, return to the FLEX command mode.
Q	A Q command will print the queue contents again.
R,#N,X	An R command repeats the file at position #N X times. If X is omitted the repeat count will be cleared. Example: R,#3,5
D,#N	A D command removes the file at queue position #N. If N=1, the current print job will be terminated. Example: D,#3
T	A T command will terminate the current print job. This will cause the job currently printing to quit and printing of the next job to start. If the current files RPT count was not zero, it will print again until the repeat count is 0. To completely terminate the current job use use the D,#1 command.
N,#N	A N command will make the file at position #N the next one to be printed after the current print job is finished. Typing Q after this operation will show the new queue order. Example: N,#3
S	An S command will cause printing to stop. After the current job is finished, printing will halt until a G command is issued.

G — A G command will re-start printing after an S command has been used to stop it.

K — A K command will kill the current print process. All printing and queued jobs will be removed from the queue. The files are not deleted from disk.

The QCHECK command is not supported under FLEX9S and Multi-User Basic.

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.

The RM command is used to Reserve Memory in which to load printer drivers that are too large to fit in the actual printer driver area. This command is primarily intended to permit the use of large printer drivers (such as the QUME™ driver) with programs like BASIC that use all available memory.

DESCRIPTION

The general syntax of the RM command is:

RM[,<size>] or RM?

where <size> is a decimal number indicating the amount of memory to be reserved for the printer drivers. If not specified, the size parameter defaults to 512 bytes. If memory has previously been reserved for printer drivers, the RM command will adjust the amount of memory available as necessary. In order to get rid of the reserved memory, it is only necessary to run the RM command with a size value of zero. If the second form of the RM command is used, the amount of reserved memory will be reported.

Flex™ itself reserves approximately 56 bytes of memory for printer driver code and defines the printer vector locations. While this is sufficient to run simple parallel or serial printers, it is not enough memory to hold drivers for more complicated devices, for example, serial printers requiring buffers and line protocol. The RM command allows the user to reserve a memory area for the printer drivers that will be off limits to programs like BASIC that use all available memory. Note that it is NOT necessary to use the RM command to use the printer commands like P and Q.

Some examples of using the RM command follow:

```
RM,700
RM?
RM,0
```

The first example will reserve 700 bytes of memory for use as a printer area. The second example will display that 700 bytes are reserved, and the third example will free the memory reserved for the printer.

READPROM

The READPROM command is used to read the data from a 2716 compatible EPROM in a SWTPC MP-R EPROM programmer to a binary disk file. Its primary use is for copying and modifying the contents of EPROMs.

DESCRIPTION

The general syntax of the READPROM command is:

READPROM[#n], <file spec>

Where #n is optional and is the port number in which the PROM programmer is installed and <file spec> is the name to be assigned to the output file. The default extension on the file is .BIN and the default drive is the working drive. The default port for READPROM is #4. Some examples will clarify the use of READPROM.

```
+++READPROM,JUNK
+++READPROM#7,JUNK.1
```

The first example will store the contents of the EPROM on the programmer in port #4 to the file JUNK.BIN on the working drive. The second example will read the EPROM on the programmer in port 7 and store its contents in the file JUNK.BIN on drive 1. 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.

The output file generated by READPROM will be 2048 (2K) bytes in length with a load address of 0000. No transfer address will be assigned to the file.

The EPROM should not be installed in the programmer until the READPROM command tells you to.

Default Port Addresses

If desired, the default port address can be changed by using the FIX utility on READPROM.CMD.

READPROM Address	Contents
C100	S/09, 69A, 69K Default Port Address
C102	/09 Default Port Address

S

The S command is used to direct the output of commands in the Utility Command Set to a serial printer. It is normally used to produce hard copy output from text processors, assemblers, and other utility programs.

DESCRIPTION

The general syntax of the S command is:

S[#n],<command string>

where #n is optional and is the port number of the serial interface connected to the printer and <command string> is a valid command line to be passed to FLEX. If the port number is not specified, it will default to port seven. If the S command is used with multiple commands per line using the end of line chracter, it will only affect the command immediately following it in the command line. Some examples will clarify the use of the S command:

S,CAT

This example will produce a printed listing of the catalog of the working drive. The port number is omitted and defaults to port seven.

S#3,LIST,TEXT

This example will produce a printed listing of the file TEXT.TXT. The port has been specified as port number three.

When using dual serial ports, the port side may be specified by using an A or B suffix to the port number. If the suffix is omitted, the A side of the port is used. For example, a port specification of 5B will use the B side of port number five.

The S command initially loads into the utility command space at $C100. If sufficient memory has been reserved for the printer driver (see the RM command documentation) it will be relocated there, otherwise it will be relocated into the highest available locations in user memory and the end of user memory pointer will be updated. When the command to be printed has completed any user memory allocated to the printer driver will be released.

The serial data output to the printer is 8-bit ASCII with no parity. The S command does not use an ACK protocol, however it will honor the data terminal ready line if it is connected to the interface.

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

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 loa Those interested in creating their own utility commands should consult the 'Advanced Programmer's Guide' for further details.

The SBOX command is used to configure Flex™ to run with various configurations of memory and I/O. The selected configuration is then reported to the programmer.

DESCRIPTION

The general syntax of the SBOX command is:

SBOX [,<parameter>=<value>]

where <parameter> is one of the keywords described below, and <value> is a string appropriate to the keyword. Values are either a decimal number or one of the words "YES" or "NO". If a yes or no response is appropriate, you can specify the single letters "Y" or "N". If no equal sign follows the parameter, an implicit "=Y" is assumed.

The following paragraphs describe each of the keyword parameters and what values they may assume. However, these values are normally set via the Flex™ bootstrap configurator. Using the SBOX command to arbitrarily change these parameters may result in unpredictable results. For example, you can use the SBOX "CPU" parameter to set the "2MHz" bit in the CPU type flag. This does NOT mean that the computer will then be running at 2 MHz. Programs that contain internal timing loops may make use of this flag byte and will then function improperly.

CPU=1 or CPU=2

This parameter is used to set the CPU speed in MHz in the CPU type flag. It is normally used on non-S/09 systems that have been modified to run at 2 Mhz. The flag is set by the bootstrap configurator in S/09 systems.

PORT=4 or PORT=16

This parameter is used to set the number of addresses per I/O port. The default value is set by the configurator and is four for /09 systems only. For S/09, 69/A and 69/K systems, the proper value is sixteen. For compatability purposes, the keyword "IO=" is also accepted for this parameter.

PLF=50 or PLF=60

This parameter is the Power Line Frequency. It is set by the configurator on S/09 systems only and defaults to 60 Hz. for other computer systems.

EXT=YES or EXT=NO

This value determines whether the system is using extended 20-bit addresses. It is set by the bootstrap if an extended memory unit (such as the Motorola SMS3509) is installed in the computer.

TIMER=YES or TIMER=NO

This parameter is set by the configurator on S/09 systems to reflect the presence of the 68B40 programmable Timer Module on the MP-ID interface. For convenience, the keyword "INT=" (For INTerval timer) is also accepted.

UC=YES or UC=NO

This parameter is defaulted to YES by the bootstrap, and causes Flex™ to internally map all file names into upper case. If both upper and lower case file names are desired, the parameter may be set to "NO".

NOMSG

The final parameter is not a keyword, and may be specified to supress printing of the option flag settings. This parameter is useful when the SBOX command is included in STARTUP files. When this parameter is not specified, the format of the SBOX report is as follows:

```
SWTPC Configurator -- Version 2
-- Memory Size = 128K
-- I/O Port Size = 16
-- CPU Clock Rate = 2 MHz
-- Power Line Frequency = 60 Hz
-- Extended Addressing = Yes
-- Interval Timer = Yes
-- Real Time Clock = Yes
-- Upper Case Only = Yes
```

The SP command is used to direct the output of commands in the Utility Command set to an IBM Electronic Typewriter Model 50 connected to an MP-WP interface. It is normally used to produce hard copy output from text processors, formatters, and other utility programs.

DESCRIPTION

The general syntax of the SP command is:

SP[#n],<command string>

where #n is optional and is the port number of the MP-WP interface connected to the typewriter and <command string> is a valid command line to be passed to FLEX. If the port number is not specified, it will default to port seven. If the SP command is used with multiple commands per line using the end of line character, it will only affect the command immediately following it in the command line. Some examples will clarify the use of the SP command:

SP,CAT

This example will produce a printed listing of the catalog of the working drive. The port number is omitted and defaults to port seven.

SP#3,LIST,TEXT

This example will produce a printed listing of the file TEXT.TXT. The port has been specified as port number three.

The SP command initially loads into the utility command space at $C100. If sufficient memory has been reserved for the printer driver (see the RM command documentation) it will be relocated there, otherwise it will be relocated into the highest available locations in user memory and the end of user memory pointer will be updated. When the command to be printed has completed any user memory allocated to the printer driver will be released.

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:

```
+++BUILD,STARTUP
  =BASIC
  =#
+++
```

The above procedure using the BUILD 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 FLEX.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.

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 EXEC 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 crea 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 $CD03. The STARTUP file may then be deleted and if desired, modified. Directing execution to CD03, the DOS warm start address, bypasses the DOS STARTUP function.

The SUM command is used to calculate a 32-bit checksum over a file. This checksum can be used to verify the integrity of various files without the requirement of a sector by sector comparison.

DESCRIPTION

The general syntax of the SUM command is:

SUM,<file spec>

where <file spec> is the name of the file you wish to check. Note that both a file name and extension are required. The SUM program will read the file in its entirity and calculate a 32-bit checksum. This sum is then reported to the user in decimal format. Certain heuristics are invoked if the file is a text file, in order to prevent extraneous null characters from altering the resultant checksum.

The checksum produced consists of an eight bit longitudinal parity, an eight bit skewed parity, and a sixteen bit polynomial Cyclic Redundancy Code (CRC). This combination was chosen through statistical sampling of several algorithms and is highly reliable in detecting files that have been corrupted. Since the program can produce a maximum of 2 ** 32 - 1 different checksums, it is obviously possible to generate two distinct files that will produce identical checksums. The probability of this occurance is extremely low if the files being checked bear considerable similarity. This would be the case if the files were a binary file, and a patched version of the same binary file. Similarly, two text files that differ by only a few percent of their lines will produce distinct checksums in nearly all circumstances.

The use of the SUM program to compare files that are unrelated (like comparing a binary file to a text file) is somewhat more suspect. The probability of generating an identical checksum given two long arbitrary random data files is approximately equal to one in 800 million. This probability decreases as the length of the input files is reduced.

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.

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.

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.

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.

The touch command is used to change the last-altered date in a file directory entry to reflect the current date.

DESCRIPTION

The general syntax of the TOUCH command is:

TOUCH,<file spec>

where <file spec> is the name of the file to be touched. If no extension is specified, a default of .TXT is assumed. The directory entry of the file is updated so that the last altered date is set to the current date. The contents of the file itself are not altered.

The TIME command is used to determine the amount of real time required by other Flex programs. It requires that the computer system have the MP-ID interval timer installed and functional.

DESCRIPTION

The general syntax of the TIME command is:

```
TIME,<command line>
```

where <command line> is a valid Flex command string. The TIME command will set up the interval timer, invoke FLEX to process the command string, and report the amount of time required to process the command. Time is reported in hours, minutes, seconds, and tenths of seconds, with leading zero values supressed. An example will clarify the use of the TIME command:

```
+++TIME,COPY,FILE1,FILE2
-- Elapsed Time was 1:23.4
```

In this example, 1 minute and 23.4 seconds were required to copy FILE1 into FILE2. The time command will generate valid time values for programs running up to 38 hours. After this period, the reported time will wrap back around to zero. Time values are generated using the power line frequency as a reference. You should note that if the 50Hz/60Hz flag set by the configurator is incorrect, all of the values returned by the time command will be incorrect. For more information on the configurator, see the documentation of the SBOX command.

The TIME command initially loads into the utility command space at $C100. It initializes the timer and preserves certain Flex parameters. It then gets approximately 400 bytes from user memory and relocates itself into the reserved memory. If insufficient memory is available for the relocation, a message is generated and the TIME command aborts. Once relocated, the TIME command invokes Flex to process the command line. When the command processing is finished, the interval timer is used to determine the elapsed time required by the command which is then reported. The preserved Flex parameters are restored and the user memory is returned.

The T command is used to direct the output of commands in the Utility Command Set to a parallel printer attached to an 8200 series terminal. It is normally used to produce hard copy output from text processors, assemblers, and other utility programs.

DESCRIPTION

The general syntax of the T command is:

T,<command string>

where <command string> is a valid command line to be passed to FLEX. If the T command is used with multiple commands per line using the end of line character, it will only affect the command immediately following it in the command line.

The T command assumes that the printer attached to the terminal does not have form feed capability (see documentation of the NF command) and generates line feed characters for each form feed encountered. The T command cannot be used from BASIC. Some examples will clarify the use of the T command:

```
+++T,CAT
+++T,LIST,TEXT
```

The first example will produce a printed listing of the catalog of the working drive. The second example will produce a listing of the file TEXT.

The USEMF command is used to enable access to an MF-69, D-5, or DT-5 MiniFloppy diskette unit attached to a 6809 computer system with two 8-inch DMAF drives.

DESCRIPTION

The general syntax of the USEMF command is:

```
USEMF [,<controller>] [,<drive type>]
```

where <controller> is an optional minifloppy controller type, and <drive type> is an optional diskette drive type. Supported controllers are the DC-4 (Default), the DC-3 (No Double Density), and the DC-2 (No Double Density or Drive Ready). The DC-1 controller is not supported and should not be used. Supported drive types are QUME (Default), TANDON (3 Ms. step rate), and SHUGART (30 Ms. step rate). Additionally, FAST and SLOW are supported for 3 Ms. step and 30 Ms. step, respectively.

When the USEMF command is executed, a position independant MF-69 driver is loaded into the utility command space at $C100. The USEMF program then obtains approximately 330 bytes of memory from the top end of the user memory and relocates the MF-69 driver program into this area. The FLEX® memory end pointer is updated, and the MF-69 driver program is linked into the resident (DMAF) diskette drivers. A message is then printed informing the user that the MF-69 drives are now online. The MF-69 drivers will remain resident until the system is re-booted.

Once the USEMF program is executed, any file references with drive specifications of 2 or 3 are directed to MF-69 drives 0 and 1, respectively. The following example shows how USEMF can be used to copy files from a 5-inch diskette to an 8-inch diskette. The 5-inch diskette drives are Shugart SA-400 on a DC-2 controller.

```
+++USEMF SA400 DC2
-- DT-5 or MF-69 Now Online.
+++COPY,2,0,.TXT

  FILE    STUFF    .TXT     TO DRIVE #0    COPIED.
  FILE    MORESTUF.TXT      TO DRIVE #0    COPIED.
  FILE    STILLMOR.TXT      TO DRIVE #0    COPIED.

                    etc.
```

The USEMF command has the following restrictions:

1. The command must not be used when the system has been BOOTED from a 5-inch diskette. This would result in having multiple sets of driver routines accessing a single MF-69 controller. Such an occurance could result in damaged file structures.

2. The command should not be used more than once between boots. A convenient way to enforce this limitation is to place the command into the startup file. The USEMF command initialization checks for this error and issues a nasty message if used multiple times.

3. This command must not be used in conjunction with the UCAL command. Again, the initialization code checks and issues an error message if the UCAL drivers have been invoked.

The UCAL command is used to enable access to a CalComp Marksman CDS-1 fixed disk unit attached to a 6809 computer system running FLEX9 (Not FLEX9S which includes the CDS-1 drivers) having two 8-inch DMAF drives or two 5-inch MF-68 drives.

DESCRIPTION

The general syntax of the UCAL command is:

```
UCAL
```

When the UCAL command is executed, a position independant CalComp Marksman CDS-1 driver is loaded into the utility command space at $C100. The UCAL program then obtains approximately 280 bytes of memory from the top end of the user memory and relocates the CDS-1 driver program into this area. The FLEX® memory end pointer is updated, and the CDS-1 driver program is linked into the resident disk drivers. A message is then printed informing the user that the CDS-1 drive is now online. The CDS-1 drivers will remain resident until the system is re-booted.

Once the UCAL program is executed, any file references with drive specification of 2 is directed to the CDS-1 disk unit. The following example shows how UCAL can be used to copy files from an 8-inch DMAF or 5-inch MF-68 diskette to the CalComp CDS-1 disk unit:

```
+++UCAL
Marksman Now Online.
+++COPY,0,2,.TXT

  FILE    STUFF    .TXT     TO DRIVE #2     COPIED.
  FILE    MORESTUF.TXT      TO DRIVE #2     COPIED.
  FILE    STILLMOR.TXT      TO DRIVE #2     COPIED.

                        etc.
```

The UCAL command has the following restrictions:

1. The command must not be used when the system has been BOOTED with a FLEX9S system already supporting the CalComp drive.

2. The command must not be used more than once between boots. A convenient way to enforce this limitation is to place the command into the startup file.

3. This command must not be used in conjunction with the USEMF command.

The UNITERM command allows a FLEX computer system to act as a UniFLEX terminal. A special serial cable from a serial port at $E000 to a serial port on the host UniFLEX system is used to transmit terminal data from the FLEX console terminal to the UniFLEX system.

DESCRIPTION

The general syntax of the UNITERM command is:

UNITERM

To halt the UNITERM program and allow operation as a normal FLEX system, enter one of the following control sequences on the CRT terminal: for a 8212W enter "CTRL-DEL"; for 8209,8212, or CT-82 terminals enter "SHIFT-CTRL-DEL"; for other terminals, operation may be terminated by entering whatever terminal control sequence produces a hexadecimal $1F (decimal 31).

All characters transmitted from the FLEX terminal will be buffered and transmitted to the UniFLEX system as though there were only a CRT terminal attached to the UniFLEX system. Characters transmitted from the UniFLEX system are placed in a large buffer and passed to the CRT terminal as fast as the terminal will accept them. If for some reason the CRT cannot accept data as fast as the UniFLEX system is transmitting it (for instance, during graphics processing) the UNITERM program will temporarily halt transmission from the UniFLEX system to allow the terminal to catch up.

The specific serial cable connections from the FLEX serial port to the UniFLEX serial port are as follows:

Port 0, Side A, MP-S2 FLEX DB-25 connector	Any Unused MP-S2 Port UniFLEX DB-25 connector
2	3
3	2
7	7
8	20
jump 12 to 20	

Make the connections above on the serial cable attaching the serial port at $E000 on the FLEX system to the UniFLEX port. When the UNITERM program is invoked, it will configure the serial port at $E000 on the FLEX system and send a prompt to the CRT terminal. All transmission from then on is sent to the UniFLEX system and should allow completely transparent operation.

The VER command is used to display the version number of a utility or program.

DESCRIPTION

The general syntax of the VER command is:

VER,<file spec>

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

+++VER,0.CAT

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

VERIFY

The VERIFY command is used to set the File Management System's write verify mode. If VERIFY is on, every sector which is written to the disk is read back from the disk for verification (to make sure there are no errors in any sectors). With VERIFY off, no verification is performed.

DESCRIPTION

The general syntax of the VERIFY command is:

```
VERIFY[,ON]
   or
VERIFY[,OFF]
```

where ON or OFF sets the VERIFY mode accordingly. If VERIFY is typed without any parameters, the current status of VERIFY will be displayed on the terminal. Example:

```
+++VERIFY,ON
+++VERIFY
```

The first example sets the VERIFY mode to ON. The second line would display the current status (ON or OFF) of the VERIFY mode. VERIFY causes slower write times, but it is recommended that it be left on for your protection.

WRITPROM

The WRITPROM command is used to write the data contained in a disk file to a 2716 compatible EPROM using a SWTPC MP-R EPROM programmer. WRITPROM first checks the EPROM to be sure that it is erased, writes the data, then verifies the contents.

DESCRIPTION

The general syntax of the WRITPROM command is:

WRITPROM[#n],<file spec>,<load address>,[+opt]

Where #n is optional and is the port number in which the PROM programmer is installed, <file spec> is the name of the input file, <load address> is the beginning address of the input file that will be written to the EPROM, and [+opt] are the special options (described later) which can be selected. The default extension on the file is .BIN and the default drive is the working drive. The default port for WRITPROM is port #4 and the default file load address is 0000.

Some examples will clarify the syntax of WRITPROM:

```
+++WRITPROM,JUNK
+++WRITPROM #7,JUNK.CMD.1,C800,+R
```

The first example will write the 2K segment starting at address 0000 of the file JUNK.BIN on the working drive to the EPROM on the programmer in port #4. The second example will write the contents of the file JUNK.CMD on drive 1 starting at address C800 to the EPROM on the programmer in port #7 using extended retry capability.

Specifying the Load Address

The load address entered in the command line specifies the beginning address of the 2K byte segment of the file which will be written to the EPROM. As an example say that we want to store the 4K program DIAG.CMD which resides from D000-DFFF into two EPROMs which will be used in some type of ROM board or controller application. The first EPROM should be written by the command WRITPROM,DIAG.CMD,D000. The second EPROM should be written by WRITPROM,DIAG.CMD,D800. If no address is specified, an address of 0000 is assumed. If an address is specified that does not exist in the file, the message "SPECIFIED FILE CONTAINS ALL BYTES = FF" will be displayed when attempting to write the EPROM since no part of the file will be loaded into WRITPROM's write buffer.

Options

R - Retry

Specifying the +R option in the command line will instruct WRITPROM to do multiple re-trys on programming. This feature should only be used when the normal programming sequence fails to program an EPROM correctly.

C - Check +25 volts

The C option is used to turn on the +25 volts on the MP-R board. BE SURE THAT NO EPROM IS INSTALLED WHEN USING THIS OPTION. Entering any character thru the keyboard will turn off the 25 volts and return control the the operating system. To be sure that a file that always exists on the disk is used for the input file, the following sequence should always be used to invoke the C option:

```
+++WRITPROM,WRITPROM.CMD,+C
```

Remember that this option is for diagnostic purposes only and is normally not used.

Error Messages

A number of messages can be displayed if an error is encountered during the programming procedure. Most of the messages are self explanatory. If an EPROM which is not completely erased is used, during the "is the EPROM erased" test any bytes that are not erased will be displayed by the message "BYTE AT (ADDRESS) IS (DATA) - CONTINUE?". Entering a Y will cause WRITPROM to check the next byte. Entering a B will cause the rest of the "is the EPROM erased" check to be bypassed and an N will cause the programming sequence to terminate. When verifying the contents of the EPROM after writing, a similar output is generated for those bytes which did not program correctly. Again the verification can continue by typing Y, be bypassed by typing B or exited by typing N.

If the message "SPECIFIED FILE CONTAINS ALL BYTES = FF" is ever displayed then either the specified input file contains only $FF's as data or an incorrect load address was entered.

The EPROM should not be installed in the programmer until WRITPROM tells you to do so.

Default Port Addresses

If desired, the default port addresses can be changed by using the FIX utility on WRITPROM.CMD.

WRITPROM Address	Contents
1000	S/09, 69A, 69K Default Port Address
1002	/09 Default Port Address

XOUT is a special form of the delete command which deletes all files having the extension .OUT.

DESCRIPTION The general syntax of XOUT is:

XOUT[,<drive spec>]

where <drive spec> is the desired drive number. If no drive is specified all, .OUT files on the working drive will be deleted and if auto drive searching is enabled, all .OUT files on drives 1 and 2 will be deleted. XOUT will not delete any files which are delete protected or which are currently in the print queue.

Example:
+++XOUT
+++XOUT 1

The Y command is used to automatically answer "Y" to prompts produced by various FLEX utilities. This facility is especially useful when writing EXEC files.

DESCRIPTION

The general syntax of the Y command is:

```
Y,<command string>
```

where <command string> is a valid command line to be passed to FLEX. If the Y command is used with multiple commands per line using the end of line character, it will only affect the command immediately following it. Some examples will show the usefulness of the Y command:

```
+++Y,DELETE,JUNK.TXT
DELETE "0.JUNK.TXT" ? Y
ARE YOU SURE? Y
```

This example shows how to delete a file without having to reply to the "ARE YOU SURE?" prompt. This is especially useful in EXEC files. In a similar vein, you can perform COPY commands that automatically overwrite existing files, as shown in the following example:

```
+++Y,COPY,0,1,PROG

  FILE    PROG1   .TXT    TO DRIVE #1    COPIED.
  FILE    PROG2   .TXT    TO DRIVE #1  FILE EXISTS
            DELETE ORIGINAL? Y
            ARE YOU SURE? Y     COPIED.
  FILE    PROG3   .TXT    TO DRIVE #1  FILE EXISTS
            DELETE ORIGINAL? Y
            ARE YOU SURE? Y     COPIED.
```

GENERAL SYSTEM INFORMATION

I. DISK CAPACITY

Each sector of a FLEX diskette contains 252 bytes of user data since 4 bytes of each 256 byte sector is used by the system. The various capacities of disks are as follows:

5-inch Single Sided	340 sectors,	85,680 bytes
5-inch Double Sided	680 sectors,	171,360 bytes
8-inch Single Sided, Single Density	1140 sectors,	287,280 bytes
8-inch Single Sided, Double Density	1976 sectors,	497,952 bytes
8-inch Double Sided, Single Density	2280 sectors,	574,560 bytes
8-inch Double Sided, Double Density	3952 sectors,	995,904 bytes

II. WRITE PROTECT

Floppy disks can normally 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 5-inch disk can be write protected by placing a piece of opaque tape over the small rectangular cutout on the edge of the disk. 8-inch disks are the opposite, i.e., tn order to write on a full size disk, you must place tape over the notch on the rear edge of the diskette. To write protect them, remove the tape. Some 8-inch disks do not have a write protect notch and 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 various printer commands initially load into the utility command space at $C100. A check is then made for a reserved printer driver area (See the RM command) and if available, the printer driver is relocated there. If there is no reserved printer area, the printer drivers are relocated into the high end of user memory and the end of user memory pointer is updated. When the command to be printed completes, the memory occupied by the printer driver is returned to the user memory area.

V. ACCESSING DRIVES NOT CONTAINING A DISKETTE

If an attempt is made to access a drive not containing a disk, an error message is normally issued. However, if you are using 5-inch disks and a DC-1 or DC-2 controller, the system will hang up attempting to read until a disk is inserted and the door closed.

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 file 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

VII. SYSTEM MEMORY MAP

The following is a brief list of the RAM space required by the FLEX Operating System. All address are in hex.

0000 - BFFF	User RAM *Note: Some of this space is used by NEWDISK, COPY, and the printer utilities.
C000 - C07F	System Stack
C080 - C0FF	Line Input Buffer
C100 - C6FF	Utility command space
C700 - DFFF	Disk Operating System
CD00	FLEX cold start entry address
CD03	FLEX warm start entry address

For a more detailed memory map, consult the 'Advanced Programmer's Guide'.

VIII. FLEX OPERATING SYSTEM INPUT/OUTPUT SUBROUTINES

In order for the FLEX I/O functions to operate properly, all user program character input/output subroutines should be vectored thru the FLEX operating system rather than the computer's monitor. Below is a list of FLEX's I/O subroutines and a brief description of each. All given addresses are in hexadecimal.

GETCHR at $CD15

This subroutine is functionally equivalent to S-BUG's character input routine. 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 $CD15 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
B,X,Y,U	not affected

PUTCHR at $CD18

This subroutine is used to output one character from the computer to the control port (I/O port #1). It is functionally equivalant to the output character routine in S-BUG.

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    #$41
JSR    $CD18
```

The processor's registers are affected as follows:

ACC. A	changed internally
B,X,Y,U	not affected

PSTRNG at $CD1E

PSTRNG is a subroutine used to output a string of text on the control terminal. When address $CD1E 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
X	Contains the memory location of the last character read from the string (usually the 04 unless stopped by the ESC key)
Y,U	Unchanged

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.

STAT at $CD4E
This routine is used to determine the "status" of the input device. That is, to see if a character has been typed on the input terminal keyboard. Its function is to check for characters such as the ESCAPE key in FLEX which allows breaking of the output. This routine returns an EQual condition if no character was hit and a Not-Equal condition if a character was hit. No registers, except for the condition codes, may be altered.

For additional information consult the 'Advanced Programmer's Manual'.

IX. BOOTING THE FLEX DISK OPERATING SYSTEM

In order to read FLEX from the system disk upon powering up your system, you must have a short program in RAM or ROM memory. This program is called a 'bootstrap' loader.

If you are using a Southwest Technical Products disk system and the S-BUG monitor, there are bootstraps stored in this ROM which you can use. They are executed by simply typing a 'D' for the full size floppy or a 'U' for the mini floppy.

Those users of other hardware or monitor ROM should use the boot supplied with the hardware if compatible with FLEX. A sample boot (for the SWTPc mini system) is given here for reference.

If the system does not boot properly, re-position the system disk in the drive and re-execute the bootstrap loader.

```
0100 B6  E018    START   LDA     COMREG    TURN MOTOR ON
0103 86  00              LDA     #0
0105 B7  E014            STA     DRVREG
0108 8E  0000            LDX     #0000
010B 3D          OVR     MUL               DELAY FOR SPEED UP
010C 30  1F              LEAX    -1,X
010E 26  FB              BNE     OVR
0110 C6  0F              LDB     #$0F      RESTORE
0112 F7  E018            STB     COMREG
0115 8D  2B              BSR     RETURN
0117 F6  E018    LOOP1   LDB     COMREG
011A C5  01              BITB    #1
011C 26  F9              BNE     LOOP1
011E 86  01              LDA     #1
0120 B7  E01A            STA     SECREG
0123 8D  1D              BSR     RETURN
0125 C6  8C              LDB     #$8C      READ WITH LOAD
0127 F7  E018            STB     COMREG
012A 8D  16              BSR     RETURN
012C 8E  C000            LDX     #$C000
012F C5  02      LOOP2   BITB    #2        DRQ?
0131 27  05              BEQ     LOOP3
0133 B6  E01B            LDA     DATREG
0136 A7  80              STA     0,X+
0138 F6  E018    LOOP3   LDB     COMREG
013B C5  01              BITB    #1        BUSY?
013D 26  F0              BNE     LOOP2
013F 7E  C000            JMP     $C000
0142 8D  00      RETURN  BSR     RTN
0144 39          RTN     RTS
```

X. REQUIREMENTS FOR RELOCATABLE PRINTER DRIVERS

There are four routines that must be furnished in all printer drivers --

1) An OPEN routine which is called to perform all necessary printer initialization,

2) A CLOSE routine which is called to perform all necessary printer cleanup and termination operations,

3) A PUT routine which is called to output the next character to the printer, and

4) A CHECK routine which is called to determine if the printer is ready to accept another character.

The assembled printer driver which contains these routines is then combined with the "P.COR" binary file furnished with the operating system. For the procedure used to combine these two files see the documentation on P.COR, section P.2.

All four of the required routines may be located anywhere in the space provided for the routines (C312-C6FF hex). The only requirement is that one of the long branch (LBRA) instructions in the entry point vector (located at C302-C30D hex) will branch to the appropriate routine. All four routines must end with a return from subroutine (RTS) instruction. All four routines must preserve the contents of the Y and U registers.

The OPEN and CLOSE routines have no input or output parameterization, but must preserve the contents of the Y and U registers. The PUT routine expects the character to be output in the A register on entry to the routine, while the B, X, Y and U registers must be preserved. The CHECK must return a minus indication if the printer is ready to accept another character, otherwise it must return a plus indication. The contents of all of the registers must be preserved by the CHECK routine.

Since the printer driver will be relocated from the $C300 location where it is assembled, the driver must be written in position independent code. Two good examples of position independent code are the parallel printer driver and the serial printer driver listed in chapter 3, section XI of this manual. The following are good rules of thumb that may be used while writing position independent code.

1) If an instruction that will be relocated needs to reference a byte of data that will be relocated as well, "program counter relative" addressing must be used. All this means is that a ",PCR" must be added to the instruction. As an example refer to both the position independent parallel printer driver listed in section XI of chapter 3 and the non position independent parallel printer driver listed in section XII of chapter 3. Look at the references to variable "PFLAG". "COM PFLAG" is not position independent, while "COM PFLAG,PCR" is.

2) If the location referenced by a JMP or JSR instruction will be relocated, do not use JMP or JSR; use BRA, BSR, LBRA or LBSR instead.

3) It is okay to use any of the "indexed" addressing modes in position independent code. Instuctions that make use of "indexed" addressing may have operands that look like any of the following: 0,X or 5,X or 0,Y and so on.

4) It is okay to use the "immediate" addressing mode in position independent code. An immediate value is one preceded by the number sign character (#).

The following table describes the location where all items of the printer driver must be assembled.

C300 - C301 Two byte count of the number of bytes in the driver. This count includes all of the bytes between C302 and the end of the driver.

C302 - C304	LBRA	OPEN	PRINTER INITIALIZE
C305 - C307	LBRA	CLOSE	PRINTER TERMINATE
C308 - C30A	LBRA	PUT	PRINT CHARACTER
C30B - C30D	LBRA	CHECK	PRINTER READY CHECK

C30E - C30F Port address of the printer interface. The default port address should be assembled into these locations. If a port number is supplied on the command line, a new port address will be stored here by the P.COR processing.

C310 Interface Side designator. If a "side" is specified in the command port specification, the sign bit of this byte is set. A side specification of "A" will result in $80, "B" will be $81, etc.

C311 Reserved byte - assemble a zero into this location.

C312 - C6FF Space for printer driver routines. These routines must fit within the space provided. The entire space does not need to be used.

XI. PARALLEL AND SERIAL PRINT DRIVERS

The following parallel and serial print drivers are provided to assist the programmer in the creation of position independant relocatable print drivers for use with the P.COR file. The parallel driver assumes a parallel port having an address of $E01C which is the default address for port seven in a /09 computer system. It makes use of the CENTRONIX® handshake, and (although position independant) is essentially equivalent to the non-relocatable driver listed in section XII. The serial driver assumes an acia at address $E01C (also defaulting to port 7) and transmits with 8 data bits, no parity, and two stop bits. The baud rate is set at the interface.

By comparing the position independant and non-relocatable parallel print drivers, it will be simple to write your own custom printer drivers. Just remember to use program counter relative addressing for all variables defined within the printer driver itself.

PARALLEL PRINTER DRIVER 10-23-79 TSC 6809 XASMB PAGE 1

```
                        *
                        *   THE FOLLOWING PRINTER DRIVER WAS WRITTEN
                        *   IN POSITION INDEPENDANT CODE AND IS
                        *   DESIGNED TO BE APPENDED TO THE P.COR
                        *   FILE TO MAKE A PRINTER COMMAND
                        *
                        *
                        *   SET UP ADDRESSES AND DRIVER LENGTH
                        *
C300                            ORG     $C300       MUST START AT C300

C300 0050                       FDB     ENDS-POPEN LENGTH OF DRIVER
                        *
                        *   ENTRY VECTORS
                        *
C302 16  000E   POPEN   LBRA    OPEN        PRINTER INITIALIZE
C305 16  001B   PQUIT   LBRA    CLOSE       PRINTER TERMINATE
C308 16  001A   PCHAR   LBRA    PUT         PRINT CHARACTER
C30B 16  002F   PCHEK   LBRA    CHECK       PRINTER READY CHECK
                        *
                        *   PARALLEL PRINTER FILE CONTROL BLOCK
                        *
C30E E01C       PIA     FDB     $E01C       DEFAULT PORT ADDRESS
C310 00         SIDE    FCB     0           INTERFACE SIDE VALUE
C311 00                 FCB     0           -- RESERVED BYTE --
C312 FF         PFLAG   FCB     $FF         PRINTER READY FLAG

0000            DR      EQU     0           DATA REGISTER OF PIA
0000            DDR     EQU     0           DATA DIRECTION REG OF PIA
0001            CR      EQU     1           CONTROL REGISTER OF PIA
                        *
                        *   PRINTER INITIALIZATION
                        *
C313 AE  8C F8  OPEN    LDX     PIA,PCR     PICK UP PIA ADDRESS
C316 86  3A             LDA     #$3A        SELECT DATA DIRECTION REG
C318 A7  01             STA     CR,X        BY WRITING 0 IN DDR CONTROL
C31A 86  FF             LDA     #$FF        SELECT ALL OUTPUT LINES
C31C A7  84             STA     DDR,X       PUT IN DATA DIRECTION REG
C31E 86  3E             LDA     #$3E        SET UP FOR TRANSITION CHECKS
C320 A7  01             STA     CR,X        AND ENABLE OUTPUT REGISTER
C322 39                 RTS
                        *
                        *   TERMINATE PRINTER PROCESSING
                        *
C323 86  0D     CLOSE   LDA     #$0D        PRINT CARRIAGE RETURN
```

PARALLEL PRINTER DRIVER 10-23-79 TSC 6809 XASMB PAGE 2

```
                        *
                        *   PRINTER OUTPUT CHARACTER ROUTINE
                        *
C325 8D   16    PUT     BSR     CHECK       TEST FOR PRINTER READY
C327 2A   FC            BPL     PUT         LOOP UNTIL PRINTER READY
C329 34   10            PSHS    X           SAVE INDEX REGISTER
C32B 6F   8C E4         CLR     PFLAG,PCR   SET PRINTER FLAG NOT READY
C32E AE   8C DD         LDX     PIA,PCR     GET PRINTER ADDRESS
C331 A7   84            STA     DR,X        SET DATA IN OUTPUT REGISTER
C333 86   36            LDA     #$36        SET DATA READY, HIGH TO LOW
C335 A7   01            STA     CR,X        STORE IN CONTROL REGISTER
C337 86   3E            LDA     #$3E        THEN SEARCH FOR TRANSITION
C339 A7   01            STA     CR,X        OF LOW LEVEL TO HIGH LEVEL
C33B 35   90            PULS    X,PC
                        *
                        *   CHECK FOR PRINTER READY
                        *
C33D 34   10    CHECK   PSHS    X           SAVE INDEX REGISTER
C33F 6D   8C D0         TST     PFLAG,PCR   CHECK READY FLAG
C342 2B   0C            BMI     CHEXIT      IF NEGATIVE, PRINTER READY
C344 AE   8C C7         LDX     PIA,PCR     PICK UP INTERFACE ADDRESS
C347 6D   01            TST     CR,X        CHECK FOR TRANSITION
C349 2A   05            BPL     CHEXIT      IF PLUS, PRINTER NOT READY
C34B 6D   84            TST     DR,X        RESET TRANSITION STATUS
C34D 63   8C C2         COM     PFLAG,PCR   SET PRINTER READY FLAG
C350 35   90    CHEXIT  PULS    X,PC

C352             ENDS    EQU     *           END OF DRIVER

                         END
```

0 ERROR(S) DETECTED

SERIAL PRINTER DRIVER 10-23-79 TSC 6809 XASMB PAGE 1

```
                        *   SET UP ADDRESS AND DRIVER LENGTH

C300                            ORG     $C300       MUST START AT $C300

C300 0044                       FDB     ENDS-POPEN LENGTH OF DRIVER

                        *   ENTRY VECTORS

C302 16  000D   POPEN   LBRA    OPEN        PRINTER INITIALIZE
C305 16  0025   PQUIT   LBRA    CLOSE       PRINTER TERMINATE
C308 16  0024   PCHAR   LBRA    PUT         PRINT CHARACTER
C30B 16  002E   PCHEK   LBRA    CHECK       PRINTER READY CHECK

                        *   SERIAL PRINTER DRIVER FILE CONTROL BLOCK

C30E E01C       ACIA    FDB     $E01C       DEFAULT PORT ADDRESS
C310 00         SIDE    FCB     0           INTERFACE SIDE VALUE
C311 00                 FCB     0           -- RESERVED BYTE --

                        *   SEE IF A SIDE IS SPECIFIED

C312 A6  8C FB  OPEN    LDA     SIDE,PCR
C315 2A  08             BPL     ARESET      IF POSITIVE NO SIDE SPECIFIED
C317 48                 ASLA
C318 48                 ASLA                MULTIPLY BY FOUR
C319 AB  8C F3          ADDA    ACIA+1,PCR
C31C A7  8C F0          STA     ACIA+1,PCR STORE INTO DEVICE ADDRESS

                        *   RESET ACIA DEVICE

C31F AE  8C EC  ARESET  LDX     ACIA,PCR    GET ACIA ADDRESS
C322 86  03             LDA     #%00000011
C324 A7  84             STA     0,X         DO MASTER RESET ON ACIA
C326 86  11             LDA     #%00010001
C328 A7  84             STA     0,X         SET UP NO PARITY BIT,
C32A 6D  01             TST     1,X         8 DATA BITS, 2 STOP BITS
C32C 39                 RTS

                        *   CLOSE DOWN PRINTER PROCESSING

C32D 86  0D     CLOSE   LDA     #$0D        SEND OUT CARRIAGE RETURN

                        *   WRITE A CHARACTER TO PRINTER

C32F 8D  0B     PUT     BSR     CHECK       WAIT FOR PRINTER READY
C331 2A  FC             BPL     PUT
C333 34  10             PSHS    X
C335 AE  8C D6          LDX     ACIA,PCR    GET ACIA ADDRESS
C338 A7  01             STA     1,X         STORE INTO DATA REGISTER
C33A 35  90             PULS    X,PC
```

```
                        *   CHECK TO SEE IF PRINTER IS READY

C33C 34   04       CHECK   PSHS   B
C33E E6   9C CD            LDB    [ACIA,PCR] GET ACIA STATUS
C341 56                    RORB
C342 56                    RORB              SHIFT READY BIT
C343 56                    RORB              INTO SIGN FLAG
C344 35   84             PULS   B,PC

C346                 ENDS  EQU    *          END OF DRIVER

                           END

0 ERROR(S) DETECTED
```

XII. FORMER P AND PRINT.SYS

FLEX, as originally supplied, included a printer driver that will work with most parallel type printers, such as the SWTPC PR-40. Although this printer driver has been superseded, a source listing has been included for compatability purposes. If desired, these drivers may be used with the PO print command. The requirements for this type of driver are as follows:

1) The driver must be in a file called PRINT.SYS

2) Three separate routines must be supplied, a printer initialization routine (PINIT at $CCC0), a check ready routine (PCHK at $CCD8), and an output character routine (POUT at $CCE4).

3) When the POUT routine is called by FLEX, the character to be output will be in the A accumulator. The output routine must not destroy the B, X, Y, or U registers. PINIT may destroy the A, B, and X registers. PCHK may not alter any registers.

4) The routines MUST start at the addresses specified, but may be continued anywhere in memory if there is not room where specified. If placed elsewhere in memory, be certain they do not conflict with any utilities or programs which will use them.

5) All three routines must end with a return from subroutine (RTS) instruction.

```
                    *
                    * PRINT.SYS PIA DRIVERS FOR GENERAL CASE PRINTER
                    *
                E01C PIA     EQU    $E01C     PIA ADDRESS FOR PORT #7
                    *
                    * PRINTER INITIALIZATION (MUST BE AT $CCC0)
                    *
CCC0                        ORG    $CCC0     MUST RESIDE AT $CCC0
CCC0 86  3A         PINIT   LDA    #$3A      SELECT DATA DIRECTION REG.
CCC2 B7  E01D               STA    PIA+1     BY WRITING 0 IN DDR CONTROL
CCC5 86  FF                 LDA    #$FF      SELECT ALL OUTPUT LINES
CCC7 B7  E01C               STA    PIA       PUT IN DATA DIRECTION REG.
CCCA 86  3E                 LDA    #$3E      SET UP FOR TRANSITION CHECKS
CCCC B7  E01D               STA    PIA+1     AND ENABLE OUTPUT REGISTER
CCCF 39                     RTS
```

```
                        *
                        * PRINTER READY ROUTINE
                        *
CCD0 7D   E01C  PREADY  TST     PIA       RESET PIA READY INDICATION
CCD3 73   CCE3          COM     PFLAG     SET THE PRINTER READY FLAG
CCD6 39                 RTS
                        *
                        * CHECK FOR PRINTER READY (MUST BE AT $CCD8)
                        *
CCD8                    ORG     $CCD8     PRINT TEST AT $CCD8
CCD8 7D   CCE3  PCHK    TST     PFLAG     TEST FOR PRINTER READY
CCDB 2B   05            BMI     PCHKX     IF NEGATIVE, PRINTER READY
CCDD 7D   E01D          TST     PIA+1     CHECK FOR TRANSITION
CCE0 2B   EE            BMI     PREADY    IF MINUS, PRINTER NOW READY
CCE2 39                 RTS
                        *
                        * PRINTER READY FLAG
                        *
CCE3 FF                 FCB     $FF       PRINTER READY FLAG
                        *
                        * PRINTER OUTPUT CHARACTER ROUTINE (MUST BE AT $CCE4)
                        *
CCE4                    ORG     $CCE4     MUST RESIDE AT $CCE4
CCE4 8D   F2    POUT    BSR     PCHK      TEST FOR PRINTER READY
CCE6 2A   FC            BPL     POUT      LOOP UNTIL PRINTER READY
CCE8 7F   CCE3          CLR     PFLAG     SET PRINTER FLAG NOT READY
CCEB B7   E01C          STA     PIA       SET DATA IN OUTPUT REGISTER
CCEE 86   36            LDA     #$36      SET DATA READY, HIGH TO LOW
CCF0 8D   02            BSR     POUTB     STUFF BYTE INTO THE PIA
CCF2 86   3E            LDA     #$3E      THEN SEARCH FOR TRANSITION
CCF4 B7   E01D  POUTB   STA     PIA+1     OF LOW LEVEL TO HIGH LEVEL
CCF7 39                 RTS

                        END
```

This mechanism for creating printer drivers is adequate for those drivers that will fit entirely within the printer driver area located from $CCC0 to $CCF7. For those drivers that require additional memory, it is recomended that the P.COR program be used with position independant printer drivers.

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

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

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

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

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

C4MAT
    Description Page: C.5

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

COPY,<file spec>,<file spec>
COPY,<file spec>,<drive>
COPY,<drive>,<drive>[,<match list>]
    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

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

FIX,<file spec>[,<file spec>]
    Default Extension: .BIN
    Description Page: F.1

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

I,<file spec>,<command>
    Default Extension: .TXT
    Description Page: I.1

```
JUMP,<hex address>
    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

MON
    Description Page: 1.7

MV,<file spec>,<file spec>
    Default Extension: .TXT
    Description Page: M.2

N,<command>
    Description Page: N.2

NEWDISK,<drive>
    Description Page: N.1

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

P,<command>
    Description Page: P.1

P.COR
    Description Page: P.2

PO,<command>
    Description Page: P.3

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

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

PUTBOOT,<drive>
PUTBOOT,<file spec>
    Default Extension: .SYS
    Description Page: P.6
```

Q,<command>
    Description Page: Q.1

QCHECK
    Description Page: Q.2

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

RM[,<size>]
    Description Page: R.2

READPROM,<file spec>
    Default Extension: .BIN
    Description Page: R.3

S,<command>
    Description Page: S.1

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

SAVE.LOW
    Description Page: S.2.2

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

SUM,<file spec>
    Description Page: S.6

SP,<command>
    Description Page: S.4

STARTUP
    Description Page: S.5

TIME,<command>
    Description Page: T.3

TOUCH,<file spec>
    Default Extension: .TXT
    Description Page: T.2

TTYSET[,<parameter list>]
    Description Page: T.1

UCAL
    Description Page: U.2

USEMF
    Description page: U.1

VER
VER <file spec>[,<file spec>...]
    Default Extension: .CMD
    Description Page: V.1

VERIFY[,<ON or OFF>]
    Description Page: V.2

WRITPROM,<file spec>[,<address>]
    Default Extension: .BIN
    Description Page: W.1

XOUT[,<drive spec>]
    Description Page: X.1

Y,<command>
    Description Page: Y.1

NOTES: