DMAF1
DISK OPERATING SYSTEM
FLEX© VER. 1.0

USER'S GUIDE

Southwest Technical Prods. Corp.
219 W. Rhapsody
San Antonio, Texas 78216

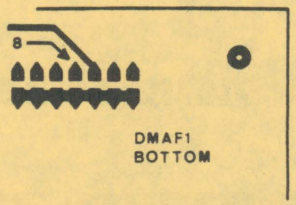
DMAF1 Important Notes

After completing assembly of the DMAF1 disk controller board, there is one modification which must be made to insure proper operation of the system. Use the light gauge wire supplied with the kit to make the following changes:

( ) Connect IC22 pin 29 to IC19 pin 12

( ) Connect IC19 pin 13 to IC19 pin 14

( ) Connect IC19 pin 11 to pin number 8 of the ribbon cable connector. The location of pin 8 is shown in the figure below.



Jot down the above changes on the bottom of page 3 of the instruction set for possible future reference.

Also, change the value of capacitor C1 on the FD-M motor control board to a 0.047 @250 V capacitor. This new value should be written in on page 11 of the instruction set.

Before installing the drives in the chassis check the programming of the JPR1-JPR4 option on both drives. On drive 1 the only thing that should be installed on any of these pins is the two-wire cable that goes back to the motor control board as described in the instructions. On drive Ø no programming jumper should be installed on any position of JPR1, JPR2, JPR3 or JPR4. If one is installed, remove it.

The hardware and software documentation for this kit are being shipped separately. Therefore, if you have received one but not the other, be patient. The rest of the kit should arrive shortly.

There is an addition to the list of modifications to the MP-A (not MP-A2) for compatibility with the DMAF1. It should be noted on page 23 of your instruction set under the **Modifying the MP-A Processor Board** heading.

Carefully cut and lift away pin 6 of integrated circuit IC12 on the MP-A board from the top side of the board. Cut the pin as near the top surface of the board as you can. Now using a short piece of light gauge wire, connect together pins 12 and 13 of IC12 on the MP-A board by soldering the wire to the bottom side of the board.

Assembly Instructions – DMAF1 Floppy Disk System

Introduction

The Southwest Technical Products Corporation DMAF1 is a dual drive, single density, double sided 8" floppy disk system. The hardware consists of a SS-50 bus (SWTPC 6800) compatible DMA (direct memory access) controller capable of handling up to four drives, two CalComp 143M double density rated disk drives, 5 3/8" H x 17 1/8" W x 20 1/2" D aluminum chassis, regulated power supply, drive motor control board, cooling fan, diskette and interfacing cables.

The DMA controller board for the disk system is a 5½" x 9" circuit board that plugs onto one of the unused 50-pin connector rows on the SWTPC 6800 computer system. The board contains all of the circuitry required to interface up to four disk drives. Connections between the controller and the drives are made thru a daisy chained 50 conductor flat ribbon cable with the terminating connector along the top edge of the controller board. The controller board contains a 1771 disk controller chip, 6844 DMA controller, programmable address decoding, full address/data line buffering and on board regulation. The board utilizes low power Schottky technology and has a current consumption of approximately 600 ma.

Although the board features selectable address decoding in 1K byte blocks from 32K thru 40K and 48K thru 56K, the supplied software assumes positioning at 36K (9ØØØ hex). The SWTPC MP-B Mother Board requires a minor patch to allow this. The newer MP-B2 Mother Board requires no modifications. The controller board requires this memory allocation since the disk controller chip, DMA controller chip and drive select latch are addressed and accessed just like computer memory and, hence, require memory addresses.

The DOS (disk operating system) and BASIC Interpreter require a minimum of 20K of RAM memory on the computer system. The first 16K bytes must be located in the lower 12K bytes of addressable memory. The remaining 8K bytes must be located at 40K thru 48K (AØØØ thru BFFF hex). Since this is the address range allocated to the scratchpad RAM on the SWTPC MP-A and MP-A2 processor boards, the scratchpad must be switch disabled on the MP-A2 board and patch disabled on the earlier MP-A processor board for proper operation. The DOS itself and most utility programs either reside or are loaded into this 8K segment. This frees the entire Ø thru 32K byte address range for user programs, interpreters, and compilers. That small area of memory assigned for scratchpad RAM (AØØØ thru AØ7F) is not over written by the DOS, therefore, the ROM monitor is not affected.

The DMAF1 disk system is being made available in kit and assembled form. Although this instruction set has been written for the kit version, it is being supplied with both the kit and assembled versions of the system. The assembled system owner should skip over those sections of the manual that involve system assembly.

When assembling the DMAF1 disk system, work on only one assembly at a time. Start with the large disk controller board, followed by the drive configuration, small motor control board, power supply, and then system checkout. This instruction set has been written in this order.

The MOS integrated circuits supplied with this kit are susceptible to static electricity damage and for this reason have been packed with their leads impressed onto a special conductive foam or possibly wrapped in a conductive foil. In either case, **do not** remove the protective material until told to do so later in the instructions.

DMAF1 Controller PC Board Assembly

NOTE: Since all of the holes on the PC board have been plated thru, it is only necessary to solder the components from the bottom side of the board. The plating provides the electrical connection from the "BOTTOM" to the "TOP" foil of each hole. Unless otherwise noted, it is important that none of the connections be soldered until **all** of the components of each group have been installed on the board. This makes it much easier to interchange components if a mistake is made during assembly. Be sure to use a low wattage iron (not a gun) with a small tip. Do not use acid core solder or any type of paste flux. We will not guarantee or repair any kit on which either product has been used. Use only the solder supplied with the kit or a 60/40 alloy resin core equivalent. Remember all of the connections are soldered on the bottom side of the board **only**. The plated-thru holes provide the electrical connection to the top foil.

( ) Before installing any parts on the circuit board, check both sides of the board over carefully for incomplete etching and foil "bridges" or "breaks". It is unlikely that you will find any, but should there be, especially on the "TOP" side of the board, it will be very hard to locate and correct after all of the components have been installed on the board.

( ) Attach all of the resistors to the board. As with all other components unless noted, use the parts list and component layout drawing to locate each part and install from the "TOP" side of the board bending the leads along the "BOTTOM" side of the board and trimming so that 1/16" to 1/8" of wire remains. Solder.

( ) Install all of the capacitors on the board. Be sure to install the electrolytic capacitor oriented exactly as shown on the component layout drawing. Solder.

( ) Install the transistors and diodes on the board. The diodes must be turned so the banded end corresponds with that shown on the component layout drawing, and the transistors must be turned to match the outline on the component layout drawing as well. Solder.

( ) Starting from one end of the circuit board install each of the five , 10-pin Molex female edge connectors along the lower edge of the board. These connectors must be inserted from the "TOP" side of the board and **must** be pressed down firmly against the board. Makes sure the body of the connector seats firmly against the board and that each pin extends completely into the holes on the circuit board. Not being careful here will cause the board to either wobble and/or be crooked when plugged onto the mother board. It is suggested that you solder only the two end pins of each of the five connectors until all have been installed; at which time, if everything looks straight and rigid you should solder the as yet unsoldered pins.

( ) Insert the small nylon indexing plug into the lower edge connector pin indicated by the small triangle on the "BOTTOM" side of the circuit board. This prevents the board from being accidently plugged on incorrectly.

( ) Install the three programming headers on the board. There is one sixteen pin and two three pin headers. These must be installed from the "TOP" side of the board with the shorter pin side going into the board. Using the component layout drawing, install one three-pin header in the NOR/MR position, the other three-pin in the 32K/48K position and the sixteen-pin header in the Ø1234567 position. The three small shorting blocks are used for programming the board and will be detailed later in this instruction set. Until then, temporarily plug one shorting block between the center pin and NOR terminal on the NOR/MR header. Plug another between the center pin and the 32K terminal on the 32K/48K header. Plug the remaining shorting block between the 4 and the pin immediately **below** it on the Ø1234567 header.

( ) The 50-pin ribbon cable connector should now be attached to the board. Install the connector from the "TOP" side of the board and orient the connector such that the pins face the top edge of the board. Solder.. The connector supplied may have locking ears either attached to or detached from the connector. The locking ears may be inserted and/or removed from the connector even after it is soldered in place. If you will be operating the 6800 Computer System with the cover off you should install the locking ears. If you will be operating the 6800 Computer System with the cover on you will have to remove the locking ears because the cover will not go on with the ears in place. The locking ears lock the flatribbon cable connector in place when installed and when folded back, eject the mating connector for easy removal. If you are not using the locking ears, you will have to use a small bladed screwdriver to unplug the mating connector whenever removal is necessary. Do not attempt to remove the mating connector by pulling on the flat ribbon cable.

( ) The crystal should now be installed on the board. Bend the crystal's leads at a 90^{o} angle approximately 1/8" from its body and mount from the top side of the board. After soldering, fasten the crystal to the board using a short piece of stripped wire by passing the wire through the two holes next to the crystal.

( ) Install all integrated circuits, except IC6, IC9, IC22, IC26 and IC29. As each one is installed, make sure it is down firmly against the board and solder only two of the leads to hold the pack in place while the other IC's are being inserted. **Do not** bend the leads on the back side of the board. Doing so makes it very difficult to remove the integrated circuits should replacement

ever be necessary. The semicircle notch, dot or bar on the end of the package is used for orientation purposes and must match with the outlines shown on the component layout drawing for each of the IC's. After inserting all of the integrated circuits, go back and solder each of the as yet unsoldered pins.

( ) Install the integrated circuit sockets for IC6 and IC22. The sockets have a pin 1 index mark on them, so orient them so they match with the component layout drawing. Solder.

( ) Install integrated circuit IC29 and its heatsink on the circuit board. This component must be oriented so its metal face is facing the circuit board and is secured to the circuit board with a #4-40 x 1/4" screw and nut. The three leads of the integrated circuit must be bent down into each of their respective holes. The hole on the heatsink should be positioned to allow maximum contact area between the regulator and the heatsink. Solder.

( ) **NOTE: READ THE FOLLOWING BEFORE REMOVING MOS INTEGRATED CIRCUITS FROM CONDUCTIVE FOAM.**

MOS devices are susceptible to damage by relatively low DC voltages applied across the leads of the device. It is easy to acquire a potential difference of many hundreds of volts between your body and ground by walking on dry carpet in tennis shoes. The same sort of potential differences may be present on your workbench.

Although it is not necessary to ground yourself, your tools, and the components to an Earth ground, it is necessary to bring yourself, your tools and the components to the **same** potential before handling and installing the MOS devices. Your skin is a conductor and can be used to equalize potentials between everything involved.

Remove and install integrated circuits IC9 and IC26 **ONE AT A TIME**. Handle the foam with both hands as you remove the IC. Hold the IC between thumb and forefinger touching all the pins. With the other hand, touch the printed cicuit foil pattern where the IC is to be installed. Press the IC into place, simultaneously making a double check of correct position and orientation. The semicircular notch or dot on the end of the IC package must match with the outlines printed on the circuit board. Never cut or bend leads of the IC packages. While still holding the component in the board, handle your solder and soldering iron with the other hand. Several seconds of handling is long enough in humid climates. Handle the parts and tools longer if you work in a dry climate. Even with the devices in the circuit, it is still possible to damage them with stray voltage. Avoid touching the conductors or devices in the circuit any more than necessary. Use this procedure for installing each MOS device. Press the IC's firmly onto the board before soldering. Most soldering irons with 3 wire line cords have grounded tips. If you do not use a soldering iron with a grounded tip, you should assure yourself that the iron you use does not carry an AC or DC voltage on the tip. When you feel confident that you have done everything possible to avoid damaging the integrated circuit with stray high voltage, solder all connections and repeat the procedure for each of the MOS devices.

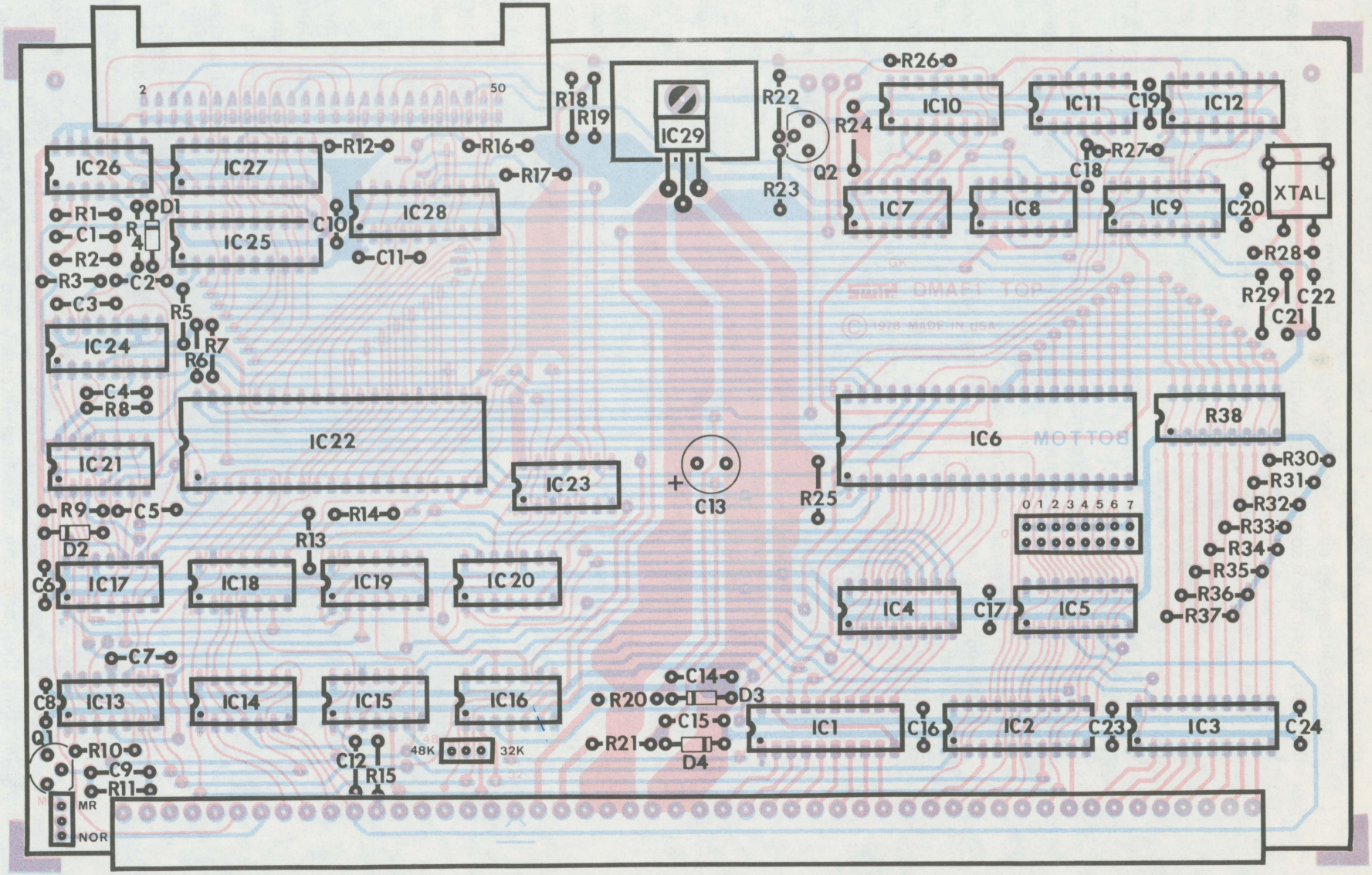
( ) Install MOS integrated circuits IC6 and IC22 into the sockets provided following the precautions given in the preceding step. Be sure to orient them as shown in the component layout drawing. Make sure that none of the IC pins fold under the IC instead of going into the socket pins.

( ) Working from the "TOP" side of the circuit board, fill in all of the feed-thru's with molten solder. The feed-thru's are those unused holes on the board whose internal plating connects the "TOP" and "BOTTOM" circuit connections. Filling these feed-thru's with molten solder guarantees the integrity of the connections and increases the current handling capability.

( ) Now that all of the components have been installed on the board, double check to make sure all have been installed correctly in their proper location.

( ) Check very carefully to make sure that all connections have been soldered. It is very easy to miss some connections when soldering which can really cause some hard to find problems later during checkout. Also look for solder "bridges" and "cold" solder joints which are another common problem.

This completes the DMAF1 Controller Board assembly. Since the circuit board now contains MOS devices, it is susceptible to damage from severe static electrical sources. One should avoid handling the board any more than necessary and when you must, avoid touching or allowing anything to come into contact with any of the conductors on the board.


2
50
IC26
IC27
IC28
IC25
IC24
IC22
IC21
IC23
IC29
IC10
IC11
IC12
IC7
IC8
IC9
IC6
IC4
IC5
IC17
IC18
IC19
IC20
IC13
IC14
IC15
IC16
IC1
IC2
IC3
R1
C1
R2
R3
C2
C3
D1
R4
C10
C11
R5
R6
R7
C4
R8
R9
C5
D2
C6
R12
R16
R17
R18
R19
R22
R23
Q2
R24
R26
C18
R27
C19
C20
XTAL
R28
R29
C22
C21
R38
R25
C13
R13
R14
C7
C8
Q1
R10
C9
R11
MR
NOR
C12
R15
48K
32K
C14
R20
D3
C15
R21
D4
C16
C23
C24
C17
R30
R31
R32
R33
R34
R35
R36
R37
0 1 2 3 4 5 6 7

Parts List – DMAF1 Disk Controller

Resistors

——	R1	47K ohm	¼ watt resistor
——	R2	100K ohm	" " "
——	R3	470K ohm	" " "
——	R4	10K ohm	" " "
——	R5	10K ohm	" " "
——	R6	10K ohm	" " "
——	R7	10K ohm	" " "
——	R8	7.5K ohm	" " "
——	R9	10K ohm	" " "
——	R10	22K ohm	" " "
——	R11	4.7K ohm	" " "
——	R12	150 ohm	" " "
——	R13	10K ohm	" " "
——	R14	10K ohm	" " "
——	R15	470 ohm	" " "
——	R16	150 ohm	" " "
——	R17	150 ohm	" " "
——	R18	330 ohm	" " "
——	R19	150 ohm	" " "
——	R20	100 ohm	¼ watt resistor
——	R21	680 ohm	" " "
——	R22	10K ohm	" " "
——	R23	10K ohm	" " "
——	R24	150 ohm	" " "
——	R25	10K ohm	" " "
——	R26	10K ohm	" " "
——	R27	10K ohm	" " "
——	R28	10M ohm	" " "
——	R29	1K ohm	" " "
——	R30	10K ohm	" " "
——	R31	10K ohm	" " "
——	R32	10K ohm	" " "
——	R33	10K ohm	" " "
——	R34	10K ohm	" " "
——	R35	10K ohm	" " "
——	R36	10K ohm	" " "
——	R37	10K ohm	" " "
——	R38	10K ohm integrated resistor optionally used in place of R30-R37	

Capacitors

——	C1	.022 mfd	capacitor
——	C2	470 pfd	"
——	C3	0.22 mfd	"
——	C4	100 pfd	"
——	C5	470 pfd	"
——	C6	0.1 mfd	"
——	C7	100 pfd	"
——	C8	0.1 mfd	"
——	C9	20 pfd	"
——	C10	0.1 mfd	"
——	C11	100 pfd	"
——	C12	470 pfd	"
——	*C13	220 mfd @ 10 VDC	electrolytic cap.
——	C14	0.1 mfd	film capacitor
——	C15	0.1 mfd	film capacitor
——	C16	0.1 mfd	capacitor
——	C17	0.1 mfd	"
——	C18	470 pfd	"
——	C19	0.1 mfd	"
——	C20	0.1 mfd	"
——	C21	60 pfd	"
——	C22	20 pfd	"
——	C23	0.1 mfd	"
——	C24	0.1 mfd	"

Transistors and Diodes

——	* D1	1N4148 silicon diode
——	* D2	1N4148 " "
——	* D3	1N4742 12.0 volt 1W zener diode
——	* D4	1N4733 5.1 volt 1W zener diode
——	* Q1	2N5210 NPN transistor
——	* Q2	2N5210 NPN transistor

Integrated Circuits

—— *IC1 74LS245 octal non-inverting bi-directional transceiver
—— *IC2 74LS244 octal non-inverting buffer
—— *IC3 74LS245 octal non-inverting bi-directional transceiver
—— *IC4 74LS139 dual 1 of 4 decoder
—— *IC5 74LS138 1 of 8 decoder
—— *IC6 6844 DMA controller (MOS)
—— *IC7 74LS00 quad NAND gate
—— *IC8 74LS74 dual D flip-flop
—— *IC9 4049B or 14049B hex inverter (MOS)
—— *IC10 74LS175 quad D flip-flop
—— *IC11 74LS08 quad AND gate
—— *IC12 74LS163 4-bit counter
—— *IC13 74121 one shot
—— *IC14 74125 quad tri-state buffer
—— *IC15 74LS00 quad NAND gate
—— *IC16 74LS32 quad OR gate
—— *IC17 74LS00 quad NAND gate
—— *IC18 74LS04 hex inverter
—— *IC19 74LS86 quad exclusive OR gate
—— *IC20 74LS00 quad NAND gate
—— *IC21 74LS32 quad OR gate
—— *IC22 1771 disk controller (MOS)
—— *IC23 74LS74 dual D flip-flop
—— *IC24 74LS123 dual one shot
—— *IC25 74LS273 octal D flip-flop
—— *IC26 14541 or 40541 timer (MOS)
—— *IC27 74LS240 octal inverting buffer
—— *IC28 74LS240 octal inverting buffer
—— *IC29 7805 5 volt regulator

Miscellaneous

—— XTAL 4.0000 MHz crystal

* Note: All components flagged with a * must be oriented as shown on the component layout drawing.

Drive Programming

In order to use the supplied disk drives they must first be programmed to enable the functions required by the system. The following description should be used in conjunction with the DRIVE PROGRAMMING PICTORIAL to program the CalComp 143M disk drives.

( ) Carefully remove one of the drives from the chassis. It is attached to the bottom of the chassis with #6 screws. Be careful not to pick up the drive by its door or front panel. Remove the two plastic bags taped to the back containing the power connectors.

( ) Turn the drive upside down so that the door is toward you and so that you are looking at the component side of the circuit board. Jumper selections are made on these drives by small shorting blocks which look like connectors without any wires attached. Notice that at each jumper location has a "center" pin which is unmarked. The shorting block always goes between this center pin and an adjacent labeled pin. In most cases, the desired jumpers will be in the correct position as supplied.

( ) Install the JPR-7 jumper. This jumper will cause the LED on the front panel to activate when the drive is selected. Use the attached pictorial for reference.

( ) Install the JPR-11 jumper. This jumper selects this drive as drive #Ø.

( ) The JPR-14, JPR-15 and JPR-16 jumper is for selecting a hard sectored drive and is not used in this system. If any jumper is installed at these locations, it should be removed.

( ) Install the JPR-13 jumper. This will cause the heads to load when the drive is selected.

( ) Switch S1 should now be programmed by flipping the switch to the desired OPEN (off) or CLOSED (on) positions. The OPEN or CLOSED position will be noted on the switch. The switch section A, B, C or D is marked along side the switch on the circuit board. The next four steps describe how to program this switch.

( ) Flip switch section A to the CLOSED (on) postion.

( ) Flip switch section B to the OPEN (off) position.

( ) Flip switch section C to the OPEN (off) position.

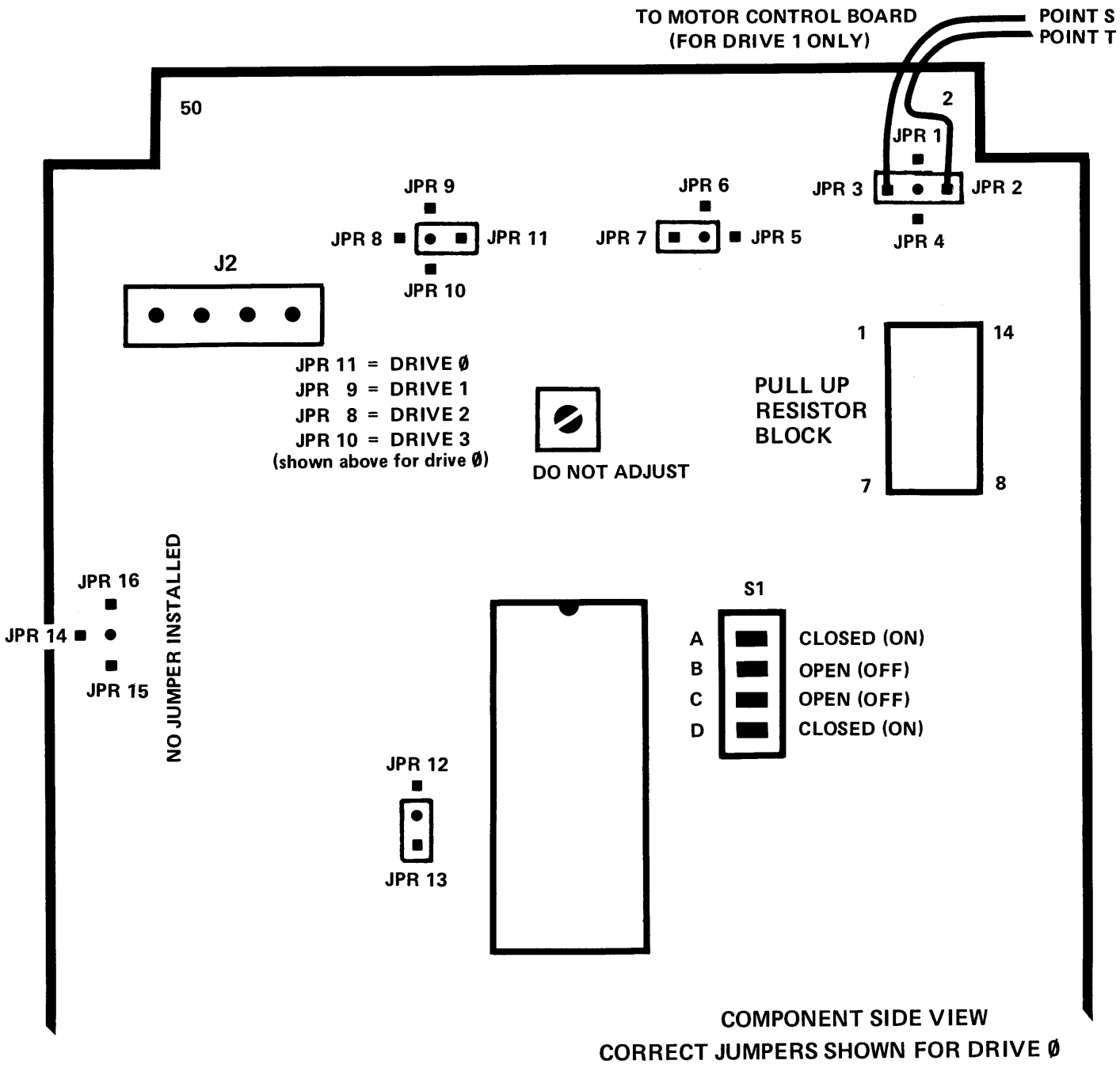
( ) Flip switch section D to the CLOSED (on) position.

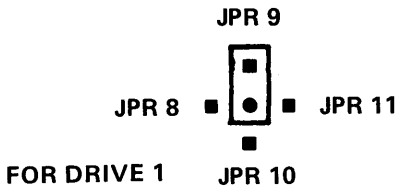
This drive is now programmed as drive #Ø. Using a pencil or felt pen, write Ø on the back of the drive for future reference.

( ) The above procedure should now be followed for the second drive except that the JPR-9 jumper should be installed instead of the JPR-11 jumper. This jumper selects this drive as drive #1. Using a pencil or felt pen, write 1 on the back of the drive for future reference.

( ) Included with the kit is what appears to be a 14-pin integrated circuit, but it is separate and probably blue. This is not an IC but is a network of resistors. This resistor network should be installed in the "resistor block socket" on **drive #1**. Be sure to install the block so that pin 1 as marked agrees with the DRIVE PROGRAMMING PICTORIAL.

This completes the disk drive programming. Set the drives aside until they are called for later during assembly.





DRIVE PROGRAMMING PICTORIAL

Assembly Instructions – FD-M Motor Control Board

The FD-M motor control board is used to turn the AC disk drive motors off when disk data is not being continually accessed. This technique reduces noise and minimizes drive and diskette wear. The actual ON/OFF and timing circuitry is on the DMAF controller board. The FD-M board contains the optical isolation and noise suppression circuitry required to isolate the system's logic circuitry from the AC power line.

The board itself is a 3 5/8" x 2" single sided circuit board and, once assembled, is installed in the right rear corner of the disk system chassis. Anytime the disk system is plugged into the AC line various points on the FD-M board carry AC line voltage regardless of whether or not the unit is turned on or running. So be careful! Never touch any components on the FD-M circuit board while the disk system's AC plug is attached to an AC power receptacle.

PC Board Assembly

( ) Clean the copper side of the circuit board with a piece of Scotchbrite® to remove all oxidation.

( ) Attach all of the resistors to the board. As with all other components unless noted, use the parts list and component layout drawing to locate each part and install from the "TOP" side of the board bending the leads along the "BOTTOM" side of the board and trimming so that 1/16" to 1/8" of wire remains. Solder.

( ) Install all of the capacitors on the board. Solder.

( ) Install integrated circuit, IC1. As it is installed, make sure it is down firmly against the board. Do not bend the leads on the back side of the board. Doing so makes it very difficult to remove the integrated circuit should replacement ever be necessary. The semicircular notch, dot or bar on the end of the package is used for orientation purpose and must match with the outline shown on the component layout drawing. Solder.

( ) Install triac Q1 and its heatsink on the circuit board. This component must be oriented so its metal face is facing the circuit board and is secured to the circuit board with a #4-40 x 1/4" screw, lockwasher and nut. The three leads of the triac must be bent down into each of their respective holes. The hole on the heatsink should be positioned to allow maximum contact area between the regulator and heatsink. Solder.

( ) Twist together two 16" lengths of the heavy gauge wire supplied with the kit to form a twisted pair. Attach and solder one end of the pair, with one wire to point Y and the other to point Z on the FD-M board.

( ) Attach and solder the female pins supplied with the three-pin connector shell packed with one of the disk drives, to the as yet unattached ends of these wires.

( ) Snap one of the pins (with wire attached) into pin 1 of the shell. Snap the other into pin 3. Polarity is not important. Nothing is connected to pin 2.

( ) Twist together two 9½" lengths of the heavy gauge wire supplied with the kit to form a second twisted pair. Attach and solder one end of the pair, with one wire to point Y and the other to point Z on the FD-M board.

( ) Attach and solder the male pins supplied with the three-pin connector shell packed with the remaining disk drive to the as yet unattached ends of these wires.

( ) Snap one of the pins (with wire attached) into pin 1 of the shell. Snap the other into pin 3. Polarity is not important. Nothing is connected to pin 2.

( ) Twist together two 9" different colored lengths of the light gauge supplied with the kit to form a third twisted pair. Attach and solder one end of the pair with one wire to point S and the other to point T.

( ) Attach and solder the small connector pins supplied with the miniature three-pin connector shell to the as yet unattached ends of these wires.

( ) Carefully snap one of the connector pins with a wire attached into one of the outer positions on the miniature connector shell. Bend the ears of the connector pin tight against the wire if necessary. Insert the pin completely into the shell until it snaps into place.

( ) Carefully snap the remaining connector pin with a wire attached into the remaining outer posi-ion on the miniature connector shell. Follow the procedure given in the previous step. Nothing is installed in the center pin position.

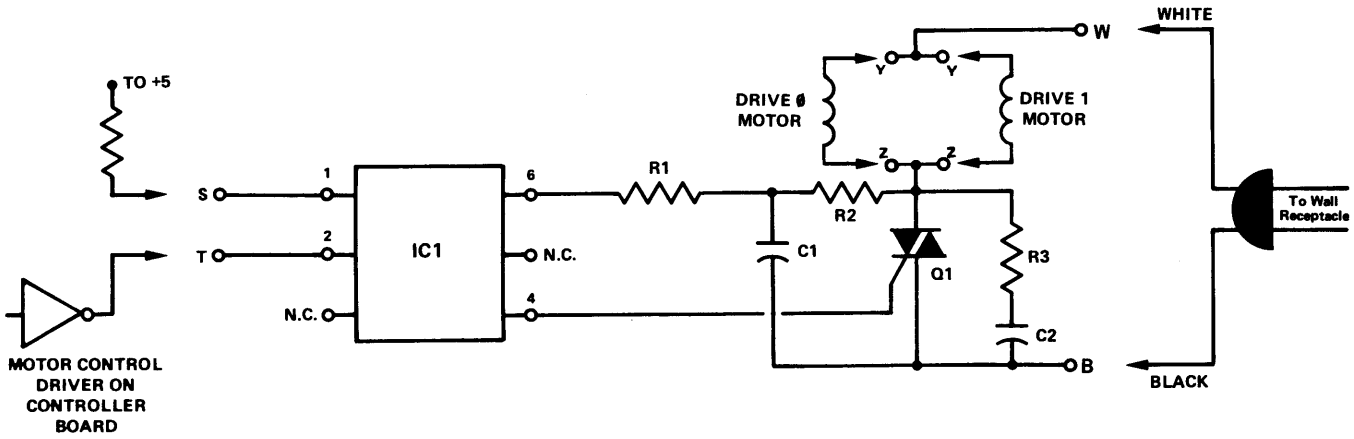
( ) Attach and solder a 8¼" piece of the heavy gauge wire supplied with the kit to point B.

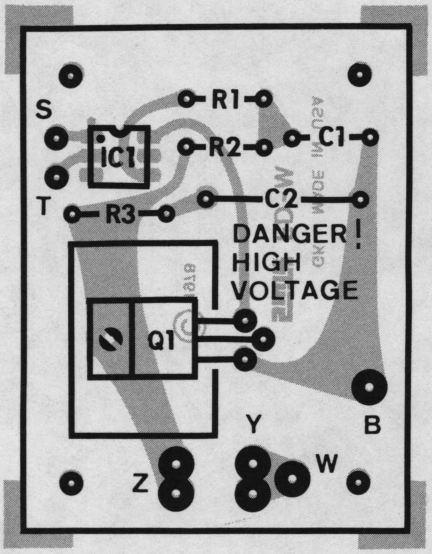
( ) Attach and solder a separate 7½" piece of the heavy gauge wire to point W.

( ) Now that all of the components have been installed on the board, double check to make sure all have been installed correctly in their proper location.

( ) Check very carefully to make sure that all connections have been soldered. It is very easy to miss some connections when soldering which can really cause some hard to find problems later during checkout. Also look for solder "bridges" and "cold" solder joints which are another common problem.

This completes the FD-M motor control board assembly.





Parts List FD-M Motor Control Board

Resistors

—— R1 180 ohm ½ watt resistor
—— R2 220 ohm " " "
—— R3 4.7 K ohm ½ watt resistor

Capacitors

—— C1 0.01 mfd @ 400 V capacitor
—— C2 0.1 mfd @ 400 V capacitor

Semiconductors

—— Q1 TIC206D or TIC216D 400 V triac
—— IC1 MOC3011 optically coupled driver

Note: All voltage specifications are minimums.

P-200 DF Power Supply

The P-200 Power Supply is a fixed voltage power supply designed to be used with several SWTPC kits. The P-200 DF version is supplied with those parts required to power the DMAF1 Dual Floppy Disk System. The P-200 DF provides the following outputs:

+24 VDC ± 5% @ 1.5 amps.
+ 5 VDC ± 5% @ 3.0 amps

The power supply consists of the 3 1/8" W x 3 3/8" L x 3 3/4" H power transformer, two large diameter filter capacitors, 3 1/4" W x 4" L circuit board and two regulator transistors. The design utilizes integrated regulators for adjustment free outputs and built-in overload protection. The power supply itself may be operated from 120 or 240 VAC, 50 to 60 Hz power systems; however, the disk drives themselves are manufactured to operate at only one voltage and line frequency.

PC Board Construction

( ) Clean the copper foil side of the circuit board with a piece of Scotchbrite® (available at most hardware stores) to remove any oxidation. Scotchbrite® is a registered trademark of 3M Corp.

( ) Attach all of the resistors to the circuit board. Use the parts list and the component layout drawing to locate the proper position for each part. As with all components unless otherwise noted, mount each flush with the top of the board, bend the leads parallel to the board on the foil side and trim so that 1/16" to 1/8" of wire remains. Solder.

( ) Attach all of the diodes to the circuit board. Be sure the banded end of each diode matches with the outline shown in the component layout drawing. Solder.

( ) Using some #18 gauge bus wire, install a jumper in the location indicated with the number "24". Solder. Make sure NO jumper is installed in the "12" positions. Use the component layout drawing. Solder.

( ) Attach the twelve pin Wafercon connector to the circuit board. Be **sure** to orient the connector as shown in the component layout drawing. Solder.

( ) Complete the first half of steps 1 thru 10 of the wiring table. Cut each wire to the specified length and attach and solder it to the specified point on the circuit board from the top side. Do not connect the other ends of the wires to their destination terminations yet.

( ) This completes the circuit board assembly. Check to make sure that all connections have been soldered and that there are no cold solder joints. Also make sure that all components have been installed correctly as called for in the instructions. Take note that there are many power supply components not used in the P-200 DF version of the kit. Install only those components listed in the parts list.

Attaching the Connector to the Power Transformer

Leave all of the power transformer **secondary** leads full length and trim the ends of the wires so that only 1/8" protrudes beyond the insulation. Attach and solder the specified connector pins to each of the leads using the table below for reference. Use the connector reference sheet contained within this instruction set if you have any problem distinguishing between the connector pins. Do **NOT** insert the connector pins into the connector shell until told to do so later in the instructions.

Transformer Secondary Wire	Connector pin Gender	Connector Pin #
yellow	female	1
green-white	female	2
green-yellow	male	3
green	female	4
blue-white	male	7
brown	female	9
blue	female	10
blue	female	11
brown	male	12

( ) Take note that the backside of the male connector shell is numbered. Using the previous table, carefully insert each of the specified connector pins into the correct numerical position of the connector shell. Insert the pins from the back or numbered side of the connector and be careful not to make a mistake. The pins cannot be removed without destroying them once they have been pressed into place. This completes the transformer connector assembly.

Power Supply onto Chassis Assembly

( ) Attach the rear panel of the chassis to the base plate using three #6-32 x 3/8" screws, lockwashers, and nuts. Slip a ground lug under the mounting screw indicated in the chasses pictorial.

( ) Attach the four stick on rubber feet to the bottom of chassis base plate. Inset each about 1½" from each corner.

( ) Attach the clamps for electrolytic capacitors C3 and C4 to the chassis using #6-32 x 3/8" screws, lockwashers and nuts. Orient the clamps as shown in the chassis pictorial. Leave the mounting screws loose until the capacitors have been installed as called for later in the instructions. Attach a ground lug under the C4 capacitor clamp screw nearest the front of the chassis.

( ) Place the fan guard on the outside of the chassis and attach the cooling fan to the chassis using four #8-32 screws, flatwashers and nuts. Orient the fan as shown in the chassis pictorial.

( ) Attach fuseholder F1 to the chassis. Sandwich the rubber washer if supplied between the fuseholder and the outside of the chassis. Orient the fuseholder as shown in the chassis pictorial.

( ) Snap power switch S1 into the chassis oriented so its contacts are oriented as shown in the chassis pictorial.

( ) Strip off about 4½" of outer insulation from the end of the line cord.

( ) Using a pair of pliers, crimp the strain relief onto the line cord at a point about 5½" from the end of the line cord and insert the compressed strain relief and line cord assembly into the 5/8" hole provided on the rear of the chassis from the outside of the chassis, then release.

( ) Loosely install the #6-32 x ½" capacitor hold down screws in the C3 and C4 capacitor clamps. Secure with #6 lockwashers and nuts.

( ) Now insert electrolytic capacitors C3 and C4 into their clamps. Use the parts list and chassis pictorial to determine position and orientation. Install them exactly as shown in the pictorial. These capacitors are polarized so the + terminal must be positioned as shown in the drawings. Secure the capacitors with the #6-32 x ½" screws, lockwashers and nuts.

( ) Tighten all of the capacitor clamp mounting screws.

( ) Using #10-32 x ¼" screws attach two terminal lugs to the (+) terminal and five terminal lugs to the (-) terminal of capacitor C4. Use the chassis pictorial to show proper orientation.

( ) Put a loop in each of capacitor C3's terminals so that connecting wires can easily be attached and soldered.

( ) Orient the power transformer so the nine wire secondary side is nearer the left side of the chassis and secure with four #8-32 x 3/8" screws, flatwashers and nuts.

( ) Remove the precoated insulators from their packages and place over the pins on the bottom of regulator transistors Q3 and Q4.

( ) Install transistors Q3 and Q4 onto the chassis in the appropriate set of holes from the outside of the chassis. Be sure you have put the right transistor in the right set of holes. Secure each transistor with #6-32 x 3/8" screws, insulated shoulder washers, ground lugs and nuts. **NOTE:** The case of each power transistor is electrically a transistor junction and hence must be electrically isolated from all other electrical junctions including the chassis. The mounting screws are electrically connected to each transistor case and you must be sure the screws do not contact the chassis as they pass through. Keep in mind also that the wire leads of each power transistor

must be centered in the large holes through which they pass. The mounting screws must be tightened evenly and with enough pressure to slightly compress the transistor insulators. The entire bottom of the transistor case must be in solid contact with the insulator for good heat transfer.

( ) Orient the P-200 power supply printed circuit board as shown in the chassis pictorial and secure it to the chassis using four #6-32 x 5/8" screws, ¼" spacers, lockwashers and nuts.

( ) Orient the FD-M motor control board as shown in the chassis pictorial and secure it to the chassis using four #4-40 x 5/8" screws, 1/4" spacers, lockwashers and nuts.

( ) For American standard 120 VAC line operation complete steps 11 thru 14 of the wiring table. For European standard 240 VAC operation complete steps 15 thru 17 of the wiring table.
NOTE: Although it is a simple matter to change the power supply for 120/240 VAC, 50/60 Hz operation, it is not so easy with the disk drives themselves. Since their AC motor uses the AC line voltage, the drive motor and/or pulley must be changed in some instances. Those systems supplied directly by SWTPC and its U.S. dealers will probably be configured for 120 VAC 60Hz operation. Those systems supplied by SWTPC overseas dealers will vary depending upon the locality. If you are not sure of the disk drive's AC electrical requirements, remove the cover plate on the top of the drive near the back. The motor's specifications will be stamped on the motor housing. Changing from 120 to 240 VAC or vise versa requires a new motor. Changing from 60 Hz to 50 Hz or vise versa requires a new pulley. Be sure to reinstall the cover plate.

( ) Now go back and complete the second half of wiring steps 1 thru 10. When attaching the wires to the regulator transistors Q3 and Q4, slip a 1" piece of heat shrinkable tubing over each of the wires to be attached first. Solder the wire directly to the transistor pin, slip the heat shrinkable tubing over the exposed connection and shrink the tubing with the heat from your soldering iron. The female pins specified in steps 9 and 10 are those for the four pin connectors J2-Ø and J2-1 which attach to the disk drives. These connectors and their pins are individually packed in small plastic bags. Use a 1" piece of heat shrinkable tubing over all of the terminals on power switch S1. This means that you must run each wire thru the tubing and after all wires have been attached and soldered, the tubing is slipped over the terminal and shrunk with the heat from the tip of your soldering iron. This is done later during assembly.

( ) Complete wiring steps 18 thru 35 of the wiring table. Use a 1" piece of heat shrinkable tubing over all of the terminals on power switch S1. This means that you must run each wire thru the tubing and after all wires have been attached and soldered, slip the tubing over the terminal and shrink with the heat from the tip of your soldering iron. Connector J2 is the four-pin connector supplied with the disk drives. The Ø or 1 suffix specifies the drive number.

( ) Go back and double check all wiring steps and solder connections for correctness and completion. Even a simple mistake can cause costly damage to your power supply and or disk drives.

( ) Plug the twelve-pin male connector attached to the power transformer's secondary leads onto the twelve-pin receptacle on the power supply printed circuit board. Be sure to orient the connector correctly. It will fit only one way.

( ) Install fuse F1 into the fuseholder.

( ) Without having anything plugged onto power connectors JØ-Ø, JØ-1, J2-Ø or J2-1 and after making sure these connectors are not inadvertantly touching anything they shouldn't,
plug the line cord into a wall outlet and turn the power switch ON. When ON the fan should run.

( ) Using one of the (–) terminals on capacitor C3 or C4 as a ground reference, measure the following voltages on the two DC power connectors J2-Ø and J2-1 listed below. If you find that any of the voltages do not measure as specified, immediately remove power and recheck all wiring and solder connections.

Connnectors J2-Ø and J2-1

Pin #	Voltage	Tolerance
1	+24 VDC	±5%
2	0 VDC	±5%
3	+5 VDC	±5%
4	0 VDC	±5%

( ) DO NOT make any voltage checks on connectors JØ-Ø or JØ-1 on the FD-M motor control board.

( ) If everything checks out as called for then remove power and unplug the unit. Once you are convinced that the power supply is working as it should be, use the wire ties supplied with the kit to bundle the wires where necessary. If the power supply voltages do not measure as specified, unplug the unit and recheck all previous assembly steps.

( ) Double check to make sure the unit is unplugged.

This completes the power supply assembly.

Parts List P-200 MF Power Supply

Resistors

——	R1	243 ohm 1% resistor
——	R2	4320 ohm 1% resistor

Diodes

——	D1*	1N5402 high current diode
——	D2*	" " " "
——	D3*	" " " "
——	D4*	" " " "
——	D5*	1N4003 diode
——	D7*	1N5402 high current diode
——	D8*	" " " "
——	D9*	" " " "
——	D10*	" " " "
——	D11*	1N4003 diode

Capacitors

——	C3*	4,000 mfd @ 50 VDC electrolytic capacitor
——	C4*	29,000 mfd @ 15 VDC " "

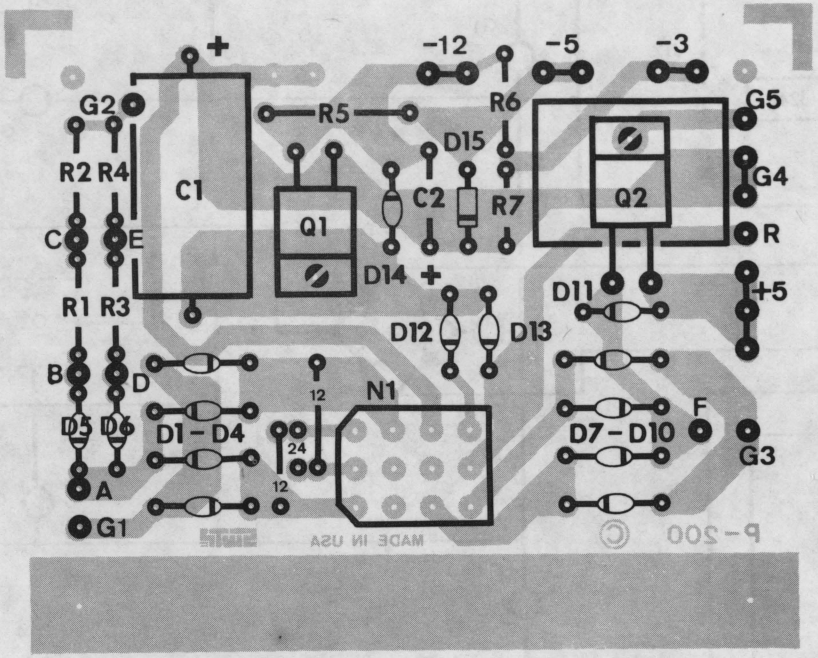
Regulators

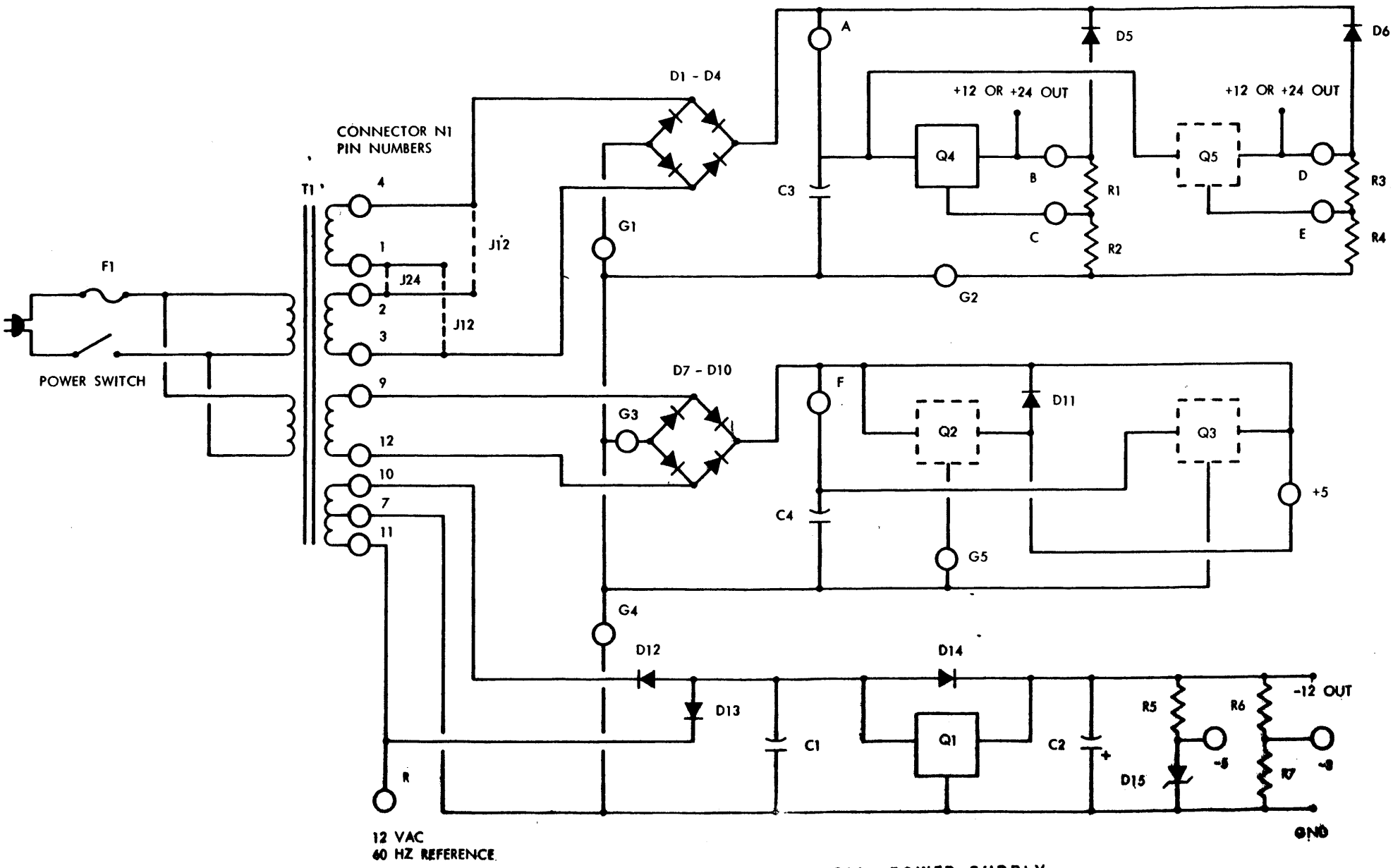
——	Q3*	LM323 +5 VDC regulator
——	Q4*	LM317 adjustable regulator

Miscellaneous

——	T1*	Power transformer 50-60 Hz Primary: 120/240 VAC Secondaries: 12 VAC @ 1.5 amp 12 VAC @ 1.5 amp 7 VAC @ 3 amp 24 VAC @ 200 ma
——	F1	2 ½ amp slo-blo fuse

All components flagged with a (*) must be oriented as shown in the component layout drawing and pictorials.


+
G2
R2 R4
C
E
R1 R3
B
D
D5 D6
A
G1
C1
R5
Q1
D14
C2
D15
R6
R7
-12
-5
-3
G5
G4
R
+5
Q2
D11
D12
D13
N1
12
24
12
D1 - D4
D7 - D10
F
G3
MADE IN USA
P-200



SCHEMATIC P - 200 POWER SUPPLY

WIRE			FROM			TO		
STEP	LENGTH	GAUGE	PART	TERMINAL	SOLDER	PART	TERMINAL	SOLDER
1	6¾"	#18	PC board	A	yes	C3 lug	(+)	no
2	8 "	#18	PC board	B	yes	Q4	B3	yes
3	7½"	#18	PC board	C	yes	Q4	B1	yes
4	15½"	#18	PC board	F	yes	C4 lug	(+)	yes
5	6¼"	#18	PC board	G1	yes	C3 lug	(–)	no
6	6½"	#18	PC board	G2	yes	C3 lug	(–)	no
7	14"	#18	PC board	G3	yes	C4 lug	(–)	yes
8	9 "	#18	PC board	+5	yes	Q3	A2	yes
9	9½"	#18	PC board	+5	yes	female pin	J2-Ø pin 3	yes
10	17"	#18	PC board	+5	yes	female pin	J2-1 pin 3	yes
120 VAC OPERATION								
11	full	———	T1	black	———	S1	A	no
12	full	———	T1	white	———	S1	A	no
13	5"	———	T1	blk-wht	———	F1	A	no
14	5"	———	T1	blk-red	———	F1	A	no
240 VAC OPERATION								
15	———	———	T1	black	———	S1	A	no
16	———	———	T1	white	———	T1	blk-red	yes
17	———	———	T1	blk-wht	———	F1	A	no
18	———	———	line cord	black	———	S1	B	yes
19	———	———	line cord	white	———	F1	B	yes
20	———	———	line cord	green	———	gnd lug	———	yes
21	full	———	fan	A	———	S1	A	no
22	full	———	fan	B	———	F1	A	no
23	8¼"	———	FD-M	B	———	S1	A	yes
24	7½"	———	FD-M	W	———	F1	A	yes
25	8½"	#18	C3 lug	(–)	yes	gnd lug	———	no
26	9"	#18	C4 lug	(–)	yes	gnd lug	———	yes
27	9½"	#18	C4 lug	(–)	yes	Q3	A3	yes
28	10½"	#18	C4 lug	(+)	yes	Q3	A1	yes
29	12"	#18	C3 lug	(+)	yes	Q4	B2	yes
30	22½"	#18	C4 lug	(–)	yes	female pin	J2-Ø pin 4	yes
31	28½"	#18	C4 lug	(–)	yes	female pin	J2-1 pin 4	yes

P-200 DF Wiring Table — Continued on next page

WIRE			FROM			TO		
STEP	LENGTH	GAUGE	PART	TERMINAL	SOLDER	PART	TERMINAL	SOLDER
32	20½"	#18	C3 lug	(–)	yes	female pin	J2-Ø pin 2	yes
33	29¼"	#18	C3 lug	(–)	yes	female pin	J2-1 pin 2	yes
34	16"	#18	Q4	B4	no	female pin	J2-Ø pin 1	yes
35	24"	#18	Q4	B4	yes	female pin	J2-1 pin 1	yes

Configuring the DMAF1 Disk for the SWTPC 6800 Computer System and Vice-Versa.

When using the DMAF1 Disk Controller Board with the SWTPC 6800 Computer System, program the DMAF1 Controller Board as follows:

1.) Plug one shorting block between the center pin and NOR terminal on the NOR/MR header.

2.) Plug another shorting block between the center pin and 32K terminal on the 32K/48K header.

3.) Plug the remaining shorting block between the 4 and the pin immediately below it on the Ø1234567 header.

If you are using the DMAF1 Disk Controller Board with a SWTPC 6800 Computer System with a MP-B2 mother board and MP-A2 processor board, no modifications to the computer system other than with one of the memory board(s) are necessary. Check to see which boards are being used in your computer. The nomenclature for each board is printed on the "TOP" side of the board.

Modifying the MP-B Mother Board

If your computer system has a MP-B (not MP-B2) mother board, power down the system, unplug the connector going to the power supply board and remove the board. Make the following modifications.

1.) Cut the foil conductor connecting pin 10 to pin 12 of IC4, the 7400 NAND gate, on the "BOTTOM" side of the mother board.

2.) Attach and solder an insulated jumper between pin 11 of IC4 (7400 NAND gate) to pin 6 of IC6 (74S138 decoder) on the "BOTTOM" side of the board.

3.) Attach and solder a separate insulated jumper between pin 12 of IC4 (7400 NAND gate) and address line A12 on the "BOTTOM" side of the board.

4.) Tape the two jumper wires to the "BOTTOM" side of the board so they do not break off or get pinched.

5.) Reinstall the mother board and reconnect the connector going to the MP-P power supply board.

Modifying the MP-A Processor Board

If your computer system has a MP-A (not MP-A2) processor board, power down the system, carefully remove the MP-A processor board and make the following modifications:

1.) First, check to see that integrated circuit IC15 (second IC from the left on the bottom row as viewed from the "TOP" side) is a DM8098 or 74368. If it has a DM8096 or 74366 substituted the IC will have to be replaced replaced with a DM8098 or 74368.

2.) Carefully cut pin 15 of IC15 from the "TOP" side of the board very close to the surface of the board. Bend the IC pin 15 up and away from the foil trace.

3.) Attach and solder an insulated jumper from now unconnected pin 15 to pin 8 of IC15 on the "TOP" side of the board. This effectively grounds pin 15.

4.) Now cut the foil trace connecting pin 10 to pin 13 on IC16 from the "TOP" side of the board. IC16 is the third IC from the left on the bottom row as viewed from the "TOP" side. It is a 7420.

5.) Attach and solder an insulated jumper from the now unconnected pin 13 on IC16 to pin 3 of IC7 on the "BOTTOM" side of the board. IC7 is the third IC from the right on the bottom row as viewed from the "TOP" side. IC7 is a DM8097 or 74367.

6.) Cut the foil trace connecting to pin 13 of IC3 on the "BOTTOM" side of the board. IC3 is the 6810 RAM memory on the board. Now run a jumper from pin 13 of IC3 to pin 12 of IC3. This effectively grounds pin 13 of IC3.

7.) Reinstall the modified MP-A processor board on the computer system. NOTE: The previous modifications have disabled the 6810 scratchpad RAM memory on the processor board. This was necessary so that an external 8K byte contiguous block of memory from AØØØ thru BFFF (40K thru 48K) may be installed in the computer system for the DOS. The processor board **will not** now function at all without this extra memory installed. The ROM monitor will not work without RAM memory at memory locations AØØØ thru AØ7F. The DOS is loaded into locations AØ8Ø thru BFFF.

Memory Board Modification

Regardless of which processor or mother board your system uses, one of the MP-8M 8K or two of the MP-M 4K memory boards will have to be modified for operation from AØØØ thru BFFF (40K thru 48K) where the scratchpad and disk operating system (DOS) reside. The MP-A processor board should have already been modified for external memory from AØØØ thru BFFF. If you are using the MP-A2 processor board, it will be necessary to switch OFF the RAM dip switch on the processor board once the modified memory boards are plugged onto the computer system.

Modifying the 4K MP-M Memory Board

To modify the MP-M memory board for operation above 32K break the conductor foil between pin 6 of integrated circuit IC22 and pin 1 of IC24 as well as the conductor foil between pin 4 of IC22 and connector pin A15. Break the conductors near IC22 using a small screwdriver or knife to scribe a small line across the trace deep enough to break the conductive path. Using a piece of light gauge hookup wire connect pin 6 of IC22 to connector pin A15. Using a separate piece of hookup wire connect pin 4 of IC22 to pin 2 of IC24. Check your modifications and wiring for accuracy. This completes the modification. Use the table below to determine the proper position for the address select programming jumper which must be installed on the memory board. One 4K board must be modified and jumper programmed for #2. Another 4K board must be modified and programmed for #3. This will provide RAM memory from AØØØ thru BFFF.

TABLE 1

MP-M Memory Address Assignment Table (Hex above 32K)

Programming Jumper #	Memory Quandrant (K of memory)	Starting Address	Ending Address
	1	AØØØ	A3FF
	2	A4ØØ	A7FF
2	3	A8ØØ	ABFF
	4	ACØØ	AFFF
	1	BØØØ	B3FF
	2	B4ØØ	B7FF
3	3	B8ØØ	BBFF
	4	BCØØ	BFFF

MP-M/MP-MX Memory IC Assignment Map

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit Ø
Quadrant 1 (1K)	IC15	IC13	IC11	IC9	IC7	IC5	IC3	IC1
Quadrant 2 (2K)	IC16	IC14	IC12	IC10	IC8	IC6	IC4	IC2
Quadrant 3 (3K)	IC40	IC38	IC36	IC34	IC32	IC30	IC28	IC26
Quadrant 4 (4K)	IC25	IC39	IC37	IC35	IC33	IC31	IC29	IC27

ØØ hex = ØØØØ ØØØØ binary
Ø1 hex = ØØØØ ØØØ1 binary
Ø2 hex = ØØØØ ØØ1Ø binary
Ø4 hex = ØØØØ Ø1ØØ binary
Ø8 hex = ØØØØ 1ØØØ binary
1Ø hex = ØØØ1 ØØØØ binary
2Ø hex = ØØ1Ø ØØØØ binary
4Ø hex = Ø1ØØ ØØØØ binary
8Ø hex = 1ØØØ ØØØØ binary

Modifying the 8K MP-8M Memory Board

To modify the MP-8M memory board for operation above 32K first flip all of the address select slide switches on the memory board to their OFF position. For operation from 40K to 48K (8ØØØ to 9FFF) solder a piece of light gauge hookup wire from pin 1 of IC22 to pin 10 of IC18. Check your wiring for accuracy. This completes the modification. Table II gives the new memory assignments for each of the memory integrated circuits. All of the switches must be left "OFF". The board is now configured for operation for AØØØ thru BFFF.

TABLE II
MP-8M Memory Address Assignment Table (Hex) above 32K

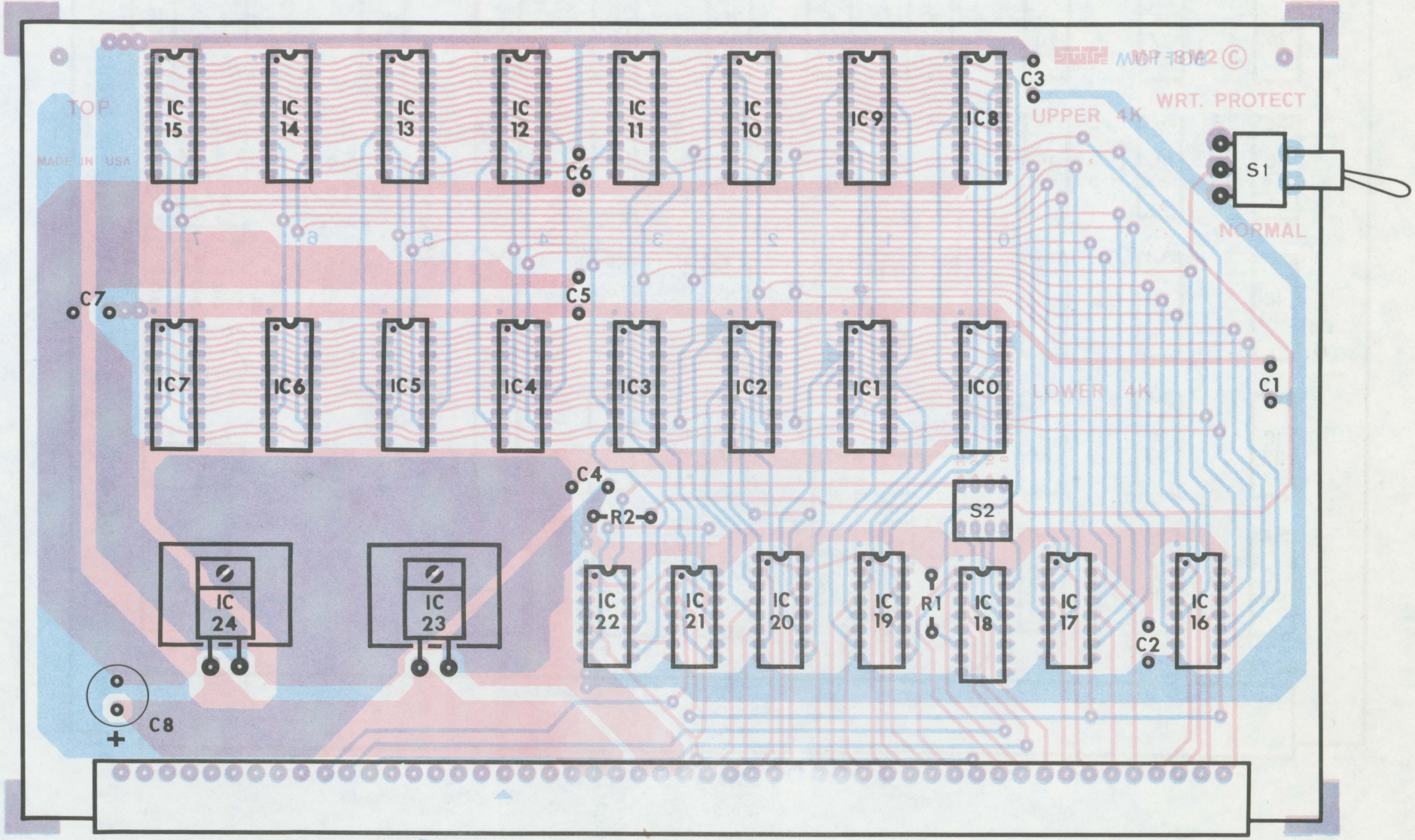
Half of Memory	Starting Address	Ending Address
lower	AØØØ	AFFF
upper	BØØØ	BFFF

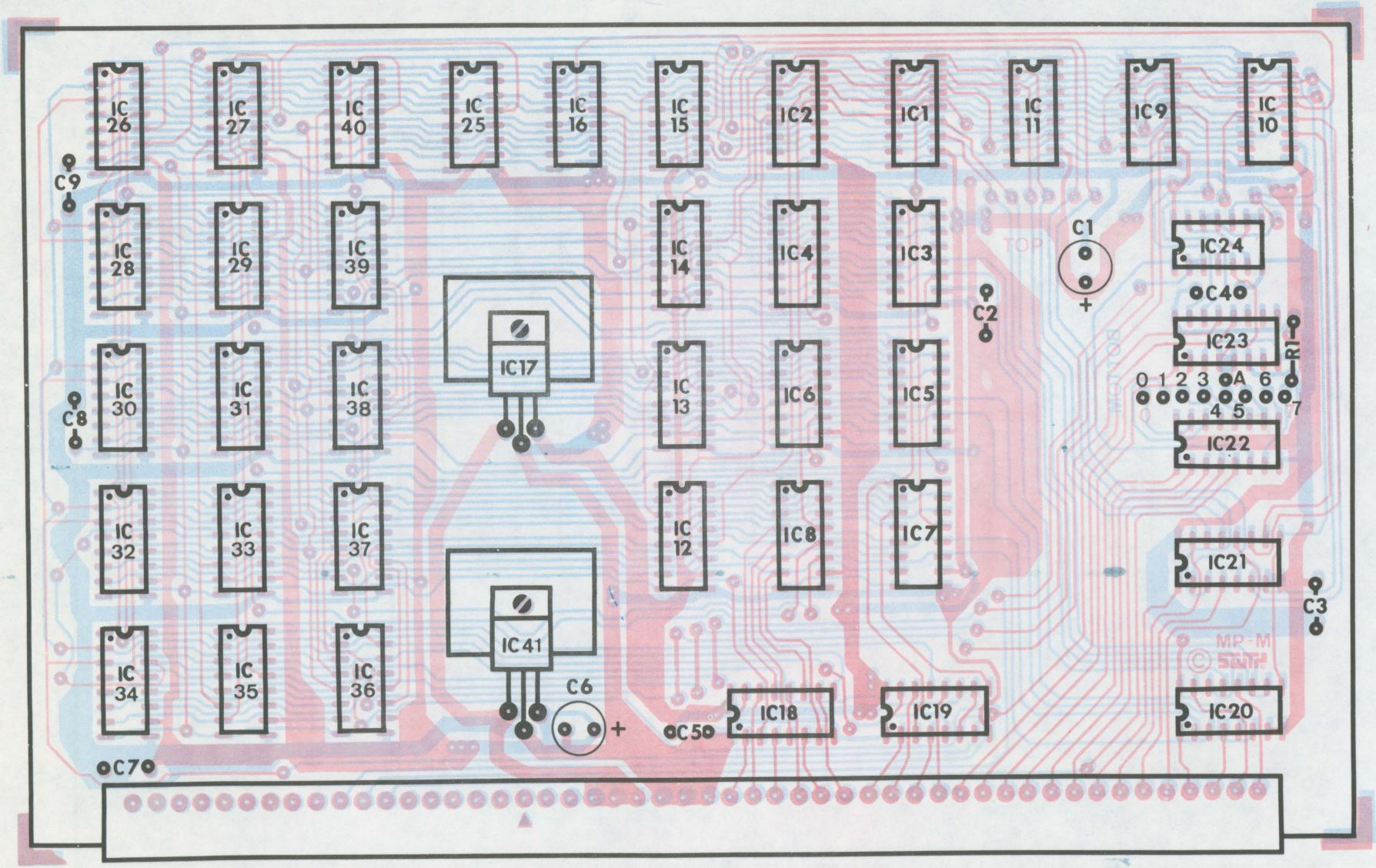
MP-8M Memory IC Assignment Table

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit Ø
Lower 4K	IC7	IC6	IC5	IC4	IC3	IC2	IC1	ICØ
Upper 4K	IC15	IC14	IC13	IC12	IC11	IC10	IC9	IC8

Hex to Binary Conversion

ØØ hex = ØØØØ ØØØØ binary
Ø1 hex = ØØØØ ØØØ1 binary
Ø2 hex = ØØØØ ØØ1Ø binary
Ø4 hex = ØØØØ Ø1ØØ binary
Ø8 hex = ØØØØ 1ØØØ binary
1Ø hex = ØØØ1 ØØØØ binary
2Ø hex = ØØ1Ø ØØØØ binary
4Ø hex = Ø1ØØ ØØØØ binary
8Ø hex = 1ØØØ ØØØØ binary


TOP
MADE IN USA
IC 15
IC 14
IC 13
IC 12
IC 11
IC 10
IC9
IC8
C3
C6
C5
C7
IC7
IC6
IC5
IC4
IC3
IC2
IC1
IC0
C4
R2
S2
IC 24
IC 23
IC 22
IC 21
IC 20
IC 19
R1
IC 18
IC 17
IC 16
C2
C1
C8
+
WRT. PROTECT
UPPER 4K
LOWER 4K
S1
NORMAL


IC 26
IC 27
IC 40
IC 25
IC 16
IC 15
IC2
IC1
IC 11
IC9
IC 10
C9
IC 28
IC 29
IC 39
IC 14
IC4
IC3
C1
TOP
+
IC24
C2
C40
IC23
R1
0 1 2 3 A 6 7
4 5
IC 30
IC 31
IC 38
IC17
IC 13
IC6
IC5
C8
IC22
IC 32
IC 33
IC 37
IC 12
IC8
IC7
IC21
C3
IC 34
IC 35
IC 36
IC41
C6
+
MP-M
IC18
IC19
IC20
C5
C7

Before attempting to check out the DMAF1 Disk System, let's go thru the hardware just to make sure everything required is plugged in and configured properly. First of all the computer system must have a minimum of 20K of RAM memory installed (24K recommended) with 8K positioned from AØØØ thru BFFF (40K thru 48K). The remaining 12K thru 32K of RAM must be the contiguous memory starting at memory location ØØØØ.

If your system has a MP-B (not MP-B2) mother board, it must be modified as per the instructions in the preceding section of this manual. If your system has a MP-B2 mother board, no mother board modifications are necessary.

If your system has a MP-A (not MP-A2) processor board, it must also be modified as per the instructions in the preceding section of this manual. If your system has a MP-A2 processor board, it will be necessary to switch the DIP switch marked RAM on the MP-A2 board to the OFF position.

Before attaching the disk and its controller board to the system, it is suggested that you run memory diagnostics on all areas of RAM memory in the system. If there are any signs of problems especially on more than one of the memory boards, make sure you are not drawing too much power from the computer system's power supply. It is suggested that your memory boards not draw a total of more than 7.5 amps from the power supply. Each MP-M 4K and MP-8M 8K memory board consumes approximately 1.5 amps. The advantage of the 8K board is, of course, that you may have twice as much memory for the same amount of power. The 16K/32K dynamic memory boards now available consume approximately 1.5 amps.

Most of the SWTPC memory diagnostics are loaded into the scratchpad RAM memory from AØØØ thru AØ7F so when you run the memory diagnostics on the 40K thru 48K memory board(s) start the diagnostic at AØ8Ø rather than AØØØ otherwise the diagnostic will destroy itself.

At the time of this writing three monitor ROMs are available for the SWTPC 6800 Computer System.

The newest monitor and the one suggested for those DMAF1 owners with an MP-A2 processor board is DISKBUG® . DISKBUG®is presently the only monitor containing a boot for the DMAF1 disk system. The SWTBUG® and MIKBUG® monitors require that this bootstrap program be entered by hand. The bootstrap, for those confused, is a short program that configures the disk controller and initiates the loading of the disk operating system (DOS) into computer memory. DISKBUG®may be installed only on a MP-A2 processor board. It is a preprogrammed 2716 PROM and is installed in one of the PROM sockets. It sells for $60.00 ppd. in the continental U.S. #DISKBUG–2716.

Although the SWTBUG®ROM contains a disk boot, it is for the SWTPC MF-68 Mini-Floppy Disk System and is not compatible with the DMAF1 disk system.

Unlike SWTBUG® and MIKBUG® , DISKBUG®has been written to operate with a MP-S serial interface only and will not work with a MP-C control interface.

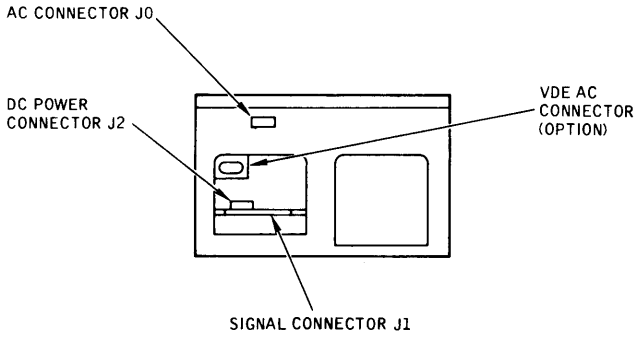
For those who do not have the DISKBUG monitor, the eighty-byte program may be entered by hand using the monitor's memory examine and change function. Once entered it may be saved to cassette tape (if available) or blown in a 2716 EPROM by the user with a MP-A2 and MP-R EPROM programmer, for future convenience. The code for the boot is listed in the appendix of the DOS User's Guide.

Once you are sure that the computer system's memory is good, remove power. Check over the DMAF1 controller board once more for proper soldering and correct installation of the shorting blocks as detailed earlier in this manual. Carefully install the controller board onto one of the unused 50-pin board positions on the computer mainframe. Make sure the board is seated properly. With nothing plugged onto the connector along the top edge of the controller board, apply power to the computer system. The terminal should respond with the monitor's prompt. Use the monitor's memory examine/change function to write a 55 into hex location 9ØØØ. Using the same function, go back and read hex 9ØØØ. It should still contain a 55. If it does not, then data is not being written to and/or read from the DMA chip on the disk controller board. Now try writing a 55 into hex location 9Ø22. Using the memory examine and change function, go back and read hex 9Ø22. It

should still contain a 55. If it does not, then data is either not being written to and/or read from the disk controller chip on the disk controller board.

If you have a voltmeter handy, measure the DC voltage of the rightmost lead of the voltage regulator, IC using the center lead as a ground. It should measure +5 VDC ±5%. Do not continue assembly if you have difficulty with any of these three tests. Instead, remove power, remove the controller board and recheck component installation and orientation. Check carefully for missed solder connections, solder and foil bridges or breaks. If you cannot find the problem, carefully pack up just the controller board and return it to SWTPC attn: "Repair Dept." with a description of the problem.

If the board successfully performs the previous tests, continue with assembly. Power down the computer system. Set the disk chassis base plate within four feet of the computer system mainframe Set each of the two disk drives vertically onto the base plate so the circuit boards attached to the drives face to the right. Place the drive programmed for Ø on the left and the one programmed for 1 on the right. This is important. Do not get them reversed. Make absolutely sure the line cord for the disk system is not attached to an AC receptacle. Attach the three-pin AC power connectors JØ-Ø and JØ-1 to the rear of each of the disk drives. The three-pinned connectors are interchangable and their length should dictate which cable connects to which drive. Attach the four-pinned DC power connectors J2-Ø and J2-1 to the rear of each of the disk drives. The four-pinned connectors are interchangable and their length should dictate which cable connects to which drive.



Connector Locations

Run the three-pinned miniature connector from the FD-M motor control board thru the ½" x 4" opening on the rear to the right hand drive (1) from the outside. This connector must be oriented as shown in the drive programming pictorial earlier in this manual and plugged onto the JPR 3 and JPR 2 posts. It is important that the JPR 3 terminal connect to point S and the JPR 2 terminal connect to point T on the FD-M motor control board. Getting these reversed will not damage anything but will prevent the drive motors from functioning, so be careful.

Now attach the terminating PC connector of the flat ribbon cable to the drive 1 (the rightmost drive). The connector may be keyed and if so will go on only one way. If not, orient the connector so the tracer on the flat ribbon cable is nearer the side of the PC connector with the cutout. Attach the adjacent PC connector of the flat ribbon cable to drive Ø (the leftmost drive) using the same orientation procedures. Now plug the remaining flat ribbon connector onto the controller board connector along the top edge of the board. The connector may be keyed and thus will plug on only one way. If not, orient the connector so the tracer on the flat ribbon cable is nearer the left edge of the controller board as viewed from the front of the computer system. Double check all wiring for accuracy.

The following information should be read to familiarize you with diskettes and their use in the system. Read it carefully before proceeding with the diagnostics.

Final Checkout

Using the Disk System

The DMAF1 Floppy Disk System is designed to be as straightforward and easy to use as possible. There are certain things that the user must be aware of, however, for correct operation.

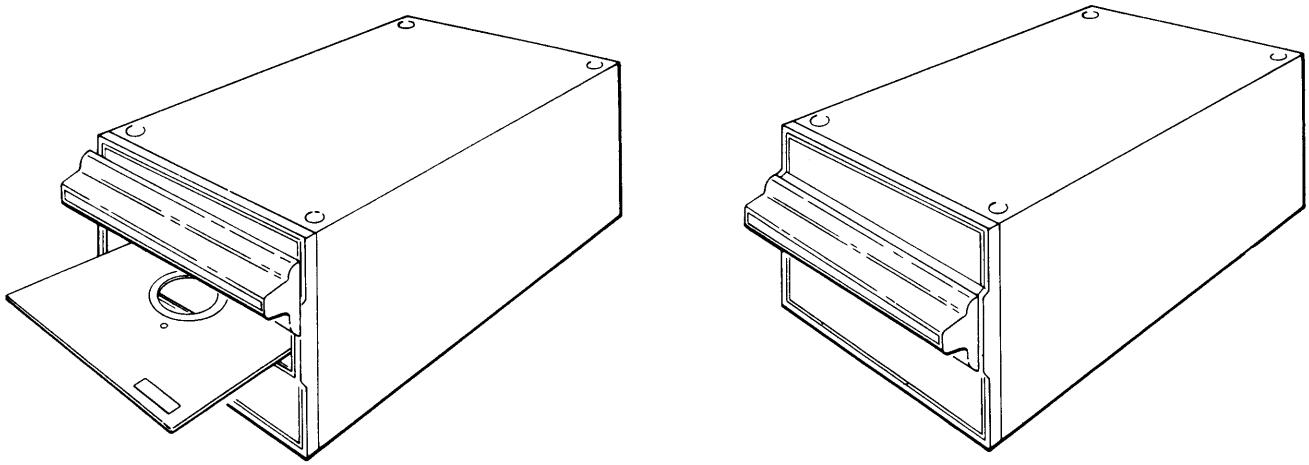
All RAM memory in the system must be operational for the disk to operate properly. If any doubt exists, run the memory diagnostics to verify correct operation.

Loading Flexible Disks

To load a flexible disk, simply depress the pushbutton located on the center of the loading handle. The loading handle is spring loaded and will then expose the load aperture.

Insert the flexible disk in the load aperture with the label toward the operator and facing the loading handle (see figure). Ensure that the flexible disk is inserted fully within the drive.

Grasp the bar on the loading handle and close it firmly; it will lock shut.

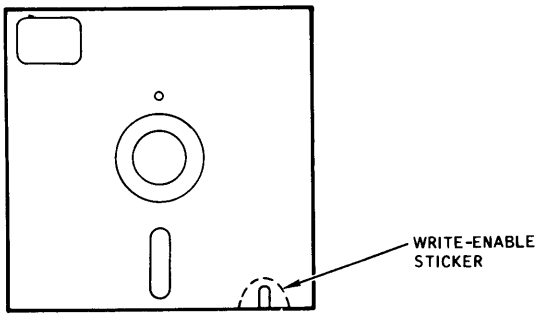


A. FLOPPY DISK IN LOAD POSITION

B. FLOPPY DISK LOADED

Flexible Disk Loading

Having the write protect notch on a diskette closed with a piece of tape will allow the diskette to be written on. Leaving the notch open will disable write privileges.



Write-Protect Option

The LED's on the drives are drive select/head load lights and are activated only when the head for a particular drive is loaded on a diskette. They are not power indicators.

The disk controller has been designed to load the head and turn on the drive motors only when necessary. When the computer requests data from the disk both motors will activate and the correct head will load. After the information has been retrieved the head will unload, and after approxi-. mately 60 seconds, both motors will turn off.

Just like cassette tapes, diskettes are made of magnetic materials and can be erased by stray magnetic fields. Also, it is an excellent idea to back-up all important disks on a spare diskette. The following precautions should be followed concerning diskettes.

1. Return the diskette to its storage envelope whenever it is removed from a drive.
2. Keep diskettes away from magnetic fields and from ferromagnetic materials which might become magnetized. Strong magnetic fields can distort recorded data on the disk.
3. Replace storage envelopes when they become worn, cracked or distorted. Envelopes are designed to protect the disk.

4. Do not write on the plastic jacket with a lead pencil or ball-point pen. Use a felt tip pen.
5. Heat and contamination from a carelessly dropped ash can damage the diskette.
6. Do not expose diskettes to heat or sunlight.
7. Do not touch or attempt to clean the disk surface. Abrasions may cause loss of stored data.
8. Flexible disks must be in the same temperature and humidity environment as the disk drive for a minimum of 5 minutes before installing the diskette in the disk drive. These environmental requirements are as follows:

Temperature	50°F (10°C) to 125°F (51.6°C); maximum gradient of 20°F (11.1°C) per hour
Relative Humidity	8% to 80%, maximum wet bulb 85°F (29.4°C)

Diagnostics

It is recommended that you try running the two following diagnostics before attempting to boot the disk system. Power up the computer system and the DMAF1 disk system. The disk system is on when the cooling fan runs. If you do run into a problem, try to pin it down to the DMAF1 controller board, motor control board, power supply or one of the disk drives. If you appear to be having a problem with a particular drive, reprogram them both so that drive Ø is drive 1 and vise versa. If it is a drive problem, this should confirm it. If you do run into some kind of problem(s) check the following:

1.) Make sure you have at least 12K of continguous RAM memory from ØØØØ thru 2FFF (0-12K).
2.) Check to see that you also have 8K of contiguous RAM memory from AØ8Ø thru BFFF (40K-48K).
3.) Run the memory diagnostics and verify this memory to be good.
4.) Make sure the DMAF1 disk controller board is properly programmed with the shorting blocks.
5.) Make sure there are no cold solder joints or foil bridges on the controller board. Also make sure all components are installed and oriented correctly.
6.) Make sure the DMAF1 disk controller board is plugged onto the computer system and is properly seated.
7.) Make sure the flat ribbon cable is plugged on the disk controller board and is oriented correctly.
8.) Make sure the flat ribbon cable is plugged onto both disk drives and is oriented correctly. Check the orientation very carefully.
9.) Make sure the J2-Ø, J2-1, JØ-Ø and JØ-1 power connectors are plugged into their mates on the back of both drives.
10.) Make sure the miniature connector going to the FD-M motor control board is oriented properly and plugged onto disk drive connector pins specified in the instruction set.
11.) Confirm that the cooling fan on the disk system is running. It should run whenever the disk system is turned on.
12.) Make sure you have modified your computer's MP-A processor and MP-B mother boards if applicable and have flipped the RAM switch OFF on the MP-A2 processor board if applicable.
13.) Run the REGTEST diagnostic (details follow).
14.) Run the STEPTEST diagnostic (details follow).
15.) Try booting the system (details follow) if the diagnostics run correctly.

REGTEST

The REGTEST diagnostic can be used to verify that the various registers on a DMAF1 disk controller board are being accessed properly. REGTEST assumes that the disk drives are connected to the controller and power is applied as described in the checkout instructions.

To use the REGTEST diagnostic, the code must be entered into the computer instruction by instruction using the memory examine and change function of the computer's monitor. Program execution should then be started at hex location Ø1ØØ by either setting AØ48 and AØ49 to Ø1ØØ and typing G or by typing J Ø1ØØ depending on your monitor. REGTEST will then check each register and will alternately select drive Ø and then drive 1. Correct drive selection is indicated if the LED on the front of drive Ø lights briefly and if the LED on drive 1 lights briefly when the one on drive Ø goes out. Both LEDs should never be on at the same time. Both drive motors should be running. If any register errors are seen an **X** will be displayed. If no errors are seen a + will be displayed. The diagnostic should be allowed to run until 256 +'s have been displayed.

REGTEST may be exited by depressing the RESET switch on the computer. A diskette need not be installed in the drive.

```
                        NAM     REGTEST
0100 CE 90 00  START    LDX     #$9000
0103 F6 01 61           LDA B   BYTE
0106 E7 00              STA B   0,X         STORE IN ALL LOCATIONS
0108 E7 01              STA B   1,X
010A E7 02              STA B   2,X
010C E7 03              STA B   3,X
010E E7 21              STA B   $21,X
0110 E7 22              STA B   $22,X
0112 E7 23              STA B   $23,X
0114 E6 00              LDA B   0,X         START READING DATA
0116 8D 37              BSR     TEST
0118 E6 01              LDA B   1,X
011A 8D 33              BSR     TEST
011C E6 02              LDA B   2,X
011E 8D 2F              BSR     TEST
0120 E6 03              LDA B   3,X
0122 8D 2B              BSR     TEST
0124 E6 21              LDA B   $21,X
0126 8D 27              BSR     TEST
0128 E6 22              LDA B   $22,X
012A 8D 23              BSR     TEST
012C E6 23              LDA B   $23,X
012E 8D 1F              BSR     TEST
0130 7C 01 61           INC     BYTE
0133 26 CB              BNE     START
0135 86 FE              LDA A   #$FE        SELECT DRIVE 0
0137 B7 90 24           STA A   DRVREG
013A 8D 1E              BSR     DELAY
013C 86 FD              LDA A   #$FD        SELECT DRIVE 1
013E B7 90 24           STA A   DRVREG
0141 8D 17              BSR     DELAY
0143 86 2B              LDA A   #'+
0145 BD E1 D1           JSR     OUTEEE
```

—Continued on next page—

```
0148 86 FF              LDA A   #$FF
014A B7 90 24           STA A   DRVREG
014D 20 B1              BRA     START       STORE NEW DATA

014F F1 01 61  TEST     CMP B   BYTE
0152 27 05              BEQ     OK
0154 86 58              LDA A   #'X         ERROR FOUND
0156 BD E1 D1           JSR     OUTEEE
0159 39        OK       RTS

015A CE FF FF  DELAY    LDX     #$FFFF
015D 09        DEC      DEX
015E 26 FD              BNE     DEC
0160 39                 RTS

0161 00        BYTE     FCB     00
```

STEPTEST

The STEPTEST diagnostic can be used to verify that the track selection circuitry of the system drive and the controller is working properly. If desired, a blank diskette may be used in place of the supplied system diskette.

To use the STEPTEST diagnostic, it should first be entered into the computer instruction by instruction using the memory examine and change function of the computer's monitor. A diskette (perferably blank or at least write protected) should then be installed in drive Ø and the door closed. Program execution should then be started at hex location ØlØØ by setting AØ48 and AØ49 to ØlØØ and typing G or by typing JØlØØ depending on the computer's monitor. STEPTEST will then select drive #Ø and check to see if the index hole sensing and motor control circuitry is working properly. If so, the disk drive heads will be moved back and forth between track ØØ and track 76. STEPTEST ouputs no information to the user—if the heads move back and forth across the diskette proper operation is assumed.

```
                        NAM     STEPTEST
0100 CE 90 20  START    LDX     #COMREG
0103 C6 FE              LDA B   #$FE        SELECT DRIVE 0
0105 E7 04              STA B   4,X
0107 E6 00              LDA B   0,X         WAIT UNTIL READY
0109 C5 80              BIT B   #$80
010B 26 F3              BNE     START
010D C6 08     LOOP     LDA B   #08         RESTORE
010F E7 00              STA B   0,X
0111 8D 12              BSR     WAIT        WAIT UNTIL THRU
0113 C6 4C              LDA B   #76         TRACK 76
0115 E7 03              STA B   3,X
0117 8D 13              BSR     DEL28       DELAY
0119 C6 18              LDA B   #$18        SEEK
011B E7 00              STA B   0,X
011D 8D 06              BSR     WAIT
011F 20 EC              BRA     LOOP
0121 E7 00     EXEC     STA B   0,X
0123 8D 07              BSR     DEL28
0125 E6 00     WAIT     LDA B   0,X         LOOP UNTIL 1771 THRU
0127 C5 01              BIT B   #1
0129 26 FA              BNE     WAIT
012B 39                 RTS
012C 8D 00     DEL28    BSR     DEL14
012E 8D 00     DEL14    BSR     DEL
0130 39        DEL      RTS
```

If you do not have an MP-A2 processor board with the DISKBUG® monitor installed, it will be necessary to enter the code for the bootstrap program by hand using the memory examine and change function of the monitor. The instructions and listing for the bootstrap program are contained in the DOS User's Manual. The boot will load the disk operating system from a system diskette installed in drive Ø (the left drive) only. Make sure the door on the disk drive is closed after inserting the diskette.

How It Works

The DMAF1 disk system can be broken down into four major parts: disk controller board, motor control board, power supply and the disk drives.

Disk Controller Board

The purpose of the disk controller board is to interface the disk drives to the computer system. Most of the control logic for the drives is handled by IC22, the 1771 disk controller chip. Since the data exchange rate between the computer system and the disk controller chip is a little too fast for a byte by byte transfer, a 6844 direct memory access (DMA) chip IC6 is used for disk data transfers between the 6800 Computer System and the 1771 disk controller chip. Much of the logic on the disk controller board is provided for interfacing these two chips together and to the rest of the system. Eight bit latch IC25 is used for drive and side select. It is a write only register and may be accessed at hex 9024. Timer IC24 is responsible for feeding the motor control board which turns the disk drive motors off if they have not been accessed for sixty seconds or so. This reduces drive and media wear and cuts down noise. Integrated circuits IC7, IC8, IC9, IC10, IC11 and IC12 are part of the clocked data separator that is external to the 1771 controller chip. The advantage here is that there are no adjustments to be made as with some data separators. The data separator internal to the 1771 is not reliable at the 2.0 MHz clock rate. Address decoding for the board is provided by integrated circuits IC4 and IC5. The board uses a 1K block of memory addresses anywhere from 32K thru 40K or 48K thru 54K jumper programmable. Bidirectional transceiver IC1 and buffer IC2 buffer the address lines to and from the board. Voltage regulator IC29 supplies +5 VDC power for the board while zener diodes D3 and D4 provide the +12 and -5 voltages required by the 1771 disk controller chip.

FD-M Motor Control Board

The motor control board turns AC power to the disk drives motors on and off as determined by the timer on the controller board. Integrated circuit IC1 on the board is an optically coupled triac which in turn drives Q1, a larger triac. The optical coupling is required to isolate the disk system's ground from the AC power line. NOTE: The motor control signal from the disk controller board is fed to two unused pins on both of the disk drives thru the flat ribbon cable. The miniature three-pin connector on the motor control board must be plugged onto these unused pins on one of the two drives otherwise the drive motors will not turn on. The orientation of the connector is important. Check the instruction set for complete details. Some of the components on this motor control board are connected to one side of the AC power line whenever the disk system's line cord is plugged into an AC receptacle regardless of whether or not it is turned on or running. Therefore, exercise extreme caution and never put your hands or tools near the board while the disk system is plugged into an AC receptacle.

P-200DF Power Supply

The power supply on the disk system is just about as simple as you can make it. The secondaries of the transformer are wired to provide 7 VAC and 24VAC outputs. These in turn are rectified by separate full wave bridge rectifier circuits providing +9 VDC and +30 VDC outputs. These outputs are then run thru separate integrated regulators Q3 and Q4 to yield the +5 VDC and +24 VDC outputs required by the disk drives. Power for the disk controller board is supplied by the 6800 Computer System.

Final Assembly

If everything seems to be working properly, you may now fasten down the disk drives and install the cover. Turn the power off and unplug the line cord on the disk system. Carefully turn each of the disk drives so they set flat on the chassis baseplate with the LED indicators nearer the bottom of the chassis. Let the plastic front panel of the disk drives overhang the front of the chassis. Make sure that none of the inter-connecting cables are twisted or sandwiched between the disk drives and chassis base plate. Charefully lift up the front of the chassis and secure the disk drives with #6-32 x ½" screws. Six are used to hold each drive in place. Install all of the screws finger tight until all twelve have been installed then tighten the screws being careful not to overtighten. The disk drive castings are aluminum and strip easily.

Before installing the cover, snap the two tinnerman nuts into the 3/16" holes on the sides of the chassis back panel. Carefully check once more to make sure that all of the interconnection cables are routed properly and that none are crimped. Route the flat ribbon cable so it lies neatly in the slot provided in the back panel. Carefully install the cover so that the center mounting hole on each side aligns with the one in the disk drive. Secure the cover using six black 3/8" screws. Here again be careful not to overtighten the screws.

IN CASE OF PROBLEMS

If your DMAF1 Disk System fails to operate properly we suggest that you first go back and double check all component installation and orientation. Be sure that they are turned as shown on the drawings and that each is the correct part number. The majority of problems turn out to be incorrect assembly. Using the printed pattern as a guide look over the board for solder bridges. Accidental solder bridges are the second most common problem in kits that are returned for repair. Be sure that all programming jumpers called for are in place and that all connections have been soldered.

If you suspect that one of the CalComp 143M disk drive units is not working properly, you may reprogram the drives and interchange the two. If your suspicions are correct remove the drive and return it to us for testing. **Do not** attempt to adjust, or repair the drive unit. Special equipment and tools are required and considerable damage can be done by attempting to work on these units without proper training.

If you have difficulty getting the disk system to work, repair services are available, however, it is advisable to try to determine which element of the system is not working and return only that portion rather than the entire assembly. The power supply circuitry is very straightforward and may be checked with a DC voltmeter. If it has a problem and you cannot repair it yourself, remove the power transformer and disk drives and return the entire chassis base plate. Do not return just the power supply board itself. If the disk drive motors do not turn on it is more than likely a problem with the disk controller board rather than the motor control board. If you have to return any part of your system, it would be a good idea to include your supplied system diskette containing the FDOS. Repairs are performed for a flat labor charge per item **plus** parts and postage.

Item	**Labor Charge**
Controller board and cable	$35.00
Power supply	$15.00
Disk drives	depends on individual drive

If we find that the controller board, drive or power supply is functional as received and does not require service, the Checkout Charge is $15.00

A confirmation sheet will be sent upon receipt of the kit. Please do not ask for a detailed report on exactly what was done in repairing your unit as we cannot provide this service.

It is not necessary to enclose any funds with the kit, you will be billed for authorized repairs.

SHIPPING INSTRUCTIONS

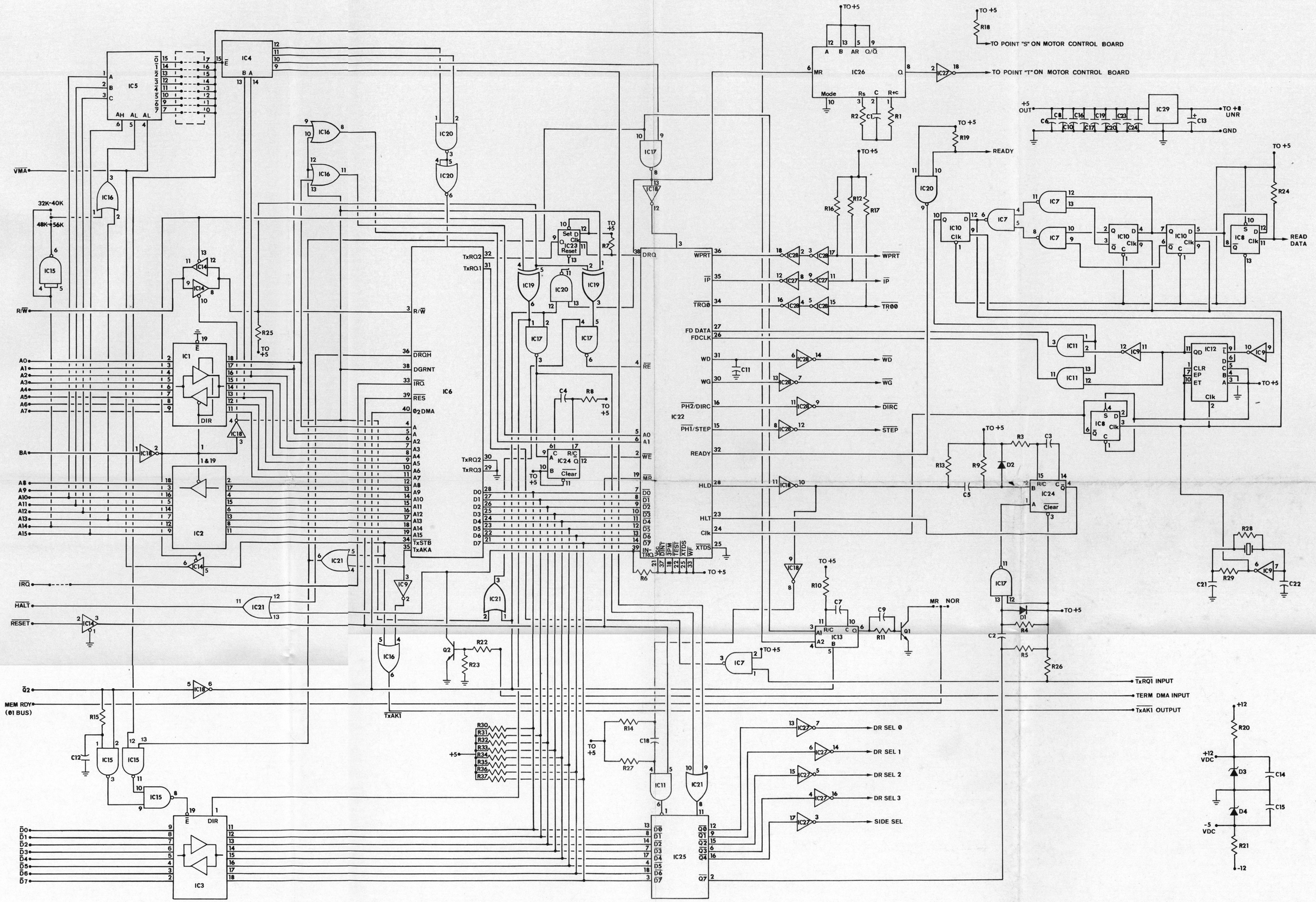
Pack in a large carton with at least 3 inches of padding on all sides. We will not service a kit if there is any postal damage until the claim is settled.

Include all relevant correspondence and a brief description of the difficulty.

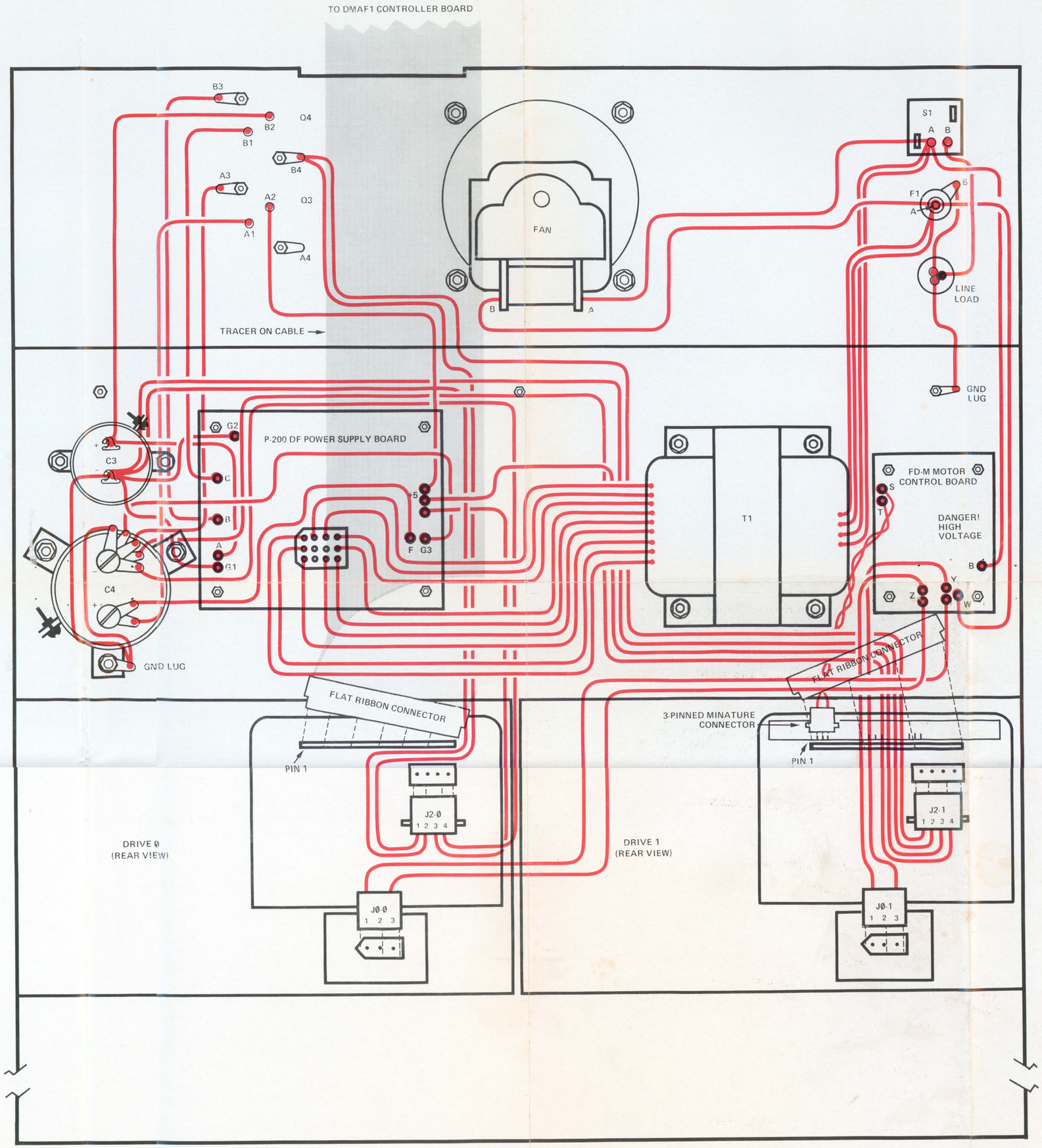
Ship prepaid by UPS or insured Parcel Post. We cannot pick up repairs sent by bus.

Ship to:

Southwest Technical Products Corp.
Repair Department - Digital Group
219 W. Rhapsody
San Antonio, Texas 78216



Schematic – DMAF1 Disk Controller Board



DMAF-1 DISK SYSTEM CHASSIS PICTORIAL

FLEX USER'S MANUAL

Technical Systems Consultants, Inc.
P. O. Box 2574
West Lafayette, Indiana 47906

COPYRIGHT NOTICE

IMPORTANT NOTE

Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible. Southwest Technical Products Corporation and Technical Systems Consultants assumes no responsiblity for any damages incurred or generated by such material. Southwest Technical Products Corporation and Technical Systems Consultants also reserve the right to make changes in such material at anytime.

PREFACE

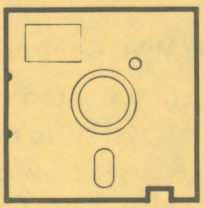
The purpose of this User's Guide is to provide the user of the FLEX Operating System with the information required to make effective use of the available system commands and utilities.The user should keep this manual close at hand while becoming familiar with the system. It is organized to make it convenient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

Notice to FLEX Users

After reading the User's Guide but before experimenting with the FLEX operating system, it is a good idea to follow the steps given below to make a duplicate diskette in case you accidentally enter a command which would erase the supplied system diskette.

1.) Power up the computer system and disk system. Be sure that all memory is good and be sure that you have memory installed from hex ØØØØ - 2FFF (12K) and from hex AØØØ - BFFF.

2.) If possible write protect your supplied diskette by removing the small piece of tape that covers the small rectangular hole on the edge of the diskette. Some diskettes do not have this notch and can not be write protected.



WRITE PROTECT NOTCH

3.) Install the supplied system diskette in drive Ø (the left hand drive) as described in the disk manual and close the door.

4.) Install a blank diskette in drive #1 and close the door. The write protect notch on this diskette (if it has one) should be covered.

5.) Boot up the FDOS system as described in the manual by either entering the boot by hand or by typing D depending on your monitor. The D command on the SWTBUG® monitor will not boot a DMAF1 system.

6.) The system should respond with FLEX and ask for the current date. Enter the date such as 5, 4, 78. If the system will not respond, try to boot again as described in the manual. If after several trys the system cannot be booted, the system diskette should be removed and all hardware checked.

7.) When the system is booted, type **NEWDISK 1** followed by a carriage return. Follow the instructions given in the NEWDISK command to answer any prompts.

8.) The system will then take several minutes to intialize the diskette. When finished, type **COPY Ø,1** followed by a carriage return.

9.) Flex will copy the system disk in about 5 minutes. When finished type **LINK 1.DOS** followed by a carriage return.

10.) The supplied system diskette should now be removed and set aside. The copy can be tested and used as desired.

Advanced Programmer's Manual

Throughout this manual you will find references to the DOS Advanced Programmer's Guide. This manual contains detailed information on the operation of the Disk Operating System at the machine language level. It is written for the individual who wishes to write his own utilities, interface to the DOS thru machine language programs, or just understand how it all works. It has been written for the individual who understands programming at the machine language level and it is not recommended for the novice. It is not being supplied with the DMAF1 kit but is sold separately for $20.00 ppd. in the continental U.S. It should be available sometime in July, 1978. When ordering please designate as the DMAF1 Advanced Programmer's Guide.

Blank Diskettes

For those of you who are having trouble purchasing double sided diskettes locally, you can order them from SWTPC. The order number for a blank double sided 8-inch diskette is FD-DS. Diskettes are $9.95 each ppd. in the continental U.S. Remember that single sided diskettes which are commonly available will also work with the DMAF1 system.

MP-T Interrupt Timer

For those of you wishing to implement the printer spooling function of FLEX, an MP-T Interrupt Timer board is needed. SWTPC offers the MP-T (in kit form only) for $39.95 ppd in the continental U.S.

Notice to Owners of the MP-C Control Interface and MIKBUG

FLEX will **not** work with an MP-C control interface, therefore systems containing the earlier MIKBUG® monitor will have to update the system to a monitor ROM which will support an MP-S serial interface. For serious disk users we suggest using the DISKBUG® monitor with an MP-A2 processor board. If you wish to use the earlier MP-A processor board, the SWTBUG® monitor will work but does not contain a DMAF1 boot command.

DISKBUG®

Serious disk users may be interested in purchasing the new monitor ROM DISKBUG® for their system. DISKBUG® contains a boot compatible with the DMAF1 disk system. DISKBUG® is currently available only in a 2716 EPROM for use in a MP-A2 processor board and is sold for $50.00 ppd in continental U.S. DISKBUG® requires an MP-S serial interface available for $35.00 (in kit form only) ppd in the continental U.S. DISKBUG® is not compatible with the earlier MP-A processor board since there is no provision for a EPROM on the board.

IMPORTANT NOTE

Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible. Southwest Technical Products Corporation and Technical Systems Consultants will not assume responsibility for any damages incurred or generated by such material. Also, Southwest Technical Products Corporation and Technical System Consultants reserve the right to make changes in such material at any time.

The hardware and software documentation for this kit are being shipped separately. Therefore, if you have received one but not the other, be patient. The rest of the kit should arrive shortly.

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.

II. SYSTEM REQUIREMENTS

The minifloppy version of FLEX requires random access memory from location ØØØØ through location 2FFF hex (12K). Memory is also required from AØØØ (40K) through BFFF hex (48K), 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 #Ø and #1. You should consult the disk drive instructions for this information. FLEX®will work only with Southwest Technical Product's SWTBUG® or DISKBUG® monitor ROMs and a **MP-S** Serial Interface.

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 if DISKBUG® is installed in the system, simply type "D" to call the disk boot loader from ROM.

As a specific example, suppose the system we are using has DISKBUG® installed and we wish to run FLEX. The first step is to power on all equipment and make sure the DISKBUG® prompt is present ($). Next insert the system diskette into drive 0 (the boot must be performed with the disk in drive Ø) and close the door on the drive. Type "D" on the terminal. 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 1Ø, Ø3, 78. 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 228Ø sectors may be used on any one diskette, but the user need not keep count, for the system does this automatically. A file will always be at least one sector long and can have a maximum length of 228Ø sectors. If single sided diskettes are being used a maximum of 114Ø sectors are available. 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
INVENTRY
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 'Ø' 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 of 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 extension 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 characters following the period 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 number 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: 'name', 'extension' and 'drive'. The rules for the file specification can be stated quite concisely using the following notation:

{(drive).}(name){. (extension)}
(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 '???', which 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 Ø
```

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 Ø 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. 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 Ø 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 is Ø and the working drive is 1, and the command line was:

```
+++LIST TEXTFILE
```

FLEX would go to drive Ø 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 (utility command set) 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 Ø.

MON

MON is used to exit FLEX and return to the hardware monitor system such as SWTBUG®. 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 ADØ3 hex. If using SWTBUG® or DISKBUG® simply typing 'G' will return you to the FLEX operating system.

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 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 two or more files, creating a new file as the result. Any type of file may be appended but it only makes sense to append files of 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.

DESCRIPTION

The general syntax for the APPEND command is as follows:

APPEND. (file spec) {.(file list) }, (file spec)

Where (file list) can be an optional list of the specifications. The last file name specified should not exist on the disk since this will be the name of the resultant file. If the last file name given does exist on the disk, the question "MAY THE EXISTING FILE BE DELETED?" will be displayed. A **Y** response will delete the current file and cause the APPEND operation to be completed. A **N** response will terminate the APPEND operation. All other files specified must exist since they are the ones to be appended together. If only 2 file names are given, the first file will be copied to the second file. The extension default is TXT unless a different extension is used on the FIRST FILE SPECIFIED, in which case that extension becomes the default for the rest of the command line. Some examples will show its use:

```
APPEND, CHAPTER1,CHAPTER2,CHAPTER3,BOOK
APPEND,FILE1,1.FILE2.BAK,GOODFILE
```

The first line would create a file on the working drive called 'BOOK.TXT' which would contain the files 'CHAPTER1.TXT', CHAPTER2.TXT', and 'CHAPTER3.TXT' in that order. The second example would append 'FILE2.BAK' from drive 1 to FILE1.TXT from the working drive and put the result in a file called 'GOODFILE.TXT' on the working drive. The file GOODFILE defaults to the extension of TXT since it is the default extension. Again, after the use of the APPEND command, all of the original files will be intact, exactly as they were before the APPEND operation.

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. As the system is initialized the automatic drive searching mode will be selected. An example will show how the system defaults to these values:

APPEND,FILE1,FILE2,FILE3

Upon receiving the above command line the operating system will try to execute the APPEND utility stored on drive Ø. If a file APPEND.CMD is not found on drive Ø, drive 1 would be searched, and if not there, drive 2, etc. The referencing of FILE1 and FILE2 would be done in the same way. The created file, FILE3, will be saved on the lowest drive number, drive Ø. When using the auto searching mode the lowest drive number in the system is always accessed first.

If the system drive is assigned to be Ø and the working drive is assigned to drive 1, then the above example will perform the following operation: get the APPEND command from drive Ø (the system drive), then append FILE2 from drive 1 (the working drive) to FILE 1 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.

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 #Ø
THE WORKING DRIVE IS #Ø
```

Some examples of using the ASN command are:

```
ASN,W=1
ASN,S=1,W=Ø
```

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 Ø. Careful use of drive assignments will 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
```

BACKUP

The BACKUP command allows for the making of copies of entire FLEX disks. These copies are different from those produced by the COPY command in that BACKUP makes a "mirror image" copy of the input disk, where COPY always reorganizes a disk so that a file's sectors are all grouped together. There are trade-offs involved when deciding whether to use the BACKUP command or the COPY command. Reorganization will speed up file accesses which have become slow due to the sectors of a file not being grouped together. Generally, COPY should be used if there are only a few files on the disk, or if the disk is very slow in access times. COPY will also allow single files to be copied as well as copying files to partially used sides. The BACKUP command, which in most cases will run faster than the COPY routine, will only copy entire disks, and the output disk will be entirely overwritten. Experience will help determine which command to use and when.

DESCRIPTION

The general syntax for the BACKUP command is:

BACKUP, (input drive), (output drive)

where the drives are specified with single digits. The input drive contains the disk we wish to copy the information from, and the output drive contains the disk on which we wish the data to be placed. As an example, to BACKUP drive Ø to drive 1, the following should be typed:

+++BACKUP,Ø,1

There are several situations which can exist at the start of a BACKUP operation. Since the BACKUP command copies every sector from the input drive to the output drive, not caring if there is actually information on those sectors, it requires that the output disk be formatted (initialized) and have no bad sectors.

If an attempt is made to back up to a diskette that has not be formatted, the message DISK FILE WRITE ERROR will be displayed. An error message will also be displayed if a bad sector is encountered on the destination disk or on the source disk.

One final note will be of interest. If the input disk had DOS.SYS on it, and it had been previously linked to the boot (see LINK command), then the new disk will also have DOS.SYS and it will be linked to the boot as well.

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 operation.

After you are in the 'BUILD' mode, the terminal will respond with the equals sign ('=') as the prompt character. This is similar to the Text Editing Systems'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 into the file. 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 separated 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,Ø,1
+++CAT,Ø,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 drives if auto drive searching is selected) whose names start with 'PR'. The next line causes all files on both drive Ø and drive 1 to be cataloged. Finally, the last example will catalog the files on drive Ø 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 This file is delete protected (delete or rename prohibited)

W This 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, groups of name–similar files, 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. When copying entire disks it is sometimes more desirable to use the BACKUP command. Refer to its description for details of the tradeoffs involved between the two methods of copying disks. 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 on 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,Ø.TEST.TXT,1.TEXT25
```

This command line would cause the file TEST.TXT on drive Ø 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,Ø.LIST.CMD,1
```

Here the file named LIST.CMD on drive Ø 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,Ø,1
+++COPY,1,Ø,.CMD,.SYS
+++COPY,Ø,1,A,B,CA.T
```

The first example will copy all files from drive Ø 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 Ø. No other files will be copied. The last example will copy the files from drive Ø 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 ouput 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 ouput to the terminal, as well as the drive to which it is being copied.

DATE

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

DESCRIPTION

The general syntax of the DATE command is:

DATE (mo., day, year)

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

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

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

Example:

+++ DATE
May 2, 1978

DELETE

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

DESCRIPTION

The general syntax of the DELETE command is:

DELETE,(file spec){,(file list)}

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

DELETE "FILE NAME"?

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

```
+++DELETE,MATHPACK.BIN
+++DELETE,1.TEST.TXT,Ø.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 Ø.

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.

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 prodedures 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 EXEC 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,Ø,1,.CMD,.OV,.LOW,.SYS
  =LINK,1.DOS
  = #
+++
```

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 Ø to drive 1. Finally the LINK will be performed. Now when we want to create a system disk in drive 1 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 memory beginning at hex location 7CØØ. Do not attempt to use EXEC if your system does not have memory at this address.

I

The I command can be used to force characters to all operator input requests (questions) in FLEX utilities.

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 the 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 Ø.DATA.DAT?" question.
          and 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 from 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 already in memory 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 1Ø3 hex. To start its execution from FLEX, type the following:

+++JUMP,1Ø3

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.

LINK

The LINK command is used to tell the bootstrap loader where the DOS.SYS 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 DOS. The default extension is SYS. Some examples of the use of LINK follow:

```
+++LINK,DOS
+++LINK,1.DOS
```

The first line will LINK DOS.SYS on the working drive, while the second example will LINK DOS.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 file 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 ØC formfeed character. Entering a +NP will select both options. A few examples will clarify the syntax used:

+++LIST,RECEIPTS,
+++LIST,CHAPTER1,3Ø,2ØØ,+NP
+++LIST,LETTER,1ØØ

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 3Øth line through the 2ØØth 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 selected 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 interpreted 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 1ØØ to the end of the file. No line numbers will be output since the 'N' was omitted.

The MEMTEST1 utility can be used to verify the integrity of the computer's memory. MEMTEST1 should be run periodically on your computer to alert you of any memory failures.

DESCRIPTION

The general syntax of the MEMTEST1 utility is:

MEMTEST1

MEMTEST1 does not have any arguments or file specifications associated with it. MEMTEST1 will then prompt you for the beginning and ending memory addresses. A four digit hexadecimal number should be entered in each case. In the case of a 32K system, the response would be as follows:

```
+++MEMTEST1
ENTER THE STARTING MEMORY ADDRESS (Ø2ØØ min) Ø2ØØ
ENTER THE ENDING MEMORY ADDRESS (7FFF max) 7FFF
```

If no errors are found in the memory being checked a & will be displayed on the screen. To completely test an area of memory, MEMTEST1 must be allowed to run until 256 &'s have been displayed on the screen. Each time a & is displayed on the screen MEMTEST has successfully cycled through memory storing and reading a different pattern.

After the selected region of memory has been tested (256 &'s displayed) MEMTEST1 will then cycle thru the RAM memory that FLEX uses (AØØØ - BFFF). Each + displayed on the screen denotes one successful cycle thru the memory. The diagnostic should run until 256 +'s have been displayed. MEMTEST1 will then exit to the computer system's monitor.

If an error is detected the output will be similar to the following:

```
$Ø6      2Ø          16AØ
(PATTERN #)  (ERRANT BITS)  (ADDRESS)
```

An error message such as this says that MEMTEST1 cycled thru memory five times without error, but on the sixth try a pattern was used that detected an error. The Ø6 tells what pattern number MEMTEST1 was working on when the error was detected. The 2Ø (hexadecimal) tells which bit(s) were in error. 2Ø converted to binary is ØØ1ØØØØØ—the location of the 1 is the bit(s) that were in error, in this case bit 5. Bit numbers start from Ø as shown.

```
          7 6 5 4 3 2 1 Ø  BIT #
2Ø16 = Ø Ø 1 Ø Ø Ø Ø Ø
```

The 16AØ is the address where the error was detected. This address may not store a particular number or possibly writing into another address, such as 16BØ, changed the contents of 16AØ.

The IC assignments table supplied with the memory board should be used to help locate the problem. In the above case on an MP-8M 8K memory board the bit # 5 IC in the upper 4K of memory should be suspected.

After running MEMTEST1, FLEX may be re-entered only by re-booting the system.

NEWDISK is used to format a new diskette. Diskettes as purchased will not work with FLEX until certain system information has been put on them. The NEWDISK utility puts this information on the diskette, as well as checking the diskette for defective sectors (bad spots on the surface of the disk which may cause data errors).

DESCRIPTION

The general syntax of the NEWDISK command is:

NEWDISK,(drive)

where (drive) represents a single digit drive number and specifies the drive to be formatted. After typing the command, the system will ask if you are sure you want a NEWDISK, and if the disk to be initialized is a scratch disk. Type 'Y' as the response to these questions if you are sure the NEWDISK command should continue. NEWDISK will also ask you if you have a double sided disk installed. If so, type 'Y'. If you are using single sided diskettes, type 'N'. SWTPC supplies only double sided diskettes with the DMAF1. NEWDISK then prompts for a volume name and number. This gives you the ability to "name" the diskette for future reference.

The NEWDISK process takes approximately five minutes to initialize a disk, assuming there are no bad spots on it. Defective sectors will make NEWDISK run even slower, depending on the number of bad sectors found. As bad sectors are detected, a message will be output to the terminal such as:

BAD SECTOR AT xxyy

where "xx" is the disk 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 disk has several bad sectors on it, it is still usable. When NEWDISK finishes, FLEX will report the number of available sectors remaining on the disk. If no defective sectors were detected, the total should be 2280 for double sided disks and 1140 for single sided.

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

FATAL ERROR–FORMATTING ABORTED

and FLEX will regain control. You should not immediately assume the disk to be useless if this occurs, but instead, remove the disk from the drive, re-insert it, and try NEWDISK again. If after several attempts the formatting is still aborted, you should assume the disk is unusable. You may not BACKUP onto a diskette with bad sectors on it. See the BACKUP documentation for more information.

CREATING SYSTEM DISKETTES

A system disk is one from which the disk operating system can be loaded. Normally the system disk 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 .CMD files desired to the new disk.
3. COPY all .SYS files to the new disk. It should be noted that steps 2 and 3 can be done with one command; 'COPY,Ø,1,.CMD,.OV,.LOW,.SYS', assuming you are copying from Ø to 1 and all command files and their overlays are desired. (the .OV copies overlay files and .LOW copies the utility 'SAVE.LOW').
4. Last it is necessary to LINK the file DOS.SYS to the system using the LINK command.

A very convenient way to get the above process performed without having to type all of the commands each time is to create a command file and use the EXEC command. Consult the EXEC documentation for details.

It is not necessary to make every disk a system diskette. It is also possible to create 'working' diskettes, disks 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. To create a working disk, simply run NEWDISK on a diskette. It will now have all of the required information to enable FLEX to make use of it. This disk, however, does not contain the disk operating system and is not capable of booting the system.

NEWDISK is used to format a new diskette. Diskettes as purchased will not work with FLEX until certain system information has been put on them. The NEWDISK utility puts this information on the diskette, as well as checking the diskette for defective sectors (bad spots on the surface of the disk which may cause data errors).

DESCRIPTION

The general syntax of the NEWDISK command is:

NEWDISK,(drive)

where (drive) represents a single digit drive number and specifies the drive to be formatted. After typing the command, the system will ask if you are sure you want a NEWDISK, and if the disk to be initialized is a scratch disk. Type 'Y' as the response to these questions if you are sure the NEWDISK command should continue. NEWDISK will also ask you if you have a double sided disk installed. If so, type 'Y'. If you are using single sided diskettes, type 'N'. SWTPC supplies only double sided diskettes with the DMAF1. NEWDISK then prompts for a volume name and number. This gives you the ability to "name" the diskette for future reference.

The NEWDISK process takes approximately five minutes to initialize a disk, assuming there are no bad spots on it. Defective sectors will make NEWDISK run even slower, depending on the number of bad sectors found. As bad sectors are detected, a message will be output to the terminal such as:

BAD SECTOR AT xxyy

where "xx" is the disk 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 disk has several bad sectors on it, it is still usable. When NEWDISK finishes, FLEX will report the number of available sectors remaining on the disk. If no defective sectors were detected, the total should be 2280 for double sided disks and 1140 for single sided.

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

FATAL ERROR—FORMATTING ABORTED

and FLEX will regain control. You should not immediately assume the disk to be useless if this occurs, but instead, remove the disk from the drive, re-insert it, and try NEWDISK again. If after several attempts the formatting is still aborted, you should assume the disk is unusable. You may not BACKUP onto a diskette with bad sectors on it. See the BACKUP documentation for more information.

CREATING SYSTEM DISKETTES

A system disk is one from which the disk operating system can be loaded. Normally the system disk 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 .CMD files desired to the new disk.
3. COPY all .SYS files to the new disk. It should be noted that steps 2 and 3 can be done with one command; 'COPY,Ø,1,.CMD,.OV,.LOW,.SYS', assuming you are copying from Ø to 1 and all command files and their overlays are desired. (the .OV copies overlay files and .LOW copies the utility 'SAVE.LOW').
4. Last it is necessary to LINK the file DOS.SYS to the system using the LINK command.

A very convenient way to get the above process performed without having to type all of the commands each time is to create a command file and use the EXEC command. Consult the EXEC documentation for details.

It is not necessary to make every disk a system diskette. It is also possible to create 'working' diskettes, disks which do not have the operating system on them, for use with text files or BASIC

O

The O (not zero) command can be used to route all displayed output from a utility to an output file instead of to the terminal. The function of O is similar to P (the printer commend) except that output is stored in a file rather than being printed on the terminal or printer. Other SWTPC and TSC software may support this utility. Check the supplied software instruction 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 very special and unlike any others currently in the UCS. P is the system print routine and will allow the output of any command to be routed to the printer. This is very useful for getting printed copies of the CATalog or when used with the LIST command will allow the printing of FLEX text files.

DESCRIPTION

The general syntax of the P command is:

P,(command)

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

+++P,CAT

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

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

The P command tries to load a file named PRINT.SYS from the same disk which P itself was retrieved. The PRINT.SYS file which is supplied with the system diskette contains the necessary routines to operate a SWTPC PR 40 printer connected through a parallel interface on PORT 7 of the computer. If you wish to use a different printer configuration, consult the 'Advanced Programmer's Guide' for details on writing your own printer driver routines to replace the PRINT.SYS file. The PR 40 drivers, however, are compatible with many other parallel interfaced printers presently on the market.

PRINT

FLEX has the ability to output file stored data to a printer at the same time that it is performing other tasks. This feature is especially useful when it is necessary to print a long listing without tying up the computer. This method of printing is called PRINTER SPOOLING. In order for the printer spooling function to work, a SWTPC MP-T interrupt timer board must be installed in I/O position #4 on the computer's mother board.

DESCRIPTION

The general syntax of the PRINT command is as follows:

PRINT (file spec), {repeat #}

where (file spec) is the name of the file to be printed. The default extension on (file spec) is .OUT. {Repeat #}is the number of **additional** copies of the file you wish to be printed.

For example, say that your disk had a very large number of files on it and a printed 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

+++PRINT,CAT.OUT or +++PRINT,CAT

should be entered.

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

After the file name to be printed has been stored in the print queue, control will return to the FLEX operating system. At this time you may perform any disk operation you want, such as deleting files, copying disks, etc. While you are using FLEX, PRINT will be outputting the desired file to the printer. PRINT 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 utility.

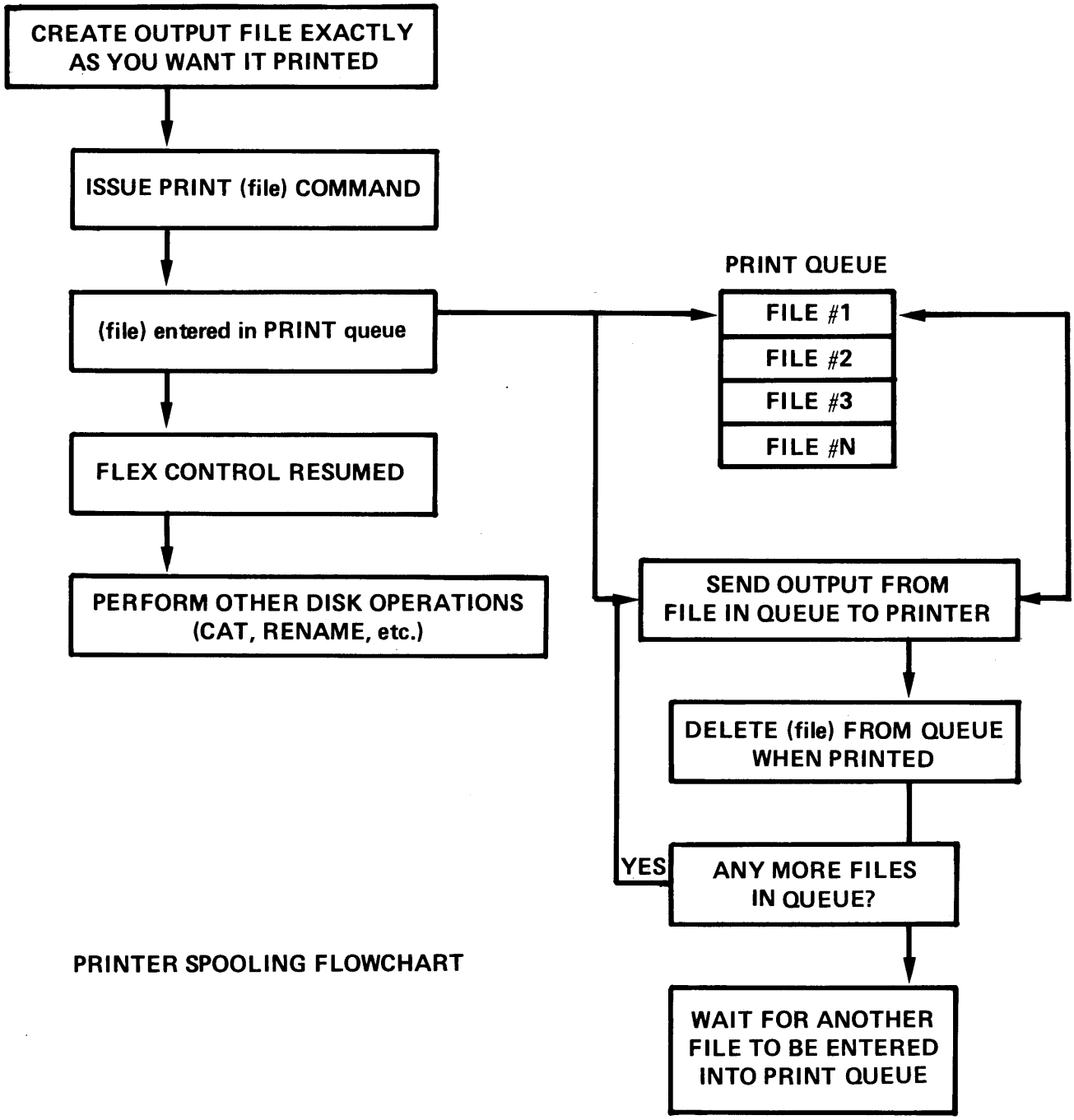
NOTE: There are several things that the user should be aware of when using printer spooling:

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 untility.

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

3.) The P command should not 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.



PRINTER SPOOLING FLOWCHART

PROT

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

DESCRIPTION

The general syntax of the PROT command is:

PROT, (file spec), {option list}

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

D —A **D** will delete protect a file. A delete protected file cannot be affected by using the DELETE or RENAME commands, or by the delete functions of SAVE, APPEND, etc.

W —A **W** will write protect a file. A write protected file can not 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.

The QCHECK utility can be used to examine the contents of the print queue and to modify its 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       Ø
       3      CHPTR2.     .TXT       Ø
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 awaiting 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 will repeat 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 will delete the file at queve 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 Ø. To completely terminate the current job 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 command 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, 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 deleted. No files are actually deleted, however.

RENAME

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

DESCRIPTION

The general syntax of the RENAME command is:

RENAME,(file spec 1),(file spec 2)

where (file spec 1) is the name of the file you wish to RENAME and (file spec 2) is the new name you are assigning to it. The default extension for file spec 1 is TXT and the default drive is the working drive. If no extension is given on (file spec 2), it defaults to that of (file spec 1). No drive is required 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,Ø.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 Ø to be renamed FIND.CMD. This is useful for making binary files created by an assembler into command files (changing the extension from BIN to CMD). If you try to give a file a name which already exists in the directory, the message:

FILE EXISTS

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

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

SAVE

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, 1Ø0,1FF
+++SAVE,1.GAME,Ø,168Ø,1Ø0
```

The first line would SAVE the memory locations 1Ø0 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 Ø through 168Ø to be SAVEd on the disk in file GAME.BIN on drive 1. Since a transfer address of 1Ø0 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 1Ø0.

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

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 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 necssary 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 DOS.SYS and must have been LINKed in order for the boot to work properly. Second, it is assumed that a file called BASIC.CMD actually exists on the disk.

Another example of the use of STARTUP is to set system environment parameters 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=6Ø:ASN,W=1:ASN:CAT,Ø
```

each time the system was booted the following actions would occur. First, TTYSET would set the 'depth' to 16 and the 'width' to 6Ø. 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 Ø 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 using the single line.

```
EXEC,(file name)
```

where (file name) would be replaced by the name assigned to the command file created. A little imagination and experience will show may 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, setting the program counter addresses AØ48 and AØ49 to ADØ3 and typing G for go. The STARTUP file may then be deleted and if desired, modified. Directing execution to ADØ3, the DOS warm start address, bypasses the DOS STARTUP function.

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' character to the character having the ASCII hex value of hh. This character is initially a 'control H' (hex Ø8), 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=Ø 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 the SWTPC CT-64.

If a hex Ø8 is specified as the echo character, FLEX will output a space (2Ø) then another Ø8. This feature is very useful for terminals which decode a hex Ø8 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 Ø8) 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 (2Ø) to erase the Y and another Ø8 (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 16). The action of the delete character is to 'erase' the current input line before it is accepted into the computer for execution. Setting DL=Ø 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 Ø 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=Ø 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 5Ø and a line of 125 characters is to be displayed, the first 5Ø characters are displayed on a physical line at the terminal, the next 5Ø characters are displayed on the next physical line, and the last 25 characters are displayed on the first physical line. If WD is set to Ø, 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 intial value is 4. Users using CRT type terminals may want to set NL=Ø 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 or printers with fan fold paper so as 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 intial 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. At this time (output halted), typing another 'escape character' will resume output, while typing the 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 termination always happens at the end of a line.

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.

VERSION

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

DESCRIPTION

The general syntax of the VERSION command is:

VERSION,(file spec)

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

+++VERSION,Ø.CAT

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

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 all on line drives 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
```

1. DISK CAPACITY

Each double sided diskette when used with FLEX is capable of holding 2280 sectors. Each sector can contian a maximum of 252 characters (4 bytes in each sector are used by the system). The total capacity of the diskette is then 574,560 characters or bytes of information. When using single sided diskettes with FLEX exactly one-half the amount of space is available.

II. WRITE PROTECT

It is possible to write on some diskettes only by placing a piece of opaque tape over the small rectangular cutout on the edge of the diskette. Any attempts to write files or delete files on a protected disk (no tape, hole exposed) will cause an error message to be issued. It is good practice to write protect disks which have important files on them. Some diskettes, however, do not contain this write protect notch.

III. THE 'RESET' BUTTON

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

IV. NOTES ON THE P COMMAND

The P command tries to load a file named PRINT.SYS from the same disk which P itself was retrieved. The PRINT.SYS file which was supplied with the system diskette contains the necessary routines to operate a SWTPC PR40 printer connected through a parallel interface on PORT 7 of the computer. If you wish to use a different printer configuration, consult the 'Advanced Programmer's Guide' for details on writing your own printer driver routines to replace the PRINT.SYS file.

V. ACCESSING DRIVES NOT CONTAINING A DISKETTE

If an attempt is made to access a drive not containing a diskette, a DRIVES NOT READY message will be output on the terminal.

VI. SYSTEM ERROR NUMBERS

Any time that FLEX detects an error during an operation, an appropriate error message will be displayed on the terminal. FLEX internally translates a derived error number into a plain language statement using a look-up table called ERRORS.SYS. If you have forgotten to copy this .SYS file onto a disk that you are using. FLEX will report a corresponding error 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 FMAS 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 DIRECTOR ERROR—REBOOT SYSTEM
6	THE SYSTEM DIRECTORY SPACE 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	DRIVES 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

For more details concerning the meanings of these error messages, consult the 'Advanced Programmer's Guide'.

VII. SYSTEM MEMORY MAP

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

0000 - 7FFF	User RAM
	*Note: Some of this space is ued by NEWDISK, BACKUP and other utilities.
A000 - BFFF	Disk Operating System
AD00	FLEX cold start entry address
AD03	FLEX warm start entry address
A100 - A6FF	Utility command space
A04A - A07F	System stack

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 jumps and a brief description of each. All given addresses are in hexadecimal.

AD15

This subroutine is functionally equivalent to SWTBUG®'s or DISKBUG®'s character input routine EIAC. This routine will look for one character from the control terminal (I/O #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 $ AD15 should be used.

AD15 automatically sets the 8th bit to Ø and does not check for parity. When using the subroutine the processor's registers are affected as follows:

ACC A	loaded with the character input from the terminal
ACC B	not affected
IXR	not affected

AD18

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

To use AD18 the character to be output should be placed in the A accumulator in its ASCII form. To output the letter A on the control terminal, the following program could be used:

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

The processor's registers are affected as follows:

ACC A	changed internally
ACC B	not affected
IXR	not affected

This routine is functionally equivalent to EIDI in SWTBUG® and DISKBUG®monitors.

ADIE

ADIE is the entry point of the subroutine used to output a string of text on the control terminal. When address ADIE 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 Ø4 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 ADIE are as follows:

ACC A	Changed during the operation
ACC B	UNCHANGED
IXR	Contains the memory location of the last character read from the string (usually the Ø4 unless stopped by the ESC key)

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

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

IX. BOOTING THE FLEX SYSTEM

Below is a short bootstrap program which will load the FLEX operating system from the system diskette. This boot is not necessary for user's having a DISKBUG® monitor—DISKBUG® already contains this boot.

To bring up the FLEX operating system, enter the bootstrap program below instruction by instruction using the memory examine and change function of your monitor. As shown, the bootstrap loads from hex address Ø1ØØ to Ø15A. After entering the bootstrap, set the computer's program counter AØ48 and AØ49 to Ø1ØØ. After a system diskette is installed in drive Ø, a G may be entered to execute the bootstrap.

If the system will not boot properly, re-position the system diskette in the drive and re-execute the bootstrap. The diskette to be booted must be initialized and must also contain the disk operating system software.

```
0100 C6 01      DISK    LDA B  #1          LOAD DRIVE NUMBER
0102 F7 90 22           STA B  SECREG      SET SECTOR TO 1
0105 53                 COM B              GENERATE INVERTED DATA
0106 F7 90 24   DISKSC  STA B  DRVREG
0109 8D 4A              BSR    DISKST      TEST SELECTED DRIVE STATUS
010B 27 0A              BEQ    DISKRC      GO ISSUE A RESTORE
010D 0D         DISKND  SEC
010E 59                 ROL B
010F C5 10              BIT B  #$10        TEST DRIVE SELECT BIT
0111 26 F3              BNE    DISKSC      SCAN IF SO
0113 C6 FE              LDA B  #$FF-1      SET BACK TO DRIVE 1
0115 20 EF              BRA    DISKSC
0117 86 08      DISKRC  LDA A  #$08        LOAD THE RESTORE COMMAND
0119 B7 90 20           STA A  COMREG
011C 8D 2F              BSR    DISKWT      WAIT FOR COMMAND TO FINISH
011E CE FE FF           LDX    #$FFFF-BYTES LOAD DMA BYTE COUNT
0121 FF 90 02           STX    CNTREG      STORE IN COUNT REGISTER
0124 CE 5E FF           LDX    #$FFFF-$A100 SET THE LOAD ADDRESS
0127 FF 90 00           STX    ADDREG
012A 86 FD              LDA A  #$FF-2      LOAD THE CHANNEL REGISTER
012C B7 90 10           STA A  CCREG
012F 86 FE              LDA A  #$FF-1      SET CHANNEL 0
0131 B7 90 14           STA A  PRIREG
0134 86 8C              LDA A  #$8C        SET SINGLE SECTOR READ
0136 B7 90 20           STA A  COMREG
0139 FE 90 02   DISKDW  LDX    CNTREG      GET THE BYTE COUNT
013C 8C FE FF           CPX    #$FFFF-BYTES
013F 27 F8              BEQ    DISKDW      LOOP AGAIN
0141 86 FF              LDA A  #$FF        SET THE PRI REGISTER TO 0
0143 B7 90 14           STA A  PRIREG
0146 8D 05              BSR    DISKWT      WAIT FOR COMMAND TO FINISH
0148 BD A1 00           JSR    $A100       JUMP TO SECTOR LOADED OFF OF DISK
014B 20 C0              BRA    DISKND      IF RTS, USE NEXT DRIVE
014D 8D 06      DISKWT  BSR    DISKST      CHECK READY STATUS
014F 26 FC              BNE    DISKWT      LOOP
0151 47                 ASR A              TEST BUSY BIT
0152 25 F9              BCS    DISKWT      WAIT FOR NOT BUSY
0154 39                 RTS
0155 B6 90 20   DISKST  LDA A  COMREG
0158 85 80              BIT A  #$80        TEST DRIVE READY BIT
015A 39                 RTS
```

X Requirements for the PRINT.SYS Printer Driver

FLEX, as supplied, includes a printer driver that will work with most parallel type printers, such as the SWTPC PR-40. If desired, the printer driver may be changed to accommodate other types of printers. Included is the source listing for the supplied driver. Additional information on the requirements for the PRINT.SYS driver can be found in the 'Advanced Programmer's Guide'.

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

2.) Hex location ØØ1Ø must contain the starting address of the port initialization routine.

3.) Hex location ØØ12 and location 71ØD must contain the address of the character output routine.

4.) When the printer character output routine is called by FLEX, the character to be output will be in the A accumulator. The output routine must not destroy the index register or the B accumulator.

5.) Both the initialization and output routine may reside anywhere in memory, but must not conflict with any utilities or programs which will use P.

6.) Both the initialization and the output routine must end with a return from subroutine RTS.

```
 1                              NAM     PRINT
 2                              OPT     PAG
 3                      *GENERATES THE PRINT SYS FILE FOR USE
 4                      *WITH THE P AND PRINT UTILITIES
 5                      *VERSION 1

 7  801C                PIA     EQU     $801C       PORT #7

 9                      *PRINTER INITIALIZATION (MUST BE AT $ACC0)
10  ACC0                        ORG     $ACC0
11  ACC0 86 FF          PINIT   LDA A   #$FF
12  ACC2 B7 80 1C               STA A   PIA         ALL OUTPUTS
13  ACC5 86 3E                  LDA A   #$3E
14  ACC7 B7 80 1D               STA A   PIA+1       SET UP HANDSHAKE TYPE
15  ACCA 39                     RTS

17                      *CHECK IF PRINTER READY ROUTINE
18                      *MUST BE LOCATED AT $ACD8
19  ACD8                        ORG     $ACD8
20  ACD8 7D 80 1D       PCHK    TST     PIA+1
21  ACDB 39                     RTS
22  ACDC B7 80 1D       PCHK3   STA A   PIA+1
23  ACDF 39                     RTS

25                      *OUTPUT CHARACTER ROUTINE
26                      *MUST BE LOCATED AT $ACE4
27  ACE4                        ORG     $ACE4
28  ACE4 8D F2          POUT    BSR     PCHK
29  ACE6 2A FC                  BPL     POUT
30  ACE8 7D 80 1C               TST     PIA
31  ACEB B7 80 1C               STA A   PIA
32  ACEE 86 36                  LDA A   #$36
33  ACF0 B7 80 1D               STA A   PIA+1
34  ACF3 86 3E                  LDA A   #$3E
35  ACF5 20 E5                  BRA     PCHK3
36                              END
```

CONTINUED

FLEX© VER. 1.0

ADVANCED PROGRAMMER'S GUIDE

IMPORTANT NOTE

Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Southwest Technical Products Corporation and Technical Systems Consultants will not assume responsibility for any damages incurred or generated by such material. Also, Southwest Technical Products Corporation and Technical Systems Consultants reserve the right to make changes in such material at any time.

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY SAN ANTONIO, TEXAS 78216

CONTENTS

Preface

The purpose of the Advanced Programmer's Manual is to provide the assembler language programmer with the information required to make effective use of the available system routines and functions. This manual applies to the eight inch version of FLEX. The programmer should keep this manual close at hand while learning the system. It is organized to make it convenient as a quick reference guide as well as a thorough reference manual. The manual is not written for the novice programmer and assumes the user to have a thorough understanding of assembler language programming techniques.

Copyright Notice

Introduction

The FLEX Operating System consists of three main parts: the Disk Operating System (DOS) which processes commands, the File Management System (FMS) which manages files on a diskette, and the Utility Command Set, which are the user-callable commands. The Utility Command Set is described in the FLEX User's Guide. Details of the Disk Operating System and File Management System portions of FLEX are described in this manual, which is intended for the programmer who wishes to write his own commands or process disk files from his own program.

When debugging programs which use disk files and the File Management System, the user should take the following precautions:

1. Write-protect the system diskette by exposing the write-enable cutout on the diskette. This will prevent destruction of the system disk in case the program starts running wild.

2. Use an empty scratch diskette as the working diskette to which your program will write any data files. If something goes wrong and the diskette is destroyed, no valuable data will have been lost.

3. Test your program repeatedly, especially with "special cases" of data input which may not be what the program is expecting. Well-written programs abort gracefully when detecting errors, not dramatically.

A careful programmer, using the information in this manual, should be able to make the fullest use of his floppy disk system.

DISCLAIMER

This product is intended for use only as described in this document and the FLEX User's Guide. Technical Systems Consultants and Southwest Technical Products Corporation will not be responsible for the proper functioning of features or parameters. The user is urged to abide by the warnings and cautions issued in this document lest valuable data or diskettes be destroyed.

PATCHING "FLEX"

It is not possible to patch FLEX. Technical Systems Consultants cannot be responsible for any destructive side-effects which may result from attempts to patch FLEX.

The Disk Operating System

The Disk Operating System (DOS) forms the communciation link between the user (via a computer terminal) and the File Management System. All commands are accepted through DOS. Functions such as file specification parsing, command argument parsing, terminal I/O, and error reporting are all handled by DOS. The following sections describe the DOS global variable storage locations (Memory Map), the DOS user callable subroutines, and give examples of some possible uses.

Memory Map

The following is a description of those memory locations within the DOS portion of FLEX which contain information of interest to the programmer. The user is cautioned against utilizing for his own purposes any locations documented as being either "reserved" or "system scratch", as this action may cause destruction of data.

$A080 - $A0FF – Line Buffer

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

$AC00 – TTYSET Backspace Character

This is the character which the routine INBUF will interpret as the Backspace character. It user definable through the TTYSET DOS utility. Default = $08, a Control–H (ASCII BS).

$AC01 – TTYSET Delete Character

This is the character which the routine INBUF will interpret as the line cancel or delete character. It is user definable through the TTYSET DOS Utility. Default = $18, a control–X (ASCII CAN).

$AC02 – TTYSET End of Line Character

This is the character DOS recognizes as the multiple command per line separator. It is user definable through the TTYSET Utility. Default = $3A, a colon (:).

$AC03 – TTYSET Depth Count

This byte determines how many lines DOS will print on a page before Pausing or issuing Ejects. It may be set by the user with the TTYSET command. Default = 0.

$AC04 – TTYSET Width Count

This byte tells DOS how many characters to output on each line. If zero, there is not limit to the number output. This count may be set by the user using TTYSET. Default = 0.

$AC05 – TTYSET Null Count

This byte informs DOS of the number of null or pad characters to be output after each carriage return, line feed pair. This count may be set using TTYSET. Default = 4.

$AC06 – TTYSET Tab Character

This byte defines a tab character which may be used by other programs, such as the Editor. DOS itself does not make use of the Tab character. Default = 0, no tab character defined.

$AC07 – TTYSET Backspace Echo Character

This is the character the routine INBUF will echo upon the receipt of a backspace character. If the backspace echo character is set to a $08, and the backspace character is also a $08, FLEX will output a space ($20) prior to the outputting of the backspace echo character. Default = 0.

$AC08 – TTYSET Eject Count

The Eject Count instructs DOS as to the number of blank lines to be output after each page. (A page is a set of lines equal in number to the Depth Count.) If this byte is zero, no Eject lines are output. Default = 0.

$AC09 – TTYSET Pause Control

The Pause byte instructs DOS what action to take after each page is output. A zero value indicates that the pause feature is enabled; a non-zero value, pause is disabled. Default = $FF, pause disabled.

$AC0A – TTYSET Escape Character

The Escape character causes DOS to pause after an output line. Default = $18, ASCII ESC.

$AC0B – System Drive Number

This is the number of the disk drive from which commands are loaded. If this byte is $FF, all ready drives will be searched. Default = $FF, all drives enabled.

$AC0C – Working Drive Number

This is the number of the default disk drive referenced for non-command files. If this byte is $FF, all ready drives will be searched. Default = $FF, all drives enabled.

$AC0D – System Scratch

$AC0E - $AC10 – System Date Registers

These three bytes are used to store the system date. It is stored in binary form with the month in the first byte, followed by the day, then the year. The year byte contains only the tens and ones digits.

$AC11 – Last Terminator

This location contains the most recent non-alphanumeric character encountered in processing the line buffer. See commentary on the routines NXTCH and CLASS in the section "User-Callable System Routines".

$AC12 - $AC13 – User Command Table Address

The programmer may store into these locations the address of a command table of his own construction. See the section called "User-Written Commands" for details. Default = 0000, no user command table is defined.

$AC14 - $AC15 – Line Buffer Pointer

These locations contain the address of the next character in the Line Buffer to be processed. See documentation of the routines INBUFF, NXTCH, GETFIL, GETCHR, and DOCMND in the section "User-Callable System Routines" for instances of its use.

$AC16 - $AC17 – Escape Return Register

These locations contain the address to which to jump if a RETURN is typed while output has been stopped by an Escape Character. See the FLEX User's Guide, TTYSET, for information on Escape processing. See also the documentation for the routine PCRLF in the section called "User-Callable System Routines".

$AC18 – Current Character

This location contains the most recent character taken from the Line Buffer by the NXTCH routine. See documentation of the NXTCH routine for additional details.

$AC19 – Previous Character

This location contains the previous character taken from the Line Buffer by the NXTCH routine. See documentation of the NXTCH routine for additional details.

$AC1A – Current Line Number

This location contains a count of the number of lines currently on the page. This value is compared to the Line Count value to determine if a full page has been printed.

$AC1B - $AC1C – Loader Address Offset

These locations contain the 16-bit bias to be added to the load address of a routine being loaded from the disk. See documentation of the System Routine LOAD for details. These locations are also used as scratch by some system routines.

$AC1D – Transfer Flag

After a program has been loaded from the disk (see LOAD documentation), this location is non-zero if a transfer address was found during the loading process. This location is also used as scratch by some system routines.

$AC1E - $AC1F – Transfer Address

If the Transfer Flag was set non-zero by a load from the disk (see LOAD documentation), these locations contain the last transfer address encountered. If the Transfer Flag was set zero by the disk load, the content of these locations is indeterminate.

$AC2Ø – Error Type

This location contains the error number returned by several of the File Management System functions. See the "Error Numbers" section of this document for an interpretation of the error numbers.

$AC21 – Special I/O Flag

If this byte is non-zero, the PUTCHR routine will ignore the TTYSET Width feature and also ignore the Escape Character. The routine RSTRIO clears this byte. Default = Ø.

$AC22 – Output Switch

If zero, output performed by the PUTCHR routine is through the routine OUTCH. If non-zero, the routine OUTCH2 is used. See documentation of these routines for details.

$AC23 – Input Switch

If zero, input performed by GETCHR is through the routine INCH. If it is non-zero, the routine INCH2 is used. See documentation of these routines for details.

$AC24 - $AC25 – File Output Address

These bytes contain the address of the File Control Block being used for file output. If the bytes are zero, no file output is performed. See PUTCHR description for details. These locations are set to zero by RSTRIO.

$AC26 - $AC27 – File Input Address

These bytes contain the address of the File Control Block being used for file input. If the bytes are zero, no file input is performed. The routine RSTRIO clears these bytes. See GETCHR for details.

$AC28 – Command Flag

This location is non-zero if DOS was called from a user program via the DOCMND entry point. See documentation of DOCMND for details.

$AC29 – Current Output Column

This location contains a count of the number of characters currently in the line being output to the terminal. This is compared to the TTYSET Width Count to determine when to start a new line. The output of a control character resets this count to zero.

$AC2A – System Scratch

$AC2B - $AC2C – Memory End

These two bytes contain the end of user memory. If 32K of memory exists in the machine, the DOS reserves a portion of the upper memory for future utility use. This location is set during system boot and may be read by programs requiring this information.

$AC2D - $AC2E – Error Name Vector

If these bytes are zero, the routine RPTERR will use the file ERRORS.SYS as the error file. If they are non-zero, they are assumed to be the address of an ASCII string of characters (in directory format) of the name of the file to be used as the error file. See the description of RPTERR for more details.

$AC2F – File Input Echo Flag

If this byte is non-zero (default) and input is being done through a file, the character input will be echoed by the output channel. If this byte is zero, the character retrieved will not be echoed.

$AC3Ø - $AC4D – System Scratch

$AC4E - $ACBF — System Constants

$ACCØ - $ACD7 — Printer Initialize

This area is reserved for the overlay of the system printer initialization subroutine.

$ACD8 - $ACE3 — Printer Ready Check

This area is reserved for the overlay of the system "check for printer ready" subroutine.

$ACE4 - $ACF7 — Printer Output

This area is reserved for the overlay of the system printer output character routine. See Printer Routine descriptions for details.

$ACF8 - $ACFF — System Scratch

User-Callable System Routines

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

$ADØØ (COLDS) Coldstart Entry Point

The BOOT program loaded from the disk jumps to this address to initialize the FLEX system. Both the Disk Operating System (DOS) portion and the File Management System portion (FMS) of FLEX are initialized. After initialization, the FLEX title line is printed and the STARTUP file, if one exists, is loaded and executed. This entry point is only for use by the BOOT program, not by user programs. Indiscriminate use of the Coldstart Entry Point by user programs could result in the destruction of the diskette. Documentation of this routine is included here only for completeness.

$ADØ3 (WARMS) Warmstart Entry Point

This is the main re-entry point into DOS from user programs. A JMP instruction should be used to enter the Warmstart Entry Point. Here, the system stack is reset, the monitor (SWTBUG/DISKBUG) program counter ($AØ48) is reset, as well as the Escape Return Register. At this point, the main loop of DOS is entered. The main loop of DOS checks the Last Terminator location for a TTYSET end-of-line character. If one is found, it is assumed that there is another command on the line, and DOS attempts to process it. If no end-of-line is in the Last Terminator location DOS assumes that the current command line is finished, and looks for a new line to be input from the keyboard. If, however, DOS was called from a user program through the DOCMND entry point, control will be returned to the user program when the end of a command line is reached.

$ADØ6 (RENTER) DOS Main Loop Re-entry Point

This is a direct entry point into the DOS main loop. None of the Warmstart initialization is performed. This entry point must be entered by a JMP instruction. Normally, this entry point is used internally by DOS and user-written programs should not have need to use it. For an example of use, see "Printer Driver" section for details.

$ADØ9 (INCH) Input Character

$ADØC (INCH2) Input Character

Each of these routines inputs one character from the keyboard, returning it to the calling program in the A-register. The address portion of these entry points is set to the SWTBUG/DISKBUG Input Character routine. It is not possible to patch this address to refer to some other routine. The GETCHR routine normally uses INCH but may be instructed to use INCH2 by setting the "Input Switch" non-zero (see Memory Map). The user's program may change the jump vector at the INCH address to refer to some other input routine such as a routine to get a character from paper tape. The OUTCH2 address should never be altered. The Warmstart

Entry Point resets the INCH jump vector to the same routine as INCH2 and sets the Input Switch to zero. RSTRIO also resets these bytes. User programs should use the GETCHR routine, documented below, rather than calling INCH, because INCH does not check the TTYSET parameters. The B and X registers are preserved.

$AD0F (OUTCH) Output Character

$AD12 (OUTCH2) Output Character

On entry to each of these routines, the A-register should contain the character being output. Both of these routines output the character in the A-register to an output device. The OUTCH routine usually does the same as OUTCH2; however, OUTCH may be changed by programs to refer to some other output routine. For example, OUTCH may be changed to drive a line printer. OUTCH2 is never changed, and always points to the SWTBUG/DISKBUG Output Character routine. This address may not be patched to refer to some other output routine. The routine PUTCHR, documented below, calls one of these two routines, depending on the content of the location "Output Switch" (see Memory Map). The Warmstart Entry Point resets the OUTCH jump vector to the same routine as OUTCH2, and sets the Output Switch to zero. RSTRIO also resets these locations. User routines should use PUTCHR rather than calling OUTCH or OUTCH2 directly since these latter two do not check the TTYSET parameters. The B and X registers are preserved.

$AD15 (GETCHR) Get Character

This routine gets a single character from the keyboard. The character is returned to the calling program in the A-register. The Current Line Number location is cleared by a call to GETCHR. Because this routine honors the TTYSET parameters, its use is preferred to that of INCH. If the location "Input Switch" is non-zero, the routine INCH2 will be used for input. If zero, the byte at "File Input Address" is checked. If it is non-zero, the address at this location is used as a File Control Block of a previously opened input file and a character is retrieved from the file. If zero, a character is retrieved via the INCH routine. The X and B registers are preserved.

$AD18 (PUTCHR) Put Character

This routine outputs a character to a device, honoring all of the TTYSET parameters. On entry, the character should be in the A-register. If the "Special I/O Flag" (see Memory Map) is zero, the column count is checked, and a new line is started if the current line is full. If an ACIA is being used to control the monitor terminal, it is checked for a TTYSET Escape Character having been typed. If so, output will pause at the end of the current line. If the location "Output Switch" is non-zero, the routine OUTCH2 is used to send the character. If zero, the location File Output Address is checked. If it is non-zero the contents of this location is used as a address of a File Control Block of a previously opened for write file, and the character is written to the file. If zero, the routine OUTCH is called to process the character. Normally, OUTCH sends the character to the terminal. The user program may, however, change the address portion of the OUTCH entry point to go to another character output routine. The X and B registers are preserved.

$AD1B (INBUFF) Input into Line Buffer

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

$AD1E (PSTRNG) Print String

This routine is similar to the PDATA routine in SWTBUG and DISKBUG. On entry, the X-register should contain the address of the first character of the string to be printed. The string must end with an ASCII EOT character ($Ø4). This routine honors all of the TTYSET conventions when printing the string. A carriage return and line feed are output before the string. The B register is preserved.

$AD21 (CLASS) Classify Character

This routine is used for testing if a character is alphanumeric (i.e. a letter or a number). On entry, the character should be in the A-register. If the character is alphanumeric, the routine returns with the carry flag cleared. If the character is not alphanumeric, the carry flag is set and the character is stored in the Last Terminator location. All registers are preserved by this routine.

$AD24 (PCRLF) Print Carriage Return and Line Feed

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

$AD27 (NXTCH) Get Next Buffer Character

The character in location Current Caaracter is placed in location Previous Character. The character to which the Line Buffer Pointer points is taken from the Line Buffer and saved in the Current Character location. Multiple spaces are skipped so that a string of spaces looks no different than a single space. The line Buffer Pointer is advanced to point to the next character unless the character just fetched was a RETURN or TTYSET End-of-Line character. Thus, once an end-of-line character or RETURN is encountered, additional calls to NXTCH will continue to return the same end-of-line character or RETURN. NXTCH cannot be used to cross into the next command in the buffer. NXTCH exits through the routine CLASS, automatically classifying the character. On exit, the character is in the A-register, the carry is clear if the character is alphanumeric, and the B-register and X-register are preserved.

$AD2A (RSTRIO) Restore I/O Vectors

This routine forces the OUTCH jump vector to point to the same routine as does the OUTCH2 vector. The Output Swtich location and the Input Switch location are set to zero. The INCH jump vector is reset to point to the same address as the INCH2 vector. Both the File Input Address and the File Output Address are set to zero. The A-register and B-register are preserved by this routine.

$AD2D (GETFIL) Get File Specification

On entry to this routine, the X-register must contain the address of a File Control Block (FCB) and the Line Buffer Pointer must be pointing to the first character of a file specification in the Line Buffer. This routine will parse the file specification, storing the various components in the FCB to which the X-register points. If a drive number was not specified in the file specification, the working drive number will be used. On exit, the carry bit will be clear if no error was detected in processing the file specification. The carry bit will be set if there was a format error in the file specification. If no extension was specified in the file specification, none is

stored. The calling program should set the default extension desired after GETFIL has been called by using the SETEXT routine. The Line Buffer Pointer is left pointing to the character immediately beyond the separator, unless the separator is a carriage return or End of Line character. If an error was detected, error number 21 is stored in the error status byte of the FCB. The X-register is preserved with a call to this routine.

$AD3Ø (LOAD) File Loader

On entry, the X-register must contain the address of a File Control Block which has been opened for binary reading of the desired file. This routine is used to load binary files only, not text files. The file is read from the disk and stored in memory, normally at the load addresses specified in the binary file itself. It is possible to load a binary file into a different memory area by using the Loader Address Offset locations. The 16-bit value in the Loader Address Offset locations is added to the addresses read from the binary file. Any carry generated out of the most significant bit of the address is lost. The transfer address, if any is encountered, is not modified by the Loader Address Offset. Note that the setting of a value in the Loader Address Offset does not modify any part of the content of the binary file. It does not act as a program relocator in that it does not change any addresses in the program itself, merely the location of the program in memory. On exit, the Transfer Address Flag is zero if no transfer address was found. This flag is non-zero if a transfer address record was encountered in the binary file, and the Transfer Address locations contain the last transfer address encountered. The disk file is closed on exit. If a disk error is encountered, an error message is issued and control is returned to DOS at the Warmstart Entry Point.

$AD33 (SETEXT) Set Extension

On entry, the X-register should contain the address of the FCB into which the default extension is to be stored if there is not an extension already in the FCB. The A-register, on entry, should contain a numeric code indicating what the default extension is to be. The numeric codes are described below. If there is already an extension in the FCB (possibly stored there by a call to GETFIL), this routine returns to the calling program immediately. If there is no extension in the FCB, the extension indicated by the numeric code in the A-register is placed in the FCB File Extension area. The legal codes are:

Ø	–	BIN
1	–	TXT
2	–	CMD
3	–	BAS
4	–	SYS
5	–	BAK
6	–	SCR
7	–	DAT
8	–	BAC
9	–	DIR
1Ø	–	PRT
11	–	OUT

Any values other than those above are ignored, the routine returning without storing any extension. The X-register is preserved in this routine.

$AD36 (ADDBX) Add B-register to X-register

The content of the B-register is added to the content of the X-register. The content of the B-register is destroyed on exit.

$AD39 (OUTDEC) Output Decimal Number

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

$AD3C (OUTHEX) Output Hexadecimal Number

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

$AD3F (RPTERR) Report Error

On entry to this routine, the X-register contains the address of a File Control Block in which the Error Status Byte is non-zero. The error code in the FCB is stored by this routine in the Error Type location. A call to the routine RSTRIO is made and location Error Vector is checked. If this location is zero, the file ERRORS.SYS is opened for random read. If this location is non-zero, it is assumed to be an address pointing to an ASCII string (containing any necessary null pad characters) of a legal file name plus extension (string should be 11 characters long). This user provided file is then opened for random read. The error number is used in a calculation to determine the record number and offset of the appropriate error string message in the file. Each error message string is 63 characters in length, thus allowing 4 messages per sector. If the string is found, it is printed on the terminal. If the string is not found (due to too large of error number being encountered) or if the error file itself was not located on the disk, the error number is reported to the monitor terminal as part of the message:

DISK ERROR #nnn

Where "nnn" is the error number being reported. A description of the error numbers is given elsewhere in this document.

$AD42 (GETHEX) Get Hexadecimal Number

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

$AD45 (OUTADR) Output Hexadecimal Address

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

$AD48 (INDEC) Input Decimal Number

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

$AD4B (DOCMND) Call DOS as a Subroutine

This entry point allows a user-written program to pass a command string to DOS for processing, and have DOS return control to the user program on completion of the commands. The command string must be placed in the Line Buffer by the user program, and the Line Buffer Pointer must be pointing to the first character of the command string. Note that this will destroy any as yet unprocessed parameters and commands in the Line Buffer. The command

string must terminate with a RETURN character ($D hex). After the commands have been processed, DOS will return control to the user's program with the B-register containing any error code received from the File Management System. The B-register will be zero if no errors were detected. Caution: do not use this feature to load programs which may destroy the user program in memory. An example of a use of this feature of DOS is that of a program wanting to save a portion of memory as a binary file on the disk. The program could build a SAVE command in the Line Buffer with the desired file name and parameters, and call the DOCMND entry point. On return, the memory will have been saved on the disk.

User-Written Commands

The programmer may write his own commands for DOS. These commands may be either disk-resident as disk files with a CMD extension., or they may be memory-resident in either RAM or ROM.

Memory-Resident Commands

A memory-resident command is a program, already in memory, to which DOS will transfer when the proper command is entered from the keyboard. The command which invokes the program, and the entry-point of the program, are stored in a User Command Table created by the programmer in memory. Each entry in the User Command Table has the following format:

```
FCC     'command'      (Name that will invoke the program)
FCB     Ø
FDB     entry address  (This is the entry address of the program)
```

The entire table is ended by a zero byte. For example, the following table contains the commands DEBUG (entry at $3ØØØ) and PUNT (entry at $32ØØ).

```
FCC     'DEBUG'        Command Name
FCB     Ø
FDB     $3ØØØ          Entry address for DEBUG
FCC     'PUNT'         Command Name
FCB     Ø
FDB     $32ØØ          Entry address for PUNT
FCB     Ø              End of command table
```

The address of the User Command Table is made known to DOS by storing it in the User Command Table Address locations (see Memory Map).

The User Command Table is searched before the disk directory, but after DOS's own command table is searched. The DOS command table contains only the GET and MON commands. Therefore, the user may not define his own GET and MON commands.

Since the User Command Table is searched before the disk directory, the programmer may have commands with the same name as those on the disk. However, in this case, the commands on the disk will never be executed while the User Command Table is known to DOS. The User Command Table may be deactivated by clearing the User Command Table Address locations.

Disk-Resident Command

A disk-resident command is an assembled program, with a transfer address, which has been saved on the disk with a CMD extension. The ASMB section of the FLEX User's Guide describes the way to assign a transfer address to a program being assembled.

Disk commands, when loaded into memory, may reside anywhere in the User RAM Area; the address is determined at assembly time by using an ORG statement. Most commands may be assembled to run in the Utility Command Space (see Memory Map). Most of the commands supplied with FLEX run in the Utility Command Space. For this reason, the SAVE command cannot be used to save information which is in the Utility Command Space or System FCB space as this information would be destroyed when the SAVE command is loaded.The SAVE.LOW command is to be used in this case. The SAVE.LOW command loads into memory at location $1ØØ and allows the saving programs in the $A1ØØ region.

The System FCB area is used to load all commands from the disk. Commands written to run in the Utility Command Space must not overflow into the System FCB area. Once loaded, the command itself may use the System FCB area for scratch or as an FCB for its own disk I/O. See the example in the FMS section.

General Comments About Commands

User-written commands are entered by a JMP instruction. On completion, they should return control to DOS by jumping (JMP instruction) to the Warmstart Entry Point (see Memory Map).

Processing Arguments

User-written commands are required to process any arguments entered from the keyboard. The command name and the arguments typed are in the Line Buffer area (see Memory Map). The Line Buffer Pointer, on entry to the command, is pointing to the first character of the first argument, if one exists. If there are no arguments, the Line Buffer Pointer is pointing to either an end-of-line character or a carriage return. The DOS routines NXTCH, GETFIL, and GETHEX should be used by the command for processing the arguments.

Processing Errors

If the command, while executing, receives an error status from either DOS or FMS of such a nature that the command must be aborted, the program should jump to the Warmstart Entry Point of DOS after issuing an appropriate error message. Similarly, if the command should detect an error on its own, it should issue a message and return to DOS through the Warmstart Entry Point.

Examples of Using DOS Routines

1. Setting up a file spec in the FCB can be done in the following manner. This example assumes the Line Buffer Pointer is pointing to the first character of a file specification, and the desired resulting file spec should default to a TXT extension.

```
LDX     #FCB        Point to FCB
JSR     GETFIL      Get file spec into FCB
BCS     ERROR       Report error if one
LDAA    #1          Set extension code (TXT)
JSR     SETEXT      Set the default extension
```

   The user may now open the file for the desired action, since the file spec is correctly set up in the FCB. Refer to the FMS examples for opening files.

2. The following examples demonstrate some simple uses of the basic I/O functions provided by DOS.

```
LDAA    #'A         Setup an ASCII A
JSR     PUTCHR      Call DOS out character

LDX     #STRING     Point to string
JSR     PSTRNG      Print CR & LF + string
```

   The above simple examples are to show the basic mechanism for calling and using DOS I/O routines.

The File Management System

The File Management System (FMS), forms the communication link between the DOS and the actual disk hardware. The FMS performs all file allocation and removal on the disk. All file space is allocated dynamically, and the space used by files is immedately reusable upon that file's deletion. The user of the FMS need not be concerned with the actual location of a file on the disk, or how many sectors it requires.

Communication with the FMS is done through File Control Blocks. These blocks contain the information about a file, such as its name and what drive it exists on. All disk I/O performed through FMS is "one character at a time" I/O. This means that programs need only send or request a single character at a time while doing file data transfers. In effect, the disk looks no different than a computer terminal. Files may be opened for either reading or writing. Any number of files may be opened at any one time, as long as each one is assigned its own File Control Block.

The FMS is a command language whose commands are represented by various numbers called Function Codes. Each Function Code tells FMS to perform a specific function such as open a file for read, or delete a file. In general, making use of the various functions which the FMS offers is quite simple. The index register is made to point to the File Control Block which is to be used, the Function Code is stored in the first byte of the File Control Block, and FMS is called as a subroutine (JSR). At no time does the user ever have to be concerned with where its directory entry is located. The FMS does all of this automatically.

Since the file structure of FLEX is a linked structure, and the disk space is allocated dynamically, it is possible for a file to exist on the disk in a set of non-contiguous sectors. Normally, if a disk has just been formatted, a file will use consecutive sectors on the disk. As files are created and deleted, however, the disk may become "fragmented". Fragmentation results in the sectors on the disk becoming out of order physically, even though logically they are still all sequential. This is a characteristic of "linked list" structures and dynamic file allocation methods. The user need not be concerned with this fragmentation, but should be aware of the fact that files may exist whose sectors seem to be spattered all over the disk. The only result of fragmentation is the slowing down of file read times, because of the increased number of head seeks necessary while reading the file.

The File Control Block (FCB)

The FCB is the heart of the FLEX File Management System (FMS). An FCB is a 32Ø byte long block of RAM, in the user's program area, which is used by programs to communicate with FMS. A separate FCB is needed for each open file. After a file has been closed, the FCB may be re-used to open another file or to perform some other disk function such as Delete or Rename. An FCB may be placed anywhere in the user's program area (except page zero) that the programmer wishes. The memory reserved for use as an FCB need not be preset or initialized in any way. Only the parameters necessary to perform the function need be stored in the FCB; the File Management System will initialize those areas of the FCB needed for its use.

In the following description of an FCB, the byte numbers are relative to the beginning of the FCB; i.e. byte Ø is the first byte of the FCB.

Description of an FCB

Byte Ø Function Code

The desired function code must be stored in this byte by the user before calling FMS to process the FCB. See the section describing FMS Function Codes.

Byte 1 Error Status Byte

If an error was detected during the processing of a function, FMS stores the error number in this byte and returns to the user with the CPU Z-Condition Code bit clear, i.e. a non-zero condition exists. This may be tested by the BEQ or BNE instruction.

Byte 2 Activity Status

This byte is set by FMS to a "1" if the file is open for read, or "2" if the file is open for writing. This byte is checked by several FMS function processors to determine if the requested operation is legal. A Status Error is returned for illegal operations.

The next 12 bytes (3-14) comprise the "File Specification" of the file being referenced by the FCB. A "File Specification" consists of a drive number, file name, and file extension. Some of the FMS functions do not require the file name or extension. See the documentation of the individual function codes for details.

Byte 3 Drive Number

This is the hardware drive number whose diskette contains the file being referenced. It should be binary Ø to 3.

The next 24 bytes (4-27) comprise the "Directory Information" portion of the FCB. This is the exact same information which is contained in the diskette directory entry for the file being referenced.

Bytes 4-11 File Name

This is the name of the file being referenced. The name must start with a letter and contain only letters, digits, hyphens, and/or underscores. If the name is less than 8 characters long, the remaining bytes must be zero. The name should be left adjusted in its field.

Bytes 12-14 Extension

This is the extension of the file name for the file being referenced. It must start with a letter and contain only letters, digits, hyphens, and/or underscroes. If the extension is less than 3 characters long, the remaining bytes must be zero. The extension should be left adjusted. Files with null extensions should not be created.

Byte 15 File Attributes

At present, only the most significant 4 bits are defined in this byte. These bits are used for the protection status bits and are assigned as follows:

BIT 7 = Write Protect
BIT 6 = Delete Protect
BIT 5 = Read Protect
BIT 4 = Catalog Protect

Setting these bits to 1 will activate the appropriate protection status. All undefined bits of this byte should remain Ø.

Byte 16 Reserved for future system use

Bytes 17-18 Starting disk address of the file

These two bytes contain the hardware track and sector numbers, respectively, of the first sector of the file.

Bytes 19-2Ø Ending disk address of the file

These two bytes contain the hardware track and sector numbers, respectively, of the last sector of the file.

Bytes 21-22 File Size

This is a 16-bit number indicating the number of sectors in the file.

Byte 23 File Sector Map Indicator

If this byte is non-zero (usually $Ø2), the file has been created as a random access file and contains a File Sector Map. See the description of Random Files for details.

Byte 24 Reserved for future system use

Bytes 25-27 File Creation Date

These three bytes contain the binary date of the files creation. The first byte is the month, the second is the day, and the third is the year (only the tens and ones digits).

Bytes 28-29 FCB List Pointer

All FCBs which are open for reading or writing are chained together. These two bytes contain the memory address of the FCB List Pointer bytes of the next FCB in the chain. These bytes are zero if this FCB is the last FCB in the chain. The first FCB in the chain is pointed to by the FCB Base Pointer. (See Global Variables.)

Bytes 30-31 Current Position

These bytes contain the hardware track and sector numbers, respectively, of the sector currently in the sector buffer portion of the FCB. If the file is being written, the sector to which these bytes point has not yet been written to the diskette; it is still in the buffer.

Bytes 32-33 Current Record Number

These bytes contain the current logical Record Number of the sector in the FCB buffer.

Byte 34 Data Index

This byte contains the address of the next data byte to be fetched from (if reading) or stored into (if writing) the sector buffer. This address is relative to the beginning of the sector, and is advanced automatically by the Read/Write Next Byte function. The user program has no need to manipulate this byte.

Byte 35 Random Index

This byte is used in conjunction with the Get Random Byte From Sector function to read a specific byte from the sector buffer without having to sequentially skip over any intervening bytes. The address of the desired byte, relative to the beginning of the sector, is stored in Random Index by the user, and the Get Random Byte From Sector function is issued to FMS. The specified data byte will be returned in the A-register. A value less than 4 will access one of the linkage bytes in the sector. User data starts at an index value of 4.

Bytes 36-46 Name Work Buffer

These bytes are used internally by FMS as temporary storage for a file name. These locations are not for use by a user program.

Bytes 47-49 Current Directory Address

If the FCB is being used to process directory information with the Get/Put Information Record functions, these three bytes contain the track number, sector number, and starting data indexof the directory entry whose content is in the Directory Information portion of the FCB. The values in these three bytes are updated automatically by the Get Information Record function.

Bytes 50-52 First Deleted Directory Pointer

These bytes are used internally by FMS when looking for a free entry in the directory to which to assign the name of a new file.

Bytes 53-63 Scratch Bytes

These are the bytes into which the user stores the new name and extension of a file being renamed. The new name is formatted the same as described above under File Name and File Extension.

Byte 59 Space Compression Flag

If a file is open for read or write, this byte indicates if space compression is being performed. A value of zero indicates that space compression is to be done when reading or writing the data. This is the value that is stored by the Open for Read and Open for Write functions. A value of $FF indicates that no space compression is to be done. This value is what the user must store in this byte, after opening the file, if space compression is not desired. (Such as for binary files.) A positive non-zero value in this byte indicates that space compression is currently in progress; the value being a count of the number of spaces processed thus far. (Note that although this byte overlaps the Scratch Bytes described above, there is no conflict since the Space Compression Flag is used only when a file is open, and the Scratch Bytes are used only by Rename, which requires that the file be closed.) In general, this byte should be 0 while working with text type files, and $FF for binary files.

Bytes 64-319 Sector Buffer

These bytes contain the data contained in the sector being read or written. The first four bytes of the sector are used by the system. The remaining 252 are used for data storage.

$B400 – FMS Initialization

This entry point is used by the DOS portion of FLEX to initialize the File Management System after a coldstart. There should be no need for a user-written program to use this entry point. Executing an FMS Initialization at the wrong time may result in the destruction of data files, necessitating a re-initialization of the diskette.

$B403 – FMS Close

This entry point is used by the DOS portion of FLEX at the end of each command line to close any files left open by the command processor. User-written programs may also use this entry point to close all open files; however, if an error is detected in trying to close a file, any remaining files will not be closed. Thus the programmer is cautioned against using this routine as a substitute for the good programming practice of closing files individually. There are no arguments to this routine. It is entered by a JSR instruction as though it were a subroutine. On exit, the CPU Z-Condition code is set if no error was detected (i.e. a "zero" condition exists). If an error was detected, the CPU Z-Condition code bit is clear and the X-register contains the address of the FCB causing the error.

$B406 – FMS Call

This entry point is used for all other calls to the File Management System. A function code is stored in the Function Code byte of the FCB, the address of the FCB is put in the X-register, and this entry point is called by a JSR instruction. The function codes are documented elsewhere in this document. On exit from this entry point, the CPU Z-Condition code bit is set if no error was detected in processing the function. This bit may be tested with a BEQ or BNE instruction. If an error was detected, the CPU Z-Condition code bit is cleared and the Error Status byte in the FCB contains the error number. Under all circumstances, the address of the FCB is still in the X-register on exit from this entry point. Some of the functions require additional parameters in the A and/or B-registers. See the documentation of the Function codes for details. The X and B registers are always preserved with a call to FMS.

Global Variables

This section describes those variables within the File Management System which may be of interest to the programmer. Any other locations in the FMS area should not be used for data storage by user programs.

$B409 - $B40A FCB Base Pointer

These locations contain the address of the FCB List Pointer bytes of the first FCB in the chain of open files. The address in these locations is managed by FMS and the programmer should not store any values in these locations. A user program may, however, want to chain through the FCBs of the open files for some reason,and the address stored in these locations is the proper starting point. Remember that the address is that of the FCB List Pointer locations in the FCB, not the first word of the FCB. A value of zero in these locations indicates that there are no open files.

$B40B - $B40C Current FCB Address

These locations contain the address of the last FCB processed by the File Management System. The address is that of the first word of the FCB.

$B435 Verify Flag

A non-zero value in this location indicates that FMS will check each sector written for errors immediately after writing it. A zero value indicates that no error checking on writes is to be performed. The default value is "non;zero".

FMS Function Codes

The FLEX File Management System is utilized by the user through function codes. The proper function code number is placed, by the user, in the Function Code byte of the File Control Block (FCB) before calling FMS (Byte Ø). FMS should be called by a JSR to the "FMS Call" entry. On entry to FMS, the X-register should contain the address of the FCB. On exit from FMS, the CPU Z-Condition code bit will be clear if an error was detected while processing the function. This bit may be tested by the BNE and BEQ instructions. Note: In the following examples, the line "JSR FMS" is referencing the FMS Call entry at $B4Ø6.

Function Ø – Read/Write Next Byte/Character

If the file is open for reading, the next byte is fetched from the file and returned to the calling program in the A-register. If the file is open for writing, the content of the A-register on entry is placed in the buffer as the next byte to be written to the file. The Compression Mode Flag must contain the proper value for automatic space compression to take place, if desired (see description of the FCB Compression Mode Flag for details). On exit, this function code remains unchanged in the Function Code byte of the FCB; thus, consecutive read/writes may be performed without having to repeatedly store the function code. When reading, an End-of-File error is returned when all data in the file has been read. When the current sector being read is empty, the next sector in the file is prepared for processing automatically, without any action being required of the user. Similarly, when writing, full sectors are automatically written to the disk without user intervention.

Example:

```
If reading–
LDX     #FCB    Point to the FCB
JSR     FMS     Call FMS
BNE     ERROR   Check for error
The character read is now in A.

If writing–
LDAA    CHAR    Get the character
LDX     #FCB    Point to the FCB
JSR     FMS     Call FMS
BNE     ERROR   Check for errors
The character in A has been written.
```

Function 1 – Open for Read

The file specified in the FCB is opened for read-only access. If the file cannot be found, an error is returned. The only parts of the FCB which must be preset by the programmer before issuing this function are the file specification parts (drive number file name, and file extension) and the function code. The remaining parts of the FCB will be initialized by the Open process. The Open process sets the File Compression Mode Flag to zero, indicating a text file. If the file is binary, the programmer should set the File Compression Mode Flag to $FF, after opening the file, to disable the space compression feature. On exit from FMS, after opening a file, the function code in the FCB is automatically set to zero (Read/Write Next Byte Function) in anticipation of Input/Output on the file.

Example:

```
LDX     #FCB    Point to the FCB
[ Set up file spec in FCB ]
LDAA    #1      Set open function code
STAA    Ø,X     Store in FCB
JSR     FMS     Call FMS
BNE     ERROR   Check for errors
The file is now open for text reading

To set for binary–continue with the following
LDAA    #$FF    Set FF for sup. flag
STAA    59,X    Store in suppression flag
```

Function 2 – Open for Write

This is the same as Function 1, Open for Read, except that the file must not already exist in the diskette directory, and it is opened for write-only access. A file opened for write may not be read unless it is first closed and then re-opened for read-only. The space compression flag should be treated the same as described in "Open for Read". A file is normally opened as a sequential file but may be created as a random file by setting the FCB location File Sector Map byte non-zero immediately following an open for write operation. Refer to the section on Random Files for more details. The file will be created on the drive specified unless the drive spec of $FF in which case the file will be created on the first drive found to be ready.

Example:

```
LDX     #FCB    Point to FCB
[ Setup file spec in FCB ]
LDAA    #2      Setup open for write code
STAA    Ø,X     Store in FCB
JSR     FMS     Call FMS
BNE     ERROR   Check for errors
File is now open for text write
```

For binary, follow example in Read open.

Function 3 – Open for Update

This function opens the file for both read and write. The file must not be open and must exist on the specified drive. If the drive spec is $FF, all drives will be searched. Once the file has been opened for update, four operations may be performed on it: 1. sequential read, 2. random read, 3. random write, and 4. close file. Note that it is not possible to do sequential writes to a file open for update. This implies that it is not possible to increase the size of a file which is open for update.

Function 4 – Close File

If the file was opened for reading, a close merely removes the FCB from the chain of open files. If the file was opened for writing, any data remaining in the buffer is first written to the disk, padding with zeroes if necessary, to fill out the sector. If a file was opened for writing but never written upon, the name of the file is removed from the diskette directory since the file contains no data.

Example:

```
LDX     #FCB    Point to FCB
LDAA    #4      Setup close code
STAA    Ø,X     Store in FCB
JSR     FMS     Call FMS
BNE     ERROR   Check for errors
File has now been closed.
```

Function 5 – Rewind File

Only files which have been opened for read may be rewound. On exit from FMS, the function code in the FCB is set to zero, anticipating a read operation on the file. If the programmer wishes to rewind a file which is open for writing so that it may now be read, the file must first be closed, then re-opened for reading.

Example:

```
Assuming the file is open for read:
LDX     #FCB    Point to FCB
LDAA    #5      Setup rewind code
STAA    Ø,X     Store in FCB
JSR     FMS     Call FMS
BNE     ERROR   Check for errors
File is now rewound and ready for read
```

Function 6 – Open Directory

This function opens the directory on the diskette for access by a program. The FCB used for this function must not already be open for use by a file. On entry, the only information which must be preset in the FCB is the drive number, no file name is required. The directory entries are read by using the Get Information Record function. The Put Information Record function is used to write a directory entry. The normal Read/Write Next Byte function will not function correctly on an FCB which is opened for directory access. It is not necessary to close an FCB which has been opened for directory access after the directory manipulation is finished. The user should normally not need to access the directory.

Function 7 – Get Information Record

This function should only be issued on an FCB which has been opened with the Open Directory function. Each time the Get information Record function is issued, the next directory entry will be loaded into the Directory Information area of the FCB (see Description of the FCB for details of the format of a directory entry). All directory entries, including deleted and unused entries are read when using this function. After an entry has been read, the FCB is said to "point" to the directory entry just read; the Current Directory Address bytes in the FCB refer to the entry just read. An End-of-File error is returned when the end of the directory is reached.

Example:

```
        To get the 3rd directory entry–
        LDX     #FCB    Point to FCB
        LDAA    DRIVE   Get the drive number
        STAA    3,X     Store in the FCB
        LDAA    #6      Setup open directory code
        STAA    Ø,X     Store in FCB
        JSR     FMS     Call FMS
        BNE     ERROR   Check for errors
        LDAB    #3      Set counter to 3
LOOP    LDAA    #7      Setup get rec code
        STAA    Ø,X     Store in FCB
        JSR     FMS     Call FMS
        BNE     ERROR   Check for errors
        DECB            Decrement the counter
        BNE     LOOP    Repeat til finished
        The 3rd entry is now in the FCB
```

Function 8 – Put Information Record

This function should only be issued on an FCB which has been opened with the Open Directory function. The directory information is copied from the Directory Information portion of the FCB into the directory entry to which the FCB currently points. The directory sector just updated is then re-written automatically on the diskette to ensure that the directory is up-to-date. A user program should normally never have to write into a directory. Careless use of the Put Information Record function can lead to the destruction of data files, necessitating a re-initialization of the diskette.

Function 9 – Read Single Sector

This function is a low-level interface directly to the disk driver which permits the reading of a single sector, to which the Current Position bytes of the FCB point, into the Sector Buffer area of the FCB. This function is normally used internally within FLEX and a user program should never need to use it. The Read/Write Next Byte function should be used instead, whenever possible. On return from FMS, the B-register is zero if no error was detected. If the B-register is non-zero on exit, a non-recoverable error was detected and the B-register contains the hardware status returned by the disk driver, not a FLEX error number. The error code

is not stored in the Error Status byte by this function, nor are any of the pointers in the FCB updated. Extreeme care should be taken when using this function since it does not conform to the usual conventions to which most of the other FLEX functions adhere.

Example:

```
LDX     #FCB     Point ot FCB
LDAA    TRACK    Get track number
STAA    3Ø,X     Set current track
LDAA    SECTOR   Get sector number
STAA    31,X     Set current sector
LDAA    #9       Setup function code
STAA    Ø,X      Store in FCB
JSR     FMS      Call FMS
BNE     ERROR    Check for errors
The sector is now in the FCB
```

Function 1Ø ($ØA hex) – Write Single Sector

This function, like the Read Single Sector function, is a low-level interface directly to the disk driver which permits the writing of a single sector. As such, it requires extreme care in its use. This function is normally used internally by FLEX and a user program should never need to use it. The Read/Write Next Byte function should be used whenever possible. Careless use of the Write Single Sector Function may result in the destruction of data, necessitating the re-initialization of the diskette. The disk address being written is taken from the Current Position bytes of the FCB; the data is taken from the FCB Sector Buffer. On return, the B-register is zero if no error was detected. This function honors the Verify Flag (see Global Variables section for a description of the Verify Flag), and will check the sector after writing it if directed to do so by the Verify Flag. If the B-register is non-zero on exit, an unrecoverable error was detected, and the B-register contains the hardware status returned by the driver, not A FLEX error number. The error status is not stored in the Error Status byte of the FCB, nor are any of the pointers in the FCB updated.

Function 11 ($ØB hex) – Reserved for future system use

Function 12 ($ØC hex) – Delete File

This function deletes the file whose specification is in the FCB (drive numbers, file name, and extension). The sectors used by the file are released to the system for re-use. The file should not be open when this function is issued. The file specification in the FCB is altered during the delete process.

Example:

```
LDX     #FCB     Point to FCB
[ setup file spec in FCB ]
LDAA    #12      Setup function code
STAA    Ø,X      Store in FCB
JSR     FMS      Call FMS
BNE     ERROR    Check errors
File has now been deleted.
```

Function 13 ($ØD hex) – Rename File

On entry, the file must not be open, the old name must be in the File Specification area of the FCB, and the new name and extension must be in the Scratch Bytes area of the FCB. The file whose specification is in the FCB is renamed to the name and extension stored in the FCB Scratch Bytes area. Both the new name and the new extension must be specified; neither the name nor the extension can be defaulted.

Example:

```
LDX     #FCB    Point to FCB
[ setup both file specs in FCB ]
LDAA    #13     Setup funciton code
STAA    Ø,X     Store in FCB
JSR     FMS     Call FMS
BNE     ERROR   Check for errors
File has been renamed
```

Function 14 ($ØE hex) – Reserved for future system use.

Function 15 ($ØF hex) – Next Sequential Sector

On entry the file should be open for either reading or writing (not update). If the file is open for reading, this function code will cause all of the remaining (yet unread) data bytes in the current sector to be skipped, and the data pointer will be positioned at the first data byte of the next sequential sector of the file. If the file is open for write, this operation will cause the remainder of the current sector to be zero filled and written out to the disk. The next character written to that file will be placed in the first available data location in the next sequential sector. It should be noted that all calls to this function code will be ignored unless at least one byte of data has either been written or read from the current sector.

Function 16 ($1Ø hex) – Open System Information Record

On entry, only the drive number need be specified in the FCB, there is no file name associated with this function. The FCB must not be open for use by a file. This function accesses the System Information Record for the diskette whose drive number is in the FCB. There are no separate functions for reading or changing this sector. All references to the data contained in the System Infomation Record must be made by manipulating the Sector Buffer directly. This function is used internally within FLEX, there should be no need for a user-written program to change the System Information Record. Doing so may result in the destruction of data, necessitating the re-initialization of the diskette. There is no need to close the FCB when finished.

Function 17 ($11 hex) – Get Random Byte From Sector

On entry, the file should be open for reading or update. Also, the desired byte's number should be stored in the Random Index byte of the FCB. This byte number is relative to the beginning of the sector buffer. On exit, the byte whose number is stored in the Random Index is returned to the calling program in the A-register. The Random Index should not be less than 4 since there is no user data in the first four bytes of the sector.

Example:

To read the 54th data byte of the current sector–

```
LDX     #FCB    Point to the FCB
LDAA    #54+4   Set to item + 4
STAA    35,X    Put it in random index
JSR     FMS     Call FMS
BNE     ERROR   Check for errors
Character is now in acc. A
```

Funciton 18 ($12 hex) – Put Random Byte in Sector

The file must be open for update. This function is similar to Get Random Byte except the character in the A accumulator is written into the sector at the data location specified by Random Index of the FCB. The Random Index should not be less than 4 since only system data resides in the first 4 bytes of the sector.

Example:

To write into the 54th data byte of the current sector—

```
LDX     #FCB    Point to the FCB
LDAA    #54+4   Set to item + 4
STAA    35,X    Put it in Random Index
LDAA    #18     Setup Function Code
STAA    Ø,X     Store in FCB
LDAA    CHAR    Get character to be written
JSR     FMS     Call FMS
BNE     ERROR   Check for errors
```

Character has been written

Function 19 ($13 hex) – Reserved for future system use

Function 2Ø ($14 hex) – Find Next Ready Drive

This function is used to find the next online drive which is in the "ready" state. If the Drive Number in the FCB is hex FF, the search for drives will start with drive Ø. If the Drive Number is Ø, 1, or 2, the search will start with drive 1, 2, or 3 respectively. If a ready drive is found, its drive number will be returned in the Drive Number of the FCB and the carry bit will be cleared. If no ready drive is found, the carry bit will be set and error #16 (Drives Not Ready) will be set.

Function 21 ($15 hex) – Position to Record N

This is one of the 2 function codes provided for random file accessing by sector. The desired record number to be accessed should be stored in the FCB location Current Record Number (a 16-bit bianary value). The file must be open for read or update before using this function code. The first data record of a file is record number one. Positioning to record Ø will read in the first sector of the File Sector Map. After a successful Position operation, the first character read with a sequential read will be the first data byte of the specified record. An attempt to position to a nonexistent record will cause an error. For more information on random files, see the section titled 'Random Files'.

Example:

To position to record #6—

```
LDX     #FCB    Point to the FCB
LDAA    #6      Set position
STAA    33,X    Put in FCB
CLR     32,X    Set M. S. B to Ø
LDAA    #21     Setup Function Code
STAA    Ø,X     Store in FCB
JSR     FMS     Call FMS
BNE     ERROR   Check for errors
```

Record ready to be read.

Function 22 ($16 hex) – Backup One Record

This is also used for random file accessing. This function takes the Current Record Number in the FCB and decrements it by one. A Position to the new record is performed. This has the effect of back spacing one full record. For example, if the Current Record Number is 16 and the Backup One Record function is performed, the file would be positioned to read the first byte of record #15. The file must be open for read or update before this function may be used. See 'Random Files' section for more details.

The FCB that was too large

The specification for a file already

A file was requested on an FCB non-
not exist in the diskette directory.

Random Files

The 8 inch version of FLEX supports random files. The random access technique allows access by record number of a file and can reach any specified sector in a file, no matter how large it is, in a maximum of two disk reads. With a small calculation using the number of data bytes in a sector (252), the user may also easily reach the Nth character of a file using the same mechanism.

Not all files may be accessed in a random manner. It is necessary to create the file as a random file. The default creation mode is sequential and is what all of the standard FLEX Utilities work with. The only random file in a standard FLEX system is the ERROR.SYS file. FLEX uses a random access technique when reporting error messages. A file which has been created as a random access file may read either randomly or sequentially. A sequential file may only be read sequentially.

To create a random file, the normal procedure for opening a file for write should be used. Immediately following a successful open, set the File Sector Map location of the FCB to any non-zero value and proceed with the file's creation. It only makes sense to create text type files in the random mode. As the file is built, the system creates a File Sector Map. This File Sector Map (FSM) is a map or directory which tells the system where each record (sector) of the file is located on the disk. The FSM is always two sectors in length and is assigned record number Ø in the file. This implies that a data file requiring 5 sectors for the data will actually be 7 sectors in length. The user has no need for the FSM sectors and they are automatically skipped when opening a file for read. The FMS uses them for the Position and Backup function code operations.

The directory information of a file states whether or not a file is a random file. If the File Sector Map byte is non-zero, the file is random, otherwise it is sequential only. It should be noted that random files can be copied from one disk to another without losing its random properties, but it can not be appended to another file.

FLEX Error Numbers

1 – Illegal FMS Function Code Encountered
FMS was called with a function code in the Function Code byte of the FCB that was too large or illegal.

2 – The Requested File is in Use
An Open for Read, Update, or Write function was issued on an FCB that is already open.

3 – The File Specified Already Exists
a. An Open for Write was issued on an FCB containing the specification for a file already existing in the diskette directory.
b. A Rename function was issued specifying a new name that was the same as the name of a file already existing in the diskette directory.

4 – The Specified File Could Not Be Found
An Open for Read or Update, a Rename, or a Delete function was requested on an FCB containing the file specification for a file which does not exist in the diskette directory.

5 – System Directory Error–Reboot System
Reserved for future system use.

6 – The System Directory Space is Full
This error should never occur since the directory space is self expanding, and can never be filled. Only disk space can be filled (error #7).

7 – All Available Disk Space Has Been Used
All of the available space on the diskette has been used up by files. If this error is returned by FMS, the last character sent to be written to a file did not actually get written.

attempting to [illegible] a sector. DOS has
[illegible] times, without success, before reporting
[illegible] and sector addresses being put in the

[illegible] attempting to write a sector. DOS has
[illegible] the failing sector before reporting the
[illegible] numbers being put in the FDC's A
[illegible] detected by DOS in attempting to

[illegible] write-protected by exposing the
[illegible] the write protect bit set

[illegible] protect bit set and can not be deleted.

[illegible] the open FCB chain, but it was not in the chain.

[illegible] is open.

18 -- System File Sharing Error

a. A Sharing Control was attempted when [illegible] was already closed or open for write access.

b. A [illegible] attempted on [illegible] was closed or open for read access.

19 -- FMS Record Locking Error

The [illegible] from Sector function was issued with a Random Byte number greater than [illegible]

1A -- FMS [illegible] Sharing Violation
Reserved [illegible]

1B -- Illegal File Name Format

A format error was encountered in a file name specification. The name must begin with a letter and contain only letters, digits, hyphens, and/or underscores. Similarly with the extensions. The name can contain [illegible] characters, extension to 3.

[illegible] Reserved

8 – Read Past End of File

A read operation on a file encountered an end-of-file. All of the data in the file has been processed. This error will also be returned when reading a directory with the Get Information Record function when the end of the directory is reached.

9 – Disk File Read Error

A checksum error was encountered by the hardware in attempting to read a sector. DOS has already attempted to re-read the failing sector several times, without success, before reporting the error. This error may also result from illegal track and sector addresses being put in the FCB.

10 – Disk File Write Error

A checksum error was detected by the hardware in attempting to write a sector. DOS has already tried several times, without success, to re-write the failing sector before reporting the error. This error may also result from illegal track and sector numbers being put in the FCB. A write-error status may also be returned if a read error was detected by DOS in attempting to update the diskette directory.

11 – The File or Disk is Write Protected

An attempt was made to write on a diskette which has been write-protected by exposing the write-enable cut in the diskette or to a file which has the write protect bit set.

12 – The File is Protected–File Not Deleted

The file attempted to be deleted has its delete protect bit set and can not be deleted.

13 – Illegal File Control Block Specified

An attempt was made to access an FCB from the open FCB chain, but it was not in the chain.

14 – Illegal Disk Address Encountered

Reserved for future system use

15 – An Illegal Drive Number Was Specified

Reserved for future system use.

16 – Drives Not Ready

The drive does not have a diskette in it or the door is open.

17 – The File is Protected–Access Denied

Reserved for Future system use.

18 – System File Status Error

a. A Read or Rewind was attempted on a file which was closed, or open for write access.

b. A write was attempted on a file which was closed, or open for read access.

19 – FMS Data Index Range Error

The Get Random Byte from Sector function was issued with a Random Byte number greater than 255.

20 – FMS Inactive–Reboot System

Reserved for future system use.

21 – Illegal File Specification

A format error was detected in a file name specification. The name must begin with a letter and contain only letters, digits, hyphens, and/or underscores. Similarly with file extensions. File names are limited to 8 characters, extensions to 3.

22 – System File Close Error

Reserved for future system use.

the file was specified [illegible]

[illegible] correct record. The file is probably

[illegible] computer for a particular
[illegible] timer board installed.

Disk Drivers

[illegible] own disk drivers to interface
[illegible] Neither the vendor nor Technical
[illegible] no do these configurations
[illegible] written drivers.
[illegible] driving the floppy
[illegible] are designed to interface with the
[illegible] SBFDC - SBFDF. All disk functions are ver-
[illegible] bad not handle retries in case of errors
[illegible] should exit with the disk hard-
[illegible] (Issue a TST B before reading
[illegible] produced by the Western Digital 1771
[illegible] simulated if [illegible] is used. All drivers should
[illegible] with the X register [illegible] with a JSR instruction.

[illegible] – Read
[illegible] (X) = [illegible] Buffer Address
(A) = [illegible]
(B) = Sector No. Code

[illegible] sector [illegible] sector numbers is to be read into the Sector Buffer area
[illegible] indicated by FCB.

[illegible] – Write
[illegible] (X) = FCB Sector Buffer Address
(A) = Track Number
(B) = Sector Number

[illegible] of the Sector Buffer area of the indicated FCB is to be written to the sector re-
[illegible] by the track and sector numbers.

[illegible]
(No parameters)

[illegible] just [illegible] to determine if there are CRC errors

23 – Sector Map Overflow–Disk Too Segmented
An attempt was made to create a very large random access file on a disk which is very segmented. All record information could not fit in the 2 sectors of the File Sector Map. Recreating the file on a new diskette will solve the problem.

24 – Non-Existent Record Number Specified
A record number larger than the last record number of the file was specified in a random position access.

25 – Record Number Match Error–File Damaged
The record located by the FSM random search is not the correct record. The file is probably damaged.

26 – Command Syntax Error–Retype Command
The command line just typed has a syntax error contained in it.

27 – That Command is Not Allowed While Printing
The command just entered is not allowed to operate while the system printer spooler is activated.

28 – Wrong Hardware Configuration
This error usually implies insufficient memory installed in the computer for a particular function or trying to use the printer spooler without the hardware timer board installed.

Disk Drivers

The following information is for those users who wish to write their own disk drivers to interface with some other disk configuration than is supplied by the vendor. Nether the vendor nor Technical Systems Consultants is in a position to write disk drivers for other configurations, no do these companies guarantee the proper functioning of FLEX with user-written drivers.

The disk drivers are the interface routines between FLEX and the hardware driving the floppy disks themselves. The drivers released with the FLEX System are designed to interface with the Western Digital 1771 Floppy Disk Formatter/Controller chip.

The disk drivers are located in RAM at addresses $BE8Ø - $BFFF. All disk functions are vectored jumps at the beginning of this area. The disk drivers need not handle retries in case of errors; FLEX will call them as needed. If an error is detected, the routines should exit with the disk hardware status in the B-register and the CPU Z-Condition code bit clear (issue a TST B before returning to accomplish this). FLEX expects status responses as produced by the Western Digital 1771 Controller. These statuses must be simulated if some other controller is used. All drivers should return with the X-register unchanged. All routines are entered with a JSR instruction.

$BE8Ø – Read
Entry - (X) = FCB Sector Buffer Address
(A) = Track Number
(B) = Sector Number
The sector referenced by the track and sector numbers is to be read into the Sector Buffer area of the indicated FCB.

$BE83 – Write
Entry - (X) = FCB Sector Buffer Address
(A) = Track Number
(B) = Sector Number
The content of the Sector Buffer area of the indicated FCB is to be written to the sector referenced by the track and sector numbers.

$BE86 – Verify
Entry - (No parameters)
The sector just written is to be verified to determine if there are CRC errors.

Diskette Initialization

Description of a Directory Sector

$BE89 – Restore

Entry - (X) = FCB Address
Exit - CC, NE, & B=$B if write protected
CS, NE, & B=$F if no drive

A Restore Operation (also known as a Seek to Track ØØ) is to be performed on the drive whose number is in the FCB.

$BE8C – Drive Select

Entry - (X) = FCB Address

The drive whose number is in the FCB is to be selected.

$BE8F – Check Drive Ready

Entry - (X) = FCB Address
Exit - NE & CS if drive not ready
EQ & CC if drive ready

The drive whose number is in the FCB is checked for a ready status after selecting that drive and delaying long enough for the drive motor to come up to speed (approx. 2 seconds).

$BE92 – Quick Check Drive Ready

This routine is the same as Drive Check Ready except the 2 second delay is not done. This assumes the drive motor is already to speed.

Diskette Initialization

The NEWDISK command is used to "initialize" a diskette for use by the FLEX Operating System. The initialization process writes the necessary track and sector addresses in the sectors of a "soft-sectored" diskette such as is used by FLEX. In addition, the initialization process links together all of the sectors on the diskette into a chain of available sectors.

The first track on the diskette, track Ø, is special. None of the sectors on track Ø are available for data files, they are reserved for use by the FLEX system. The first two sectors contain a "boot" program which is loaded by the "D" command of the DISKBUG monitor. The boot program, once loaded, then loads FLEX from the diskette. Another sector on track Ø is the System Information Record. This sector contains the track and sector addresses of the beginning and ending sectors of the chain of free sectors and those available for data files. The rest of track Ø is used for the directory of file names.

After initialization, the free tracks on the diskette have a common format. The first two bytes of each sector contain the track and sector number of the next sector in the chain. The next two bytes are used to store the logical record number of the sector in the file. The remaining 252 bytes are zero. Initially, all record number bytes are zero. When data is stored in a file, the two linkage bytes at the beginning of each sector are modified to point to the next sector in the file, not the next sector in the free chain. The sectors in the diskette director on track Ø also have linkage bytes similar to those in the free chain and data files.

A FLEX diskette is not initialized in the strict IBM standard format. In the standard format, the sectors on the diskette should be physically in the same order as they are logically, i.e. sector 2 should follow sector 1, 3 follow 2, etc. On a FLEX diskette, the sectors are interleaved so that there is time, after having read one sector, to process the data and request the next sector before it has passed under the head. If the sectors are physically adjacent, the processing time must be very short. The interleaving of the sectors allows more time for processing the data. The phenomena of missing a sector because of long processing times is called "missing revolutions", and results in very slow running time for programs. The FLEX format reduces the number of missed revolutions, thus speeding up programs.

Description of a Directory Sector

Each sector in the directory portion of a FLEX diskette contains 1Ø directory entries. Each entry refers to one file on the diskette. In each sector, the first four bytes contain the sector linkage in-

Description of a Data Sector

Description of a Binary File

formation and the next 12 bytes are not used. When reading information from the directory using the FMS Get Information Record function, these 16 bytes are skipped automatically as each sector is read; the user need not be concerned with them.

Each entry in the directory contains the exact same information that is stored in the FCB bytes 4-27. See the description of the File Control Block (FCB) for more details.

A directory entry which has never been used has a zero in the first byte of the file name. A directory entry which has been deleted has the leftmost bit of the name set (i.e. the first byte of the name is negative).

Description of a Data Sector

Every sector on a FLEX diskette (except the two BOOT sectors) has the following format:

Bytes Ø-1 Link to the next sector
Bytes 2-3 File Logical Record Number
Bytes 4-255 Data

If a file occupies more than one sector, the "link to the next sector" portion contains the track and sector numbers, respectively, of the next sector in the file. These bytes are zero in the last sector of a file, indicating that no more data follows (an "end-of-file" condition). The user should never manually change the linkage bytes of a sector. These bytes are automatically managed by FMS. In fact, the user need not be concerned at all with sector linkage information.

Description of a Binary File

A FLEX binary file may contain anything as data; all ASCII characters are allowed. Each binary file is composed of one or more binary records. There may be more than one binary record in a single sector.

A binary record looks as follows: (byte numbers are relative to the start of the record, not the beginning of a sector)

Byte Ø Start of record indicator ($Ø2, the ASCII STX)
Byte 1 Most significant byte of the load address
Byte 2 Least significant byte of the load address
Byte 3 Number of data bytes in the record
Byte 4-n The binary data in the record

The load address portion of a binary record contains the address where the data resided when it was written to the file with the FLEX SAVE command. When the file is loaded for execution or use, it will be put in the same memory areas from which it was SAVED.

A binary file may also contain an optional transfer address record. This record gives the address in memory of the entry point of a binary program. The format of a transfer address record is as follows:

Byte Ø Transfer Address Indicator ($16, ASCII ACK)
Byte 1 Most significant byte of the transfer address
Byte 2 Least significant byte of the transfer address

If a file contains more than one transfer address record (caused by appending binary files which contain transfer addresses), the last one encountered by the load process is the one that is used, the others are ignored.

When reading or writing a binary file through the File Management System from a user program, the calling program must process the record indicator bytes and load addresses itself; FLEX does not supply or process this information for the user.

Writing Utility Commands

[illegible] utility commands are best [illegible] by the use of an assembler. FLEX reserves a block of memory in [illegible] which utilities may be placed. This memory starts at hex location $A100 and extends [illegible] location $B6FF. The system FCB [illegible] location $A642 may also be used in user written [illegible] The remainder of the program's memory [illegible] actual code should reside in this FCB space since [illegible] it would interfere with the loading of the utility if FLEX is using the FCB while loading utilities.

An example will be given to demonstrate some of the conventions and techniques which should be used when writing utilities. The example, which can be found on the following pages, is a simple text file listing utility. Its syntax is:

LIST[,<FILESPEC>]

The default extension on the filespec is TXT. The utility will simply display the contents of a text file on the terminal, line by line.

The following is a description by section of the LIST utility. The first section of the source listing is a series of EQUATES which tell the assembler where the various DOS routines reside in memory. These equates represent the addresses given in this manual for "User Callable DOS System Routines".

The next two sections are also equates, the first to the FMS entry points, and the second references the system FCB. The actual program finally starts with the ORG statement. In this program, we will make use of the Utility Command space located at $A100, therefore, the ORG is to $A100.

One of the conventions which should be observed when writing DOS utilities is to always start the program with a BRA instruction. Following this instruction should be a VN FCB 1 which defines the version number of the utility. The 1 should of course be set to whatever the actual version number is. In this example, the version number is 1. This convention allows the FLEX VERSION Utility to correctly identify the version number of a command.

Description of a Text File

A text file (also called an "ASCII file" or "coded file") contains only printable ASCII characters plus a few special-purpose control characters. There is no "load address" associated with a FLEX text file as there is with FLEX binary files. It is the responsibility of the program which is reading the text file to put the data where it belongs.

The only control character which FLEX recognizes and processes in a FLEX text file are:

$0D (ASCII CR or RETURN)
This character is used to mark the end of a line or record in the file.

$00 (ASCII NULL)
Ignored by FLEX; if encountered in the file, it is not returned to the calling program.

$18 (ASCII CANCEL)
Ignored by FLEX; if encountered in the file, it is not returned to the calling program.

$09 (ASCII HT or HORIZONTAL TAB)
This is a flag character which indicates that a string of spaces has been removed from the file as a space-saving measure. The next byte following the flag character is a count of the number of spaces removed (2-127). The calling program sees neither the flag character nor the count character. The proper number of spaces are returned to the user program as successive characters are requested by the Read Next Byte function. When writing a file, the spaces are automatically deleted as the user program sends them to the File Management System using the Write Next Byte function. The data compression is, therefore, transparent to the calling program. (The above discussion is only valid if the file is open for Text operations. If open for Binary, the compression flag and count get passed exactly as they appear in the file.)

Writing Utility Commands

Utility commands are best prepared by the use of an assembler. FLEX reserves a bock of memory in which medium size utilities may be placed. This memory starts at hex location $A100 and extends through location $A6FF. The system FCB at location $A840 may also be used in user written utilities for either FCB space or temporary storage. No actual code should reside in this FCB space since it would interfere with the loading of the utility (FLEX is using that FCB while loading utilities).

An example will be given to demonstrate some of the conventions and techniques which should be used when writing utilities. The example, which can be found on the following pages, is a simple text file listing utility. It syntax is:

LIST, [(FILE SPEC)]

The default extension on the file spec is TXT. The utility will simply display the contents of a text file on the terminal, line for line.

The following is a section by section description of the LIST utility. The first section of the source listing is a set of EQUATES which tell the assembler where the various DOS routines reside in memory. These equates represent the addresses given in this manual for "User Callable DOS System Routines".

The next two sections are also equates, the first to the FMS entry points, and the second references the system FCB. The actual program finally starts with the ORG statement. In this program, we will make use of the Utility Command space located at $A100, therefore, the ORG is to $A100.

One of the conventions which should be observed when writing DOS utilities is to always start the program with a BRA instruction. Following this instruction should be a 'VN FCB 1' which defines the version number of the utility. The 1 should of course be set to whatever the actual version number is. In this example, the version number is 1. This convention allows the FLEX VERSION Utility to correctly identify the version number of a command.

Moving down the program to the label called 'LIST2', the program needs to retrieve the file specification and get it into the FCB. Pointing X to the FCB, we can make use of the DOS resident subroutine called 'GETFIL' to automatically parse the file spec, check for errors, and set the name in the FCB correctly. If all goes well in GETFIL, the carry should be clear, otherwise there were errors in the file spec and this fact needs reporting. If the carry is set, control is passed to the line with the label 'LIST9'. At this point, the error message is reported and control is returned to FLEX.

If the file spec was correct, and the carry was clear after the return from GETFIL, we want to set a default file name extension of TXT. The DOS subroutine named SETEXT will do exactly that. First it is necessary to put the code for TXT in the A accumulator (the code is 1). X needs to be pointing to the FCB which it still is. The '1' is also put in the FCB for the future open operation. The call is made to SETEXT and the file name is now correctly set up in the FCB. Note that no errors can be generated by a call to SETEXT.

Now that we have the file spec, it is necessary to open the requested file for read. X is still pointing to the FCB so it is not necessary to reset. The FMS Function Code for 'open a file for read' is 1 which was previously put in the FCB location Ø. A call to FMS is now made in an attempt to open the file. Upon return, if the Z-condition code is set, there were no errors. If there was an error, the 'BNE LIST9' will take us to the code to report the error. This section of code is the desired way to handle most FMS caused disk errors. The first thing to do is call the DOS routine RPTERR which will pint the disk error message on the monitor terminal. Next, all open disk files should be closed. This can be easily accomplished by a call to the FMS close entry (FMSCLS). Finally, return control back to DOS by jumping to the WARM START entry. If the file opened successfully, control will be transfered to the line with the label 'LIST4'. At this time is desirable to fetch characters one at a time from the file, printing them on the monitor terminal as they are received. Since line feeds are not stored in text files (carriage returns mark the end of lines, but the next line will follow immediately), each carriage return received from the file is not output as is, but instead a call to the DOS routine 'PCRLF' is made to print a carriage return and a line feed. As each character is received from the file (by a call to FMS at label LIST4), the error status is checked. If an error does occur, control is transferred to 'LIST6'. Since FLEX does not store an End of File character with a file, the only mechanism for determining the end of a file is by the End of File error generated by FMS. At 'LIST6', the error status is checked to see if it is 8 (end of file status). If it is not an 8, control is transferred to the error handling routine described above. If it is an End of File, we are finished listing the file so it must now be closed. The FMS Function Code for closing a file is 4. This is loaded into accumulator A and stored in the FCB. Calling FMS will attempt to close the file. Upon return, errors are checked, and if none found, control is transferred back to DOS by the jump to 'WARMS'.

This example illustrates many of the methods used when writing utilities. Many of the DOS and FMS routines were used. The basic idea of file opening and closing were demonstrated, as well as file I/O. The methods of dealing with various types of errors were also presented. Studying this example until it is thoroughly understood will make writing your own disk commands and disk oriented programs an easy task.

```
                        *
                        * SIMPLE TEXT FILE LIST UTILITY
                        *
                        * COPYRIGHT (C) [illegible] BY
                        * TECHNICAL SYSTEMS CONSULTANTS, INC
                        
                        * DOS EQUATES

AD03                    WARMS   EQU     $AD03     DOS WARMS START ENTRY
AD2D                    GETFIL  EQU     $AD2D     GET FILE SPECIFICATION
AD18                    PUTCHR  EQU     $AD18     PUT CHARACTER ROUTINE
AD24                    PCRLF   EQU     $AD24     PRINT CR & LF
AD33                    SETEXT  EQU     $AD33     SET DEFAULT NAME EXT
AD3F                    RPTERR  EQU     $AD3F     REPORT DISK ERROR

                        * FMS EQUATES

B406                    FMS     EQU     $B406
B403                    FMSCLS  EQU     $B403

                        * SYSTEM EQUATES

A840                    FCB     EQU     $A840     SYSTEM FCB

                        * LIST UTILITY STARTS HERE

A100                            ORG     $A100

A100 20 01              LIST    BRA     LIST2     GET AROUND TEMPS

A102 01                 VN      FCB     1         VERSION NUMBER

A103 CE A8 40           LIST2   LDX     #FCB      POINT TO FCB
A106 BD AD 2D                   JSR     GETFIL    GET FILE SPEC
A109 25 34                      BCS     LIST9     ANY ERRORS?
A10B 86 01                      LDA A   #1        SET UP CODE
A10D A7 00                      STA A   0,X       SAVE FOR READ OPEN
A10F BD AD 33                   JSR     SETEXT    SET TXT EXTENSION
A112 BD B4 06                   JSR     FMS       CALL FMS - DO OPEN
A115 26 22                      BNE     LIST8     CHECK FOR ERROR
A117 CE A8 40           LIST4   LDX     #FCB      POINT TO FCB
A11A BD B4 06                   JSR     FMS       CALL FMS - GET CHAR
A11D 26 0E                      BNE     LIST6     ERRORS?
A11F 81 0D                      CMP A   #$D       IS CHAR A CR?
A121 26 05                      BNE     LIST5
A123 BD AD 24                   JSR     PCRLF     OUTPUT CR & LF
A126 20 EF                      BRA     LIST4     REPEAT
A128 BD AD 18           LIST5   JSR     PUTCHR    OUTPUT THE CHARACTER
A12B 20 EA                      BRA     LIST4     REPEAT SEQUENCE
```

```
                        *
                        * SIMPLE TEXT FILE LIST UTILITY
                        *

                        * COPYRIGHT (C) 1978 BY
                        *
                        * TECHNICAL SYSTEMS CONSULTANTS, INC.

                        * DOS EQUATES

AD03                    WARMS   EQU     $AD03       DOS WARMS START ENTRY
AD2D                    GETFIL  EQU     $AD2D       GET FILE SPECIFICATION
AD18                    PUTCHR  EQU     $AD18       PUT CHARACTER ROUTINE
AD24                    PCRLF   EQU     $AD24       PRINT CR & LF
AD33                    SETEXT  EQU     $AD33       SET DEFAULT NAME EXT
AD3F                    RPTERR  EQU     $AD3F       REPORT DISK ERROR

                        * FMS EQUATES

B406                    FMS     EQU     $B406
B403                    FMSCLS  EQU     $B403

                        * SYSTEM EQUATES

A840                    FCB     EQU     $A840       SYSTEM FCB

                        * LIST UTILITY STARTS HERE

A100                            ORG     $A100

A100 20 01              LIST    BRA     LIST2       GET AROUND TEMPS

A102 01                 VN      FCB     1           VERSION NUMBER

A103 CE A8 40           LIST2   LDX     #FCB        POINT TO FCB
A106 BD AD 2D                   JSR     GETFIL      GET FILE SPEC
A109 25 34                      BCS     LIST9       ANY ERRORS?
A10B 86 01                      LDA A   #1          SET UP CODE
A10D A7 00                      STA A   0,X         SAVE FOR READ OPEN
A10F BD AD 33                   JSR     SETEXT      SET TXT EXTENSION
A112 BD B4 06                   JSR     FMS         CALL FMS - DO OPEN
A115 26 28                      BNE     LIST9       CHECK FOR ERROR
A117 CE A8 40           LIST4   LDX     #FCB        POINT TO FCB
A11A BD B4 06                   JSR     FMS         CALL FMS - GET CHAR
A11D 26 0E                      BNE     LIST6       ERRORS?
A11F 81 0D                      CMP A   #$D         IS CHAR A CR?
A121 26 05                      BNE     LIST5
A123 BD AD 24                   JSR     PCRLF       OUTPUT CR & LF
A126 20 EF                      BRA     LIST4       REPEAT
A128 BD AD 18           LIST5   JSR     PUTCHR      OUTPUT THE CHARACTER
A12B 20 EA                      BRA     LIST4       REPEAT SEQUENCE
```

```
A12D A6 01      LIST6   LDA A   1,X         GET ERROR STATUS
A12F 81 08              CMP A   #8          IS IT EOF ERROR?
A131 26 0C              BNE     LIST9
A133 86 04              LDA A   #4          CLOSE FILE CODE
A135 A7 00              STA A   0,X         STORE IN FCB
A137 BD B4 06           JSR     FMS         CALL FMS - CLOSE FILE
A13A 26 03              BNE     LIST9       ERRORS?
A13C 7E AD 03           JMP     WARMS       RETURN TO FLEX

A13F BD AD 3F   LIST9   JSR     RPTERR      REPORT ERROR
A142 BD B4 03           JSR     FMSCLS      CLOSE ALL FILES
A145 7E AD 03           JMP     WARMS       RETURN TO FLEX

                        END     LIST
```

The DOS LINK Utility

The LINK Utility provided with FLEX is a special purpose command, its only function is to inform the "disk boot" which track & sector the program resides which is to be loaded during the boot operation. Normally, LINK is used to set the pointer to the DOS program. Since DOS may reside anywhere on the disk, LINK takes the starting disk address of the file and stores it in a pointer in the boot sector. When the boot program is to be executed, it simply takes this address, and loads the binary file which resides at that location. The load process is terminated upon the receipt of a transfer address record. At this time, control is transferred to the program just loaded by jumping to the address specified in the transfer address record. If the "linked" program is ever moved on the disk, then it must be re-linked so the boot knows the new disk address.

LINK may be used to store specialized applications. One is the development of custom operating systems. The user may write his own operating system, link it to the boot, and use it exactly as FLEX is used now. It may also be desirable for special utilities to boot in specialized programs rather than the operating system. In this case, the user must be sure that unless the DOS is loaded during the boot process, there will not be any disk drivers or file management system resident in memory.

Printer Routines

There are two routines to load programs provided with FLEX. One is the P Utility, the other is the PRINTSYS file which is the actual set of printer drivers. Initialize printer and output characters. The P routine and a short listing is provided on the following pages and should be self explanatory. Below you will find the requirements of the PRINT.SYS file. No source listing is provided here since one is given in the "FLEX User's Manual."

PRINT.SYS File Requirements

The PRINT.SYS file needs to provide the system with three basic printer routines, one for printer port initialization, one for printer status, and one for output character to printer routine. The P routine and the system printer should use these routines to communicate with the printer. A source listing of the provided routine is contained in the "FLEX User's Manual" and will not be duplicated here. The three routines and their requirements are listed here:

PINIT (SCOD ACDF) This routine should initialize the printer port. No registers need to be preserved.

PSTAT (SACD8-ACDA) This routine should check to see if the printer can accept another character. Return Negative CC status if can accept. Plus if can't. Preserve A, B, and X.

POUT (SACDA-ACDF) This routine should output the character in A after calling PCHK to verify the printer can accept the character. Preserve B and X.

The System Printer Spooler

FLEX contains a printer spooler module. It requires the installation of an SWTPC MP-T interrupt timer board for operation. Essentially, the spooler is a multi-tasking system, with the output to printer function being a low priority task. Any required disk service will cause the printer task to temporarily halt until the disk has been used. It should be noted that the SWI CPU vector is adjusted in this task scheduler. The PRINT command is used to activate the spooler which in turn prints the files (if any) in the print queue. Exact details of the spooling operation are not available for publication at this time.

The DOS LINK Utility

The LINK Utility provided with FLEX is a special purpose command. Its only function is to inform the "disk boot", which is on track Ø, where the program resides which is to be loaded during the boot operation. Normally, LINK is used to set the pointer to the DOS program. Since DOS may reside anywhere on the disk, LINK takes the starting disk address of the file and stores it in a pointer in the boot sector. When the boot program is later exectued, it simply takes this disk address, and loads the binary file which resides at that location. The load process is terminated upon the receipt of a transfer address record. At this time, control is transferred to the program just loaded by jumping to the address specified in the transfer address record. If the 'linked' program is ever moved on the disk, then it must be re-linked so the boot knows the new disk address.

LINK may be used in some specialized applications. One is the development of custom operating systems. The user may write his own operating system, link it to the boot, and use it exactly as FLEX is used now. It may also be desirable for special disks to boot in specialized programs rather than the operating system. If this is done, remember that unless the DOS is loaded during the boot process, there will not be any disk drivers or File Management System resident in memory.

Printer Routines

There are two printer related programs provided with FLEX. One is the P Utility, the other is the PRINT.SYS file which is the actual set of printer drivers (initialize printer and output character). The P command source listing is provided on the following pages and should be self explanatory. Below you will find the requirements of the PRINT.SYS file. No source listing is provided here since one is given in the "FLEX User's Manual."

PRINT.SYS File Requirements

The PRINT.SYS file needs to provide the system with three basic printer routines, one for printer port initialization, one for printer status, and one for output character to printer routine. The P routine and the system printer spooler use these routines to communicate with the printer. A source listing of the provided routines are included in the "FLEX User's Manual" and will not be duplicated here. The three routines and their requirements are listed here.

PINIT ($ACCØ-ACD7) This routine should initialize the printer port. No registers need to be preserved.

PCHK ($ACD8-ACE3) This routine should check to see if the printer can accept another character. Return Negative CC status if can accept, Plus if can not. Preserve A, B, and X.

POUT ($ACE4-ACF7) This routine should output the character in A after calling PCHK to verify the printer can accept the character. Preserve B and X.

The System Printer Spooler

FLEX contains a printer spooler module. It requires the installation of an SWTPC MP-T interrupt timer board for operation. Essentially, the spooler is a multi-tasking system, with the output to printer function being a low priority task. Any required disk service will cause the printer task to temporarily halt until the disk has been used. It should be noted that the SWI CPU vector is adjusted in this task scheduler. The PRINT command is used to activate the spooler which in turn prints the files (if any) in the print queue. Exact deatails of the spooling operation are not available for publication at this time.

```
*
* P - UTILITY COMMAND
*
* THE P COMMAND INITIALIZES THE PORT AND
* CHANGES THE OUTCH JUMP VECTOR IN FLEX
*
* COPYRIGHT (C) 1979 BY
* TECHNICAL SYSTEMS CONSULTANTS, INC.
* EQUATES

[illegible]          INDEX   EQU   [illegible]
A840          FCB     EQU   $A840
AD23          LOAD    EQU   $AD23
B406          FMS     EQU   $B406
B403          FMSCLS  EQU   $B403
[illegible]   RENTER  EQU   [illegible]
[illegible]   NFERR   EQU   $4
AC09          PAUSE   EQU   $AC09
AD1E          PSTRNG  EQU   $AD1E
AD3F          RPTERR  EQU   $AD3F
AD03          WARMS   EQU   $AD03
AC11          LSTTRM  EQU   $AC11
AC02          EOL     EQU   $AC02
ACC0          PINIT   EQU   $ACC0
ACE4          POUT    EQU   $ACE4
AD0F          OUTCH   EQU   $AD0F
ACFC          PR1     EQU   $ACFC

                      ORG   $A100

A100 20 01    PLINIT  BRA   PL1         BRANCH AROUND TEMPS

A102 01       VN      FCB   1           VERSION NUMBER

A103 B6 AC FC PL1     LDA A PR1         CHECK SYSTEM PROCESS REG
A106 27 05            BEQ   PL2         IS IT BUSY?
A108 CE A8 40         LDX   #FCB        POINT TO FCB
A10B C6 1B            LDA B #27         SET BUSY ERROR
A10D E7 01            STA B 1,X         STUFF IN FCB
A10F 20 43            BRA   P3          GO REPORT ERROR
A111 B6 AC 11 PL2     LDA B LSTTRM      GET LIST TERMINATOR
A114 81 0D            CMP A #$D         IS IT A CR?
A116 27 45            BEQ   P8
A118 B1 AC 02         CMP A EOL         IS IT EOL CHARACTER?
A11B 27 40            BEQ   P8
A11D 7F AC 09         CLR   PAUSE       DISABLE THE PAUSE FEATURE
```

- continued -

```
                    *
                    * "P" UTILITY COMMAND
                    *
                    * THE P COMMAND INITIALIZES A PORT AND
                    * CHANGES THE OUTCH JUMP VECTOR IN FLEX
                    *

                    * COPYRIGHT (C) 1978 BY
                    *
                    * TECHNICAL SYSTEMS CONSULTANTS, INC.

                    * EQUATES

0010                INDEX   EQU     $0010
A840                FCB     EQU     $A840
AD30                LOAD    EQU     $AD30
B406                FMS     EQU     $B406
B403                FMSCLS  EQU     $B403
AD06                RENTER  EQU     $AD06
0004                NFER    EQU     $4
AC09                PAUSE   EQU     $AC09
AD1E                PSTRNG  EQU     $AD1E
AD3F                RPTERR  EQU     $AD3F
AD03                WARMS   EQU     $AD03
AC11                LSTTRM  EQU     $AC11
AC02                EOL     EQU     $AC02
ACC0                PINIT   EQU     $ACC0
ACE4                POUT    EQU     $ACE4
AD0F                OUTCH   EQU     $AD0F
ACFC                PR1     EQU     $ACFC

A100                        ORG     $A100

A100 20 01          P       BRA     P1          BRANCH AROUND TEMPS

A102 01             VN      FCB     1           VERSION NUMBER

A103 B6 AC FC       P1      LDA A   PR1         CHECK SYSTEM PROCESS REG
A106 27 09                  BEQ     P12         IS IT BUSY?
A108 CE A8 40               LDX     #FCB        POINT TO FCB
A10B C6 1B                  LDA B   #27         SET BUSY ERROR
A10D E7 01                  STA B   1,X         STUFF IN FCB
A10F 20 43                  BRA     P3          GO REPORT ERROR
A111 B6 AC 11       P12     LDA A   LSTTRM      GET LAST TERMINATOR
A114 81 0D                  CMP A   #$D         IS IT A CR?
A116 27 45                  BEQ     P8
A118 B1 AC 02               CMP A   EOL         IS IT EOL CHARACTER?
A11B 27 40                  BEQ     P8
A11D 7F AC 09               CLR     PAUSE       DISABLE THE PAUSE FEATURE
```

- continued -

```
A120 B6 AC E4          LDA A POUT      GET 1ST BYTE OF SPACE
A123 81 39             CMP A #$39      IS IT RTS?
A125 26 13             BNE   P15       IF NOT - THEN LOADED
A127 CE A8 40          LDX   #FCB      POINT TO FCB
A12A 86 01             LDA A #1        OPEN FILE FOR READ
A12C A7 00             STA A 0,X
A12E BD B4 06          JSR   FMS       CALL FMS
A131 26 13             BNE   P2        CHECK FOR ERRORS
A133 86 FF             LDA A #$FF      SET FOR BINARY READ
A135 A7 3B             STA A 59,X      SET COMPRESSION FLAG
A137 BD AD 30          JSR   LOAD      CALL FLEX'S LOADER
A13A BD AC 00     P15  JSR   PINIT     GO INITIALIZE PORT
A13D FE AC E4          LDX   POUT      GET OUTPUT ADDRESS
A140 FF AD 10          STX   OUTCH+1   STUFF IN FLEX
A143 7E AD 06          JMP   RENTER    RETURN TO FLEX

A146 A6 01        P2   LDA A 1,X       GET ERROR CODE
A148 81 04             CMP A #NFERR    IS IT "NO SUCH FILE"?
A14A 26 08             BNE   P3
A14C CE A1 62          LDX   #NOPST    POINT TO MESSAGE
A14F BD AD 1E     P22  JSR   PSTRNG    GO PRINT IT
A152 20 03             BRA   P4

A154 BD AD 3F     P3   JSR   RPTERR    REPORT ERROR
A157 BD B4 03     P4   JSR   FMSCLS    CLOSE ALL FILES
A15A 7E AD 03          JMP   WARMS     RETURN TO FLEX

A15D CE A1 76     P9   LDX   #ERSTG    POINT TO STRING
A160 20 ED             BRA   P22       GO PRINT IT

A162 50 52 49     NOPST FCC  /PRINT.SYS NOT FOUND/
A175 04                FCB   4
A176 27 50 27     ERSTG FCC  /'P' MUST BE FOLLOWED BY A COMMAND/
A197 04                FCB   4

                  * THE FOLLOWING CODE IS LOADED INTO
                  * THE SYSTEM FCB WHEN THE P COMMAND IS
                  * LOADED INTO MEMORY
                  * IT PRESETS THE FILE NAME IN THE FCB

A843                   ORG   $A843

A843 FF                FCB   $FF
A844 50                FCC   'PRINT'
A849 00                FCB   0,0,0
A84C 53                FCC   'SYS'

                       END   P
```

```
A120 B6 AC E4          LDA A  POUT      GET 1ST BYTE OF SPACE
A123 81 39             CMP A  #$39      IS IT RTS?
A125 26 13             BNE    P15       IF NOT - THEN LOADED
A127 CE A8 40          LDX    #FCB      POINT TO FCB
A12A 86 01             LDA A  #1        OPEN FILE FOR READ
A12C A7 00             STA A  0,X
A12E BD B4 06          JSR    FMS       CALL FMS
A131 26 13             BNE    P2        CHECK FOR ERRORS
A133 86 FF             LDA A  #$FF      SET FOR BINARY READ
A135 A7 3B             STA A  $3B,X     SET COMPRESSION FLAG
A137 BD AD 30          JSR    LOAD      CALL FLEX'S LOADER
A13A BD AC C0  P15     JSR    PINIT     GO INITIALIZE PORT
A13D CE AC E4          LDX    #POUT     GET OUTPUT ADDRESS
A140 FF AD 10          STX    OUTCH+1   STUFF IN FLEX
A143 7E AD 06          JMP    RENTER    RETURN TO FLEX

A146 A6 01     P2      LDA A  1,X       GET ERROR CODE
A148 81 04             CMP A  #NFER     IS IT "NO SUCH FILE"?
A14A 26 08             BNE    P3
A14C CE A1 62          LDX    #NOPST    POINT TO MESSAGE
A14F BD AD 1E  P25     JSR    PSTRNG    GO PRINT IT
A152 20 03             BRA    P4

A154 BD AD 3F  P3      JSR    RPTERR    REPORT ERROR
A157 BD B4 03  P4      JSR    FMSCLS    CLOSE ALL FILES
A15A 7E AD 03          JMP    WARMS     RETURN TO FLEX

A15D CE A1 78  P8      LDX    #ERSTR    POINT TO STRING
A160 20 ED             BRA    P25       GO PRINT IT

A162 22        NOPST   FCC    '"PRINT SYS" NOT FOUND'
A177 04                FCB    4
A178 22        ERSTR   FCC    '"P" MUST BE FOLLOWED BY A COMMAND'
A199 04                FCB    4

               * THE FOLLOWING CODE IS LOADED INTO
               * THE SYSTEM FCB WHEN THE P COMMAND IS
               * LOADED INTO MEMORY.
               * IT PRESETS THE FILE NAME IN THE FCB.

A843                   ORG    $A843

A843 FF                FCB    $FF
A844 50                FCC    'PRINT'
A849 00                FCB    0,0,0
A84C 53                FCC    'SYS'

                       END    P
```

```
A120 B6 AC E4          LDA A  POUT      GET 1ST BYTE OF SPACE
A123 81 39             CMP A  #$39      IS IT RTS?
A125 26 13             BNE    P15       IF NOT - THEN LOADED
A127 CE A8 40          LDX    #FCB      POINT TO FCB
A12A 86 01             LDA A  #1        OPEN FILE FOR READ
A12C A7 00             STA A  0,X
A12E BD B4 06          JSR    FMS       CALL FMS
A131 26 13             BNE    P2        CHECK FOR ERRORS
A133 86 FF             LDA A  #$FF      SET FOR BINARY READ
A135 A7 3B             STA A  $3B,X     SET COMPRESSION FLAG
A137 BD AD 30          JSR    LOAD      CALL FLEX'S LOADER
A13A BD AC C0  P15     JSR    PINIT     GO INITIALIZE PORT
A13D CE AC E4          LDX    #POUT     GET OUTPUT ADDRESS
A140 FF AD 10          STX    OUTCH+1   STUFF IN FLEX
A143 7E AD 06          JMP    RENTER    RETURN TO FLEX

A146 A6 01     P2      LDA A  1,X       GET ERROR CODE
A148 81 04             CMP A  #NFER     IS IT "NO SUCH FILE"?
A14A 26 08             BNE    P3
A14C CE A1 62          LDX    #NOPST    POINT TO MESSAGE
A14F BD AD 1E  P25     JSR    PSTRNG    GO PRINT IT
A152 20 03             BRA    P4

A154 BD AD 3F  P3      JSR    RPTERR    REPORT ERROR
A157 BD B4 03  P4      JSR    FMSCLS    CLOSE ALL FILES
A15A 7E AD 03          JMP    WARMS     RETURN TO FLEX

A15D CE A1 78  P8      LDX    #ERSTR    POINT TO STRING
A160 20 ED             BRA    P25       GO PRINT IT

A162 22        NOPST   FCC    '"PRINT SYS" NOT FOUND'
A177 04                FCB    4
A178 22        ERSTR   FCC    '"P" MUST BE FOLLOWED BY A COMMAND'
A199 04                FCB    4

               * THE FOLLOWING CODE IS LOADED INTO
               * THE SYSTEM FCB WHEN THE P COMMAND IS
               * LOADED INTO MEMORY.
               * IT PRESETS THE FILE NAME IN THE FCB.

A843                   ORG    $A843

A843 FF                FCB    $FF
A844 50                FCC    'PRINT'
A849 00                FCB    0,0,0
A84C 53                FCC    'SYS'

                       END    P
```

FLEX makes extensive use of interrupts during printer spooling. Any time there are files in the PRINT Queue (as a result of using the PRINT command) the Timer board (MP T in I/O slot 4) is activated. The board is programmed to output interrupts every 2.4 milliseconds. These are IRQ type interrupts and FLEX vectors to the IRQ routine. When the PRINT Queue is empty, the timer is shut off and no interrupts are generated. The SWI instruction is also used extensively in FLEX. The SWI vector in the DISKSUB scratchpad RAM is set by FLEX to point to its SWI routine. Because of the SWI used by the LOAD command will not permit leaving FLEX while there is a file in the PRINT Queue.

All FLEX routines, the Editor, Assembler, Text Processor, the BASIC are all interruptable processes. When writing programs it may be necessary to use this printing utilities in the printout queue. They should be written to be interrupted as well. At no time should the IRQ or SWI vectors be changed in a way which is to be run while printing. A general good programming practice will yield trouble free use.

```
A120 B6 AC E4          LDA A  POUT      GET 1ST BYTE OF SPACE
A123 81 39             CMP A  #$39      IS IT RTS?
A125 26 13             BNE    P15       IF NOT - THEN LOADED
A127 CE A8 40          LDX    #FCB      POINT TO FCB
A12A 86 01             LDA A  #1        OPEN FILE FOR READ
A12C A7 00             STA A  0,X
A12E BD B4 06          JSR    FMS       CALL FMS
A131 26 13             BNE    P2        CHECK FOR ERRORS
A133 86 FF             LDA A  #$FF      SET FOR BINARY READ
A135 A7 3B             STA A  $3B,X     SET COMPRESSION FLAG
A137 BD AD 30          JSR    LOAD      CALL FLEX'S LOADER
A13A BD AC C0  P15     JSR    PINIT     GO INITIALIZE PORT
A13D CE AC E4          LDX    #POUT     GET OUTPUT ADDRESS
A140 FF AD 10          STX    OUTCH+1   STUFF IN FLEX
A143 7E AD 06          JMP    RENTER    RETURN TO FLEX

A146 A6 01     P2      LDA A  1,X       GET ERROR CODE
A148 81 04             CMP A  #NFER     IS IT "NO SUCH FILE"?
A14A 26 08             BNE    P3
A14C CE A1 62          LDX    #NOPST    POINT TO MESSAGE
A14F BD AD 1E  P25     JSR    PSTRNG    GO PRINT IT
A152 20 03             BRA    P4

A154 BD AD 3F  P3      JSR    RPTERR    REPORT ERROR
A157 BD B4 03  P4      JSR    FMSCLS    CLOSE ALL FILES
A15A 7E AD 03          JMP    WARMS     RETURN TO FLEX

A15D CE A1 78  P8      LDX    #ERSTR    POINT TO STRING
A160 20 ED             BRA    P25       GO PRINT IT

A162 22        NOPST   FCC    '"PRINT SYS" NOT FOUND'
A177 04                FCB    4
A178 22        ERSTR   FCC    '"P" MUST BE FOLLOWED BY A COMMAND'
A199 04                FCB    4

               * THE FOLLOWING CODE IS LOADED INTO
               * THE SYSTEM FCB WHEN THE P COMMAND IS
               * LOADED INTO MEMORY.
               * IT PRESETS THE FILE NAME IN THE FCB.

A843                   ORG    $A843

A843 FF                FCB    $FF
A844 50                FCC    'PRINT'
A849 00                FCB    0,0,0
A84C 53                FCC    'SYS'

                       END    P
```

FLEX makes extensive use of interrupts during printer operation. Anytime there are files in the PRINT Queue (as a result of using the PRINT command) the Timer board (MPT in I/O slot 4) is activated. The board is instructed to output interrupts every 4 milliseconds. These are IRQ type interrupts and FLEX vectors to the IRQ routine. When the PRINT Queue is empty the timer is shut off and no interrupts are generated. The SWI instruction is also used extensively in FLEX. The SWI vector in the DISKSUB scratchpad RAM is set by FLEX to point to its SWI routine. Because of the SWI used by the UNCMD command will not permit leaving FLEX while there is a file printing PRINT Queue.

All FLEX commands, the Editor, Assembler, User Programs, and BASIC are all interruptable processes. When writing programs it may be necessary to use disk printing files in the printer queue, they should be written to be interruptable as well. At no time should the IRQ or SWI vectors be changed in a user program in order to take out while printing. A special speed programming procedure will yield interruptable programs.

Interrupts In FLEX

FLEX makes extensive use of interrupts during printer spooling. Anytime there are files in the PRINT Queue (as a result of using the PRINT command) the Timer board (MP-T in I/O slot 4) is activated. This board is initialized to output interrupts every 10 milliseconds. These are IRQ type interrupts and FLEX sets the IRQ vector to point to its IRQ routine. When the PRINT Queue is empty, the timer is shut off and no interrupts are generated. The SWI instruction is also used quite extensively in FLEX. The SWI vector in the DISKBUG scratchpad RAM is set by FLEX to point to its SWI routine. Because of the SWI and IRQ use, the MON command will not permit leaving FLEX while there is a file in the PRINT Queue.

All FLEX utilities, the Editor, the Assembler, Text Processor, and BASIC are all interruptable programs. When writing your own programs, if they are to be used while printing (files in the print queue), they should be written to be interruptable as well. At no time should the IRQ or SWI vectors be changed in a utility which is to be run while printing. In general, good programming practice will yield interruptable programs!

FLEX REFERENCE SHEET

FLEX MEMORY LOCATIONS

SA00	SAVSP	Last Push
SA20		TTYGET Input Buffer
SA28		TTYPUT Output Buffer
SA30		TTYSET Terminal Type
SA38		TTYSET Echo Control
SA40		TTYSET Case Control
SA48		TTYSET Tab Control
SA50		TTYSET Line Control
SA58		TTYSET Page Control
SA60		TTYSET Special Character
SA68		TTYSET Special Function
SA70		TTYSET Special Function Control
SA78		TTYSET Special Function Character
SA80		TTYSET Special Function Control
SA88		TTYSET Special Function Character
SA90		TTYSET Special Function Control
SA98		TTYSET Special Function Character
SAA0		TTYSET Special Function Control
SAA8		TTYSET Special Function Character
SAB0		TTYSET Special Function Control
SAB8		TTYSET Special Function Character
SAC0		TTYSET Special Function Control
SAC8		TTYSET Special Function Character
SAD0		TTYSET Special Function Control
SAD8		TTYSET Special Function Character
SAE0		TTYSET Special Function Control
SAE8		TTYSET Special Function Character
SAF0		TTYSET Special Function Control
SAF8		TTYSET Special Function Character
SB00		TTYSET Special Function Control
SB08		TTYSET Special Function Character
SB10		TTYSET Special Function Control
SB18		TTYSET Special Function Character
SB20		TTYSET Special Function Control
SB28		TTYSET Special Function Character
SB30		TTYSET Special Function Control
SB38		TTYSET Special Function Character
SB40		TTYSET Special Function Control
SB48		TTYSET Special Function Character
SB50		TTYSET Special Function Control
SB58		TTYSET Special Function Character
SB60		TTYSET Special Function Control
SB68		TTYSET Special Function Character
SB70		TTYSET Special Function Control
SB78		TTYSET Special Function Character
SB80		TTYSET Special Function Control
SB88		TTYSET Special Function Character
SB90		TTYSET Special Function Control
SB98		TTYSET Special Function Character
SBA0		TTYSET Special Function Control
SBA8		TTYSET Special Function Character
SBB0		TTYSET Special Function Control
SBB8		TTYSET Special Function Character
SBC0		TTYSET Special Function Control
SBC8		TTYSET Special Function Character
SBD0		TTYSET Special Function Control
SBD8		TTYSET Special Function Character
SBE0		TTYSET Special Function Control
SBE8		TTYSET Special Function Character
SBF0		TTYSET Special Function Control
SBF8		TTYSET Special Function Character
SC00		TTYSET Special Function Control
SC08		TTYSET Special Function Character
SC10		TTYSET Special Function Control
SC18		TTYSET Special Function Character
SC20		TTYSET Special Function Control
SC28		TTYSET Special Function Character
SC30		TTYSET Special Function Control
SC38		TTYSET Special Function Character
SC40		TTYSET Special Function Control
SC48		TTYSET Special Function Character
SC50		TTYSET Special Function Control
SC58		TTYSET Special Function Character
SC60		TTYSET Special Function Control
SC68		TTYSET Special Function Character
SC70		TTYSET Special Function Control
SC78		TTYSET Special Function Character
SC80		TTYSET Special Function Control
SC88		TTYSET Special Function Character
SC90		TTYSET Special Function Control
SC98		TTYSET Special Function Character
SCA0		TTYSET Special Function Control
SCA8		TTYSET Special Function Character
SCB0		TTYSET Special Function Control
SCB8		TTYSET Special Function Character
SCC0		TTYSET Special Function Control
SCC8		TTYSET Special Function Character
SCD0		TTYSET Special Function Control
SCD8		TTYSET Special Function Character
SCE0		TTYSET Special Function Control
SCE8		TTYSET Special Function Character
SCF0		TTYSET Special Function Control
SCF8		TTYSET Special Function Character
SD00		TTYSET Special Function Control
SD08		TTYSET Special Function Character
SD10		TTYSET Special Function Control
SD18		TTYSET Special Function Character
SD20		TTYSET Special Function Control
SD28		TTYSET Special Function Character
SD30		TTYSET Special Function Control
SD38		TTYSET Special Function Character
SD40		TTYSET Special Function Control
SD48		TTYSET Special Function Character
SD50		TTYSET Special Function Control
SD58		TTYSET Special Function Character
SD60		TTYSET Special Function Control
SD68		TTYSET Special Function Character
SD70		TTYSET Special Function Control
SD78		TTYSET Special Function Character
SD80		TTYSET Special Function Control
SD88		TTYSET Special Function Character
SD90		TTYSET Special Function Control
SD98		TTYSET Special Function Character
SDA0		TTYSET Special Function Control
SDA8		TTYSET Special Function Character
SDB0		TTYSET Special Function Control
SDB8		TTYSET Special Function Character
SDC0		TTYSET Special Function Control
SDC8		TTYSET Special Function Character
SDD0		TTYSET Special Function Control
SDD8		TTYSET Special Function Character
SDE0		TTYSET Special Function Control
SDE8		TTYSET Special Function Character
SDF0		TTYSET Special Function Control
SDF8		TTYSET Special Function Character
SE00		TTYSET Special Function Control
SE08		TTYSET Special Function Character
SE10		TTYSET Special Function Control
SE18		TTYSET Special Function Character
SE20		TTYSET Special Function Control
SE28		TTYSET Special Function Character
SE30		TTYSET Special Function Control
SE38		TTYSET Special Function Character
SE40		TTYSET Special Function Control
SE48		TTYSET Special Function Character
SE50		TTYSET Special Function Control
SE58		TTYSET Special Function Character
SE60		TTYSET Special Function Control
SE68		TTYSET Special Function Character
SE70		TTYSET Special Function Control
SE78		TTYSET Special Function Character
SE80		TTYSET Special Function Control
SE88		TTYSET Special Function Character
SE90		TTYSET Special Function Control
SE98		TTYSET Special Function Character
SEA0		TTYSET Special Function Control
SEA8		TTYSET Special Function Character
SEB0		TTYSET Special Function Control
SEB8		TTYSET Special Function Character
SEC0		TTYSET Special Function Control
SEC8		TTYSET Special Function Character
SED0		TTYSET Special Function Control
SED8		TTYSET Special Function Character
SEE0		TTYSET Special Function Control
SEE8		TTYSET Special Function Character
SEF0		TTYSET Special Function Control
SEF8		TTYSET Special Function Character
SF00		TTYSET Special Function Control
SF08		TTYSET Special Function Character
SF10		TTYSET Special Function Control
SF18		TTYSET Special Function Character
SF20		TTYSET Special Function Control
SF28		TTYSET Special Function Character
SF30		TTYSET Special Function Control
SF38		TTYSET Special Function Character
SF40		TTYSET Special Function Control
SF48		TTYSET Special Function Character
SF50		TTYSET Special Function Control
SF58		TTYSET Special Function Character
SF60		TTYSET Special Function Control
SF68		TTYSET Special Function Character
SF70		TTYSET Special Function Control
SF78		TTYSET Special Function Character
SF80		TTYSET Special Function Control
SF88		TTYSET Special Function Character
SF90		TTYSET Special Function Control
SF98		TTYSET Special Function Character
SFA0		TTYSET Special Function Control
SFA8		TTYSET Special Function Character
SFB0		TTYSET Special Function Control
SFB8		TTYSET Special Function Character
SFC0		TTYSET Special Function Control
SFC8		TTYSET Special Function Character
SFD0		TTYSET Special Function Control
SFD8		TTYSET Special Function Character
SFE0		TTYSET Special Function Control
SFE8		TTYSET Special Function Character
SFF0		TTYSET Special Function Control
SFF8		TTYSET Special Function Character

FLEX SUBROUTINES

SA00	GSA DS	(Goldstar entry)
SA03	SAVSP&S	(Save Stack entry)
SA06	.SYSTEM	(Issue Dos command)
SA09	.ECHO	(Echo character)
SA0C	.RCHR	
SA0F	.PUT CH	(Output character)
SA12	.GETCH2	
SA15	.GETCHR	(Get character)
SA18	.PUTCHR	(Output character)
SA1B	.PSTRNG	(Print string)
SA1E	.PCRLF	(Print carriage return)
SA21	.SPACE	(Print a space)
SA24	.PUTBLK	(Print N blanks)
SA27	.INCHR	(Input character)
SA2A	.GETBUF	(Get a line of input)
SA2D	.GETFIL	(Get a file spec)
SA30	.LOAD	(Load a file)
SA33	.SETEXT	(Set extension)
SA36	.ERROR	(Print ERROR # and abort)
SA39	.ADDBYT	(Add A to X)
SA3C	.ADDBOL	(Add A to X with carry)
SA3F	.OUTDEC	(Output A as decimal)
SA42	.OUTHEX	(Output A as hex byte)
SA45	.OUTADR	(Output X as hex word)
SA48	.PDATA	(Print data at X until EOT)
SA4B	.GETCCL	(Get a command line)
SA4E	.GETDEC	(Get a decimal number)
SA51	.GETHEX	(Get a hex number)
SA54	.INDEC	(Get decimal number with echo)
SA57	.INHEX	(Get hex number with echo)
SA5A	.COMND	(Get a command)
SA5D	FMS Initialization	
SA60	FMS Close	
SA63	FMS Call	

FILE CONTROL BLOCK SPECIFICATION

0	FMS COMMAND (Function code)
1	ERROR STATUS
2	ACTIVITY STATUS
3	DRIVE NUMBER
4	FILL NAME
12	EXTENSION
15	FILE ATTRIBUTES
16	(Reserved System Use)
17	STARTING SECTOR ADDRESS
19	ENDING SECTOR ADDRESS
21	FILE SIZE
23	FILE SECTOR MAP INDICATOR
24	(Reserved System Use)
27	FILE CREATION DATE
29	FCB USER NUMBER
30	CURRENT RECORD
33	RANDOM INDEX
36	(Reserved System Use)
37	SECTOR BUFFER ADDRESS
39	FILE ATTRIBUTE DESCRIPTION
40	(Reserved System Use)
41	CURRENT SECTOR
43	SECTOR BUFFER

FUNCTION MAP

FUNCTION (Hex)	FUNCTION
01	.SETFIL
02	.PERINP
03	.PEROUT
04	CLOSE FILE
05	.RENAME
06	.DELETE
07	.GETINFORMATION
08	.SETINFORMATION
09	.READSECTOR
0A	.WRITESECTOR
0B	.READNEXT
0C	.WRITENEXT
0D	.SEEKREC
0E	.READVERIFYNEXT
0F	.WRITEVERIFYNEXT
10	.SEEKMOD
11	.GETFREE
12	.RESERVED
13	.RESERVED
14	.RESERVED
	.RESERVED
	.RESERVED
	.RESERVED

FLEX REFERENCE SHEET

FLEX MEMORY LOCATIONS

$A080 - $A0FF	Line Buffer
$AC00	TTYSET Backspace Character
$AC01	TTYSET Delete Character
$AC02	TTYSET End of Line Character
$AC03	TTYSET Depth Count
$AC04	TTYSET Width Count
$AC05	TTYSET Null Count
$AC06	TTYSET Tab Character
$AC07	TTYSET Backspace Echo Character
$AC08	TTYSET Eject Count
$AC09	TTYSET Pause Control
$AC0A	TTYSET Escape Character
$AC0B	System Drive Number
$AC0C	Working Drive Number
$AC0D	System Scratch; future use
$AC0E - $AC10	System Date Registers
$AC11	Last Terminator
$AC12 - $AC13	User Command Table Address
$AC14 - $AC15	Line Buffer Pointer
$AC16 - $AC17	Escape Return Register
$AC18	Current Character
$AC19	Previous Character
$AC1A	Current Line Number
$AC1B-$AC1C	Loader Address Offset
$AC1D	Transfer Flag
$AC1E-$AC1F	Transfer Address
$AC20	Error Type
$AC21	Special I/O Flag
$AC22	Output Switch
$AC23	Input Switch
$AC24-$AC25	File Output Address
$AC26-$AC27	File Input Address
$AC28	Command Flag
$AC29	Current Output Column
$AC2A	System Scratch
$AC2B-$AC2C	Memory End
$AC2D-$AC2E	Error Name Vector
$AC2F	File Input Echo Flag
$AC30-$AC4D	System Scratch
$AC4E-$ACBF	System Constants
$ACC0-$ACD7	Printer Initialize
$ACD8-$ACE3	Printer Ready Check
$ACE4-$ACF7	Printer Output
$ACF8-$ACFF	System Scratch
$B409-$B40A	FCB Base Pointer
$B40B-$B40C	Current FCB Address
$B435	Verify Flag

FMS COMMANDS

FUNCTION (HEX)	FUNCTION
01	OPEN FOR READ
02	OPEN FOR WRITE
03	OPEN FOR UPDATE
04	CLOSE FILE
05	REWIND FILE
06	OPEN DIRECTORY
07	GET INFORMATION RECORD
08	PUT INFORMATION RECORD
09	READ SINGLE SECTOR
0A	WRITE SINGLE SECTOR
0B	reserved/future use
0C	DELETE FILE
0D	RENAME FILE
0E	reserved/future use
0F	NEXT SEQUENTIAL SECTOR
10	OPEN SYSTEM INFORMATION RECORD
11	GET RANDOM BYTE FROM SECTOR
12	PUT RANDOM BYTE IN SECTOR
13	reserved/future use
14	FIND NEXT READY DRIVE
15	POSITION TO RECORD N
16	BACKUP ONE RECORD

FLEX SUBROUTINES

$AD00	.COLDS (coldstart entry)
$AD03	.WARMS (warm start entry)
$AD06	.RENTER (main loop re-entry)
$AD09	.INCH (input character)
$AD0C	.INCH2
$AD0F	.OUTCH (output character)
$AD12	.OUTCH2
$AD15	.GETCHR (preferred get character)
$AD18	.PUTCHR (preferred output character)
$AD1B	.INBUFF (input to line buffer)
$AD1E	.PSTRNG (print string)
$AD21	.CLASS (classify character)
$AD24	.PCRLF (print C/R, L/F)
$AD27	.NXTCH (next character)
$AD2A	.RSTRIO (restore I/O vectors)
$AD2D	.GETFIL (parse file spec.)
$AD30	.LOAD (file loader)
$AD33	.SETEXT (set extension)
$AD36	.ADDBX (add ACC-B to X)
$AD39	.OUTDEC (output decimal number)
$AD3C	.OUTHEX (output hex number)
$AD3F	.RPTERR (report error)
$AD42	.GETHEX (get hex number)
$AD45	.OUTADR (output hexadecimal address)
$AD48	.INDEC (input decimal number)
$AD4B	.DOCMND (call DOS)
$B400	.FMS Initialization
$B403	.FMS Close
$B406	.FMS Call

FILE CONTROL BLOCK SPECIFICATION

BYTE (DECIMAL)	FUNCTION
0	.FMS COMMAND (function code)
1	.ERROR STATUS
2	.ACTIVITY STATUS
3	.DRIVE NUMBER
4-11	.FILE NAME
12-14	.EXTENSION
15	.FILE ATTRIBUTES
16	.reserved/future use
17-18	.STARTING DISK ADDRESS
19-20	.ENDING DISK ADDRESS
21-22	.FILE SIZE
23	.FILE SECTOR MAP INDICATOR
24	.reserved/future use
25-27	.FILE CREATION DATE
28-29	.FCB LIST POINTER
30-31	.CURRENT POSITION
32-33	.CURRENT RECORD NUMBER
34	.DATA INDEX
35	.RANDOM INDEX
36-46	.NAME WORK BUFFER (internal)
47-49	.CURRENT DIRECTORY ADDRESS
50-52	.FIRST DELETED DIRECTORY POINTER
53-63	.SCRATCH BYTES (for RENAME)
59	.SPACE COMPRESSION FLAG
64-319	.SECTOR BUFFER

DISK BASIC VER. 3.5

USER'S GUIDE

IMPORTANT NOTE

Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Southwest Technical Products Corporation and Technical Systems Consultants will not assume responsibility for any damages incurred or generated by such material. Also, Southwest Technical Products Corporation and Technical Systems Consultants reserve the right to make changes in such material at any time.

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY SAN ANTONIO, TEXAS 78216

SWTPC Disk BASIC

SWTPC Disk BASIC is a complete BASIC interpreter for use in both home and business applications. Features of SWTPC Disk BASIC include nine significant digit binary coded decimal addition, subtraction, multiplication and division, seven digit trigonometric functions and numerous string operations. Disk data files are also supported for manipulating or storing data.

This manual is designed to acquaint the user with the various features of SWTPC Disk BASIC— It is not designed to be a complete course on the BASIC language. This manual also assumes that the user is familiar with the supplied disk operating system (DOS) and its respective user's guide.

Definitions

Before actually describing each BASIC function, several terms need to be defined and manual notation described.

A **command** is a BASIC operation that generally has an immediate effect on the operation of BASIC.

A **statement** is a word or group of words that directs the execution of a BASIC program.

A **function** is a BASIC operation that usually results in a numerical operation or string processing.

A **variable** is a letter, or a letter and a number, that is used to represent a numeric or string value. Variables may be named by any single alphabetic character (A-Z) or any single alphabetic character followed by a number (0-9). Variables of this type represents a numerical value.

Example: A can be equated to 3.44
B1 can be equated to -7.2315 + SIN(3)

A **string variable** is a single letter followed by a $ that is used to represent literal (alphanumeric or text) data.

Example: A$ can be equated to "1234" but not to 1234. (the quotation marks make 1234 a string).
Note: a string may **not** be represented by a letter and a number such as A3$.

When BASIC initializes, the string variable length is set equal to a maximum of 32 characters.

This manual uses the following notation conventions:

/line N/	denotes a BASIC line number such as 0090
/var/	denotes a variable name such as A3
/exp/	denotes a mathematical expression such as 3+5-2
/rel.exp./	denotes a relational expression such as A=5
/string/	denotes a collection of literal alphanumeric characters enclosed by quotation marks such as "TEST1"
X	denotes a variable or expression that has a numerical result
X$	denotes a string variable

Restrictions on Program Lines

The following restrictions are placed on all BASIC program lines:

1.) Every line must have a line number ranging between 1 and 9999. Do not use line # 0.
2.) Line numbers are used by BASIC to order program statements sequentially.
3.) In any program, a line number can be used only once.
4.) A previously entered line may be changed by entering the same line number along with the desired statement(s). Typing a line number followed immediately by a carriage return deletes that line and line number.
5.) Lines need not be entered in numerical order since BASIC will automatically order them in ascending order.
6.) A line may contain no more than 72 characters including blanks.

7.) Blanks, unless within a character string and enclosed by quotation marks, are not processed by BASIC and their use is optional. Numbers can contain no imbedded blanks.
Example:

```
110 LET A=B + (3.5*5E2)
    is equivalent to
110 LETA = B+(3.5*5E2)
```

8.) Multiple statement lines are accepted with a colon (:) used as the separator. BASIC will process the line from left to right.
Example:

```
10 A=3 : B=5 : C=A*B
```

Data Format

The range of numbers that can be represented in this version of BASIC is 1.0E-99 to 9.999999-99E99. E99 represents 10^{99} while E-99 represents 10^{-99}. The E stands for exponent.

There are nine digits of significance in this version of BASIC. Numbers are internally truncated (last digits dropped) to fit this precision.

Numbers may be displayed and entered in three formats: integer, decimal and exponential.

Example: 153 34.52 136 E-2

Transcendental functions (SIN, COS, TAN, ATAN, SQR, LOG and ↑) are all evaluated by a limited infinite series. For these functions accuracy is limited to seven significant digits.

Mathematical Operators

The mathematical operators used in BASIC are as follows:

↑	Exponentiation (raises to a power)
– (unary)	Negate (used for denoting negative numbers
+	Addition and string concatenation
-	Subtraction
*	Multiplication
/	Division

No two mathematical operators may appear in sequence, and no operator is ever assumed. (A++B and (A+2)(B-3) are not valid). Exception: 5+-3 is allowed.

Examples:

A=B↑C	A is evaluated to B raised to the C power
A=B+-5	A is evaluated to B plus a negative 5
A=3/2	A is evaluated to 3 divided by 2

Priority of Operations

BASIC recognizes the priority of operation in the following order:

1. Exponentiation (↑)
2. Negation (-)
3. Multiplication (*) and division (/)
4. Addition (+) and subtraction (-)

A BASIC expression is evaluated from left to right in the above priority sequence unless parenthesis are encountered. The operators within the parenthesis are evaluated first utilizing the above priority structure.

Examples:

```
LET A=2
LET B=3
LEC C=4
```

B↑2 + C/A↑2	gives a result of 10
C↑2 - C/A	gives a result of 14
A * (A+B*2)-22	gives a result of 0
A↑A↑B	gives a result of 64

String Concatenation

Although any one string variable may be a maximum of 32 characters (or whatever the length is set equal to using the STRING= command), strings may be joined up to a maximum of 126 characters for printing. The concatenation symbol is +.

```
Example: A$= (32 char. string)
         B$= (32 char. string)
         PRINT A$+B$ (prints a 64 character string)
   also: A$= "HELLO"
         B$= "JOHN"
         C$= A$ + B$    (C$ still limited to 32 char.)
```

Arrays

Sometimes it is convenient for a variable to represent several values at one time. A variable such as this type can be considered as an array and each element can be accessed independently. In referencing an array variable, the element number in the array must be specified along with the variable name. For example, say we wanted the variable A to represent 4 values. The following program would assign a different value to each element of A.

```
10  DIM A(4)      Dimension A to hold four elements
20  A(1)=1 : A(2)=2 : A(3)=3 : A(4)=4
```

As seen above, a particular element is referenced by a subscript N, such as A(N), where 1 is the first element in the array.

Two dimensional arrays are also accepted by BASIC. Two dimensional arrays are useful when working with data which is easily represented as a matrix.

```
Example:    10  DIM A (3,3)
            20  A(1,1)=1 : A(1,2)=2 : A(1,3)=3
            30  A(2,1)=4 : A(2,2)=5 : A(2,3)=6
            40  A(3,1)=7 : A(3,2)=8 : A(3,3)=9
gives the following matrix:    1  2  3
                               4  5  6
                               7  8  9
```

String variables may also be dimensioned as arrays. (A$(5,2))

If no DIM statement is used to specifically dimension an array, a dimension of either 10 or 10 by 10 is assumed.

Program Preparation and System Operation

At the time that BASIC is executed, BASIC will automatically determine the range of working storage. If you wish to limit the amount of memory BASIC uses, refer to Appendix D of this manual. This is normally not necessary unless external machine language subroutines are being used.

The system is then ready to accept commands or lines of statements. For example the user might enter the following program:

```
150  REM DEMONSTRATION
160  PRINT "ENTER A NUMBER";
170  INPUT A
180  LET P = A*A*3.1415926
185  PRINT
190  PRINT "THE AREA OF A CIRCLE WITH";
200  PRINT "RADIUS"; A; "IS"; P
210  STOP
```

If the user wishes to insert a statement between two others, he need only type a statement number that falls between the other two. For example:

```
183  REM THIS IS INSERTED BETWEEN 180 and 185.
```

If it is desired to replace a statement, a new statement is typed that has the same number as the one to be replaced. For example:

180 P=(A*A)*3.1415926 replaces the previous LET statement.

Each line entered is terminated by a Carriage Return and is not processed by BASIC until this key is depressed. BASIC then positions the print unit to the correct position on the next line.

If a mistake is made during type in before typing the Carriage Return, a BACKSPACE may be used to delete erroneous characters. The backspace character for BASIC is a hexadecimal ASCII Ø8 (Control H). BASIC assumes that the terminal automatically generates a "cursor left" when a control H is entered.

Example:

30 REM THIS IS A TESZ (CTL.H)T

The CTL.H moves the cursor back over the Z so that the result is TEST

If it is desired to remove a complete line that was typed in before typing the Carriage Return, the CANCEL key (hex ASCII 18, control X) may be depressed. This will delete all information that was typed in on the current command or statement line. BASIC will respond with DELETED.

Example:

10 FOR 1 to 10 (CTL.X)
DELETED

PATCH (CTL. X)
DELETED

If the user wishes to execute a program at this point, the RUN command , as described in the command section, should be entered.

Program Abort

If, at any time, it is desired to abort a looping or otherwise malfunctioning program, BASIC has a provision for exiting the program and returning to the command (READY) mode. The abort (break) character for BASIC is a control C, hex ASCII Ø3. Entering one control C will immediately halt the execution of the current BASIC program and will return BASIC to the command mode. During a printout sequence, such as listing a program, typing one ESC (escape) character will cause the current printout to halt. Typing another ESC will cause printout to resume while typing a RETURN will force BASIC back to the command mode.

NOTE:When in the middle of a machine code USERroutine, control C will have no effect. If necessary, the computer's RESET button can be depressed. Resetting the computer's program counter to 0103 before re-entering BASIC will keep the current BASIC program intact.

Commands

It is possible to communicate in BASIC by typing direct commands at the terminal device. Also, certain other statements can be directly executed when they are entered without statement numbers.

Commands have the effect of causing BASIC to take immediate action. A typical BASIC language program, by contrast, is first entered statement by statement into the memory and then executed only when the RUN command is given.

When BASIC is ready to receive commands, the word READY is displayed on the terminal device. After each entry, the system will prompt with a "#".

Commands are typed without statement numbers. After a command has been executed, READY will again be displayed indicating that BASIC is ready for more input—either another command or program statements.

APPEND

The APPEND command causes a program on tape to be loaded into memory. The APPEND command operates the same as the LOAD command except that the current BASIC program is not cleared from memory.

CONT

A CONT (continue) command can be entered after a program has halted from a STOP command or after a program has been aborted with a control C. Between the time that the program has stopped and the time that CONT is entered no changes should be made in the program. The program will then continue with the next statement after the STOP command or wherever the program was when control C was pressed.

DIGITS=X

The DIGITS= command sets the number of digits that will be printed to the **right** of the decimal point when displaying numeric variables. This will truncate (not round) any digits greater than the number printed, and will force "0"s if there aren't enough significant digits to fill the number of positions specified in the "DIGITS =" command.

A DIGITS=0 command resets BASIC to the floating point mode.

The DIGITS= command may also be used as a program statement.

DOS

The DOS command causes computer control to be returned to the DOS operating system.

LINE=X

The LINE= command is used to specify the number of print positions in a line (line length) where X is the desired number of print positions.

Example: LINE=65, LINE=80, LINE=40

Note: Each line is broken up into 16 character "zones". If the print position is within the last 25 percent of the "line" length and a "space" is printed, a C/R L/F will be output. This is so that a number or word will **not** be split up at the end of a print line. If it is desired to inhibit this feature (for precise print control) just set the line length equal to greater than 125% of the desired total print line length. This can be very important when using the TAB command.

The LINE= command can also be used as a program statement.

LIST
LIST (line #)
LIST (line #m)-(line #n)

The LIST command causes the desired lines of the current program to be displayed on the control terminal. The lines are listed in increasing numerical order by line number. A LIST command causes all lines of the current program to be displayed, a LIST (line #) command lists only the line specified and a LIST (line #m)-(line #n) command causes all lines from m to n to be listed.

The LIST command can also be used to output lines to a terminal or printer on another port by entering #N, after LIST (such as LIST #7, 110-130) where N is the desired port number.

Examples: LIST
LIST 30
LIST #3, 30-70

The LIST command can also be used as a program statement.

LOAD (file name)

The LOAD command is used to load, from disk, a previously saved program. LOAD will clear the memory of the current program and load in the desired program. The same rules apply for file names and drive specifications as in DOS.

Example: LOAD COMPUTE

If no extension is given, .BAS is assumed. Also, if you forget to type in the file name and simply type LOAD, BASIC will prompt you for the file name.

MON

The MON command causes computer control to be returned to the computer's monitor.

NEW

The NEW command causes the working storage and all variables and pointers to be reset. The effect of this command is to erase all traces of the previous program from memory. This command also sets LINE equal to 48 and DIGITS equal to Ø (floating point mode).

PORT=X

The PORT=X command defines the computer I/O Port which will serve as the 'Control Port'. "X" can be a constant, variable, or expression.

Example: PORT = 3

Warning—If you define a port without a terminal as the control port, all messages (including the "Ready") will be inputed and outputed from that port. . .therefore, you will **lose** control of your program.

NOTE: PIA ports **require** handshaking. If handshaking is not available, then you must use the PEEK command to examine the PIA registers. Also, BASIC will always accept a break from port 1, therefore never leave port 1 without a terminal connected. Appendix G defines the correct handshaking procedure. Each port # is configured by BASIC for the specified type of interface:

PORT	TYPE OF PORT
Ø	ACIA
1	MODIFIED PIA (CONTROL PORT) or ACIA
2	ACIA
3	ACIA
4	PIA
5	PIA
6	PIA
7	PIA (LINE PRINTER, BY CONVENTION, such as SWTPC PR-40)

The PORT command can also be used as a program statement.

RUN

The RUN command causes the current program resident in memory to begin execution at the first statement number. RUN always begins at the lowest statement number. RUN resets all program parameters and initializes all variables to zero.

SAVE (file name)

The SAVE command causes the current BASIC program in memory to be saved on disk. The same rules apply for file names and drive specifications as in DOS. If no extension is specified, .BAS is assumed. If a file already exists with the chosen name, an error message will be output.

Example: SAVE COMPUTE

If you forget to type in the file name and enter only SAVE, BASIC will prompt for the file name.

STRING=X

The STRING= command sets the maximum allowable length of string variables. The STRING= command may be used as part of a program and must be used before any strings are referenced in a program. X may be any number between 1 and 126. STRING is initially set to 32 characters. The NEW command will not reset the string length to 32.

TAPPEND

The TAPPEND command works the same as the TLOAD command except that the current BASIC buffer area is not cleared before the load starts.

TLOAD

The TLOAD command is used for loading BASIC programs previously saved on cassette and paper tapes. All input/output regarding the TLOAD command will be thru the control or defined port. Appropriate reader on/off commands are automatically generated.

Example: TLOAD

If desired, the input from TLOAD can be channeled thru a port other than the control port by using the TLOAD #N command where N is the desired port number. The same rules apply for port types and handshaking as described in the PORT= command.

NOTE: Both the TLOAD and TSAVE commands assume that the punch/read device is set up to decode automatic reader/punch on/off commands. If your particular unit is not automatic, the reader or punch should be turned on manually before the carriage return is entered after typing the respective TSAVE or TLOAD command.

TRACE ON

The TRACE ON command will cause BASIC to display the line number of the current statement being executed for every line. This can be an important debugging tool.

TRACE OFF

The TRACE OFF command turns off the trace function.

TSAVE

The TSAVE command is used for saving BASIC programs onto cassette or paper tape. All output from the TSAVE command will be thru the control or designated port. Appropriate punch on/off commands are automatically generated for use by the tape storage device.

Example: TSAVE

If desired, the output from TSAVE can be directed to another port by using the TSAVE #N command where N is the desired port number. The same rules apply for port types and handshaking as described in the PORT= command.

NOTE: TSAVE will dump the entire BASIC program to tape—line numbers such as TSAVE 10 - 20 **can not** be entered to transfer only a portion of the program to tape.

A statement, in BASIC, is a word or a group of words that directs the execution of a BASIC program. Statements differ from commands in that they generally do not cause the computer to immediately take action by themselves. Some statements, in fact, must be used with other statements for proper operation.

DATA N1, N2, N3, . . .
READ V1, V2, V3, . . .
RESTORE

The DATA, READ and RESTORE statements are used in conjunction with each other as one of the methods to assign values to variables. Every time a DATA statement is encountered, the values in the argument field are assigned sequentially to the next available position of a data buffer. All DATA statements, no matter where they occur in a program, cause DATA to be combined into **one** list.

READ statements cause values in the data buffer to be accessed sequentially and assigned to the variables named in the READ statement. They start with the first data element from the first data statement, then the second element, to the end of the first data statement, then to the first element of the second data statement, etc., each time a READ command is encountered. If a READ is executed, and the DATA statements are out of data, an error is generated.

Numeric and string data may be intermixed, however it must be called in the appropriate order.

Note: String data need **not** be enclosed within quotes (") as the comma (,) acts as the delimiter. However, if the string contains a (,), then it **must** be delimited by quotes (").

Example:

```
10 DATA 10,20,30,56.7,"TEST,ONE",1.67E30,8, HELLO
20 READ A,B,C,D,E$,F,G5,F$
```

Note: DATA STATEMENTS may be placed anywhere within the program.

Example:

```
110 DATA 1,2,3.5
120 DATA 4.5,7,70
130 DATA 80,81
140 READ B,C,D,E
```

is the equivalent of:

```
10 LET B=1
20 LET C=2
30 LET D=3.5
40 LET E=4.5
```

The RESTORE statement causes the data buffer pointer, which is advanced by the execution of READ statements, to be reset to point to the first position in the data buffer.

Example:

```
110 DATA 1,2,3.5
120 DATA 4.5,7,70
130 DATA 80,81
140 READ B,C
150 RESTORE
160 READ D,E
```

In this example, the variables would be assigned values equal to:

```
100 LET B=1
101 LET C=2
102 LET D=1
103 LET E=2
```

There are also versions of READ and RESTORE which are used for the manipulation of disk data files. These statements are discussed in the Disk Data Files section.

DIM/var/ (exp) or /var/ (exp), /var/(exp) or /var/(exp,exp)

The DIM statement allocates memory space for an array. In this version of BASIC, one or two dimension arrays are allowed and the maximum array size is 255 x 255 elements. All array elements are set to zero by the DIM statement.

If an array is not explicitly defined by a DIM statement, it is assumed to be defined as an array of 10 elements (or 10 x 10 if two elements are used) upon first reference to it in a program.

Caution: The dimension of an array can be detemined only once in a program, implicitly and explicitly. Also only the variables A thru Z (followed by $) may be dimensioned for strings.

Example: DIM A(10), C(R5+8), D(30,A*3), A7(20), C$(30), Z$(5)
but not A6$(5)

The DIM statement can also be used in the direct execution mode.

END

The END statement causes the current BASIC program to stop executing. When an END statement is seen, BASIC will return to the command mode. In this version of BASIC, END may appear more than once and need not appear at all.

FOR /var/ = /exp 1/ TO /exp 2/ STEP /exp/
NEXT /var/

The FOR and NEXT statements are used together for setting up program loops. A loop causes the execution of one or more statements for a specified number of times. The variable in the FOR. . .TO statement is initially set to the value of the first expression (exp1). Subsequently the statements following the FOR are executed. When the NEXT statement is encountered, the values of the named variable is added to the value specified by the STEP expression in the FOR. . . TO statement, and execution is resumed at the statement following the FOR. . .TO. If the addition of the STEP value develops a sum that is **greater than** the TO expression (exp2) or, if STEP is negative, a sum **less than** the TO expression (exp2), the next instruction executed will be the one following the NEXT statement. If no STEP is specified, a value of one is assumed. If the TO value is initially less than the initial value, the FOR. . .NEXT loop will still be executed once.

```
Example:  110 FOR I=.5 TO 10
          120 INPUT X
          130 PRINT I,X,X/5.6
          140 NEXT I
```

Although expressions are permitted for the initial, final, and STEP values in the FOR statement, they will be evaluated only the first time the loop is entered. They are not re-evaluated.

It is not possible to use the same indexed variable in two loops if they are nested.

When the statement after the NEXT statement is executed, the variable is equal to the value that caused the loop to terminate, not the TO value itself. In the above example, I would be equal to 9.5 when the loop terminates.

GOSUB /line #/

A subroutine is a sequence of instructions which perform some task that would have utility in more than one place in a BASIC program. To use such a sequence in more than one place, BASIC provides subroutines and functions.

A subroutine is a program unit that receives control upon execution of a GOSUB statement. Upon completion of the subroutine, control is returned to the statement following the GOSUB by execution of a RETURN statement in the subroutine.

```
Example:  10 A=3
          20 GOSUB 100
          30 PRINT B
          40 END
         100 LET B= SIN(A)
         110 RETURN
```

GOTO /line #/

The GOTO statement directs BASIC to execute the statements on the specified line unconditionally. Program flow continues from the line specified by /line/.

Example: 150 GOTO 270

This statement may be used in the direct execution mode.

IF /relational exp/ THEN /statement n/
IF /relational exp/ THEN /BASIC statement/ (Direct)

The IF statement is used to control the sequence of program statements to be executed, depending on specific conditions. If the /relational expression/ given in the IF is "true", then control is given to the line number declared after the THEN. If the relational expression is "false", program execution continues at the line following the IF statement.

Example: 10 IF 5>2 THEN 100

It is also possible to provide a BASIC statement after the THEN in the IF statement. If this is done and the relational expression is true, the BASIC statement will be executed and the program will continue at the statement or line following the IF statement.

Example: 10 IF 5>2 THEN LET B=7

When evaluating relational expressions, arithmetic operations take precedence in their usual order, and the relational operators are given equal weight and are evaluated last.

Example: 5+6*5>15*2 evaluates to be true

The Relational Operators

=	Equal
<>	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
>=	Greater Than or Equal

Examples:
```
110 IF A<B+3 THEN 160
180 IF A=B+3 THEN PRINT "VALUE A", A
190 IF A=B THEN T1=B
```

NOTE: If an IF test fails on a multiple statement line, the remainder of the line will not be executed.

Example:
```
10 IF 5<2 THEN 100 : PRINT 3
20 END
```
Control will go to line 20 and "3" will not be printed

The relational operators = (equal) and <> (not equal) may also be used on strings.

Example: 110 IF Y$ = "YES" THEN 100

The < (less than) and > (greater than) comparisons may also be used on strings, but **only** when the number of characters in each of the strings being compared is the same. The > and < operators compare strings by evaluating the ASCII value of the characters starting from the first (leftmost) character. When a character in one string is found to be not equal to its respective character in the other string, the greater than or less than operation is made either true or false depending on the ASCII values of these two characters.

Example: IF "AAABA" > "AAAAB" THEN 100

The first non-equal character in the comparison is the B in "AAABA". The > operator then compares this B to the respective character in the other string (an A). Since the ASCII value of B is greater than that of A, the operation evaluates to "yes, greater than".

Example:
```
"A" > "B"       FALSE
"B" > "A"       TRUE
"ABCDE"<"ABCDF" TRUE
"ABC" > "ABCD"  ILLEGAL, LENGTHS NOT EQUAL
"BZZ" > "CZZ"   FALSE
```

INPUT /var/
INPUT /var/, /var/, /var/,. . .
INPUT #N, var
INPUT "/PROMPT/",/var/

The INPUT statement allows users to enter data from the terminal during program execution.

Example: INPUT X - Inputs one numeric value
INPUT X$ - Inputs one string value
INPUT X,Y,Z,B$ - Multiple inputs may be entered, separated by ",". If the expected number of values are not entered, another "?" will be generated.
INPUT "ENTER VALUE",X - Prints the message in quotes, then a "?", and waits for input. It stores the inputed value in X.

When the program comes to an INPUT statement, a question mark is displayed on the terminal device. The user then types in the requested data separated by commas and followed by a carriage return. If insufficient data is given, the system prompts the user with '?'. If no data is entered, or if a non-numeric character is entered, the system prompts "RE-ENTER". However, for string variables a null return will be considered as valid data. Caution: for input A$,B$,C$—a null response would create three null variables. Only constants can be given in response to an INPUT statement.

The INPUT can also be used to issue a prompting message before the question mark appears.

Example: 10 INPUT "INPUT A$", A$
20 PRINT A$
would give the following results
INPUT A$? 66 (user types this 66 in)
66

INPUT may also be used with the #N, directive for input from ports other than the control port.

LET /var/=/exp/

The LET statement is used to assign a value to a variable. The use of the word LET is optional unless you are in the direct mode.

Example: 100 LET B=827
110 LET C=87E2
120 R=(X*Y)/2*A
130 C$="THIS IS TEXT"

The equal sign does not mean equivalence as in ordinary mathematics. It is the replacement operator. It says: replace the value of the variable named on the left with the value of the expression on the right. The expression on the right can be a simple numerical value or an expression composed of numerical values, variables, mathematical operators, and functions.

ON /exp/ GOTO /line (s)/
ON /exp/ GOSUB /line(s)/

This statement transfers control to the line or subroutine as defined by the value of /exp/. The expression will be evaluated, truncated (chopped off after the decimal point) and control then transferred to the nth statement number (where n is the integer value of the expression).

Example: ON N GOTO 110, 300, 500, 900
Means: If N <1You will get an error
If N=1 GOTO 110
If N=2 GOTO 300
If N=3 GOTO 500
If N=4 GOTO 900
If N>4 You will get an error
Example: ON (N+7)*2 GOSUB 1000,2000
(see GOTO and GOSUB for a further description of these statements)

PRINT /var/
PRINT /string/
PRINT /exp/

The PRINT statement directs BASIC to print the value of an expression, a literal value, a simple variable, or a text string on the user's terminal device. The various forms may be combined in the print list by separating them with commas or semicolons. Commas will give zone spacing of print elements, while semicolons will give a single space between elements. If the list is terminated with a semicolon, the line feed/carriage return at the end will be suppressed.

1. PRINT – Skips a line.
2. PRINT A,B,C – Prints the values of A, B, and C, separated into 16 space zones. Use of a ";" in place of the "," would print A, B, and C separated by one space. (**No** space is generated if a string variable.) A C/R, L/F is generated at the end of the line.
3. PRINT "LITERAL STRING" – Prints the characters contained within the quotes.
4. PRINT A,B;"LITERAL"–Prints variable A & B and the word LITERAL.

PRINT may also be used with the #N directive to specify output to another port.

Example:
```
10 PRINT #7, "TEST"
```
Prints TEST on the parallel device (printer, etc.) on port #7.

PRINT may also be used in the direct mode.

REM

The REM, or remark statement, is a non-executable statement which has been provided for the purpose of making program listings more readable. By generous use of REM statements, a complex program may be more easily understood. REM statements are merely reproduced on the program listing, they are not executed. If control is given to a REM statement, it will perform no operation. (It does, however take a finite amount of time to process the REM statement.)

Example:
```
120 REM THE FOLLOWING SUB. CONVERTS
121 REM DECIMAL VALUES TO HEX VALUES
```

RETURN

The GOSUB statement causes control to be passed to the given line number. It is assumed that the given line number is an entry point of a subroutine. The subroutine returns control to the statement following the GOSUB statement with a RETURN statement.

Example:
```
100 X=1
110 GOSUB 200
120 PRINT X
125 X=5.1
130 GOSUB 200
140 PRINT X
150 STOP
200 X=(X+3)*5.32E3
210 RETURN
211 END
```

Subroutines may be nested; that is one subroutine may use GOSUB to call another subroutine which in turn can call another. Subroutine nesting is limited to eight levels.

STOP

A STOP statement can be used within a program to halt execution at a particular place for debugging purposes. A CONT command will then cause the computer to begin execution on the line following the STOP statement.

Example:
```
10 PRINT 5
20 STOP
30 PRINT 6
```
gives the following output:
```
RUN
```

```
5
STOP AT 20
CONT
6
```

FUNCTIONS

Functions are similar to BASIC statements except that they usually relate to mathematical or string processing operations.

ABS (X)

The ABS (X) function returns the "Absolute Value" of X.

```
Example:    ABS (3.44)=3.44
            ABS (-3.44)=3.44
```

ATAN (X)

The ATAN (X) function returns the angle, in radians, whose tangent is X.

ASC (string or string var)

The ASC (string or string variable) function returns the **decimal** ASCII numeric value of the **first** ASCII character within the string. Literals must be enclosed by quotes while string variables are not.

```
Example:    ASC("?")          gives 63
            ASC("A")          gives 65
            ASC("B")          gives 66
            ASC("Z")          gives 90
            ASC("5")          gives 53
            LET B$="5" → >ASC(B$)      gives 53
```

CHR$ (X)

The CHR$ (X) function returns a single character string equivalent to the decimal ASCII numeric value of X. This is the **inverse** of the ASC function.

```
Example:    CHR$(63)    gives a ?
            CHR$(65)    gives a A
            CHR$(66)    gives a B
            CHR$(53)    gives a 5
```

COS(X)

The COS(X) function returns the cosine of the angle X. X must be in radians.

DEF FN/letter/(/variable/)=/exp/

This function allows the user to create his very own functions. The /letter/ is any alphabetic character. This names the function (i.e., you could have, say, three functions named FNA, FNB, and FNC). The /variable/ is a non-subscripted numeric variable. This is essentially a "dummy" variable (or place holder). . .This will be apparent shortly. The "Expression" is any valid expression. Note that the "variable" **must** be enclosed within parenthesis.

For example, study the following sample program:

```
10 DEF FNA(X)=3.14*X↑ 2
20 DATA 5,6,7,0
30 READ X
40 IF X=0 THEN END
50 PRINT FNA(X)
60 GOTO 30

RUN
```

```
78.5
113.04
153.86

READY
```

As you can see, the dummy variables were replaced with the variables you actually wished to use at the time the function was used.

Note: You may **not** define the same function greater than once per program, and a function **must** be defined before it is called.

EXP(X)

The EXP(X) function returns the base of natural logarithms raised to the Xth power (this is the inverse of LOG(X)) and is the equivalent of 2.718282 raised to the Xth power.

INT(X)

The INT(X) function returns the greatest integer **less than** X.

Example: INT(4.354)=4

Now note this one: INT(-4.354)=-5

LEFT$(X$,N)

The LEFT$(X$,N) function returns a string of characters from the leftmost to the Nth character in X$.

```
Example:    X$="ONE,TWO,THREE"
            LET A$=LEFT$(X$,6)
            A$ NOW EQUALS "ONE,TW"
```

LEN(X$)

The LEN(X$) function returns the number of characters contained in string X$.

```
Example:    LEN("TESTING")=7
            LEN("TEST ONE")=8
```

Note: The space **does** count.

Hint: LEN(STR$(X)) = The number of print positions required to print the number X.

LOG(X)

The LOG(X) function returns the natural logarithm of the number X.

MID$(X$,X,Y)

The MID$(X$,X,Y) function returns a string of characters from X$ beginning with the Xth character from the left, and continuing for Y characters from that point. Y is **optional.** If Y is **not** specified, then the string returned is from the Xth character of the string through the end of the string.

```
Example:    X$="ONE,TWO,THREE"
            A$=MID$(X$,3,10)
            A$ NOW EQUALS "E,TWO,THRE"
```

PEEK(X)

The PEEK(X) function returns, in **decimal,** the value contained in **decimal,** not octal, memory location X.

Example: LET A=PEEK(255)

A will now contain the decimal value contained in memory location 255_{10}.

POKE(I,J)

The POKE(I,J) function takes the decimal, not octal, value of J and places it in decimal, not octal, location I. For example, POKE (255, 10) will store a decimal 10 in decimal memory location 255.

Warning: This function can cause system program failure if improperly used.

POS

The POS function returns in decimal, not octal, the current position of the print head or cursor. The first postion (left margin) is position #1.

RIGHT$(X$,N)

The RIGHT$(X$,N) function returns a string of characters from the Nth position to the left of the rightmost character through the rightmost character.

Example:
```
X$="ONE,TWO,THREE"
A$=RIGHT$(X$,9)
AS NOW EQUALS "TWO,THREE"
```

RND AND RND(X)

The RND(X) function produces a set of uniformly distributed pseudo-random numbers. If X (the seed) is Ø, then each time RND(X) is accessed, a different number between Ø and 1 will be returned. If X <> Ø then a specific random number will be returned each time (the same number each time). RND can be called without an argument, in which case it works as if one had used an argument of Ø.

Example:
```
10 LET A=RND
20 LED B=RND(5)
```

If you require random numbers other than between 0 and 1, then:

```
PRINT INT ( (B-A+1)*RND(0)+A)
```

will yield random numbers ranging between A & B.

SGN(X)

The SGN(X) function returns the 'sign' (+,-, or 0) of X. The SGN of a negative number will yield a -1, the SGN of a positive number will yield 1 and the SGN of 0 gives 0.

Example:
```
SGN(4.5)=1
SGN(-4.5)=-1
SGN(0)=0
SGN(-0)=0
```

SIN(X)

The SIN(X) function returns the sine of the angle X. X must be in radians.

SQR(X)

The SQR(X) function returns the square root of X. X must be greater than or equal to 0 (X must be positive).

STR$(X)

The STR$(X) function returns the string value of a numeric value. This is the inverse of the VAL function.

Example:
```
A=34567
LET A$=STR$(A)
A$ NOW EQUALS "34567"
```

TAB(X)

The TAB(X) function will move the print position to the "Xth" position to the right of the left margin. If the print position is **already** to the right of the position specified in the TAB command, **no** spaces will be left and printing (if any) will commence. The first print position (left margin) is position #1.

The TAB function can be used with the PRINT statement to cause data to be printed in exact locations. The argument of TAB may be an expression.

Example:

```
  5   X=3
 10   PRINT TAB(2); X; TAB( );X*X; TAB( ); X*X*S
will print
  3        9        27
```

TAN(X)

The TAN(X) function returns the tangent of the angle X. X must be in radians. (360 degrees= 2π radians π =3.141592654

USER(X)

The USER (X) function is a BASIC function that enables a user to call a special machine language subroutine. The syntax of the USER function is of the form LET /var/=USER (/var.1/) such as LET A = USER(X). The use of the USER function assumes that the programmer is familiar with assembler level programming.

When the USER function is executed in a program, the numeric value of the variable X is stored in a special BCD (binary coded decimal) format in a seven byte series somewhere in the computer's memory. BASIC then keeps track of where this series is stored so that the machine language routine can access it. After storing this series, BASIC then looks at hex memory locations 0067 and 0068. The computer is then instructed to execute a "Jump to Subroutine" to the hex address stored in hex memory locations 0067 and 0068. To avoid accidental misuse of the USER function, 0067 and 0068 will initialize to a location which contains a hex 39, a return from subroutine. Locations 0067 and 0068 can be changed using the POKE function prior to using USER.

After the computer jumps to the location pointed to by 0067 and 0068 it is up to the machine language program to perform its special function or to manipulate the data previously stored in the seven byte BCD series. To find out where this series is located, hex memory locations 005D and 005E should be checked by the machine program. 005D and 005E contain a pointer to the **location** of the seven byte series. They do not contain the actual location of the series.

For example, say that locations 005D, 005E contain the address IDB1 This means that locations IDB1 and IDB2 contain the **address** of the seven byte series. If the series was stored beginning at 242B, then the locations would be set up as follows:

```
005D    1D
005E    B1

IDB1    24
IDB2    2B

242B    Start of seven byte series.
```

The actual number that was stored in the seven byte series is stored in a special BCD format as follows:

for + numbers		for - numbers
BYTE 1	(sign) (Dg)	(sign) ($\overline{Dg}$)
BYTE 2	(D8) (D7)	($\overline{D8}$) ($\overline{D7}$)
BYTE 3	(D6) (D5)	($\overline{D6}$) ($\overline{D5}$)
BYTE 4	(D4) (D3)	($\overline{D4}$) ($\overline{D3}$)
BYTE 5	(D2) (D1)	($\overline{D2}$) ($\overline{D1}$)
BYTE 6	(Exponent in hex)	(Exponent in hex)
BYTE 7	00	00

Where D's are digits and $\overline{D}$'s are the digits complemented.

The sign half-byte denotes whether or not the number is positive or negative. A sign of 0 denotes + while a 9 denotes -. The actual number digits are located in half-bytes D_1 - D_9. The exponent byte tells BASIC where to put the decimal point. Notice that this number is hexadecimal and not BCD.

For example, the number 1234.5678 would be stored as follows:

Byte 1	01
Byte 2	23
Byte 3	45
Byte 4	67
Byte 5	80
Byte 6	04
Byte 7	00

The number is stored as .12345678 with an exponent of 4 which moves the decimal point 4 places to the right giving 1234.5678. The 0 half-byte in byte 1 denotes a positive number.

Now look at the number -1234.5678. Negative numbers are more complicated and must be handled with great care.

Byte 1	98
Byte 2	76
Byte 3	54
Byte 4	32
Byte 5	20
Byte 6	04
Byte 7	00

Notice that the first 9 in byte 1 denotes a negative number and that all digits D_1 - D_9 are complemented. The complement of a digit is defined a 9- (the digit) with the complement of 0 still being 0. In the above example, the digits that were stored were not 12345678 but rather (9-1) (9-2) (9-3) (9-4) (9-5) (9-6) (9-7) and (1+9-8). The **last significant digit not** including any trailing 0's must have 1 added to its complement before storing in the BCD series. In the example -1234.5678 (the same as -1234.56780) the last **significant** digit is 8; therefore, 1 must be added to its complement.

The number -7.20008000 would be stored as:

Byte 1	92	
Byte 2	79	
Byte 3	99	
Byte 4	20	(the last significant digit is 8)
Byte 5	00	
Byte 6	01	
Byte 7	00	

The end of the machine language program should contain a hex 39, a RTS. This will transfer control back to BASIC. BASIC will then assign the numeric value of the number in the seven byte string to the variable A in the example A=USER(X).

VAL(X$)

The VAL(X$) function returns the numeric constant equivalent to the value in X$. This is the **inverse** of the STR$ function.

Example:
```
VAL("12.3")=12.3
VAL("5E4")=5000
VAL("TWO")=GENERATES AN ERROR. "TWO" cannot be
           equaled to a numeric constant.
VAL("-12.3")=-12.3
```

Below is a description of several commands which allow the user to interface with the various files stored on a disk.

CAT (drive number)

The CAT command can be used to display all of the files on a particular drive. All files are listed, not just BASIC files. Only the names are displayed—additional information on a file's length and other disk information may be obtained by using the CAT command of DOS.

```
Examples:    CAT
             CAT 1
```

CHAIN (file name), (optional line number)
CHAIN A$, (option line number)

The CHAIN command can be used to call one BASIC program from another program. CHAIN will **force** the extension .BAS on the file name, even if another is given. If no line number is specified, program execution of the program called will begin on the first line of the program. If specified, execution will begin on the given line.

```
Example:    25 CHAIN MONEY 11Ø
```

In the above example, the current BASIC program would be deleted from memory and the program MONEY.BAS loaded. Execution on line 11Ø of MONEY would then begin.

String variables may also be used in the file specification:

```
Example:    10 A$="MONEY"
            20 CHAIN A$, 11Ø
```

KILL (file name)
KILL A$

The KILL command is used to delete a file from a disk. Care must be executed when using the KILL command since it does not prompt with an "Are you sure" question. The same rules apply for file names and drive specifications as in DOS. The default extension for KILL is .DAT.

```
Example:    KILL COMPUTE.BAS
```

If desired, KILL may be used as a program statement and the file name may be specified as a string variable.

```
Example:    10 A$= "COMPUTE.BAS"
            20 KILL A$
```

RENAME (file 1), (file 2)
RENAME A$, B$

The RENAME command may be used to rename the various files on a disk. The default extension is .DAT.

```
Example:    RENAME TEST, JUNK
```

The above example would rename the file TEST.DAT to the name JUNK.DAT. RENAME may also be used as a program statement and may be used with string variables.

```
Example:    10 A$= "TEST"
            20 B$= "JUNK"
            30 RENAME A$,B $
```

The same rules apply for file names and drive specifications as in DOS.

Disk Data Files

SWTPC Disk BASIC contains the necessary statements to manipulate sequential disk data files. Data files give the user the ability to access large amounts of data on disk whenever necessary. These data files are very useful when working with things such as inventory and payroll data.

Working with disk data files is similar to using the DATA and READ statements described earlier. When beginning a new file, the file must be "opened". This "opening" essentially equates a file number that BASIC can understand to a file name that the file management system of DOS can understand. After a file is opened, the desired data can then be "written" on this file. If no more manipulation is desired, the file is "closed" (line designated file number is disassociated with the file). The file may later be re-opened and the data read from it by a BASIC program.

Below is a description of each of the file commands in BASIC and a sample program showing their use.

OPEN #/file no./, /file name/

The OPEN command prepares a file on disk to be used for either input or output. No actual disk operation takes place when executing the OPEN statement.

BASIC programs essentially refer to files by file number rather than file name. The function of the OPEN statement is to equate a recognizable file number to a given file name. When using the OPEN statement, the /file no./ must be the number assigned to the file and must be from 0 to 9. What you choose for a file number is completely arbitrary, but each file that is open at any one given time must have a unique file number. The /file name/ specification is the name of the file as it appears on the disk. The same rules apply for /file name/ as do in DOS.

```
Example:    10 OPEN #1, DATA.DAT
            OPEN #1, DATA.DAT, #2, JUNK
```

If no extension is given on the file name, the extension .TXT is assumed.

Note: Each file number that is opened requires 198 bytes. Re-using the same file number, after closing a file, in subsequent OPEN statements will save the allocation of new memory space.

CLOSE #/file no./,#/file no./. . .

The CLOSE statement is used to "close" an open file. The file number that is closed must have previously been opened by the OPEN statement. The CLOSE statement, in effect, disassociates the previously assigned file number with the file name. Files should always be closed when file manipulation is finished.

```
Example:    10 CLOSE #1, #2
```

READ #/file no./,/variable list/

The READ #/file no./ statement is similar to the READ statement described earlier and is used to retrieve data off of a disk file to be used in a BASIC program. For example, a READ #1, A,B will transfer the first entry of file number 1 into variable A and the second entry into variable B. Each time a read is done from a file an internal pointer is incremented so that the next read will access the next value in the file. String and numeric variables may be intermixed in /variable list/ and their format must match with that of the file being read.

Example: Suppose the file PYRL.DAT contains data in the following format:

	(EMPLOYEE #)	(NAME)	(HOURLY SALARY)	(HOURS WORKED)
	numeric	string	numeric	numeric
Such as	1	ADAMS	3.25	40
	2	BROWN	6.00	40
	3	JONES	4.87	40
		etc.		

The following program could be used to work on the file:

```
10  OPEN #, PYRL.DAT
20  READ #1, N, N$, S, T                    READS DATA ON EMPLOYEE 1
30  PRINT N, N$, $, T, S*T
40  READ #1, N, N$, S, T                    READS DATA ON EMPLOYEE 2
50  PRINT N, N$, S, T, S*T
          etc.
```

The ouput on the screen would be as follows:

```
1          ADAMS          3.25          40          130
2          BROWN          6.00          40          240
                          etc.
```

Notice that the READ operation starts at the beginning of a file and increments its way through as data is read.

Note: If the receiving element is a string variable, it will receive the data from the file up to a maximum string length. The line input buffer for a single item from a file is 72 characters. If an attempt is made to read a string variable from the file that is longer than the string length limit of the receiving string variable, the item will be truncated at the receiver's limit. If the input string variable length is greater than the 72 character buffer limit, the buffer input processing will be terminated after 72 characters.

Both the READ and WRITE commands are "line" oriented. For example, say that data was written to a file with the following command:

```
10 WRITE #1, A, B, A$
```

This imaginary "line" then consists of the amount of space that A, B and A$ take up. When reading from a file, a READ statement can not read more than one "line" at a time.

Example:

```
20 READ #1, A, B, A$
```

would read the entire "line" and enter the correct values for A, B, and A$. A statement such as:

```
20 READ #1, A, B, A$, C$, C
```

would assign the correct values to A, B and A$ but would assign the value of Ø to C and set C$ to a null since these variables attempted to read past the "line" length defined by the WRITE command. Also a statement such as:

```
20 READ #1, A, B
```

would corectly read the values of A and B with the remainder of the data on this "line" (A$) not being used. The next READ such as 26 READ #1, A, B, A$ would start reading on the next "line".

SAMPLE FILE STRUCTURE

```
A1      B1      A$1
A2      B2      A$2
A3      B3      A$3
└─defined line length──┘  ↑
```

Any reads attempted from this area would set the desired numeric variables to Ø and string variables to a null.

If the receiving element is a numeric variable, the input is scanned for a "break" character (a comma or a null) and that portion of the input, up to the break character, is then processed by a validation routine which verifies the number as being a valid numeric variable. If the number is invalid, an ERROR 3 message will result.

RESTORE #/file no./, #/file no./,. . .

The RESTORE statement causes the "where to read from" pointer for the file number specified to be set to the location of the first element of the file. The file number may be that of a file that is open for either reading or writing—the restore statement will first close the file and then reopen it for reading.

SCRATCH #/file no./,#/file no./,. . .

The SCRATCH statement is used to remove an existing file from the current disk directory and then re-enter it into the directory. The file will then be re-opened for output (write). The SCRATCH statement performs the functions of delete and open for write. The old file is lost from the disk and a new file with the same name is prepared to receive data. Care should be exercised in using this statement since it will destroy the designated data file.

Example: 35 SCRATCH #5

WRITE #/file no./ , /variable list/

The WRITE statement is essentially the same as a READ statement allowing data to be written on a disk file. The file must have been previously opened for writing by either the OPEN or SCRATCH statements.

```
Example:    10  LET A=5
            20  LET C=6
            30  LET 5$="TEST"
            40  OPEN #1, TEST.DAT
            50  WRITE #1, A, C, S$
            60  CLOSE #1
```

In the above program a file will be created by the name TEST.DAT (WRITE will create a file on disk if none exists) and the values 5 6 and TEST will be entered in the file. If the file specified currently exists on the disk, an error will result on the first execution of the WRITE statement. To insure availability of file write access, the SCRATCH statement should be used.

Note: WRITE must be followed by a list of variables only.

10 WRITE #1, 5, 6, "TEST" is not valid.

EOF (X) **X= file number**

The EOF(X) command can be used to determine if the end of a file has been reached. X is the desired file number. EOF will return a Ø if not at the end and a 1 if the end has been reached.

Example: 25 IF EOF (3) = 1 PRINT "END OF FILE"

Random Access Data Arrays

In addtion to sequential disk data files described earlier, this version of disk BASIC also supports what can be called random access data arrays. Random access data arrays gives the programmer the ability to individually and selectively access any element in a given array on disk in almost exactly the same manner as is done in memory, like in the following example:

```
10  DIM A$(5,5)                  Create a 5 x 5 string matrix
20  LET A$(1,2)="THIS IS 1,2"    Selectively reference a particular location.
```

A 5 x 5 reference like the one above will easily fit in the computer's memory, but a 100 by 100 array will not. Disk random access data arrays allow these large arrays to be stored and accessed on disk rather than in memory.

Using Random Access Arrays

In order to use a random access array (RAA) a name and number must be associated with the array. The file management system of the DOS will treat the RAA just like a file; therefore, the **OPEN** statement described earlier can be used.

Example: 10 OPEN #1, DATA.RAN

This associates the file DATA.RAN with the file #1 which is subsequently recognized by BASIC. By opening the file you are essentially specifying the area of "virtual memory" (disk addressed) to be used. If the specified file does not exist on the disk, the OPEN statement will automatically create it.

Once a file has been opened on disk, it must be dimensioned with a special form of the DIM statement. The general form of this special dimension statement is:

DIM #/file no./,/var/(exp) or
DIM #/file no./,/var/(exp,exp)

where **/var/** is a numeric or string variable name and **exp** is an expression defining the size of the array.

For example, a 100 x 100 numeric array could be dimensioned as DIM #1,A(100,100) and a string array could be dimensioned as DIM #1,A$(100,100). Any dimension of up to 65535 can be specified, however, keep the size as small as possible to conserve disk space. The DIM statement essentially formats the required disk space and can take several minutes to finish the **first time** depending on the size of the array.

Nothing special is required to access an element in a RAA once the file is opened and dimensioned—referencing an element in the disk array is exactly the same as accessing an element in a memory stored array.

```
Example: 1Ø  OPEN #1, DATA.RAN
         2Ø  DIM  #1, A$(100,100)
         3Ø  LET  A$(5,35)="ELEM 5.35"
```

Once access to the RAA is finished, the file should be closed by using the **CLOSE** statement described earlier.

The Format of Random Access Arrays

Knowing how the actual data is stored on disk in a RAA can help in writing programs which are optimized for both disk space usage and speed.

All information that is stored on disk is stored in the form of "sectors". Each sector contains 252 usable bytes of storage. In this version of BASIC each floating point number requires 6 bytes of disk storage; therefore, 42 numbers can be stored in one sector. The number of sectors required to store a 100 by 100 numeric array would be:

$$\frac{100 \times 100}{42} = 238.1 \rightarrow 239 \text{ sectors.}$$

The amount of space required for a string variable depends on the string length which is set to using the STRING=L command. If STRING=32, then each string variable will hold 32 characters and will require 32 bytes. The number of string variables stored per sector can be expressed as 252÷ L where L is the string length. For example, using a string length of 32, 252 ÷ L gives 7.875 string variables per sector. A fraction of variable cannot be stored in a sector so only 7 variables/ sector are allowed and 252–(7)(32)=28 bytes per sector are wasted. Careful selection of string length will help conserve disk use.

When addressing two dimensional random access arrays, it is very important that the dimensions be accessed in the correct "order" for speed optimization. Take, for example, the 3 x 3 array **A. A** would be stored on disk in the following manner:

```
A(1,1)      row 1   col 1
A(1,2)      row 1   col 2
A(1,3)      row 1   col 3
A(2,1)            etc.
A(2,2)
  etc.
```

Obviously it is much faster to do searches by first referencing a row than incrementing by columns. Example:

CORRECT REFERENCE ORDER	INCORRECT ORDER
A(1,1)	A(1,1)
A(1,2)	A(2,1)
A(1,3)	A(3,1)
A(2,1)	A(1,2)
A(2,2)	A(2,2)
A(2,3)	A(3,2)

Two example program parts and the required times for each show the importance:

CORRECT WAY

```
10 FOR I=1 to 100
20 FOR J=1 to 100
30 X= A(I,J)
40 IF I x J=10000 THEN END
50 NEXT J
60 NEXT I
```

Time reqd = 50 sec.

INCORRECT WAY

```
10 FOR J=1 to 100
20 FOR I=1 to 100
30 X= A(I,J)
40 IF I x J=1000 THEN END
50 NEXT I
60 NEXT J
```

Time reqd = 3 minutes

Other Notes

There are several other notes worth mentioning concerning random access arrays:

1.) All references to array elements are located relative to the start of file address. No description (name, dimension, type, etc.) is stored within the file. After originally creating an array care should be execised not to later reference it with a dimension larger than the original one.

Example:

```
DIM #1, A$(10)     original

DIM #1, B$(5)      ok

DIM #1, C$(30)     DON'T DO!
```

2.) The same file name should never be opened under two different file numbers. Don't, for example:

```
10  OPEN #1, DATA.RAN

60  OPEN #2, DATA.RAN
```

3.) The sequential file commands READ, RESTORE, SCRATCH, WRITE and EOF have no use with random access arrays.

4.) The same random access array reference should not be used on both sides of an "equate" expression.

Example:

```
10  DIM #1, A(100)
20  LET A(95)=A(13)  DON'T DO!
```

do this instead:

```
10  DIM #1, A(100)
20  LET B= A(13)
30  LET A(95)=B
```

APPENDIX A

ERROR MESSAGES

If, during the operation of BASIC, a mistake was made by the programmer, BASIC will output one of the following error messages:

ERROR #	DEFINITION
1.	Oversize variable (over 255) in TAB, CHR, subscript or "ON"
2.	Input error
3.	Illegal character or variable
4.	No ending " in print literal
5.	Dimensioning error
6.	Illegal arithmetic
7.	Line number not found
8.	Divide by zero attempted
9.	Excessive subroutine nesting (max is 8)
10	**RETURN** W/O prior **GOSUB**
11.	Illegal variable
12.	Unrecognizable statement
13.	Parenthesis error
14.	Memory full
15.	Subscript error
16.	Excessive FOR loops active (max is 8)
17.	**NEXT** "X" W/O **FOR** Loop defining "X"
18.	Misnested FOR-NEXT
19.	READ statement error
20.	Error in ON statement
21.	Input Overflow (more than 72 characters on INput line)
22.	Syntax error in DEF statement
23.	Syntax error in FN error, or FN called on Function not defined
24.	Error in STRING Usage, or mixing of numeric and string variables
25.	String Buffer Overflow, or String Extract (in MID$,LEFT$, or RIGHT$) too long
26.	I/O operation requested to a port that does not have an I/O card installed.
27.	VAL function error—string starts with a non-numeric value.
28.	LOG error—an attempt was made to determine the log of a negative number.
29.	File not open
30.	Illegal file number—must by Ø-9
31.	Illegal file name
32.	File number in use
33.	Attempt to write to a file not open for write
34.	Attempt to read from a file not open for read
35.	The same virtual array reference not allowed on both sides of the = sign
36.	Can't exit while printing

APPENDIX B

Disk Error Messages

During any disk operation, there are several possible disk error messages:

ERROR #	DEFINITION	ERROR #	DEFINITION
0	NO ERROR	12	DELETE PROTECTED
1	ILLEGAL FUNCTION CODE	13	ILLEGAL FCB
2	FILE BUSY	14	ILLEGAL DISK ADDRESS
3	FILE EXISTS	15	DRIVE NUMBER ERROR
4	NO SUCH FILE	16	NOT READY
5	DIRECTORY ERROR	17	ACCESS DENIED
6	TOO MANY FILES	18	STATUS ERROR
7	DISK FULL	19	INDEX RANGE ERROR
8	END OF FILE	20	FMS INACTIVE
9	READ ERROR (CRC)	21	ILLEGAL FILE NAME
10	WRITE ERROR (CRC)	22	CLOSE ERROR
11	WRITE PROTECTED		

When a disk error is encountered, the output will be similar to the form "DISK ERROR #11". An additional error message may also be output at this time and will be of the form:

ERROR #F (file #) IN LINE # line no.)

If, for example, the following program was attempted with a write protected disk, the error message would be shown:

```
10  OPEN #1, DATA.DAT
20  WRITE #1, A, B
30  GOTO 3
    would give
DISK ERROR #11
ERROR #F1 IN LINE #20
```

If an error #FF in line # (line no) message is encountered then an attempt was made to KILL or RENAME a non-existent file or a special file control block error was detected.

APPENDIX C
ASCII Hexadecimal to Decimal Conversion Table

CHARACTER	HEXADECIMAL	DECIMAL
NUL	00	000
SOH	01	001
STX	02	002
ETX	03	003
EOT	04	004
END	05	005
ACK	06	006
BEL	07	007
BS	08	008
HT	09	009
LF	0A	010
VT	0B	011
FF	0C	012
CR	0D	013
SO	0E	014
SI	0F	015
DLE	10	016
DC1	11	017
DC2	12	018
DC3	13	019
DC4	14	020
NAK	15	021
SYN	16	022
ETB	17	023
CAN	18	024
EM	19	025
SUB	1A	026
ESC	1B	027
FS	1C	028
GS	1D	029
RS	1E	030
US	1F	031
SP	20	032
!	21	033
"	22	034
#	23	035
$	24	036
%	25	037
&	26	038
'	27	039
(	28	040
)	29	041
*	2A	042
+	2B	043
,	2C	044
-	2D	045
.	2E	046
/	2F	047
0	30	048
1	31	049
2	32	050
3	33	051
4	34	052
5	35	053
6	36	054
7	37	055
8	38	056
9	39	057
:	3A	058
;	3B	059
<	3C	060
=	3D	061
>	3E	062
?	3F	063
@	40	064
A	41	065
B	42	066
C	43	067
D	44	068
E	45	069
F	46	070
G	47	071
H	48	072
I	49	073
J	4A	074
K	4B	075
L	4C	076
M	4D	077
N	4E	078
O	4F	079
P	50	080
Q	51	081
R	52	082
S	53	083
T	54	084
U	55	085
V	56	086
W	57	087
X	58	088
Y	59	089
Z	5A	090
(	5B	091
\	5C	092
)	5D	093
^	5E	094
_	5F	095
`	60	096
a	61	097
b	62	098
c	63	099
d	64	100
e	65	101
f	66	102
g	67	103
h	68	104
i	69	105
j	6A	106
k	6B	107
l	6C	108
m	6D	109
n	6E	110
o	6F	111
p	70	112
q	71	113
r	72	114
s	73	115
t	74	116
u	75	117
v	76	118
w	77	119
x	78	120
y	79	121
z	7A	122
{	7B	123
/	7C	124
}	7D	125
~	7E	126
DEL	7F	127

Appendix D

Memory Map

ØØØØ - ØØFF	Input buffer and temporary variable storage
Ø1ØØ - 2698	BASIC interpreter (may vary slightly—check contents of Ø14E for your version)
Ø1ØØ	Cold start address
Ø1Ø3	Warm start address
ØØ2A-ØØ2B	Contains the next available memory after the current BASIC souce program.
ØØ2E-ØØ2F	Contains the starting address of the BASIC source program.
ØØ5D-ØØ5E	Contains the address of the current arithmetic value in use during a USER call.
ØØ67 - ØØ68	Contains the pointer for USER
Ø14E-Ø14F	This location tells BASIC where to start allocating memory for program storage. This address may be changed by the user if desired to allow space for USER routines.
Ø15Ø	Contains the number of the port which BASIC will initialize to, currently Ø1.
Ø153	Contains the ASCII value of what is output to the terminal when a BACKSPACE is entered. Currently a null for terminals generating an automatic cursor left on backspace.
Ø154	Contains the ASCII value of what BASIC interprets as a BREAK character, currently a Ø3 (CTL. C.).

Below is a list of the I/O jumps in BASIC for the various ports. For each port the first is the "output character in accumulator A" jump, the second receives input and places it in accumulator A and the third is the initialization routine for a particular type port (ACIA or PIA). This I/O can be changed at the discretion of the user if desired.

```
                        *PORT 0
0106 7E 04 6C  JMPTAB   JMP     OUTACI
0109 7E 04 5D           JMP     INACIA
010C 7E 03 B2           JMP     ACIINZ
                        *PORT 1
010F 7E AD 18           JMP     OUTEEE
0112 7E AD 15           JMP     INEEE
0115 7E 19 BD           JMP     DUMRTS
                        *PORT 2
0118 7E 04 6C           JMP     OUTACI
011B 7E 04 5D           JMP     INACIA
011E 7E 03 B2           JMP     ACIINZ
                        *PORT 3
0121 7E 04 6C           JMP     OUTACI
0124 7E 04 5D           JMP     INACIA
0127 7E 03 B2           JMP     ACIINZ
                        *PORT 4
012A 7E 04 82           JMP     OUTPIA
012D 7E 04 77           JMP     INPIA
0130 7E 03 BD           JMP     PIAINZ
                        *PORT 5
0133 7E 04 82           JMP     OUTPIA
0136 7E 04 77           JMP     INPIA
0139 7E 03 BD           JMP     PIAINZ
                        *PORT 6
013C 7E 04 82           JMP     OUTPIA
013F 7E 04 77           JMP     INPIA
0142 7E 03 BD           JMP     PIAINZ
                        *PORT 7
0145 7E 04 82           JMP     OUTPIA
0148 7E 04 77           JMP     INPIA
014B 7E 03 BD           JMP     PIAINZ
```

APPENDIX E
Notes for Speeding up BASIC

1. Subscripted variables take considerable time; whenever possible use non-subscripted variables.
2. Transcendental functions (SIN, COS, TAN, ATAN, EXP, LOG) are slow because of the number of calculations involved, so use them only when necessary. Also, the ↑ operator uses both the LOG and the EXP functions, so use the format A*A to square a number.
3. BASIC searches for functions and subroutines in the source file, so place often called routines at the start of the program.
4. BASIC searches the symbol table for a referenced variable, and variables are inserted into the symbol table as they are referenced; therefore, reference a frequently called variable early in the program so that it comes early in the symbol table.
5. Numeric constants are converted each time they are encountered, so if you use a constant often, assign it to a variable and use the variable instead.

APPENDIX F
Notes on Memory Usage in BASIC

1. REM statements use space, so use them sparingly.

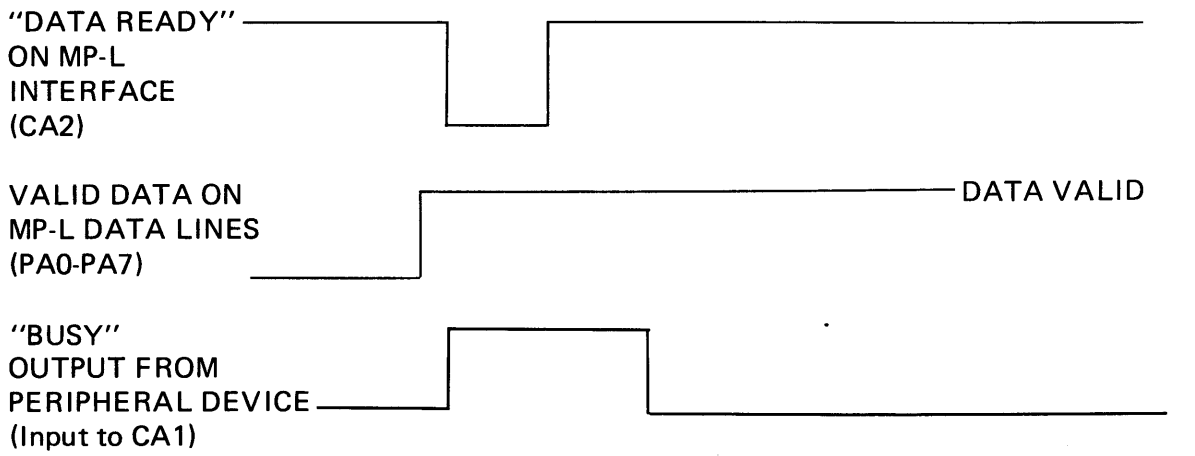
2.a. Each non-subscripted numeric variable takes 8 bytes.
   b. Each non-subscripted string variable takes 34 bytes.
   c. Each numeric array takes 6 bytes plus 6 bytes for each element.
   d. Each string array takes 6 bytes plus 32 bytes for each element.

3. An implicitly dimensioned variable creates 10 elements (or 10 by 10). If you do not intend to use all 10 elements, save memory by explicitly dimensioning the variable.
4. Each BASIC line takes 2 bytes for the line number, 2 bytes for the encoded key word, 1 byte for the end of line terminator, 1 byte for the line length, plus one byte for each character following the key word. Memory can be saved by using as few spaces as possible.
5. BASIC reserves the uppermost 256 bytes of memory for stack and buffer use.

APPENDIX G

Parallel Interface Handshaking

The parallel interface drivers are written for a conventional handshaking scheme used by many printer manufacturers and is the same as that on a SWTPC PR-40 printer. Handshaking timing is set up as follows:



↑ peripheral should generate this level change when it has accepted the previous data and is ready for more.

Commands	Statements	Functions
APPEND	CHAIN	ABS(X)
CAT	CLOSE	CHR $(X)
CONT	DATA	COS(X)
DIGITS=&	READ	DEF FN(X)
DOS &	RESTORE	EOF(X)
KILL &	FOR /exp/ to /exp/	EXP(X)
LINE= &	NEXT	INT(X)
LIST &	GOSUB	LEFT $(X$,N)
LOAD	DIM *	LEN(X$)
MON	END	MID$(X$,S.Y)
NEW	GOTO *	PEEK(X)
PORT= &	IF /rel. exp./ THEN /line/	POKE (I,J)
RENAME &	INPUT *	POS
RUN	ON /exp/ GOT /line(s)/	RIGHT(X$,N)
SAVE	ON /exp/ GOSUB /line(s)/	RND(X)
STRING= &	PRINT *	SGN(X)
TAPPEND	REM	SIN(X)
TLOAD	RETURN	SQR(X)
TRACE ON &	STOP	STR$(X)
TRACE OFF &	WRITE	TAB(X)
TSAVE		TAN(X)
		USER(X)
		VAL(X$)

* Denotes statements that may be used in the DIRECT mode

& Denotes commands that may be used as program statement

MATH OPERATORS		RELATIONAL OPERATORS	
↑	Exponentiate	=	Equal (numeric and string)
–	(unary) Negate	<>	Not Equal (numeric and string)
*	Multiplication	<	Less Than
/	Division	>	Greater Than
+	Addition, string concatenation	<=	Less Than or Equal
–	Subtraction	>=	Greater Than or Equal

LINE NUMBERS —May be from 1 thru 9999

VARIABLES —May be single character alphabetic or single character alphabetic followed by one integer 0 thru 9 or $

BACKSPACE —Is a Control H

LINE CANCEL —Is a Control X (CANCEL)

PROGRAM ABORT—Typing a Control C should bring BASIC back to the READY mode regardless of what the BASIC program is doing (except USER programs).

LINES —Each line may contain multiple statements. Each statement is separated from the other with a : .

INDEX