ATTENTION

Know Your Computer and Be Sure You Are Using the Correct Software

SWTPC has offered, or is now offering, three types of 6809 computers. In order to make SWTPC supplied software work correctly it is necessary to recognize what type of computer you have. Below is a description of each of the three types and some of the characteristics of each. The nomenclature for the motherboard of each type of computer is printed on the motherboard for easy identification.

/09

The /09 computer is any SWTPC 6809 computer which uses an MP-B or MP-B2 motherboard. This computer uses the single port MP-S serial interface and the MP-L and MP-LA parallel interfaces. This computer can have up to 56K of 4K, 8K, 16K and 32K memory boards. Its chassis RESET button is on the right side of the front panel.

69A and 69K

The 69A and 69K computers use the MP-B3 motherboard. The 69A and 69K are identical except that the A is factory.assembled and the K is the kit version. Each interface port in this type computer requires 16 addresses and uses the MP-S2 serial and MP-L2 parallel interface boards. This computer can have up to 56K of 4K, 8K, 16K and 32K memory boards. Its chassis RESET button is on the left side of the front panel.

S/09

The S/09 computer uses the MP-MB motherboard. Each interface port in this computer requires 16 addresses and uses the MP-S2 serial and MP-L2 parallel interfaces. This computer also contains a standard parallel output port and an integral interrupt timer on the MP-ID board. The S/09 can use up to 384K of 128K memory array boards. Its chassis RESET button is on the left side of the front panel.

Be Sure To Use The Correct Software

Although the /09, 69A, 69K and S/09 computers are all basically the same, small differences in I/O port assignments, speed, features and memory types dictate that certain programs, such as printer drivers, function differently on the various models. After booting the system diskette, FLEX will automatically configure the operating system as completely as it can to certain initial values of speed, CPU type, etc. A special utility (SBOX) has been supplied to examine and change the initial values and computer type. After booting the supplied diskette, this utility should be run to be sure that ALL of the displayed characteristics match EXACTLY with the computer being used. Any necessary changes can be made using the SBOX utility. This will usually be necessary only on 109 computers and 69A/69K computers operating at 2 MHZ.

Product: FLEX 2.6 DOS
Date: February 26, 1980

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

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

```
-- Interval Timer = Yes
```

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

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

```
SBOX,TIMER=NO
```

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

```
+++SBOX

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

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

General Notes

Technical Systems Consultants, Inc.

GENERAL NOTES

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

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

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

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

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

HARDWARE REQUIREMENTS

This section discusses the hardware requirements for running FLEX 9.0. This version is setup for th6 Southwest Technical Products Corporation's disk systems: the MF-68 or MF-69 5-inch minidiskette, the DMAF1 or DMAF2 8-inch diskette, and the CDS-1 Winchester disk unit.

Memory Requirements

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

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

Monitor ROM

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

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

Printer Spooling

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

DISK COMPATIBILITY

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

The one place where the disks are different is in the bootstrap loader which the NEWDISK command places on track 0 when a disk is initialized. Obviously the loader must be different for 6800 and 68C9. This simply means that a disk initialized with the 6809 NEWDISK command cannot be used to boot 6800 FLEX and vice versa.

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

SOFTWARE COMPATIBILITY

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

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

ADAPTING FLEX

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

The CUSTOM I/O DRIVER PACKAGE

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

A description of each routine and vector follows.

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

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

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

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

MONITR

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

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

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

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

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

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

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

The CUSTOM DISK DRIVER PACKAGE

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

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

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

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

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

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

PUTTING THE CUSTOM FLEX TOGETHER

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

*** IMPORTANT ***

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

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

```
END $CD00
```

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

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

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

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

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

```
                    * SKELETAL 6809 DISK DRIVER PACKAGE
                    * TECHNICAL SYSTEMS CONSULTANTS, INC.
                    * BOX 2574
                    * WEST LAFAYETTE, INDIANA  47906
                    *

                    * THE DRIVER ROUTINES PERFORM THE FOLLOWING
                    * 1.  READ SINGLE SECTOR  - DREAD
                    * 2.  WRITE SINGLE SECTOR - DWRITE
                    * 3.  VERIFY WRITE OPERATION - VERIFY
                    * 4.  RESTORE HEAD TO TRACK 00 - RESTORE
                    * 5.  DRIVE SELECTION - DRIVE
                    * 6.  CHECK READY - DCHECK
                    * 7.  QUICK CHECK READY - DQUICK
                    * 8.  COLD START INITIALIZATION - DINIT
                    * 9.  WARM START INITIALIZATION - DWARM
                    *
                    *  SYSTEM CONSTANTS
                    *
                    *  THIS SPACE IS WHERE ANY NECESSARY EQUATES MIGHT
                    *  BE PLACED, SUCH AS DISK CONTROLLER REGISTER
                    *  LOCATIONS, SECTOR LENGTH, ETC.
                    *

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

DE00                        ORG     $DE00

                    * JUMP TABLE

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

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

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

                    * VARIABLE STORAGE

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

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

                    * INIT
                    *
                    * INITIALIZES THE NECESSARY DRIVER VARIABLES.

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

                    * WARM
                    *
                    * WARM START INITIALIZATION

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

                    * READ
                    *
                    * READ ONE SECTOR

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

                    * WRITE
                    *
                    * WRITE ONE SECTOR

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

```
                    * VERIFY
                    *
                    * VERIFY LAST TRACK WRITTEN

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

                    * RST
                    *
                    * RST RESTORES THE HEAD TO 00

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

                    * DRV
                    *
                    * DRV SELECTS THE DRIVE.

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

                    * CHECK
                    *
                    * CHECK FOR DRIVE READY

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

                    * QUICK
                    *
                    * QUICK CHECK FOR READY

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


                            END
```

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

                    * SYSTEM EQUATES

               C700 CHPR    EQU    $C700       CHANGE PROCESS ROUTINE


                    *******************************************************
                    *                                                     *
                    * I/O ROUTINE VECTOR TABLE                            *
                    *                                                     *
 D3E7                       ORG    $D3E7     TABLE STARTS AT $D3E7        *
                    *                                                     *
 DEE7 D3CB          IHNDLR  FDB    IHND      IRQ INTERRUPT HANDLER        *
 D3E9 DFC2          SWIVEC  FDB    $DFC2     SWI3 VECTOR LOCATION         *
 D3EB DFC8          IRQVEC  FDB    $DFC8     IRQ VECTOR LOCATION          *
 D3ED D3C4          TMOFF   FDB    TOFF      TIMER OFF ROUTINE            *
 D3EF D3BD          TMON    FDB    TON       TIMER ON ROUTINE             *
 D3F1 D3A7          TMINT   FDB    TINT      TIMER INITIALIZATION ROUTINE *
 D3F3 F814          MONITR  FDB    $F814     MONITOR RETURN ADDRESS       *
 D3F5 D370          TINIT   FDB    INIT      TERMINAL INITIALIZATION      *
 DEF7 D39C          STAT    FDB    STATUS    CHECK TERMINAL STATUS        *
 DEF9 D38B          OUTCH   FDB    OUTPUT    TERMINAL CHAR OUTPUT         *
 D3FB D37D          INCH    FDB    INPUT     TERMINAL CHAR INPUT          *
                    *                                                     *
                    *******************************************************



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

 D370                       ORG    $D370

                    * TERMINAL INITIALIZE ROUTINE

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

                    * TERMINAL INPUT CHARACTER ROUTINE

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

```
                    * TERMINAL OUTPUT CHARACTER ROUTINE

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

                    * TERMINAL STATUS CHECK (CHECK FOR CHARACTER HIT)

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


                    * TIMER INITIALIZE ROUTINE

D3A7 BE   D3E1      TINT    LDX    TMP1A      GET PIA ADDRESS
D3AA 86   FF                LDA    #$FF
D3AC A7   84                STA    0,X
D3AE 86   3C                LDA    #$3C
D3B0 A7   01                STA    1,X
D3B2 86   8F                LDA    #$8F
D3B4 A7   84                STA    0,X
D3B6 A6   84                LDA    0,X
D3B8 86   3D                LDA    #$3D
D3BA A7   01                STA    1,X
D3BC 39                     RTS

                     * TIMER ON ROUTINE

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

                     * TIMER OFF ROUTINE

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


                    * IRQ INTERRUPT HANDLER ROUTINE

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

```
                   * ACIA AND PIA ADDRESS FOR SUPPLIED ROUTINES

D3E1                       ORG    $D3E1

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


                   * END STATEMENT HAS FLEX TRANSFER ADDRESS!

                           END     $CD00
```