May 29, 1980

APEX-65 USER'S MANUAL

VERY PRELIMINARY

This preliminary manual and version 1.0 of APEX-65 is being sent to you now so that you may make some use of the K-1013 Disk Controller board and your disk drives before the final distribution version is ready. Only a subset of the promised features are provided and none of the anticipated disk utility programs (such as generalized file copy) are provided. The most serious shortcoming of this version is that the system "overlays" are not overlayed from disk like they will be. Instead locations 4800-5CFF in the User RAM are used to keep the overlays resident. Nevertheless one should be able to effectively load and save machine language programs, BASIC programs, and aim text editor test with . EX as it is. Any files built up on your diskettes with version 1.0 of APEX-65 will be readable by the final version so you can start building a library now.

May, 1980

FIRST TIME POWER-UP USING APEX-65

FOLLOW THESE INSTRUCTIONS CAREFULLY!

1. Read the copyright notice at the front of this manual.

2. Read the disk handling cautions at the front of this manual.

3. Have you really read them? Its important!

4. Insure that you have thoroughly checked out operation of the K-1013 disk controller according to the instructions in the controller manual.

5. Insure that your disk controller is jumpered as follows:
   A. System RAM at address $8000
   B. User RAM at address $4000
   C. No interrupts

6. Insure that you have at least 2K bytes of RAM starting at $2000 (used by the COPYF program). We recommend our K-1016 RAM board addressed at 0000 (Remove the 2114 RAM chips from the AIM when starting external RAM at 0000.)

7. If a K-1008 Visible Memory is available, jumper it for 6000-7FFF.

8. Power up the AIM-65. Leave the printer on.

9. Insert the cassette with the loader program into your recorder. Later you will be sent a ROM chip to install on the disk controller board which has the loader program in it.

10. Load the file called APEXL from cassette tape according to instructions in the AIM-65 manual. (It is recorded three times in case a bad spot on the tape makes one of them unreadable.)

11. Insert the APEX-65 Distribution Diskette into drive 0.

12. Using the AIM monitor, begin execution at $0000. This is the address of the APEX bootstrap loader. The disk should "click" and show activity for about two seconds, after which two ">" characters should appear on separate lines on the printer, like this:

    >
    >

    You are now in the APEX monitor. The ">" on the display is a prompt for you to enter a command. (Note that the AIM monitor uses a " " for the prompt character allowing one to distinguish which is in control.)

13. Type FILES 0 with a carriage return. The printer should list several

File names, beginning with APEX.Z. The disk will show activity, too. The ">" indicates readiness to accept another command.

11. If all has gone smoothly so far, insert a new blank disk into drive 1 and close the door. We will now make a copy of the system.

12. Type

FORMAT 1 1002

This tells the system to Format the disk in drive 1, and give it a serial number of 1002. The serial number is arbitrary, but every disk should have a unique number (4 hex digits). This number is used to uniquely identify each disk. A sticker should later be applied to the label on the disk with this number for visual identification (don't write on the disk with a pen or pencil!). Formatting the disk erases everything on the disk (including any GUARDED files), and writes timing information on each track. Even if you buy "pre-formatted" diskettes, you must FORMAT every disk before using it with APEX. The printer will prompt you with the message,

ARE YOU SURE?

Answer with YES. The disk in drive 1 will show activity for about half a minute. When the MONITOR prompt is printed, type

COPYF APEX.Z

This will copy the operating system nucleus from drive 0 to drive 1.

13. In a like manner, type:

COPYF OVL.Z

14. Type

COPYF SYSERRMSG.Z

15. Type

COPYF STARTUP.J

16. Type

COPYF COPYF

17. Type

FILES 1

Verify that the first file listed is APEX.Z, and that there are 5 files corresponding to the 5 files on drive 0. If your FILES 0 command shows a file called VMT, type

COPYF VMT

to copy that file to drive 1.

18. Type

CLOSE 0 1

to close drives 0 and 1. Remove the distribution diskette from drive 0 and store it in its envelope in a safe place. You now should have a duplicate disk which you can use.

19. Remove the disk from drive 1, and power-down the system. Power back up the system, insert the new disk into drive 0, and "re-boot" by beginning execution at $9F00 using the AIM MONITOR. The system should come up as it did for the distribution diskette. It might be a good idea to make additional backup copies of the system using the same procedure. It is only necessary to copy the system files onto disks which will be used in drive 0. If you are only planning to use the disk in drive 1, you may just FORMAT it and CLOSE it.

NOTE: When making copies of the system, you must FORMAT the disk and COPYF APEX.Z immediately. It is not sufficient to merely ERASE all the files on a disk and then COPYF APEX.Z, nor can you do other COPYF's before doing the COPYF APEX.Z.

APEX-65 MONITOR

Executing the Bootstrap loader ($9F00) causes the operating system meory image to be loaded into memory from disk. Then a file of commands called STARTUP.J is read and all commands on it are executed. On reaching end-of-file, an interactive dialog is begun with the user. This program is called the APEX MONITOR. The AIM monitor is not normally used; however, depressing ESC will exit to the AIM monitor. Depressing F3 will re-enter the APEX Monitor. The Prompting character for command entry in the APEX monitor is ">". Unlike the AIM Monitor, the entire command line must be typed by the user, terminated by a carriage return, before any action takes place.

COMMAND EDITING

The following characters may be used to edit commands as they are typed:

DEL, CNTRL-H	Backspace1 character.
CNTRL-X	Delete entire line (start over)
ESC	Escape to AIM Monitor
F3 (while in AIM Monitor only)	Escape to APEX.
Carriage Return	End of command
;	Comment. Any characters after ";" are ignored.
!	Command abbreviaion. See description below.
blank	Seperator between command and arguments (required).

COMMANDS

There are two types of commands in APEX-65: User-Commands and Built-in commands. Built-in commands are provided automatically. User commands are easily added by writing an assembly-language program and defining it as a command using the built-in SAVE command. In the following section, only built-in commands will be discussed, so the term "command" will be understood to mean "built-in command".

To improve readability and ease the learning process, APEX commands consist of full words (usually) which suggest their ~~meaning~~ function. However, any built-in command can be abbreviated using the "!" character. Thus,

ASSIGN
ASSI!
AS!

are all equivalents for the ASSIGN command. Only enough characters need be typed before the "!" to uniquely identify the command you want.

ARGUMENTS

Most commands require one or more arguments besides the command keyword. These arguments tell the system what entities the command is to operate on. For example, the command

FORMAT 1 1002

has two arguments. The first argument tells which drive to format and the second tells what VSN (serial number) to write on the disk. Arguments are usually separated by blanks (not commas!). Any number of blanks may be used. Sometimes arguments are optional. In order to have a uniform method of describing the requirements and options

for commands, the following notation is adopted to describe the command syntax:

1. Angle Brackets, "<", and ">", are used to enclose words describing the kind of entry required.
2. Square brackets, "[" and "]", are used to enclose optional parameters, which may be included or omitted, as desired.
3. Elipses, "...", are used to indicate an arbitrary number of repetitions of the preceeding argument(s).
4. Symbols not enclosed in angle brackets are literal symbols which must be typed exactly as shown.
5. Curly brackets, "{" and "}" are used to enclose each of several mutually exclusive choices, one of which must be selected.

EXAMPLE: If we were to use the syntax descriptions above to describe some BASIC statements, then they might look like this:

GOTO <line-number>
ON <expression> GOTO <line-number> [<line-number>...]
FOR <variable> = <expression> TO <expression> [STEP <expression>] DO

TYPES OF ARGUMENTS

1. File names: File names identify logically complete entities on disk, and may consist of from 2 to 12 characters, optionally followed by a "." and a one character file extension. The first character of a file name must be alphabetic; the remaining characters may be alphabetic, numeric, or the special character, "_" (underline). The underline character is useful for improving readability. The file extension may be alphabetic or numeric. If the "." and file extension is omitted, a default file extension of ".C" is

assumed by the system. Examples of legal file names are:

```
A2
MY3RDFILE.A
HIS_STUFF.T
OLD_X_Y_DATA.8
YANK
```

The file extension is intended to provide the user with an indication of the kind of file. Although the system does not enforce any particular convention, the following extensions are suggested in the interest of uniformity:

A	Assembly language source
B	BASIC compiler source (planned for future)
C	Command (User-defined)
D	Data
G	Graphics file (future)
H	Hex file
J	Job file (i.e., list of commands)
L	Listing
T	Text file
X	Executable code other than command
Z	System file
5	AIM BASIC file

The user may devise other codes as needed. Note that if the file extension is not ".C", then it must be given explicitly.

2. CHANNELS:

TO BE CONTINUED - - -

APEX 65 DISK OPERATING SYSTEM

HARDWARE REQUIREMENTS

AIM 65, power supply, K-1013 Floppy Disk Controller with APEX "system" RAM at $8000, additional 8K RAM at $4000 and Visible Memory at $6000 (if used).

STARTUP INSTRUCTIONS

See "First Time Power-up Using APEX-65

EDITING CHARACTERS

"DEL" key or cntrl-H = BACKSPACE
"ESC" key escapes back to normal AIM 65 monitor
"F3" key escapes from AIM 65 monitor to APEX command mode
"CNTRL-X" control shifted X key deletes the entire line
"CNTRL-L" control shifted L key erases the screen if VM has been executed and in APEX command mode

APEX COMMANDS

SYNTAX DESCRIPTION

Angle brackets "<" and ">" enclose words describing the kind of entry required. Square brackets "[" and "]" enclose optional arguments. Curly brackets "{" and "}" enclose several choices, one of which must be selected. Ellipses "..." indicate that an arbitrary number of repetitions of the argument(s) is permitted.

COMMANDS (GENERAL)

Built in commands may be abbreviated using "!". Thus "AS!", "ASSIG!", and "ASSIGN" are all equivalent. User defined commands may not be abbreviated. If a command is to have arguments, they must be separated by one or more blanks (NOT COMMAS). Anywhere a numeric value is indicated, such as for <FROM>, <TO>, <CHANNEL>, etc. below, a numeric expression can be substituted. Numbers without a prefix are assumed to be HEXIDECIMAL; the hex prefix "$" can also be used. Decimal values must have a "." prefix. Operators are "+", "-", "*", "\", and "/" (backslash = remainder).

As an example, a legal <FROM> argument could be:

6000 -.10000 +34

The character ";" is the comment character for commands and any characters after it are ignored.

APEX COMMANDS

PRESENTLY IMPLEMENTED COMMANDS

1. ASSIGN CHANNEL {<DEVICE>}{<FILES> [<DRIVE#>]}...

EXAMPLES:

ASSIGN 6 C

Assigns channel 6 to the consloe (I.E., the keyboard on input and the AIM 65 display/printer on output, or the Visible Memory CRT if VM has been executed).

ASSIGN 5 MYTEXT.T

Assigns channel 5 to a file (MYTEXT.T) on drive 0. The system will respond either "new file" or "old file". If you made a mistake and did not get the file you wanted, just do it again. It is acceptable to assign a channel which is already assigned. It is also acceptable to assign several channels to the same file or device. Assigning a file always positions it to the begining of data in the file. Channels 0, 1, and 2 are used by the system and should not be reassigned until you have a good understanding of the system.

ASSIGN 4 C 7 YOURS.5:1

Assigns channel 4 to the console and channel 7 to a file (YOURS.5) on drive 1. The ".5" file extension is recommended to denote AIM 65 ROM BASIC programs (you will recall the AIM basic command is "5").

2. FREE <CHANNEL>...

EXAMPLE:

FREE 6 4 9

This command frees channels 6, 4, and 9 from their prior assignments.

3. OPEN <DRIVE#>...

EXAMPLES:

OPEN 1
OPEN 1 0

Note that upon cold startup drive 0 is opened by the system. Other drives must be opened by an open command before any disk operation on them except for "FORMAT", which can be performed on a closed drive (do NOT FORMAT the APEX MASTER distribution diskette!!!). The OPEN/CLOSE commands are needed only when changing diskettes or shutting down operations. You do not open files, as on many other systems, only entire diskettes. If you forget to close a disk before removing it from the dirve, you will get a system error message on your next operation on that disk. At this point you have 2 choices:

i. Put the old disk back in and close it (be sure to get the right one or you will kill it), or

ii. you can repeat your command in which case you will not get another error message. If you select the latter, no harm will be done unless you had:

a. An assigned disk file (not freed) on the old disk and
b. The last operation on that disk was a write (not read) from a user program (not system program); otherwise the disk will be complete.

4. CLOSE <DRIVE#>...

EXAMPLE:

CLOSE 1

Closes the diskette in DRIVE 1. See the OPEN command above. A diskette should be closed before removal from the drive.

5. SAVE <FILENAME>[<DRIVE>][=<ENTRY>]<FROM>[=<AUX.FROM>]<TO> ...

EXAMPLES:

SAVE MYFILE 300 4D5

Saves a memory block (which can be used as a user command) from $0300 to $04D5 on drive 0 (or the default drive as defined by the drive command).

SAVE RALPH_PROG.X: 0 400=2000 400+.100

Saves 100 (decimal) bytes of memory starting at $0400. However, if this file is subsequently loaded by GET or executed by a PROG command, it will be loaded starting at address $2000 instead of $0400, because the "=" designates and auxilliary load address AUX.FROM

6. GET <FILE>[:<DRIVE>][=<AUX.FROM>....]

EXAMPLES:

GET PROG
BET MYTEXT.T=2000

Loads the file MYTEXT.T into memory at address $2000 instead of the load address specified when the file was saved.

7. DRIVE <DRIVE#>

EXAMPLE:

DRIVE 1

Changes the default drive to DRIVE 1. Thereafter, any file names which do not specify a drive explicitly will reference DRIVE 1.

8. ERASE <FILE>[<DRIVE#>]...

EXAMPLES:

```
ERASE  MYDATA
ERASE PROG.C :1 YOURDATA.T HISSTUFF
```

9. GUARD <FILE>[<DRIVE#>]...

EXAMPLE:

```
GUARD MY_GOODIES
```

This sets the software WRITE/ERASE/RENAME/TRUNCATE disable on the named files, which must exist at the time of the command execution.

10. UNGUARD <FILE>[<DRIVE#>]...

EXAMPLE:

```
UNGUARD MY_GOODIES  HISSTUFF:1
```

This removes the software WRITE/ERASE/RENAME/TRUNCATE disable from the named files.

11. FILES <DRIVE#> ...

EXAMPLE:

```
FILES 0
```

This prints the list of file names on DRIVE 0 and a summary. Note that this command is only partially implemented and does not show all that it will eventually.

12. STATUS

EXAMPLE:

```
STATUS
```

Shows a list of the current channel assignments and a summary of the file sapce on the system drive. Note that this commands is not fully implemented in its present form.

13. FORMAT <DRIVE#>

EXAMPLE:

```
FORMAT 1
```

This command formats the disk currently in DRIVE 1. ***** CAUTION *****
This command is not fully implemented and does not give you a chance to change your mind, so be careful! Note that GUARD nor anything else will protect your files from an inadvertent FORMAT command!

14. AIM5

EXAMPLE:

AIM5

This command enters the AIM 65 BASIC ROM from APEX. To save a file from BASIC, type SAVE, and respond with "U" to the prompt from the AIM 65. In response to the APEX prompt "FILE=", enter FILE :DRIVE# , e.g. "STARTREK.5:1". To load, respond with "U" and a filename to the BASIC prompt after a load command. To exit BASIC, depress the ESC key and then the "F3" key to get back to APEX.

15. AIME

EXAMPLE:

AIME

This command enters the AIM 65 EDITOR from APEX. You may save EDITOR files on disk by responding as in BASIC above to the AIM 65 "L" and "R" commands. After exiting to the AIM 65 monitor, "F3" puts you back into APEX.

16. SET <FROM>{<VALUE>} {<STRING>} ...

EXAMPLE:

SET 400 FF

Sets memory address $0400 to $FF.

SET 202D .65 20 $21 "ABC" 0 0

Sets $202D to $41 (65 decimal), $202E to $20, $202F to $21, $2030 through $2032 to $41, $42, $43, and $2033 and $2034 to $00.

17. DUMP <FROM>[<TO>]

EXAMPLE:

DUMP 100

Will display the contents of location $0100 through $0107 in HEX and ASCII. Non-printable ASCII characters (including blanks) will be changed into "." on the ASCII part of the dump. The default number of bytes dumped per line is 8 but can be altered.

DUMP 100 1000

Dumps memory from $0100 to $1007.

18. UNPROTECT

EXAMPLE:

```
UNPROTECT
```

This command removes the write protect from system-reserved memory. Note ... this command is not fully implemented. The write protect is not implemented except for reading from disk to memoyr and from the SET command. UNPROTECT allows you to set system-reserved memory which will otherwise give an error message. The write protect hardware is not used.

19. BEGINOF <CHANNEL> ...

EXAMPLE:

```
BEGINOF 6 7
```

This command positions the files associated with channels 6 and 7 to the begining-of-data in the file.

20. ENDOF <CHANNEL> ...

EXAMPLE:

```
ENDOF 9
```

This command positions the files associated with channel 9 to the end-of-data in the file.

21. GO <ADDRESS>

EXAMPLE:

```
GO 1000
```

This starts execution of a program at address $1000. This command is not fully implemented. Note that entry to the program is made by a JSR so that exectuing a RTS will provide a simple way to retrun to APEX monitor.

APEX
ASSEMBLY LANGUAGE INTERFACE GUIDE

APEX ASSEMBLY LANGUAGE INTERFACE GUIDE

INTRODUCTION

This section discusses methods by which user-written assembly language programs may communicate with the outside world through the APEX-65 operating system, and take advantage of utility functions provided by the operating system. By using the functions described here, the assembly-language programmer can greatly reduce program development time and effort.

Most operating systems provide a degree of support for assembly-language programming by providing the addresses of entry points to certain systems subroutines which the user can call upon to perform I-O or other functions. APEX-65 does not do this, but instead provides a much more powerful tool called the Supervisor Call (SVC). SVC's are usually found only on large mainframe computers and some minis.

In the following discussion, an understanding of 6502 assembly language programming is presumed.

HOW SVCs WORK

A Supervisor call (SVC) consists of a BRK ($00) instruction followed by a one-byte number which tells the Supervisor what function is desired. These SVC numbers are listed in Table 1 . You may think of the SVC as a two-byte subroutine call, where the second byte tells which pre-defined system subroutine is to be called.

In order to use SVCs, a user program must first enable the Supervisor by setting the SVC enable flag, SVCENB, at address $00A0, to $80.

TABLE 1

SUPERVISOR CALL NUMBER FUNCTIONS

(SVCs)

SVC #	Description	Pass Regs.	Returned Regs.
0	Return to APEX Monitor	-	-
1	Return to APEX Monitor w/ inline message output to channel.	-	-
2	Output inline message to channel	-	-
3	Input byte from Channel(X)	X	A,CY
4	Output byte to Channel (X)	X,A	-
5	Input line from Channel (X)	X,U5	A,X,Y,CY
6	Output line to Channel (X)	X,Y,U6	-
7	Output string on Channel (X) (No carriage return)	X,Y,U6	-
8	Decode Hex Value in UO	X,Y,U5	UO,Y,A,CY
9	Decode Decimal Value in UO	X,Y,U5	UO,Y,A,CY
10 ($0A)	Encode Value in UO to Hex	UO,U6,Y,X	Y
11 ($0B)	Encode Value in UO to Decimal	UO,U6,Y,X	Y
12 ($0C)	Define system I-O buffers	-	U5,U6,Y
13 ($0D)	I-O Control Driver Function	X,A,Y	A,X,Y (special)

*NOTE: This is a preliminary list. Many more SVCs will be provided in later versions.

Failure to enable the Supervisor will result in the user program simply returning to the MONITOR at the first intended SVC. Note that the SVCENB flag must be set to $80 by the user-program, and will not work if set by the SET command from the MONITOR.

Usually some type of argument is passed to the Supervisor and/or returned to the user program. Arguments may be passed in three ways:

1. In the 6502 registers;
2. In a set of "pseudo-registers" in page 0;
3. In-line, following the SVC.

Although almost all of the SVCs communicate using only the first two methods, the first SVC we shall examine uses the third method, inline arguments.

DISPLAYING TEXT MESSAGES USING SVC 2

The first use of SVCs we will examine is displaying a message (outputting a string of bytes) on a channel. This function is so essential that it deserves our first attention. It is best illustrated by an example.

SVC EXAMPLE #1: OUTPUT INLINE MESSAGE TO CHANNEL.

PROBLEM: Write a program to display the message "HELLO THERE." on the console.

SOLUTION:

```
SVCENB  =    $A0  ;  ADDRESS OF SVC ENABLE FLAG
;
GREET   LDA  #$80
        STA  SVCENB    ;ENABLE SVCS
```

(Continued...)

```
BRK             ;  SVC...
.BYTE    2      ;  ...#2 = INLINE MESSAGE
.BYTE    2      ;  MESSAGE TO GO TO CHANNEL 2
.BYTE    'HELLO THERE.'
.BYTE    0      ;  TERMINATOR OF MESSAGE AND SVC
RTS             ;  RETURN TO MONITOR OR CALLER
```

EXPLANATION:

The program begins by enabling SVCs. (Note: once enabled, SVCs remain enabled until disabled by writing a $00 into SVCENB. It is advisable to disable SVCs when not needed, although our example program did not do this.) The BRK instruction together with the first .BYTE 2 instruction form the SVC. According to Table 1 and the detailed description of SVC 2, the Supervisor expects to find in-line parameters giving the channel and message. The second .BYTE 2 instruction therefore is the channel number over which the message is to be output. Channel 2 was selected because it is assigned by default to the console display. Of course, it could be assigned to a file or other device. After the channel is designated, the message is defined. The message can be any number of bytes terminated by a $00. This $00 also terminates the arguments passed to the Supervisor. The supervisor will output the message over channel 2 and then return control to the RTS instruction. Note that the Supervisor will restore all registers before returning control to the user program. This is a big benefit during debugging since you may freely place SVC 2 inline-message outputs anywhere in your program without fear of side-effects to registers. For all SVCs, only the registers indicated change; all other registers are restored.

Note that the SVC 2 function does not output any carriage return automatically; if you want to

output control characters, you must include them explicitily in the message, as illustrated below.

SVC EXAMPLE #2: INLINE MESSAGE ON A NEW LINE

PROBLEM: Repeat example 1 but output a carriage return before the message.

SOLUTION:

```
SVCENB  =    $A0
;
GREET   LDA  #$80
        STA  SVCENB     ;  ENABLE SVCS
        BRK             ;  SVC...
        .BYTE    2      ;...#2 = INLINE MESSAGE
        .BYTE    2      ; CHANNEL 2
        .BYTE    13     ;  13=$OD= ASCII CARRIAGE RETURN
        .BYTE    'HELLO THERE.'
        .BYTE    0      ;  TERMINATOR
        RTS
```

A final caution on SVC 2 usage --- the most common errors are (1) forgetting to enable SVCs; (2) Forgetting the channel argument; (3) Forgetting the zero-byte terminator.

PASSING ARGUMENTS TO SUPERVISOR IN 6502 REGISTERS

The above examples passed arguments to the Supervisor in-line. A much more common method of parameter-passing is the use of 6502 registers.

SVC EXAMPLE 3: CHARACTER INPUT-OUTPUT

PROBLEM: Write a program which reads a stream of bytes from channel 5 until a "." character is encountered, or End-of-File is encountered. Display a message indicating which of these two events happened. Assume channel 5 has been previously assigned to a valid file or device.

SOLUTION:

```
SVCENB  =    $A0
;
STRMIN  LDA  #$80
        STA  SVCENB     ;  ENABLE SVCS
NEXTCH  LDX  #5         ;  CHANNEL 5
        BRK             ;  SVC...
        .BYTE    3      ;  ...#3 = INPUT BYTE FROM CHANNEL(X)
        BCS  EOFENC     ;  BRANCH IF END-OF-FILE ENCOUNTERED
        CMP  #'.        ;  ELSE SEE IF "." ENCOUTNERED
        BNE  NEXTCH     ;  IF NOT, KEEP LOOKING
        BRK             ;  ELSE SVC...
        .BYTE    2      ;  ...#2 = OUTPUT INLINE MESSAGE
```

(Continued...)

```
        .BYTE     2     ;  ON CHANNEL 2
        .BYTE     13    ;  CR
        .BYTE     '"." ENCOUNTERED.'
        .BYTE     0     ;  TERMINATOR
        RTS
EOFENC  BRK             ;  SVC...
        .BYTE     2     ;  #2 = OUTPUT INLINE MESSAGE
        .BYTE     2     ;  ON CHANNEL 2
        .BYTE     13    ;  CR
        .BYTE     'E-O-F ENCOUNTERED.'
        .BYTE     0     ;  TERMINATOR
        RTS
```

EXPLANATION:

This program illustrates a number of aspects of SVC usage. The line labelled NEXTCH is used to load the channel number into X, which is in turn passed to the Supervisor. Table 1 and the descriptions of each SVC tell what registers are needed for passing arguments to the Supervisor, and which registers are used to return results to the calling program. Here we learn that the SVC 3 function returns the character read from the channel in the A register, and sets the Carry flag only if End-of-File was encountered. End-of-File is an important concept. The EOF flag (CY) is set only if no more characters can be read from the channel. This occurs when CNTRL-Z is entered from the keyboard (assuming that

channel 5 was assigned to the keyboard), or when no data remains in the file (assuming channel 5 was assigned to a file). You should always take care to check for End-of-File when doing input, so that your programs are device-independent and can therefore read from keyboard, file or another device with equal ease. In our example, once we have ascertained that EOF was not encountered, we check to see if the "." character was encountered. If not, we read another character. The SVC 2 is used to display an inline message once one of the two possible terminating conditions is reached.

WHY USE SVCS?

Before examining SVC usage in more detail, lets review why SVCs are preferred to System Subroutine calls:

1. SVCs are address-independent. This means that future system upgrades which change addresses of system routines will not adversely affect programs using SVCs. It also means that, for example, that a program on an AIM computer with APEX at $8000 can be moved to a KIM system with APEX at $E000 and run without any modification. This would be impossible using subroutine calls.

2. SVCs use less memory. Two bytes is cheaper than three!

3. SVCs handle registers in a uniform fashion.

Registers are preserved by the Supervisor except when returning values to the calling program. This saves a lot of unnecessary saving and restoring of registers.

4. SVCs are easier to debug. If an error is detected by the supervisor, the program will abort and APEX will display the exact address of the offending SVC, the values of all registers at the time of the SVC, and an error message explaining the difficulty. More than 50 error messages are defined to isolate errors. Illegal or unimplemented SVCs are also trapped in the same manner.

PASSING ARGUMENTS TO SUPERVISOR IN PSEUDO-REGISTERS

Often addresses or other 16-bit information needs to be passed to Supervisor for processing a function. The 8-bit 6502 registers are not adequate for this purpose, so a set of eight pseudo-registers are provided in page zero, as shown in figure 1. "Registers" U0 through U6 are 16 bits wide; U7 is 24 bits wide and is dedicated for file positioning, as we shall see later. Note that if SVCs are not enabled, these pseudo-registers are not used for any purpose whatsoever by the system and may be used as ordinary memory. For many SVCs, the Supervisor expects to find certain values passed in the pseudo registers. Most I-O functions, for example,

PSEUDO- REGISTERS

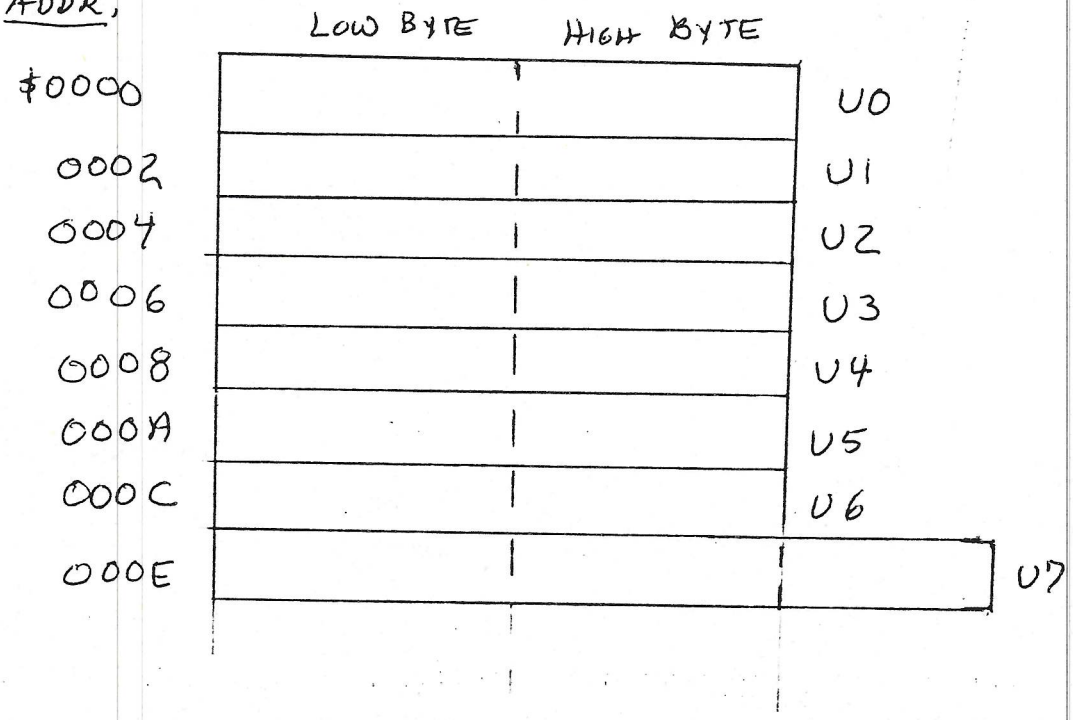

Figure 1

pass the address of the input buffer in U5 or the address of the output buffer in U6. The exact requirements are explained with each SVC description. Certain SVC functions also return information in the pseudo-registers. For example, SVC 12 ($0C) sets U5 and U6 to the addresses of the default system input and output buffers, respectively.

TEXT PROCESSING

A large class of problems deal with input and output of strings or lines of characters. Several SVCs are provided for support. The 6502 Indirect, Y addressing mode is utilized heavily in these applications. In general, U5 (for input) or U6 (for output) addresses the start of a buffer containing the current line of interest, and the Y register is used to address the particular character of interest within the line. Normally the default system line buffers are used, but the user may select other buffers if desired. An example will help introduce text-processing concepts.

SVC EXAMPLE 4: LINE-ORIENTED I-O

PROBLEM: Write a program to copy lines of input text from channel 5 to channel 6 until End-of-File is reached.

SOLUTION:

```
SVCENB  =    $A0
U5      =    $0A        ;  PSEUDO REG. U5
U6      =    $0C        ;  PSEUDO REG. U6
COPY56  LDA  #$80
```

(Continued...)

```
        STA  SVCENB          ;  ENABLE SVCS
        BRK                  ;  SVC...
        .BYTE     12         ;  12 = SET U5,U6 TO SYS. BUFS.
NEXT    LDX  #5              ;  CHANNEL 5 FOR INPUT
        BRK                  ;  SVC...
        .BYTE     5          ;  ...5 = INPUT LINE
        BCS       EOFENC     ;  BRANCH IF END-OF-FILE
        TAY                  ;  ELSE SAVE CHAR. COUNT IN Y...
        TAX                  ;  ...AND X TOO
LOOP    LDA  (U5),Y          ;  COPY INPUT BUF. CONTENTS...
        STA  (U6),Y          ;  ...TO OUTPUT BUF.
        DEY
        BPL  LOOP            ;  ...UNTIL WHOLE LINE COPIED
        TXA                  ;  THEN RECALL CHAR. COUNT
        TAY                  ; ...TO Y
        BRK                  ;  SVC...
        .BYTE     6          ;  ...6 = OUTPUT LINE
        JMP       NEXT       ;  REPEAT FOR NEXT LINE
EOFENC  RTS                  ;  RETURN ON END-OF-FILE
```

EXPLANATION:

Since this program is substantially more complex than would be necessary had we used the byte-oriented SVCs already illustrated, you might well wonder why line-oriented SVCs were used instead. The reason is that inputting an entire line allows the user to use the Backspace (Del), CNTRL-X, etc. keys to edit the line, assuming channel 5 is assigned to the Console (keyboard). SVC 3 returns control to the calling program immediately each time a key is pressed; SVC 5 does not return from the Supervisor until an entire line has been installed in the buffer (and possibly was edited using Backspace, etc.).

The example program starts by setting up U5 and U6 as buffer addresses, using the SVC 12 function.

An SVC 5 is used to input a line into the buffer addressed by U5, and End-of-File is checked. Note that the SVC 5 function returns the character count in the A register and the Y register as 0 (therefore ready to index the first character of the line). The input line is copied character by character into the output buffer, and the line is output over channel 6. Note that the SVC 6 function requires the character count to be passed in the Y register.

An alternative method to copying the buffer character-by-character would be to simply copy the contents of U5 to U6 and then issue an SVC 6. Normally, however, you will want to perform some function on the input rather than just copying it to another channel anyway, so the method illustrated is more versatile.

NUMERIC I-O USING SVCS

You may be surprised to note that no SVCs are provided to directly input or output numeric values. Instead, a combination of two SVCs must be used to perform this function. This turns out to be more versatile since you can then input or output values from memory as well as channels.

All I-O is assumed to consist of ASCII characters for purposes of this discussion. Decoding is the operation of scanningna string of numeric characters and returning the value they represent (in binary). Encoding is the inverse function, which accepts a binary value and generates a string of ASCII characters in a buffer which represent its value. The following examples will illustrate these operations.

SVC EXAMPLE 5: READ HEXADECIMAL INPUT

PROBLEM: Write a subroutine to read a hexadecimal number from channel 5 and return its value in Pseudo-register U0 (low byte in $0000, high byte in $0001).

SOLUTION:

```
SVCENB  =    $A0
;
HEXIN   LDA  #$80
        STA  SVCENB          ;  ENABLE SVCS
        BRK                  ;  SVC...
        .BYTE     12         ;  12 = SELECT SYS. BUFS.
        LDX  #5              ;  CHANNEL 5
        BRK                  ;  SVC...
        .BYTE     5          ;  INPUT LINE ON CHANNEL 5
        BRK                  ;  SVC...
        .BYTE     8          ;  ...8 = DECODE HEX VALUE TO U0
        RTS
```

EXPLANATION:

The enabling of SVCs and selection of default buffers should be familiar. In practice, these two functions would only be performed once during program initialization, and would probably not be part of the HEXIN routine, making it even simpler. The SVC 5 function inputs a line into the buffer addressed by U5, as seen before. The SVC 8 function searches the buffer (starting with the character pointed to by Y, which is 0 in our case since SVC 5 always returns Y=0) for a hex value. Note that any number of leading blanks may preceed the number, and that the number may have any number of characters, so long as the represented value does not exceed $FFFF. For example, "00D7 ", " 0D7 ", and "D7 " will all

be acceptable. The SVC 8 function keeps scanning the number until a non-hex character is encountered. Thus, for example, " 2B7,2 " will return U0 = $02B7. When control is returned to the calling program, the Y register points to the delimiter (for example, the comma above), and the A register holds the delimiter encountered. This can be very useful when scanning a line containing several values. In addition, the Carry flag is returned to the caller as a "valid data encountered" flag. Although our sample routine did not include it, it is easy to check the CY and if not set, then no valid hex digits were in the line prior to the delimiter (or end-of-line). Note that if end-of-line is encountered, the A register will contain $00.

TO BE CONTINUED - - -

APEX-65 INTERIM MEMORY MAP

NOTE: This is a pre-release version of APEX-65.
Not all features have been implemented.

Address	Description
$0000-0010	Pseudo-registers U0 through U7. Used only if SVCs are enabled, otherwise, user-RAM.
$00A0	SVCENB. SVC-enable flag, settable by user programs. See Assembly Language Interface guide.
$00A1-00FF	Page 0 system-reserved RAM, and AIM RAM.
$0100-01FF	Stack and AIM RAM.
$0200-4800	User RAM. (COPYF utility uses $2000-2900)
$4800-53FF	Interim System Ram for "overlays".
$5400-5BFF	Interim Visible Memory Terminal (VMT) driver routines, if supplied.
$5C00-5CFF	APEX default I-O line buffers(input buffer at $5C00-5C52, output buffer at 5C53).
$5D00-5FFF	Optional DMA buffers if system is modified to allow more than three simultaneous files to be active. Otherwise, unused.
$6000-7FFF	Visible memory board, if VMT is used.
$8000-85FF	Protected DMA buffers for system.
$8600-9EFF	Protected APEX-65 system nucleus.
$9F00-9FE7	Bootstrap loader ROM.
$9FE8-9FFF	Disk Controller I-O ports.
$A000-FFFF	AIM-65 ROMs.