VERY PRELIMINARY

APEX-65 OPERATING SYSTEM

USER'S MANUAL

Release 0.2: AIM (Preliminary)

July 8, 1980

Micro Technology Unlimited
2806 Hillsborough Street
P.O. Box 12106
Raleigh, NC 27605

(919) 833-1458

Micro Technology Unlimited
2806 Hillsborough Street
P.O. Box 12106
Raleigh, NC 27605
(919) 833-1458

AUG. 1 2 1980

APEX memory requirements:

0000 - 0010	Pseudo register only needed with SVC's.
00A0 - 00CF	Zero page Ram used by APEX (asm uses 00C8-00E)
0100 - 01FF	stack
(0200 - 1FFF)	(would be used in an AIM system.
2000 - 47FF	User ram ~ 10K (VMB 2000-3FFF, FDC 4000-47FF
"USER" RAM FDC 4000-5FFF: 4800 - 50FF	Interim System ram
5100 - 5BFF	User RAM 2.5K
5C00 - 5CFF	default system I/O buffers
5D00 - 5FFF	optional DMA Buffers for more than default # of buffers.
(6000 - 7FFF)	(empty - recommended loc. for VMB).
SYSTEM RAM - FDC 8000-9FFF: 8000 - 85FF	Protected system DMA buffers)
8600 - 9EFF	Protected APEX System nucleus + RAM
9FF0 - 9FFF	ROM + Disk controller I/O Ports.
A000 - FFFF	24K unused space.

APEX I/O (Console) patches:

IN	0200	4C	5A	1E	JMP GETCH ($1E5A)
OUT	0203	48			PHA save A
	0204	C9	0D		CMP #$0D to <CR>
	0206	D0	05		BNE NORMAL if not a <CR>
	0208	20	A0	1E	JSR OUTCH output the <CR>
	020B	A9	0A		LDA #$0A prepare to output a <LF>
NORMAL	020D	20	A0	1E	JSR OUTCH output <LF> or normal char.
	0210	68			PLA restore orig char
	0211	60			RTS

```
133 00D0                        .PAGE 'JUMP TABLE'
                                .=    X'8600    ;*** TABLE ORIGIN ***
                        ;
                        ;       JUMP-TABLE TO USER AND SYSTEM DEPENDENCIES
                        ;
                        ;       DEFINITIONS...
                        ;
                        ;       JCHIN:  JUMP TO USER'S CONSOLE-CHARACTER-IN SUBROUTINE. MUST
                        ;       RETURN ASCII CHARACTER IN A WITH BIT 7 = 0 (UNLESS KEYBOARD
                        ;       MASK, KBMASK, HAS BEEN ALTERED FROM ITS DEFAULT 0 VALUE FOR
                        ;       A SPECIAL KEYBOARD).  USER SUBROUTINE CAN DESTROY ANY REGISTERS
                        ;       BUT MUST RETURN WITH STACK INTACT.  THE DRIVER DOES NOT HAVE
                        ;       TO ECHO THE CHARACTER TO THE DISPLAY.  THIS IS DONE BY APEX
                        ;       (UNLESS THE HALF-DUPLEX FLAG, HFDPLX, HAS BEEN CHANGED FROM
                        ;       ITS DEFAULT OF 0 TO $80).
                        ;
                        ;       JCHOUT:  JUMP TO USER'S CONSOLE-CHARACTER-OUTPUT SUBROUTINE.
                        ;       CHARACTER TO BE OUTPUT IS PASSED IN A REGISTER, WITH BIT 7=0.
                        ;       THIS ROUTINE CAN CLOBBER ANY REGISTERS BUT MUST RETURN WITH
                        ;       THE STACK INTACT.  Must preserve A
                        ;
                        ;       JCHIF:  JUMP TO USER ROUTINE TO DETERMINE IF A KEY IS DEPRESSED
                        ;       ON THE CONSOLE KEYBOARD.  IF NOT, THE ROUTINE SHOULD SET BIT 7
                        ;       OF THE A REGISTER TO 1 (ASSUMING DEFAULT VALUE OF KBMASK) AND
                        ;       RETURN.  THE REMAINING BITS OF A ARE "DON'T CARE".  IF A KEY
                        ;       IS DEPRESSED, THEN THE ASCII KEY SHOULD BE RETURNED IN A WITH
                        ;       BIT 7 SET TO 0 (AGAIN, ASSUMING DEFAULT SETTING OF KBMASK).
                        ;       THIS ROUTINE CAN CLOBBER ANY REGISTERS BUT MUST RETURN WITH
                        ;       STACK INTACT.  IF YOUR CONSOLE DEVICE CANNOT SUPPORT THIS
                        ;       FEATURE, THEN JUST SET BIT 7 OF A TO 1 AND RETURN.
                        ;
                        ;       JCINIT:  JUMPS TO USER'S CONSOLE-INITIALIZATION ROUTINE.
                        ;       THIS ROUTINE WILL BE CALLED BY THE SYSTEM ON STARTUP PRIOR
                        ;       TO EXECUTING THE COMMANDS ON THE "STARTUP.J" FILE.  THEREFORE,
                        ;       IF THE DRIVERS FOR THE CONSOLE ARE LOADED BY A COMMAND IN THE
                        ;       STARTUP.J FILE, IT WILL BOMB THE SYSTEM IF YOU JUMP TO THE
                        ;       INITIALIZATION ROUTINE VIA JCINIT.  FOR THIS SITUATION, INCLUDE
                        ;       EXECUTION OF THE INITIALIZATION ROUTINE IN THE "STARTUP.J"
                        ;       COMMAND INSTEAD.  JCINIT DEFAULTS TO A RTS.  IT IS NORMALLY
                        ;       USED WHEN ROM-BASED DRIVERS ARE USED FOR THE CONSOLE WHICH
                        ;       CAN BE INITIALIZED BEFORE "STARTUP.J" IS READ BY THE SYSTEM.
                        ;
                        ;       JSINIT:  JUMP TO SYSTEM-DEPENDENT CODE FOR PARTICULAR SYSTEM.
                        ;       THIS ROUTINE IS PROVIDED BY MTU FOR EACH SYSTEM (AIM, KIM, ETC)
                        ;       AND IS NOT NORMALLY ALTERED BY THE USER.
                        ;
178 8600 4C9088 JCOLD:  JMP   COLD      ;JUMP TO COLD START (CANT BE RE-ENTERED!)
179 8603 4C3A8A JWARM:  JMP   COMD      ;JUMP TO WARM START ENTRY POINT
180 8606 4C~~B99D~~ JCHIN:  JMP   ~~AIMKB~~ $0200 ;TO CONSOLE CHARACTER-INPUT SUBROUTINE
181 8609 4C~~FCEE~~ JCHOUT: JMP   ~~AIMPR~~ $0203 ;TO CONSOLE CHARACTER-OUTPUT SUBROUTINE
182 860C 4CE79D JCHIF:  ~~JMP~~   ~~AIMKD~~       ;TO CONSOLE KEY-DEPRESSED TEST SUBROUTINE
183 860F 60     JCINIT: RTS                 ;TO CONSOLE INITIALIZATION. NOT NORMALLY USED.
184 8610 EA             NOP
    8611 EA             NOP
    8612 4C969D JSINIT: JMP   AIMINT    ;TO SYSTEM-DEPENDENT CODE INITIALIZATION.

                                LDA #$80
                                RTS
```

Micro Technology Unlimited
841 Galaxy Way
Manchester, NH 03103
(603) 627-1464

AUG. 1 1 1980

August 4, 1980

Mr. M. Challifer
Box 61B
S. Addison, Me. 04606

Dear Mike,

There seems to be some confusion about your status with regards to the APEX-65 software. I was under the impression, after talking to you on the phone in early July, that you were willing to wait for the initial release of the KIM version of APEX slated for August. Dave tells me you were expecting a custom-made version of the Interim AIM package adapted for KIM. If I have dropped the ball here, I apologize.

I am enclosing an AIM diskette with the Interim package on it which has been patched to run on the KIM. You are the only recipient of this version. You will therefore unfortunately have to address you disk controller as though it were an AIM, with system RAM at $8000 and user RAM at $4000. In addition, you will need RAM from $2000 to $3FFF in order to run the FORMAT and COPYF utilities. You can't run DUP because it needs RAM form $0300 to $03FF. - ? should mean 1FFF?
You will need to setup jumps to your keyboard/display I-O drivers as follows before booting up:

$0200 JMP to your keyboard subroutine
$0203 JMP to your display subroutine

See the attached comments for details of routine requirements, and in addition please not that your character-out driver will need to supply a line feed each time a carriage return is received (because the AIM does this). For example the following could be appended to the front of your normal output driver routine:

```
PATCH   CMP    #$0D      ;CARRIAGE RETURN?
        BNE    NORMAL    ;IF NOT BRANCH
        JSR    NORMAL    ;ELSE OUTPUT CARRIAGE RETURN
        LDA    #$0A      ;FOLLOWED BY LINE FEED
NORMAL...
```

Be advised that the version of the system you have is now a full 6 weeks of development out of date, but matches the interim documentation. The release version will differ substantially including addresses, commands and command syntax, utilities and SVCs. It is also much, much improved. Please note the Interim memory map in the interim manual.

Naturally you'll receive the regular KIM version as soon as its released.

AUG. 1 1 1980

I am very interested in your efforts to interface UCSD Pascal to the KIM. For that purpose, I think that the disk routines supplied directly in the Controller manual will be better suited than using APEX, since UCSD has their own system with adaptable JSRs to do the SEEK, READ, WRITE, etc.

Please feel free to call me with any questions or comments at (919)-833-1458. I regret the delays which you as a "charter" member of the APEX-65 owners have had to put up with; I feel that you will agree that the wait was worth it when you receive the distribution disk.

Sincerely,

Bruce D. Carbrey
Software Manager

DISK RELIABILITY - ITS REALLY UP TO YOU!

Floppy disks provide an excellent low-cost storage media for programs and data, with very high reliability. When used with the high-quality K-1013 Controller and APEX-65 Software, the incidence of data read-write failures should be virtually nil, provided a few simple handling precautions are observed. The way floppy disks are handled and stored will materially affect their lifetime and reliability. During the many months of development of the APEX-65 system, we did not experience a single (hard) read error during hundreds of hours of use, following the rules below:

1. Always keep the diskette in its protective envelope. Get in the habit of removing a disk from the drive directly to the paper envelope. Dust particles look like a boulder to a recorded bit!

2. Do not touch the exposed recording surface of the disk. Fingerprints are a killer, too.

3. Do not bend the disk. Its called a flexible disk, but you don't have to prove it.

4. Do not write on the disk directly with pen or pencil. Use only a soft-tip marker, and write only in the label area, or you may etch the recording media.

5. Avoid exposure to harsh environments such as extreme heat or cold.

6. Keep the diskette away from strong magnetic fields.

What kind of disks should be used?

Any quality soft-sectored 8 inch floppy disk may be used, such as Dysan, IBM, or equivalent.

WHAT IS PRELIMINARY ABOUT THIS VERSION?

APEX-65 version 0.5 is a preliminary release of the distribution operating system. It is being distributed now so that users may begin making effective use of their K-1013 disk controller systems without having to wait unduly long for the complete release.

This version of APEX-65 is preliminary primarily in the sense that a large number of features are not yet implemented, and many of the existing features are not fully implemented. From the user's viewpoint, the two most significant shortcomings of this version are:

1. This version requires that a block of memory outside the System 8K of ram on the controller be dedicated to APEX-65. This is required because many of the commands and routines will be implemented as overlays in the final version, but overlays have not yet been implemented. Therefore about 3K of additional memory is needed for this version as compared to the distribution system.

2. The SYSGEN utility program is not yet available. When it is available, the SYSGEN program will provide a conversational means of "customizing" APEX-65 to your particular system environment. For now, very little adustment is possible for individual system requirements, and little guidance is provided on how to make such modifications.

In addition, a number of commands, utility programs and SVC's are not available or not fully implemented. Some of the more major of these missing or parital features include:

1. Documentation is incomplete and somewhat sketchy. Only the first few SVCs are described. Table and Figure numbers are missing or incorrect. No index or Table of Contents is provided.
2. The utility program for generating new disks, backup disks, and copying the system does not exist. Instead, an "Interim copy procedure" must be used.
3. No test is made for defective sectors during FORMATting.
4. No disk "rebuild" utility is provided (with proper care of disks, you should have no need of one).
5. The DIR command does not exist. This is an expanded version fo the FILES command, which will show the name, size, creation date, etc., for all or selected files.
6. RENAME command does not exist. To rename a file now you must use COPYF to make a duplicate with another name and then DELETE the original.
7. The VERIFY command does not exist. This will compare a block of memory to a disk file.
8. A number of SVC's are not yet implemented.
9. Other minor features and Utilities are omitted.

INTERIM HARDWARE REQUIREMENTS

(AIM-65 VERSION)

1. AIM-65 Single Board Computer with power supply.

2. MTU K-1013 Floppy Disk Controller Board, jumpered in the following configuration (which is the default shipping configuration):

   8K of System RAM addressed $8000 to $9FFF;

   8K of User Ram addressed $4000 to $5FFF;

   Floppy Disk Controller Interrupt: Disabled.

3. One to four full size (8 inch) floppy disk drives with power supply and cabling, as described on page 3 of the K-1013 manual.

4. 16K of additional RAM addressed $0000 to $3FFF. An easy way to achieve this is to use one MTU K-1016 memory board addressed at $0000, and remove all the AIM on-board 2114 RAMs from their sockets. This memory is used by the system for the COPYF, FORMAT and DUP utility programs.

INTERIM STARTUP PROCEDURE FOR APEX-65

(AIM-65 VERSION)

1. Insure that your system has been properly setup with memory at the required addresses and a properly operating disk controller addressed with system RAM at the correct location, as indicated in Table 1.

2. Using the AIM MONITOR, load the "BOOT " program into memory from cassette. Later you will be shipped a bootstrap loader PROM chip for easy startup, but for now this file must be loaded every time you wish to startup APEX-65.

3. Insure that the printer is on. APEX-65 is designed for console devices with more than one line viewable at a time; if the printer is not on, you will need a course in speed reading to read the 20-character display before the next line is displayed!

4. Insert the distribution disk into drive 0 and close the door.

5. Again using the AIM MONITOR, begin execution at address 0000. The disk should show immediate signs of activity lasting about 2 seconds, after which several lines should be displayed containing only the character ">". The ">" is the APEX prompt for input commands. When APEX-65 is startup up, it first reads any commands that exist on a file called STARTUP.J and then reads commands from the Console (keyboard). The first few ">" characters are prompts issued while APEX reads the STARTUP.J file. When the disk activity stops and the final ">" appears on the screen, APEX-65 is up and waiting for a command. You are now in the APEX-65 MONITOR program. Anytime the ">" character appears in column 1, APEX has completed a command and is waiting for a new command.

6. You may now enter any legal command. However, if this is your first-time power-up, proceed with the following steps.

7. Type

    FILES

with a carriage return. Every APEX-65 command must be terminated with a carriage return. The names of the files on the disk should be displayed, followed by a summary of remaining disk space. To temporarily pause during the display, depress the CNTRL key. Depressing any key except CNTRL will continue the display.

8. If all has gone smoothly so far, make a backup copy of the disk as described in the next section. Do not proceed unless all is well, however.

INTERIM APEX BACKUP PROCEDURE - SINGLE DRIVE

USE THIS PROCEDURE IT YOU HAVE ONLY ONE DISK DRIVE:

1. Be sure to observe the Copyright provisions described in the front of this manual.

2. Bring up the system as described in the previous section.

3. Before the APEX-65 system or any files can be copied onto a new disk, the disk must be FORMATTED. FORMATTING irrevocably erases all material on the disk (including LOCKed Files and the Operating System memory image). All new diskettes must be FORMATTED using the FORMAT Utility, even if you buy disks which come pre-formatted. Disks distributed with software on them are already formatted. Never format the APEX-65 distribution disk!

4. With the distribution disk still in the drive, type

```
FORMAT 0
```

which tells Apex to Execute the FORMAT program for drive 0. The system will respond with

```
VOL. SER. #=?
```

5. REMOVE THE DISTRIBUTION DISKETTE! Insert the new disk to be formatted into the drive and close the door. The System is now waiting for you to enter a Volume Serial Number (VSN) which is a four digit hexadeciaml number used to uniquely identify each disk. The current system does not check the VSNs but does write the VSN on the disk at FORMAT time. Future versions will do some verification using the VSN. It is therefore a good idea to give a unique VSN for each disk, and to also put the VSN on the disk label with a magic marker (not pen or pencil!) for visual reference.

6. Type in any desired 4 digit hex number for the VSN. The System will respond,

```
DISK TO BE FORMATTED IN DRIVE 0
ARE YOU READY?=
```

7. Insure that you have really removed the Distribution disk and put in the new disk then type

```
YES
```

Any answer not starting with "Y" will abort the program. The disk will show activity for about a half minute, after which the Monitor prompt will be displayed. If you get a "FORMATTING ERROR" message, it probably means you are trying to format a hardware write-protected disk. Cover up the write-protect hole.

8. Type

CLOSE 0

9. Remove the newly formatted disk, and re-insert the distribution disk. Type

OPEN 0

10. You are now ready to copy the operating system and related files, one at a time. Type

DUP

which executes the Single-drive file duplicator Utility. The system will respond with

```
PUT SOURCE DISK IN.
FILE (OR CR IF DONE)?=
```

11. Since you already have the desired source disk for the copy in the drive, just type

APEX.Z

which is the name of the first file to be copied.

IMPORTANT: WHEN MAKING COPIES OF THE SYSTEM, THE NEW DISK MUST BE FORMATTED AND THE FILE APEX.Z MUST BE THE FIRST FILE COPIED TO IT. FAILURE TO DO SO WILL MEAN THAT THE NEW DISK WILL NOT BE ABLE TO BE BOOTED UP. IT IS NOT SUFFICIENT TO MERELY DELETE ALL OLD FILES AND THEN COPY APEX.Z; IT MUST BE DONE IMMEDIATELY AFTER THE FORMAT.

The System will respond

```
PUT DEST DISK IN,
CR WHEN READY?=
```

12. REMOVE THE DISTRIBUTION DISK, and insert the new destination disk for the copy.

13. Type a Carriage return when you are ready. The System will copy the file onto the new disk and then respond

```
PUT SOURCE DISK IN.
FILE (OR CR IF DONE)?=
```

14. Swap disks again so you have the Distribution disk in.

15. Repeat steps 11 through 14 above, but use the following file names instead of APEX.Z in step 11:

OVL.Z
SYSERRMSG.Z
STARTUP.J
FORMAT
DUP

These are all the files that are necessary for operation of the system, and generation of further backups. Using the same procedure, you may copy any other files desired.

16. When all desired files have been copied, just enter a carriage return instead of a filename in step 11 above. You may now use the new disk just like the distribution disk, and should be able to boot the system in the same fashion using it. We recommend you put away the distribution disk for safe keeping and only work with the backup. Be sure to affix the Copyright notice to the disk.

You can use the DUP utility at any time to transfer files between disks, once they are formatted.

CAUTION: IF YOU COPY A FILE TO A DISK WHICH ALREADY HAS A FILE BY THE SAME NAME, THE OLD FILE WILL SIMPLY BE OVERWRITTEN WITHOUT ANY ERROR MESSAGE.

NOTE: Some files may be too long to be copied by the DUP program in a single pass; in this case, the message

PUT SOURCE DISK IN.
CR TO CONTINUE=

will be displayed. Just enter a carriage return to continue copying the rest of the file.

INTERIM APEX BACKUP PROCEDURE - 2, 3, OR 4 DRIVES

USE THIS PROCEDURE IT YOU HAVE MORE THAN ONE DISK DRIVE:

1. Be sure to observe the Copyright provisions described in the front of this manual.

2. Bring up the system as described in the previous section.

3. Before the APEX-65 system or any files can be copied onto a new disk, the disk must be FORMATTED. FORMATTING irrevocably erases all material on the disk (including LOCKed Files and the Operating System memory image). All new diskettes must be FORMATTED using the FORMAT Utility, even if you buy disks which come pre-formatted. Disks distributed with software on them are already formatted. Never format the APEX-65 distribution disk!

4. With the distribution disk still in drive 0, insert the new disk to be formatted in drive 1 and close the door. Type

```
FORMAT 1
```

which tells APEX to Execute the FORMAT program for drive 1. The system will respond with

```
VOL. SER. #=?
```

5. The Program is now waiting for a Volume Serial Number (VSN), a four digit hexadeciaml number used to uniquely identify each disk. The current system does not check the VSNs but does write the VSN on the disk at FORMAT time. Future versions will do some verification using the VSN. It is therefore a good idea to give a unique VSN for each disk, and to also put the VSN on the disk label with a magic marker (not pen or pencil!) for visual reference.

6. Type in any desired 4 digit hex number for the VSN. The System will respond,

```
DISK TO BE FORMATTED IN DRIVE 1
ARE YOU READY?=
```

7. Type

```
YES
```

Any answer not starting with "Y" will abort the program. The disk will show activity for about a half minute, after which the Monitor prompt will be displayed. If you get a "FORMATTING ERROR" message, it probably means you are trying to format a hardware-write-protected disk. Cover up the write-protect hole.

8. You are now ready to copy the operating system and related files, one at a time. Type

    COPYF APEX.Z

which executes the File-Copy utility program for muli-drive systems. APEX.Z is the name of the first file which must be copied, if the new disk is to be used as a system disk (therefore "bootable" in drive 0). COPYF copies from drive 0 to drive 1 by default.

IMPORTANT: WHEN MAKING COPIES OF THE SYSTEM, THE NEW DISK MUST BE FORMATTED AND THE FILE APEX.Z MUST BE THE FIRST FILE COPIED TO IT. FAILURE TO DO SO WILL MEAN THAT THE NEW DISK WILL NOT BE ABLE TO BE BOOTED UP. IT IS NOT SUFFICIENT TO MERELY DELETE ALL OLD FILES AND THEN COPY APEX.Z; IT MUST BE DONE IMMEDIATELY AFTER THE FORMAT.

9. When the Monitor Prompt appears, the file is copied (it should take about four seconds). Repeat step 8 above for the following files:

    OVL.Z
    SYSERRMSG.Z
    STARUP.J STARTUP.J
    COPYF.C
    FORMAT.C

These are all the files that are necessary for a System disk. If the disk will not be used in drive 0, it is not necessary to copy any files. However, it is generally a good idea to put the system files on all disks for convenience.

10. Be sure to affix the Copyright notice to any backup disks.

11. Once the new disk is Formatted, it can be opened and used on drives except drive 0. If the disk has been formatted and the system files have been copied onto it, it can then be used in any drive. When used in drive 0, it will be the system disk. We suggest you put away the distribution disk for safekeeping and only use backups for day-to-day work. The COPYF program can be used anytime to transfer files between formatted disks.

CAUTION: IF YOU COPY A FILE TO A DISK WHICH ALREADY HAS A FILE WITH THE SAME NAME, THE OLD FILE WILL SIMPLY BE OVERWRITTEN WITHOUT ANY ERROR MESSAGE.

APEX-65 SYSTEM CONCEPTS

The APEX-65 Operating System is a powerful computer program for managing the resources of a 6502-based microcomputer. In particular, it provides a convenient method for storing and retrieving other programs and data on floppy disk storage. The user will normally interact with APEX-65 principally through two built-in facilities:

1. The SYSTEM MONITOR;
2. The SVC PROCESSOR.

The SYSTEM MONITOR provides a simple method for the user to interact directly with APEX-65 by typing commands from the keyboard (hereafter called the CONSOLE). These COMMANDS are most often used to initiate execution of other programs, examine the status of various system attributes (such as the names of files present on floppy disk), or to alter the status of the system (for example, adding a new program to floppy disk). The APEX SYSTEM MONITOR is initiated automatically when the system is "booted" up; a prompting message is issued on the console display, and the system awaits user commands. These commands may be either built-in or user-defined. Both types of commands are described in detail later.

All users of the APEX-65 system will utilize the functions of the SYSTEM MONITOR to some degree. In addition, however, programmers will also wish to utilize the facility which permits programs to interact with the operating system. For example, programmers will wish to be able to display messages on the display and input characters from the keyboard. In most conventional microcomputer systems, support for this type of activity is provided in a limited sense by making available to the programmer a list of addresses of system subroutines which perform the basic input-output functions essential to programming; the programmer can use these functions by writing a CALL (JSR) to the appropriate system subroutine from within the application program.

APEX-65 provides a different, higher-level method of support for user-written programs called the SUPERVISOR CALL (SVC). Although not found on microcomputers, SVC's are found extensively on the finest mainframe computers. Instead of a JSR instruction to a system routine, the SVC consists of a BRK ($00) instruction followed by a data byte which identifies the function desired. There are several advantages to this method, which are described later; the most important advantage of the SVC is that SVCs are address-indepenedent. This means that a program using SVCs will run without modification regardless of the location of the operating system. Thus, for example, a program written on an AIM with APEX-65 at $8000 can be run without modification on a KIM with APEX at $E000. SVCs are discussed in detail in a later section.

CHANNELS

APEX-65 provides a capability not normally found on micros called device-independent I-O. Device-independence means that a program (or SYSTEM MONITOR command) can perform input or output to or from a variety of devices or disk files without modification. For example, a program which normally displays its output on the system console device can be re-run without such that the output is directed instead to a printer, without any modification to the program. Input or output can also be re-directed to a file on disk. This provides an exceptionally powerful capability. The devices to be used can be selected by a simple MONITOR command, or by an executing program itself.

The key to device-independence in APEX-65 is the use of software I-O Channels. The SYSTEM MONITOR and programs communicate with the outside world over channels. At any time, these channels may be associated with a given device or file. The standard APEX-65 system has ten channels, numbered 0 through 9. Each of these channels may be used to send or receive data, or both. For example, a printer is normally an output-only device, but the system console (terminal) can both send and receive data. A control function can also be associated with each channel for more complex devices; this capability is discussed in a later section.

Certain channels have pre-defined meanings, and other channels have been given suggested standard meanings in the interest of uniformity among applications. These channel definitions are given in Table 1, below.

The way in which channels are used will become more clear in following sections which describe the SYSTEM MONITOR commands. The section on interfacing to user programs describes the use of channels from a programmers point of view.

TABLE 1: STANDARD CHANNELS

Channel 0: Reserved for internal APEX-65 operation.
Channel 1: Input commands to SYSTEM MONITOR.
Channel 2: Output from SYSTEM MONITOR.
Channel 3: Available. (Input preferable).
Channel 4: Available. (Input preferable).
Channel 5: Standard input for programs.
Channel 6: Standard output for programs.
Channel 7: Available.
Channel 8: Available. (Output preferable).
Channel 9: Available. (Output preferable).

Notes:
1. Channel 1 and Channel 2 are normally assigned to the console by default.
2. The notation "preferable" simply means that if it is convenient to do so, input should be assigned to the lower numbered channels and output to the higher channels. This is merely a convention and is not enforced in any way. All channels can be used in either direction or bi-directionally.

TABLE 2: DEVICES

Device Name	Description
C	Console. Input-output terminal device. (Required)
N	Null device. (Required)
P	Printer.
R	Paper tape reader and/or punch.
T	Teletype (other than console).

Notes:
1. Other devices may be named as desired during System Generation, using a single letter for each.

DEVICES

As we have already seen, APEX-65 communicates with the outside world over numbered channels. These channels can be associated with either physical devices or with files. The devices available on any given system are defined at System Generation (SYSGEN), and are identified by a single letter. Every system has at least two devices: the system console and the null device. The system console is the terminal controlling the system (normally a CRT plus keyboard), and is given the device name "C".

The null device is given the name "N" and is predefined to mean a device that doew nothing. This may seem of dubious merit, but is actually very useful. For example, if you wish to run a program which normally generates voluminous output, but you do not want any output, you can merely assign the null device and run the program.

Additional devices may be available on any given system, and may be named as desired during System Generation. In the interest of uniformity among systems, the recommended device names are given in Table 2 for selected devices.

Remember that all devices have a single letter name. The use of device names will be illustrated shortly in the section describing MONITOR commands.

FILES

Programs, text, and data of any type can be stored and retrieved from floppy disk for permanent storage from APEX-65. A File is a collection of related information stored as a logical entity on disk. Each file on disk has a unique name, designated by the creator of the file. The name consists of from two to twelve characters, optionally followed by a "." and a one-character file extension. The first character must be alphabetic. The remaining characters may be alphabetic, numeric, or the special character "_" (underline), which is used to improve readability of composite names and to facilitate searches for files, as will be discussed later. The single-character file extension may be alphabetic or numeric. If the optional file extension is omitted, a default file extension of ".C" is assumed by the system. Thus some examples of legal file names include:

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

The first two file names above will have a default extension of ".C" appended by the system. The single character file extension is intended to provide the user with an indication of the kind of file. Although APEX does not enforce any particular convention, Table 3 lists the standard file extensions which are strongly suggested for use. Unused extensions may be freely used to cover special kinds of files not included in the list. Note that the extension must be exactly one character long if given.

Remember that file names must have at least two characters; this is how APEX-65 tells the difference between a file name and a device name (which can have only one letter).

NOTE: The Undeline character is not available on the AIM-65 keyboard.

TABLE 3: FILE EXTENSIONS

Extension	Meaning
A	Assembly language source program.
B	BASIC Program source.
C	Command (User-defined MONITOR Command program).
D	Data.
G	Graphic data.
H	Hex file (i.e., paper tape type format).
J	Job file (i.e., a text file of MONITOR commands).
L	Listing.
T	Text.
X	Exexutable code other than a command (e.g., subroutine)
Z	APEX-65 reserved system file.

Notes:
1. If the extension is not given, ".C" will be assumed.
2. Other extensions may be devised by the user as needed.

APEX-65 SYSTEM MONITOR

The APEX-65 MONITOR is an interactive program which allows the user to enter commands to the system. The MONITOR is entered automatically during startup of the system. When the system is "booted" up, the APEX-65 memory image is loaded into memory from the disk in drive 0, and a file called STARTUP.J is read by the monitor and all commands on that file are executed. As each command is read and executed, the MONITOR prompting character, >"½", will appear on the console. At the completion of the startup procedure, a prompting message will be issued indicating the version of APEX-65 which is active, and the prompt, >"½" will appear. At this time, a valid command can be entered from the console keyboard.

Every command typed must be terminated by a carriage return, which signals the MONITOR to execute the command. Certain characters may be used for correcting typing errors or editing the commadn line during entry; these are summarized in Table 4.

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

In order to improve readability and ease the learning process, APEX commands ususally consist of full English words which suggest the function to be performed. However, any built-in command (not user command) can be abbreviated using the "!" character. Thus, for example,

```
ASSIGN
ASSI!
AS!
```

are all equivalents for the ASSIGN command. It is only necessary to type enough characters before the "!" to uniquely identify the command desired.

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

```
ASSIGN 6 MYFILE.T
```

has two arguments. The first argument in this case is a channel number, and the second argument is a file name. The command tells APEX-65 to associate channel 6 with the file called MYFILE.T.

Arguments must be separated from the command keyword and from each other by one or more blanks (not commas!). A few commands use other special delimiters such as "=" in certain places in the command; these will be clearly defined.

Sometimes arguments are optional, in which case the user may elect to specify the argument or else accept the default argument which will be assumed by the system. In other cases, the user has a choice of several different kinds of arguments. In order to have a uniform method of describing the syntax of various commands and arguments, the following notation is adopted:

1. Angle brackets, "<" and ">", are used to enclose words describing the kind of entry required.

2. Square brackets, "[" and "]", are used to enclose optional arguments or symbols, which may be included or omitted as desired.

3. Ellipsis, "...", are used to indicate an arbitrary number of repetitions of the previous 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, only one of which may be selected.

For example, we could use this meta-language (a meta-language is a language used to describe another language) to describe several BASIC statements as follows:

GOTO <line #>
FOR <variable> = <value> TO <value> [STEP <value>] DO

In the following section, each of the Built-in commands will be defined and illustrated. Some of the commands require numeric values for arguments. In this case, either decimal or hexadecimal values may be used. Unless otherwise indicated, all numeric arguments are assumed to be in hexadecimal. To specify a decimal argument, use the "." prefix. If desired, the "$" prefix can be used to clarify hex values. An arithmetic expression can be used anywhere a numeric value is called for. Arithmetic expressions may be formed using the ususal operators, "+", "-", "*", "/" and "\". "\" is the remainder operator. All expressions are evaluated left-to-right without any hierarchy. The value entered may not exceed 65535 decimal or be less than -32768 decimal (including any intermediate point in the computation). The following examples illustrate the evaluation of numeric expressions:

```
100   evaluates as 256 decimal (100 hex).
.100   evaluates as 100 decimal (64 hex).
B+ 10   evaluates as 27 decimal (1B hex).
1+.10*3   evaluates as 33 decimal (21 hex).
$1498/.256   evaluates as 20 decimal (14 hex).
40BC \ 100+1   evaluates as 177 decimal (BD hex).
```

TABLE 4: COMMAND EDITING CHARACTERS

Character	Meaning
DEL or CNTRL-H	Backspace 1 character.
CNTRL-X	Delete entire line (start line over).
RETURN	End-of-command.
;	Comment. Any characters after ";" are ignored.
!	Command abbreviation character. See text.
blank	Separator between arguments.
CNTRL-Q	Temporarily suspend output display
CNTRL-C	Command abort (during display)

TABLE 5: COMMANDS

Command	Purpose
ASSIGN	Assign channel to device or file.
BEGINOF	Position channel to beginning-of-data.
CLOSE	Close-out operations on disk specified.
DRIVE	Designate default drive .
DUMP	Display contents of memory.
ENDOF	Position channel to end-of-file.
DELETE	Delete file from disk directory.
FILES	List names of files on disk.
FILL	Fill block of memory with a constant.
FORMAT	Initialize a diskette to empty state.
FREE	Release channel if assigned.
GET	Load program into memory from disk.
GO	Begin execution of program in memory.
LOCK	Enable write-protect on disk file.
NEXT	Resume execution of program in memory.
OPEN	Open-up operations on a disk.
PROTECT	Enable hardware write-protect on system memory.
REG	Dispaly contents of registers.
SAVE	Save program on disk.
SET	Set memory to value(s).
STATUS	Display channel assignments and system status.
TYPE	Display contents of file.
UNLOCK	Disable write-protect on file.
UNPROTECT	Disable hardware write protect on system memory.
name	Execute User-defined command or system Utility.

COMMANDS

This section describes the function and syntax of each of the APEX-65 built-in commands.

1. ASSIGN <channel> { <device> / <file> [: <drive>] }...

The ASSIGN command is used to assign a device or file to a specified channel.

EXAMPLE:

```
ASSIGN 6 C
```

assigns channel 6 to the system console device (terminal).

```
ASSIGN 5 MYTEXT.T
```

assigns channel 5 to the disk file called MYTEXT.T on the default drive (usually drive 0). The system responds to file assignments with either "NEW FILE" of "OLD FILE" depending on whether or not the given file already exists. If a mistake is made and you get "NEW FILE" when you were expecting "OLD FILE", it probably means you mispelled the file name; you can correct this by merely doing the assignment over. Assigning a channel which is already assigned automatically frees the old assignment first. You may assign several different channels to the same file or device. Assigning a channel to a file always positions that file to beginning of data, even if it is already asssigned to another channel.

CAUTION: CHANNELS 0, 1 AND 2 ARE USED INTERNALLY BY THE SYSTEM AND SHOULD NOT BE REASSIGNED UNTIL YOU HAVE A THOROUGH UNDERSTANDING OF THE SYSTEM OPERATION!

```
ASSIGN 4 C   7 YOURS.A : 1
```

assigns channel 4 to the console and assigns channel 7 to the file called YOURS.A on drive 1. If YOURS.A does not exist, it will be created automatically and will contain nothing. Files which contain nothing disappear automatically when they are FREEd from their channel assignments.

Note that if you ASSIGN a channel to a non-existant file on a hardware write-protected disk, you will get the message, "DISK IS HARDWARE WRITE-PROTECTED" because APEX-65 will attempt to generate a NEW FILE by that name.

2. BEGINOF <channel>...

The BEGINOF command is used to position a a file associated with a given channel to beginning-of-data. If the given channel is assigned to a device rather than a file, then the command is ignored.

EXAMPLE:

BEGINOF 5

positions the file which channel 5 to beginning of file.

3. CLOSE <drive>...

The CLOSE command is used to terminate operations on the specified disk drive(s).

EXAMPLE:

CLOSE 0 1

closes drives 0 and 1. The disks may then be removed.

CAUTION: YOU SHOULD ALWAYS CLOSE EVERY DISK BEFORE REMOVING THE DISK FROM THE DRIVE OR POWERING DOWN.

If you forget to CLOSE a disk before removing it, you will get a system error message of "PREVIOUS DISK NOT CLOSED (OR RESET HIT)" when you attempt another disk operation. At this point you can do one of two things:

1. Put the old diskette back in and CLOSE it (be sure to get the right disk or you'll kill it!!!), or;

2. You can reboot the system with the new disk in place. This will ususally have no ill effects on the disk which you failed to close. At worst, files which had all of the following characteristics on the old disk may be truncated:

a. The file was assigned and not freed;
b. The file was written to by a user (not system) program.
c. The last operation was a write (not a read).

Thus it is unlikely that there will be any adverse affect on the un-closed disk. HOWEVER, IF YOU FAIL TO DO EITHER (1) or (2) ABOVE IN RESPONSE TO THE ERROR MESSAGE, YOU CAN EXPECT UNPREDICTABLE AND INVARIABLY UNPLEASANT RESULTS ON THE NEW DISK!

4. DRIVE <drive>

The DRIVE command is used to designate what drive should be used as the default drive when a file name is supplied without a drive explicitly given. When the system is initially "booted" up this value is set to drive 0.

EXAMPLE:

```
DRIVE 1
```

sets the default drive to drive 1.

5. DUMP <from>[<to> [<channel>]]

The DUMP command displays the contents of a block of memory in hexadecimal and as ASCII characters. <from> is the starting address to be displayed in memory. <to> is an optional argument. If omitted, 8 bytes will be displayed starting at from . If specified, to is the final address of the block to be displayed; however, DUMP always displays an exact multiple of 8 bytes (unless altered by a SYSGEN option).

EXAMPLE:

```
DUMP 200 213
```

displays memory starting at $0200 and will include memory through $0213. The resulting display might look similar to:

```
0200 00 21 00 AA 00 AA 00 76  .!.....v
0208 34 87 41 42 43 AA 00 AA  4.ABC...
0210 10 FF 55 FF 55 FF 55 FF  ..U.U.U.
```

Of course, the actual values displayed will depend on the content of memory. The eight rightmost characters of each line are the ASCII characters for the line, with each non-displayable character converted to ".", including blanks.

NOTES:

1. A complete line is always displayed even if the from address is not an even multiple of 8 bytes.

6. ENDOF <channel>...

The ENDOF command is used to postion a file associated with a specified channel to End-of-File. If the specified channel is assigned to a device and not to a file, then the command is ignored.

EXAMPLE:

```
ENDOF 5
```

positions the file assigned to channel 5 to End-of-File.

Note: The ENDOF command can be used (with care) in conjunction with the TYPE command to concatenate text files. See the TYPE command description for details.

7. DELETE <file> [:<drive>]...

The DELETE command removes a file name from the disk.

EXAMPLE:

```
DELETE MYDATA
```

deletes the file MYDATA.C from the default disk (usually drive 0).

```
DELETE PROG_1A:1   Y3  HIS_STUFF.T
```

deletes three files, one from drive 1 and two from the default drive.

CAUTION: USE THE DELETE COMMAND WITH CARE; THERE IS NO VERIFICATION BEFORE THE FILE IS REMOVED, SO TYPE CAREFULLY! ALL IMPORTANT FILES SHOULD BE LOCKED IMMEDIATELY AFTER THEIR CREATION TO PREVENT INADVERTANT DELETION BY AN ERRONEOUS DELETE COMMAND!

8. FILES [<drive>]...

The FILES command lists the names of all the files on the selected drive, and the remaining unused space on the disk.

EXAMPLE:

```
FILES 0
```

displays the files currently defined on drive 0.

9. FILL <from> <to> [=] { <value> / "<character>" }

The FILL command fills a block of memory with a constant. <from> is the starting address for the operation, and <to> is the ending address. <value> is an 8-bit arithmetic value or expression. <character> is any single ASCII character except the quote (").

EXAMPLE:

```
FILL 200 2FF 0
```

fills every byte between $0200 and $02FF inclusive with $00.

```
FILL 2301 2301+.20=" "
```

fills $2301 through $2315 with $20 (an ASCII blank).

Notes:

1. As each byte is deposited in memory, the result is verified by the system. An attempt to fill ROM, reserved-memory, defective memory, or non-existent memory will abort the command at the point where the error occurred.

2. The FILL command may be used to fill memory locations reserved for APEX-65 if an UNPROTECT command has been issued. Indiscriminant FILLing can lead to system crashes.

10. FORMAT <drive>[<serial no.>]

The FORMAT command is used to irrevocably erase the entire contents of a disk and prepare it in a format suitable for use with the APEX-65 operating system.

CAUTION!!! THE FORMAT COMMAND WILL IRREVOCABLY ERASE ALL INFORMATION STORED ON THE DISK, INCLUDING LOCKED FILES. DO NOT FORMAT THE APEX-65 DISTRIBUTION DISK!!!!

<drive> is the drive number containing the disk you wish to FORMAT. <serial no.> is a four-digit (or less) hexadecimal serial number which is intended to uniquely identify each disk. This serial number (called the VSN or Volume Serial Number) is presently not used by the system but will be required on future versions.

EXAMPLE;

FORMAT 1 1001

formats the disk in drive 1 with a serial number of $1001. APEX-65 will prompt you with

ARE YOU READY?

before actually formatting the disk. Reply with a "Y" or "YES" to proceed; any reply not starting with "Y" causes the command to abort.

Notes:

1. Every new disk (NOT THE DISKS ON WHICH SOFTWARE IS DISTRIBUTED!!!) must be FORMATted using the APEX-65 FORMAT command, even if the disk is already "Formatted" for IBM soft-sector operation. A recommended procedure is to FORMAT all the disks in a box of new disks as soon as you buy them.

2. Do not attempt to OPEN a disk which has not been FORMATted.

3. In the current version of APEX-65, FORMAT is implemented as a Utility program and not as a built-in command. Thus FORMAT will appear in the list of FILES on the distribution disk.

11. FREE <channel> ...

The FREE command is used to disassociate an I-O channel from a device or file.

EXAMPLE:

FREE 6

frees channel 6 from its prior assignment.

FREE 8 4

frees both channel 8 and 4.

Notes:
1. It is permissable to free an unassigned channel.

12. GET <file> [:<drive>] [=<aux. from>]..

Get is used to retrieve a memory image from a file and load it into memory. <file> is the name of the file to be loaded, <drive> is the optional drive number for the desired disk, and <aux. from> is an optional address specifying a starting location for the load in memory which is different from that which was specified when the file was SAVEd.

EXAMPLE:

GET MYPROG

loads the file called MYPROG into memory. It will be loaded at the address which was specified at the time the file was created using the SAVE command. The Program Counter will be set to the entry point address which was saved with the file.

GET OLD_PROG.X:1=100 1B00

will load the file OLD_PROG.X from drive 1 into memory. The first block (which is whatever size was SAVEd on the file) will be loaded starting at address $0100, regardless of what load address was specified when the file was created. The second block (if it exisits) will be loaded starting at address $1B00. Any additional blocks (should they exist) will be loaded at the addresses specified during the creation of the file.

Notes:
1. The file to be loaded must be an "X" format loadable file such as is generated by the SAVE command. An attempt to load a text file or other type file will result in an error.

2. The file may consist of several non-contiguous blocks of memory, all of which will be loaded. See the SAVE command description.

3. One method of moving a block of memory is to SAVE it and then LOAD it with <aux. from> specified as the new address.

13. GO [<from>]

The GO command is used to begin execution of a machine-language program in memory. <from> is the optional starting address. If ommitted, from is assumed to be the current value of the Program Counter (as displayed by the REG command).

EXAMPLE:

GO 200

begins execution of a machine language program at $0200.

Notes:

1. Upon entry to the program, the registers will be set as displayed by the (or defined) by the REG command, except the stack will be discarded.

2. The program is actually entered by a JSR instruction, so that an corresponding RTS will return control to the system. If a program re-enters APEX-65 in this manner, a subsequent REG command will display the status of all registers except the P.C. at the time of the RTS. This is useful for debugging subroutines since the GO command can be used to enter the subroutine, and the routine will return to APEX-65 on completion.

3. The difference between the NEXT command and the GO command is that the NEXT command preserves the stack and enters the program via a jump (thus effectively continuing execution), whereas the GO command discards any stack (sets stack pointer to FF) and enters the program via a JSR.

14. LOCK <file>[:<drive>]...

The LOCK command is used to enable the software write-protect for the designated file(s).

EXAMPLE:

LOCK INVENTORY.T

sets the write-protect for the file called INVENTORY.T on drive 0. This will not affect other files on the disk.

Notes:

1. The LOCK command is used to protect files against INADVERTENT destruction. It is not intended to provide any kind of file security. For floppy disk systems, the most appropriate method of securing information is physical security of the disk.

2. The LOCK command will protect files from DELETE, SAVE, RENAME, WRITE, and TRUNCATE commands or SVC's. It will NOT protect the file from FORMATting or from disk I-O using other software.

15. NEXT [<from>]

The NEXT command is used to re-enter or initiate execution of a machine langauge program. <from> is the optional starting address for execution.

EXAMPLE:

NEXT

will begin execution at the address currently asssociated with the Program Counter (P.C.), as displayed by a REG command.

Notes:

1. See the GO command description for an explanation of the differences between GO and NEXT.

16. OPEN drive ...

The OPEN command is used to prepare a disk for access by the system.

EXAMPLE:

OPEN 1

opens drive 1 for subsequent operations.

Notes:

1. Every disk must be OPENed prior to performing any command or operation on it (except FORMAT).

2. Unlike many other systems, it is not necessary to open or close individual files when using APEX-65. It is only necessary to OPEN each disk as it is inserted, and CLOSE each disk before it is removed from the drive, or before powering down.

3. See the description of the CLOSE command for more details on OPEN/CLOSE considerations.

4. The disk in drive 0 is automatically OPENed by the system when it is "booted" up.

5. OPENing a disk which is already OPEN is permissable.

17. PROTECT

The PROTECT command enables the hardware write-protect for the System RAM memory, and enables checking for reserved memory violations in page 0 and 1 by certain commands and SVCs.

EXAMPLE:

```
PROTECT
```

Notes:

1. The APEX-65 system normally "comes up" in protected mode.

2. In protected mode, the system will not allow any SET or FILL command into the portion of page 0 reserved for APEX-65, nor into any part of page 1 or the 8K block of system memory on the disk controller. Presently, any address greater than the base address of the System RAM on the controller is considered reserved. In addition, the system prohibits GETs which load into page 0, 1 or system RAM.

3. The effects of PROTECT are nullified by an UNPROTECT command.

4. PROTECT and UNPROTECT do not affect the disk or the effect of LOCK and UNLOCK commands.

18. REG [<reg. desig.> [=] {<value>} {<quoted character>}]...

The REG command is used to display or alter the contents of the microprocessors registers. <reg. desig.> is an optional register designator (register name) to be assigned the value or character specified.

EXAMPLE:

```
REG
```

will display the contents of the registers.

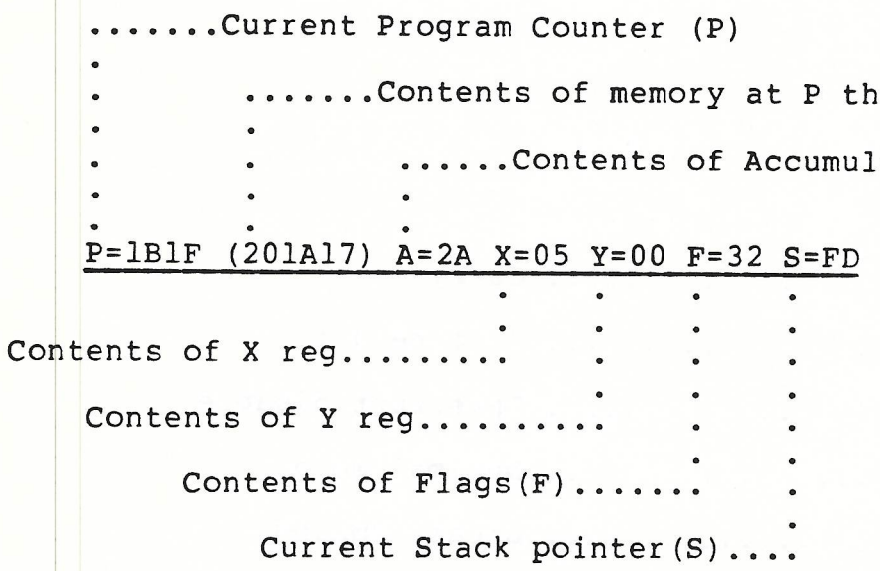
```
REG  A=0
```

sets the A register to $00.

```
REG  X .65 Y="B" A = 10
```

sets the X register to $41, the Y register to $42, and the A register to $10.

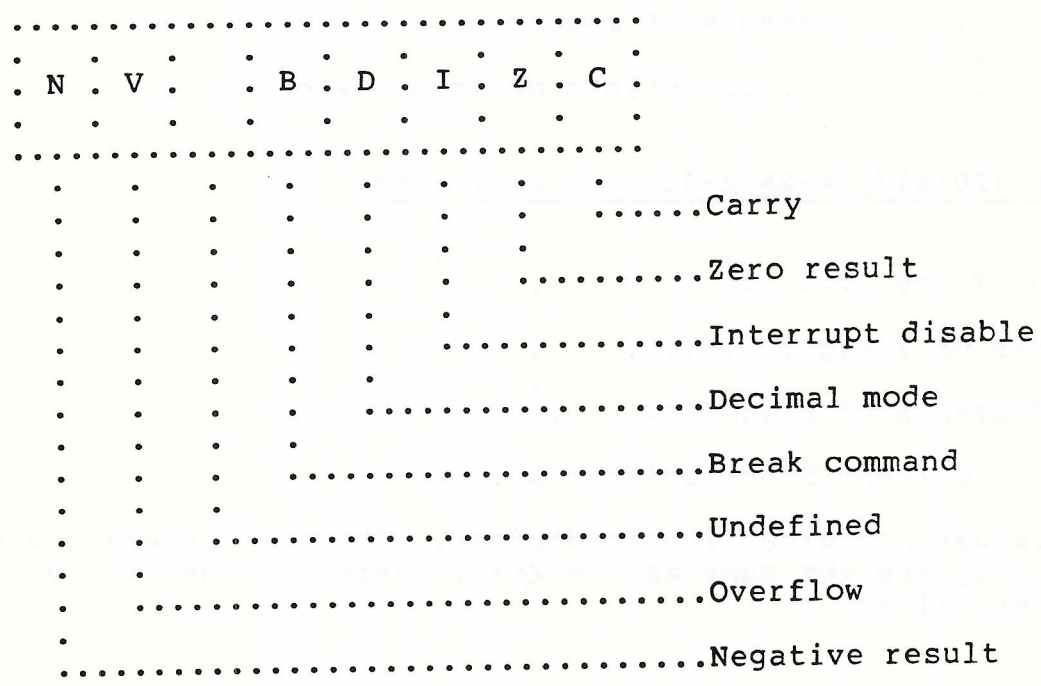
The REG command without arguments displays the register contents in the format described below:



All values are given in hexadecimal. The key letters given in the display are the same as the <reg. desig.> needed to set the register values.

(continued)

The individual bits in the Flags (F) register display are the same as the hardware Processor Status Word, as described below:



19. SAVE ⟨file⟩[:⟨drive⟩][=⟨entry⟩] ⟨from⟩ [=⟨aux.from⟩] ⟨to⟩...

The SAVE command is used to save one or more blocks of memory on floppy disk; thereafter, the saved program can be used as a user-defined command.

⟨file⟩ is the name of the file (i.e., the new command name).
⟨drive⟩ is the optional drive number.
⟨entry⟩ is an optional entry point for the program. An entry point is the address at which execution is to be initialted in the program. Normally, entry is omitted, in which case the starting address will be uised for the entry point.
⟨from⟩ is the starting address for the block of memory.
⟨aux.from⟩ is an auxilliary load address. This is normally omitted. If specified, it is used to indicate that the block of memory being saved should subsequently be loaded at a different address when retrieved using a GET command (or by executing the name of the newly created command).
⟨to⟩ is the final address of the block to be loaded.

EXAMPLE:

```
SAVE DOIT 200 2DF
```

saves the contents of memory locations $0200 though $02DF inclusive on a file called DOIT on drive 0 (by default). Since no optional arguments were specified, the entry point will be saved on the file as $0200, the same as the starting address of the block, and subsequently typing DOIT will cause the block to be loaded from disk into memory at $0200, and execution begun.

```
SAVE RALPH_PROG.C:1 = 2424 2000 20FE 340 3A0
```

saves a file called RALPH_PROG.C on drive 1. The file contains two memory blocks, the first from $2000 to $20FE, and the second from $0340 to $03A0. The entry point is $2424. Subsequently typing a RALPH_PROG:1 command will cause the two blocks of memory to be re-loaded from disk, and program execution begun at $2424.

```
SAVE SUBPKG.X 400=2000 400+.100
```

saves 100 decimal bytes of memory on a file called SUBPKG.X, starting at $0400. Since an <aux.from> address was specified, a subsequent GET SUBPKG.X command will cause the memory block to be loaded into address $2000 and up instead of the $0400 address at which it was saved.

Notes:

1. The existentce of the "=" in the command indicates the existence of one of the optional arguments entry or <aux.from>. Pay careful attention to the postion of the arguments.

2. When using <aux.from>, note that no relocation of any possible address references is made; the memory block is still exactly as saved. Therefore specifying <aux.from> is not normally a satisfactory method of relocating machine language programs.

3. The <entry> address does not have to reside inside any of the saved blocks.

4. The number of blocks saved on a single file is limited only be the number of <from>/<to> arguments you can fit on the command line.

20. SET <from> [=] { <value> / " <character> " } ...

The SET command is used to set the value of memory locations.

<from> is the starting address at which to set values.
<value> is a numeric value or expression which evaluates to an 8-bit quantity.
<character> is an ASCII character.

EXAMPLE:

SET 2000= 1B

sets address $2000 to $1B.

SET 2006 "ABC"

sets $2006 to $41 (ASCII "A"), $2007 to $42, and $2008 to $43.

SET 200 80-.10 " " 80-.20 " "

sets $0200 to $76, $0201 through $0203 to $20 (ASCII blank), $0204 to $6C, and $0205 thourh $0207 to $20.

Notes:

1. The "=" is optional and has no effect on the meaning of the command.

2. As each byte is deposited in memory, it is verified by APEX-65. If reading the byte back from memory results in a bad compare to the value deposited, an error message is issued.

3. Addresses are checked for validity before depositing each value. If an attempt is made to set Reserved memory, an error message will be issued, unless an UNPROTECT command was issued previously.

21. STATUS

The STATUS command is used to display the current channel assignments on the console.

EXAMPLE:

STATUS

displays the status of all channels which are assigned, and will display the file count and remaining disk space for the disk in drive 0.

Notes:

1. Channel 1 and 2 will always be assigned, since they are the channels for command input and output. If you FREE either of these channels, they are automatically re-ASSIGNed to the console. However, you can re-ASSIGN either or both to other devices.

22. TYPE { <file> [: <drive>] / <channel> / <device> } [[{ <file> [: <drive>] / <channel> / <device> }]]

The TYPE command is a versatile command most often used for displaying a text file on the console or printer. The first argument is required and is the file name, device name, or channel which is to be typed. The second argument is optional. If ommitted, the output will be typed on the console ("C"). If a file name is given for the second argument, then it will be the destination for the type command. A channel previously assigned can also be the destination.

EXAMPLE:

```
TYPE MYSOURCE.A
```

will display the file called MYSORCE.A on drive 0 on the console.

```
TYPE C NEW.T
```

will accept input from the console keyboard and put it on a file called NEW.T. This is one way to create a text file.

```
TYPE 5 STUFF.T:1
```

will accept input from the file or device assigned to channel 5 and output it to the STUFF.T file on drive 1.

NOTES:

1. When the source for the TYPE command is a device, for example the console, CNTRL-Z is used to enter and end-of-file and therefore terminate the TYPE command.

2. If a file name is given for either argument, the file will be automatically positioned to beginning-of-data before typing starts. However, if a channel is used for the argument, no positioning takes place. This fact can be used to advantage to copy parts of a file or concatenate files. For example:

```
ASSIGN 6 OLDTEXT.T
ENDOF 6
TYPE C 6
```

8. FILES [<drive>]...

The FILES command lists the names of all the files on the selected drive, and the remaining unused space on the disk.

EXAMPLE:

```
FILES 0
```

displays the files currently defined on drive 0.

9. FILL <from> <to> [=] { <value> | "<character>" }

The FILL command fills a block of memory with a constant. <from> is the starting address for the operation, and <to> is the ending address. <value> is an 8-bit arithmetic value or expression. <character> is any single ASCII character except the quote (").

EXAMPLE:

```
FILL 200 2FF 0
```

fills every byte between $0200 and $02FF inclusive with $00.

```
FILL 2301 2301+.20=" "
```

fills $2301 through $2315 with $20 (an ASCII blank).

Notes:

1. As each byte is deposited in memory, the result is verified by the system. An attempt to fill ROM, reserved-memory, defective memory, or non-existent memory will abort the command at the point where the error occurred.

2. The FILL command may be used to fill memory locations reserved for APEX-65 if an UNPROTECT command has been issued. Indiscriminant FILLing can lead to system crashes.

10. FORMAT <drive>[<serial no.>]

The FORMAT command is used to irrevocably erase the entire contents of a disk and prepare it in a format suitable for use with the APEX-65 operating system.

CAUTION!!! THE FORMAT COMMAND WILL IRREVOCABLY ERASE ALL INFORMATION STORED ON THE DISK, INCLUDING LOCKED FILES. DO NOT FORMAT THE APEX-65 DISTRIBUTION DISK!!!!

can be used to append lines onto the exisitng file OLDTEXT.T from the console. However,

TYPE C OLDTEXT.T

would overwrite the beginning of the file, so be careful!

23. UNLOCK <FILE>[:<DRIVE>]...

The UNLOCK command is used to disable the software protect for the file specified, which was previously set by a LOCK command.

EXAMPLE:

UNLOCK VALUABLES

removes the write-protect from the file called VALUABLES.C on drive 0.

NOTES:

1. It is permissable to UNLOCK a file which is not LOCKed.

24. UNPROTECT

The UNPROTECT command is used to remove the hardware write-protect from the system RAM on the disk controller, and to defeat the reserved-memory protection for various commands.

EXAMPLE:

UNPROTECT

NOTES:

1. Once UNPROTECTED, the SET, FILL, and GET commands will be able to freely overwrite normally-reserved areas of memory including the part of page 0 used by APEX, page 1, and the System RAM on the disk controller. Naturally, casual abuse of this facility is likely to cause strange and invariably unpleasant results.

46
35

COMMAND EXTENSIONS FOR AIM

The AIM-65 version of APEX-65 has several extensions to the standard system to support features provided in AIM ROMs. These are as follows:

1. Special keys:

a. The ESC key exits APEX-65 and enters the normal AIM Monitor.

b. Once APEX has been brought up, you may exit the AIM monitor and re-enter APEX-65 Monitor by depressing F3.

c. CNTRL will temporarily halt display of APEX-65 console ouput. Depressing CNTRL-C and any key will abort the command in progress, if it is performing output.

d. If you depress RESET, you may re-enter APEX by using the F3 key. However, using RESET may leave the system in an undefined state, especially if performing disk operations. Also, the first operation you attempt following the reset will give the error message, "PREVIOUS DISK NOT CLOSED (OR RESET HIT)". This is a warning message, indicating the previous operation may not have been properly completed.

2. Special Commands:

a. The AIM5 command is used to enter AIM BASIC in ROM from APEX-65. You may LOAD and SAVE BASIC programs on disk by answering the "IN=" or "OUT=" prompting message with "U" instead of "T". The system should reply with "FILE=". Enter any valid APEX file name. We suggest the extension "5" for AIM BASIC programs (e.g., START.5, LIFE.5:1 are valid file names). CAUTION: YOU MUST ENTER BASIC USING THE AIM5 COMMAND, NOT BY GOING TO THE AIM MONITOR AND ENTERING A "5" COMMAND, IF YOU WISH TO USE DISK FILES! The AIM5 command performs the necessary setup to activate disk files.

b. The AIME command is used to enter AIM Editor from APEX-65. You may load and store text files on disk by using the "U" reply to the "IN=" and "OUT=" prompts, and specifying a file as for BASIC above. We suggest the file extension "E". CAUTION: YOU MUST ENTER THE EDITOR USING THE AIME COMMAND, NOT BY GOING TO THE AIM MONITOR AND ENTERING AN "E" COMMAND, IF YOU WISH TO USE DISK FILES! The AIME command performs the necessary setup to activate disk files.

SUPERVISOR CALLS (SVC)

Interfacing User Assembly-language Programs to Apex

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 various utility functions provided by the system. Using the functions described here can greatly reduce program development time and effort.

Most operating systems provide a degree of support for assembly-language programming by making available the addresses of certain system subroutines which the user can call to perform I-O or other functions. For example, to output a character to the console, you might put the ASCII character into the A register and call the driver subroutine for the console display device. APEX-65 does not use this method, but instead provides a more powerful tool called the Supervisor Call Instruction (SVC). The SVC concept is not new; SVCs are found in various forms on many large mainframe computers.

The following discussion assumes a knowledge of 6502 assembly language programming on the part of the reader.

How SVC's work

The APEX-65 implementation of the Supervisor Call capability consists of a BRK instruction ($00) followed by a one-byte numeric code which tells the system what function is required. The code numbers are listed in Table 1. Effectively, the SVC is a lot like a JSR (Call Subroutine) instruction, except that it is two bytes long instead of three, and the second byte is not an address, but a code which tells what pre-defined system subroutine is to be called.

Why are SVC's better than a straightforward JSR? There are several reasons:

1. SVCs are address-independent. This is by far the most important advantage of SVCs. It means that future system upgrades which may alter the addresses of actual system routines will not affect the SVC numbers, and therefore will not adversely affect programs using SVCs. It also means that, for example, a program on an AIM-65 computer with APEX at $8000 can be transported to a KIM system with APEX at $E000 and run without modification. If subroutine calls were used instead, it would be necessary to patch all the JSRs to the system routines before execution.

TABLE 1

SUPERVISOR CALL NUMBER CODES (SVC'S)

SVC#	Description	Pass Regs.	Returns Regs.
0	Return to APEX-65 Monitor	-	-
1	Not currently defined		
2	Output inline message (see text)	-	-
3	Input byte from channel	X	A, F
4	Output byte to channel	X	-
5	Input line from channel	X,U5	A,Y,F
6	Output line to channel	X,Y,U6	-
7	Output string on channel	X,Y,U6	-
8	Decode ASCII hex to value	X,Y,U5	A,Y,F,U0
9	Decode ASCII dec. to value	X,Y,U5	A,Y,F,U0
10	Encode value to ASCII hex	X,Y,U0,U6	Y
11	Encode value to ASCII dec.	X,Y,U0,U6	Y
12	Querry default buffer addr.	-	U5,U6,Y
13	Not currently defined		
14	Querry channel assignment	X	A,F
15	Read record from channel	X,U1,U2	F,U1,U2
16	Write record to channel	X,U1,U2	F
17	Position file to beginning	X	-
18	Position file to end-of-file	X	-
19	Position file	X,U7	U7
20	Querry file position	X	X,U7
21	Assign channel to file/device	X,A	A,F
22	Free Channel	X	-

Note: This is a preliminary list. Other functions will be added later.

2. SVCs user less memory. Two bytes are cheaper than three.

3. SVCs preserve the values in registers. All registers are restored to their condition upon entry to the SVC when returning to the calling program, except when returning values to the calling program. This saves the programmer a lot of unnecessary saving and restoring registers.

4. SVC's are easier to debug. If an error is detected by the system while processing an SVC, the program will abort and APEX-65 will display the exact address of the offending supervisor call, the values of all the registers at the time of the SVC, and an error message explaining the difficulty. Illegal or unimplemented SVCs are also trapped in the same manner.

Initialization and Parameter Passing

In order to use SVCs, the user program must first enable the Supervisor by setting the SVC Enable flag, SVCENB, to $80 (bit 7 must be set to 1). If SVCs are not enabled, any BRK instruction will simply return to the Monitor with a display of the location of the BRK and register contents. Note that the SVCENB flag must be set to $80 by the user program, and will not work if set from the Monitor using the SET command.

Usually, some type of argument needs to be passed to the Supervisor and/or returned to the user program from the Supervisor. The method for passing arguments is defined for each SVC individually, and may bd done three possible ways:

1. Arguments may be passed or returned in 6502 registers.

2. Arguments may be passed in one or more "pseudo-registers" in page zero.

3. Arguments may be passed "in-line", immediately following the SVC.

Before proceeding further, an example program will illustrate SVC usage.

Example Program 1: Displaying text message.

The first SVC we shall examine in an example is SVC 2, which outputs a message over a channel. This is a very unusual SVC in that the argument is passed in-line. However, it is so frequently needed in programming that it deserves our first attention.

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

SOLUTION:

```
SVCENB    =       $11       ;SVC ENABLE FLAG LOCATION
;
          .=      $200      ;PROGRAM ORIGIN
GREET     LDA     #$80
          STA     SVCENB    ;ENABLE SVCS
          BRK               ;SVC...
          .BYTE   2         ;...#2 = OUTPUT INLINE MESSAGE...
          .BYTE   2         ;...OVER CHANNEL 2...
          .BYTE   'HELLO THERE.'
          .BYTE 0           ;0 TERMINATES MESSAGE TEXT
          RTS               ;RETURN TO MONITOR OR CALLING PROGRAM
          .END
```

EXPLANATION:

The program begins by enabling SVCs (note: once enabled, SVCs remain enabled until disabled by writing $00 into SVCENB; it is advisable to disable SVCs when not needed). The BRK instruction together with the first .BYTE 2 pseudo-instruction comprise the SVC, and Table 1 tells us that an SVC 2 is used to display an inline message. The second .BYTE 2 tells the System what channel to output the message on. Channel 2 was selected for our example because it is assigned to the console display by default. Of course, it could be re-assigned to any device or file. Following the channel is the text of the message, which can consist of up to 255 bytes and is terminated by a $00. The $00 also is the last argument of inline code. The System will output the message over channel 2 and then return control to the instruction following the $00 byte; in this case, merely the RTS whcih terminates the program.

Remember that SVCs do not alter any registers except to return values to the calling program; since SVC 2 does not need any returned values, no registers are altered. This is a big benefit, since it means that you can put inline messages anywhere you please in your program for debugging purposes without having to worry about side effects to the registers.

Note that SVC 2 does not output any carriage return automatically; if you want to output control characters, you may include them explicitly in the message, as illustrated below.

Example Program 2: Display message on a new line.

PROBLEM: Repeat Problem 1, above, but start the message on a new line.

SOLUTION:

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

EXPLANATION:

The only change to this program from Example Program 1 is the addition of the ".BYTE 13" at the start of the message, which produces a carriage return. Any control characters desired can be embedded in the message in this manner, except ASCII NUL (becuase NUL = $00, the message terminator.).

There are three very common sources of trouble when using SVC 2 to generate messages:

1. Forgetting to enable SVC's (in which case the program will simply return to the Monitor with a display of the registers when the first BRK instruction is encountered);

2. Forgetting the CHANNEL argument (which usually results in an error message of "ILLEGAL CHANNEL" or "SELECTED CHANNEL IS UNASSIGNED");

3. Forgetting the zero-byte terminatior for the message, (which often results in your program going into "hyperspace" after displaying the message).

52
41

Passing Arguments to Supervisor in 6502 Registers

The example programs above passed their arguments to the Supervisor in-line. A much more common method of parameter-passing is the use of the 6502 registers. The following example illustrates register parameter passing.

Example Program 3: Character Input-Output.

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

SOLUTION:

```
SVCENB    =       $A0
;
STRMIN    LDA     #80
          STA     SVCENB
NEXTCH    LDX     #5        ;CHANNEL 5 FOR INPUT STREAM
          BRK
          .BYTE   3         ;SVC #3 = INPUT CHARACTER FROM CHAN (X)
          BCS     EOFENC    :BRANCH IF END-OF-FILE ENCOUNTERED
          CMP     #'.       ;ELSE EXAMINE CHARACTER INPUT
          BNE     NEXTCH    :IF NOT ".", READ MORE
          BRK
          .BYTE   2         ;ELSE DISPLAY INLINE MESSAGE
          .BYTE   2         :...ON CHANNEL 2
          .BYTE   13,'"." ENCOUNTERED."',0 ;GIVE MESSAGE
          RTS
EOFENC    BRK
          .BYTE   2         ;SVC 2= INLINE MESSAGE
          .BYTE   2         ;...ON CHANNEL 2
          .BYTE   13,'E-O-F ENCOUNTERED.',0 ;GIVE MESSAGE
          RTS
```

EXPLANATION:

This program illustrates a number of aspects of SVC usage. The line labelled NEXTCH is used to load the channel number desired into X. The Supervisor expects to find the channel number in register X when the SVC is processed, as is detailed in the individual SVC descriptions. SVC 3 returns the character read in the A register, and sets the carry flag only if End-of-File was encountered. End-of-File is an important concept. The End-of-File flag (the carry flag) is set by the SVC processor only if no more characters can be read from the selected channel. If the input channel is the console keyboard, this means that CNTRL-Z was entered (the CNTRL-Z character is not returned in A). If channel 5 was assigned instead to a file, it simply means that the previous character was the last character in the file. The programmer should always check for

End-of-File when doing any kind of input operation, so that programs are device-independent. No error will occur if you attempt to read beyond end-of-file; the result in A is just not meaningful. It is the Programmer's responsibility to test the carry on every input operation and take appropriate action if it is set.

In our example, once we have ascertained that E-O-F was not encounered, the character received from channel 5 is checked to see if ti is a ".". If not, another character is read. Once one of the two terminal conditions is met, an SVC 2 is used to issue a message to the console (channel 2) indicating which event occurred.

Passing Arguments in APEX Pseudo-Registers

Sometimes it is necessary to pass addresses or other 16-bit information to the SVC processor. The 8-bit A, X, and Y registers of the 6502 are inadequate for this purpose, so a set of eight Pseudo Registers (hereafter called P-registers or simply P-regs) are providedin zero-page, as shown in figure 1. P-regs U0 through U6 are each 16 bits wide; U7 is 24 bits wide, and is used for file positioning, as we shall see later. Note that if SVCs are not enabled, these P-regs are not used for any purpose whatsoever by the system, and may be freely used as ordianry program memory by application programs. Values to be passed to the SVC processor are installed in these P-registers in the usual manner for memory. The SVC processor expects to find certain addresses or values in specific P-registers, depending on the SVC. For example, most I-O functions (except single character I-O) use U5 to hold the address of an input buffer and U6 to hold the address of an output buffer. Each SVC description tells what P-registers are used, if any. Certain SVCs return information to the application program in P-regs. For example, SVC 12 ($0C) does not pass any P-regs to the SVC processor, but the system returns U5 and U6 to the application. The addresses returned are the location of the system input and output line buffers, respectively.

Example Program 4: Line-Oriented I-O.

Most programs need to deal with input and output of strings or lines of characters. Several SVBCs are provided for support. Applications programs will make heavy use of the 6502 Indirect,Y addressing mode in these applications. In general, P-register U5 (for input) or U6 (for output) must be initialized to point to the start of a buffer containing the current line of interest. The Y register is used to index the particular character of interest within the line. Normally, the System input and Output buffer are the most convenient to use, since an SVC 12 will automatically setup the proper addresses in U5 and U6, but the programmer may select any location for the buffers. The System buffers are sufficiently large for lines of up to 80 characters.

The following problem illustrates line-processing.

PROBLEM: Write a program to copy line of input text from channel 5 to channel 6 until an End-of-File is encountered. Assume Channel 5 and 6 have been given appropriate assignments.

SOLUTION:

```
SVCENB    =       $A0
U5        =       $0A       ;P-REG U5
U6        =       U5+2      ;P-REG U-6
;
COPY56    LDA     #80
          STA     SVCENB    ;ENABLE SVCS
          BRK
          .BYTE   12        ;SVC 12 = QUERRY SYS. BUFFER ADDRESSES
NEXT      LDX     #5        ;CHANNEL 5 FOR INPUT
          BRK
          .BYTE   5         ;SVC #5 = INPUT LINE TO BUF. AT (U5)
          BCS     EOFENC    ;BRANCH IF END--OF-FILE ENCOUNTERED
          TAY               ;ELSE SAVE CHARACTER COUNT IN Y...
          TAX               ;...AND X TOO
LOOP      LDA     (U5),Y    ;COPY CONTENT OF INPUT BUFFER...
          STA     (U6),Y    ;...TO OUTPUT BUFFER
          DEY
          BPL     LOOP      ;...UNTIL WHOLE LINE COPIED
          TXA               ;THEN RECALL LINE LENGTH...
          TAY               ;...TO Y
          BRK
          .BYTE   6         ;SVC #6 = OUTPUT LINE AT (U6)
          JMP     NEXT      ;REPEAT FOR NEXT LINE
EOFENC    RTS               ;END
```

EXPLANATION:

You may have wondered why byte-oriented I-O was not used to copy the file since this would be substantially simpler. One reason is that the line-input SVC (SVC #5) supports the line editing characters Backspace (DEL or CNTRL-H) and CNTRL-X (start line over), but the byte-input SVC (SVC #3) does not. Thus using line input gives more flexibility when the input channel is assigned to the keyboard (Console). SVC number 3 (byte input) returns control to the application program immediately when a key is depressed; SVC number 5 does not return until an entire line is entered, terminated by a carriage return. The editied line is returned to the user program in the buffer pointed to by U5, and the number of characters in the line is returned in the 6502 A register. The carriage return is replaced in the line with a $00 byte, and the character count in A does not include it.

The Example program starts by enabling SVCs and setting U5 and U6 to the addresses of the system line buffers, using the SVC 12 function. An SVC 5 is then used to input the source

input line into the buffer addressed by U5, and End -of-File is tested as before. Note that the SVC 5 function returns the character count in A and Y set to 0 (therefore ready to index the first character of the line). The character count of the line is transferred to the X register as a temporary save, and the line is copied (backwards) from the input buffer to the output buffer. The output buffer is then output over channel 6. Note that the character count must be passed in the Y register. In the example, this character count was recalled from X to Y through A.

An alternative to copying the input buffer contents to the output buffer would simply be to copy the contents of U5 to U6. Normally, however, you will want to use separate input and output buffers since you will be performing other operations on the output line besides just copying the input.

Example Program 5: Read Hexadecimal Input Value.

Looking in Table 1, you may be surprised to find no direct way to input or output numeric values. Instead, a combination of two SVCs must be used to perform this function. This turns out to be a great deal more versatile. A pair of definitions are needed to get us started:

Decoding is the operation of scanning a string of ASCII characters and returning the numeric value they represent.

Encoding is the inverse operation; encoding accepts a (binary) value and returns the string of ASCII characters representing its value.

For example the ASCII string " 010B " when decoded returns the binary value 0000000100001011 ($010B), assuming that hexadecimal decoding was selected. The following problem illustrates how to input and decode a hex value.

PROBLEM: Write a subroutine which reads a hexadecimal number from channel 5 and returns its value in P-register U0.

SOLUTION:

```
SVCENB    =       $A0
;
HEXIN     LDA     #80
          STA     SVCENB   ;MAKE SURE SVCS ARE ENABLED
          BRK
          .BYTE   12       ;SVC 12 = GET BUFFER ADDRESSES
          LDX     #5       ;CHANNEL 5 FOR INPUT
          BRK
          .BYTE   5        ;SVC 5 = INPUT LINE
          BRK
          .BYTE   8        ;SVC 8 = DECODE HEX VALUE TO U0
          RTS
```

EXPLANATION:

The enabling of SVCs and selection of the System buffers should be familiar by now. In practice, these functions would probably be performed only once during program initialization, and would not be included in this subroutine, thus reducing the subroutine to a six line routine. The SVC 5 operation inputs a line into the buffer addressed by P-register U5, as previously seen. The SVC 8 function serarches the buffer (starting with the character indexed by Y, which was 0 in our case since SVC 5 always returns Y=0) for a character string representing a hex value. Note that any number of leading blanks may preceed the number, and 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. SVC 8 keeps scanning until a non-hex character is encountered. Thus, for example, " 2B7,2 " will return U0 = $02B7, because the comma will terminate the scan. When control is returned to the calling program, the Y register points to the delimiter (the comma in the example immediately above), and the A register holds the delimiter encoutnered. This is very useful when scanning a line containing multiple values. In addition, the carry flag is returned to the calling program as a "Valid Data Enocuntered" flag. Although the exaple program above did not do so, it is easy for the application program to check the status of the carry upon completion of SVC 8; if it is not set, then no valid hex digits were encountered prior to the delimiter (or end-of-line). Note that the end-of-line delimiter is returned as $00.

Note: The example programs presented have used the system input and output line buffers. In practice, during program generation and debugging, it is advisable to use other buffers because any any interaction with the system will cause your buffers to be "wiped out" (for instance, any command you enter goes into the system input buffer). To define you own buffers merely copy the address of the buffers to U5 and U6, instead of using SVC 12.

SVC DESCRIPTIONS

SVC #0 ($00)

PURPOSE: Return to APEX-65 Monitor.

ARGUMENTS: None.

ARGUMENTS RETURNED: None.

DESCRIPTION:

SVC #0 returns control to the APEX-65 MONITOR. It has two advantages over simply using an RTS to return to the Monitor:

1. It can be executed anywhere, even in a subroutine, proided that SVCs are enabled;

2. The value of the Program Counter (P) shown by the REG command after returning will show the address of the SVC 0; using a RTS to return to MONITOR will not update the P register value shown.

EXAMPLE:

```
SVCENB    =       $A0
          ...
          LDA     $80
          STA     SVCENB
          ...
          BRK
          .BYTE   0         ;RETURN TO MONITOR.
          ...
```

SVC #2 ($02)

PURPOSE: Output inline message over channel.

ARGUMENTS:

First Byte after SVC 2 = desired channel number.

Second throuth Nth byte = desired ASCII message text, terminated by a zero byte ($00).

DESCRIPTION:

SVC 2 can be used to display a message at any point in a program (provided SVCs are enabled). It does not effect any registers. The message may be any length up to 254 bytes, and can contain any byte including unprintable characters, except NUL ($00), which is the message terminator. Control will be returned to the instruction immediately following the 0-byte terminator. The channel specified must be assigned to a valid device or file.

EXAMPLE;

```
SVCENB    =       $A0       ;LOCATION OF SVC ENABLE FLAG
CR        =       13        ;ASCII CARRIAGE RETURN
          ...
          LDA     #$80
          STA     SVCENB
          ...
          JSR     DOIT7
          BRK
          .BYTE   2         ;SVC #2 = OUTPUT MESSAGE...
          .BYTE   6         ;...ON CHANNEL 6
          .BYTE   CR,'SUB. DOIT7 DONE, CALLING DOIT8.',0
          JSR     DOIT8
          ...
```

This program segment will output this message to channel 6:

DOIT7 DONE, CALLING DOIT8.

NOTES:

1. The message will always be displayed starting at the present position. If the message should start on a new line, then the carriage return should be explicitly included, as in the example above.

2. Be careful to check that you have not forgotten the CHANNEL NUMBER argument before the message, or the 0-BYTE TERMINATOR after the message!

SVC #3 ($03)

PURPOSE: Input byte from channel.

ARGUMENTS:

X = desired channel number.

ARGUMENTS RETURNED:

A = byte received from channel.

Flags: CY = 1 means End-of-File was encountered.

DESCRIPTION:

SVC 3 inputs a single byte from a selected channel, which must be assigned to a valid device or file. The value of the byte returned can be anything, including control characters ($00 to $FF), if the selected channel is assigned to a file. If assigned to a normal, character-oriented input device, such as the keyboard, then a CNTRL-Z (ASCII SUB, 16) will be interpreted as End-of-File. For files, End-of-File is true only when no more bytes can be read from the file. It is the programmer's responsibility to check the status of the Carry after every SVC 3 to insure that End-of-File was not reached. The A register is not meaningfully returned if the Carry is set.

EXAMPLE:

```
SVCENB    =       $A0
          ...
          LDA     #80
          STA     SVCENB   ;ENABLE SVCS.
          ...
          LDX     #5       ;SELECT CHANNEL 5
          BRK
          .BYTE   3        ;SVC #3 = INPUT BYTE ON CHANNEL (X)
          BCS     EOFHI    ;BRANCH IF END-OF-FILE
          CMP     #'C'     ;WAS INPUT CHARACTER 'C'?
          ...
```

This program segment inputs a character from the file or device assigned to channel 5 and checks to see if it was an ASCII "C".

NOTES:

1. The remaining flags (other than CY) are not meaningfully returned; in any case, the decimal mode flag will not be set.

SVC #4 (#04)

PURPOSE: Output byte over channel.

ARGUMENTS:

X = Channel desired.

A = Byte to be output.

ARGUMENTS RETURNED:

FLAGS: CY = 1 if at End-of-File after output operation.

DESCRIPTION:

SVC 4 outputs the byte in the accumulator over the channel specified in the X register. The channel must be assigned to a valid file or device. Although there is no need to do so, application programs may wish to test the Carry flag after SVC 4 to distingusih whether the character written was the last character of the file or was re-written over some other part of the file. If the channel is assigned to a device instead of a file, the Carry will always be returned set, since End-of-File has no meaning in this context.

EXAMPLE:

```
SVCENB    =       $A0
          ...
          LDA     #$80
          STA     SVCENB   ;ENABLE SVCS
          ...
          LDA     #$09     ;BYTE DESIRED TO OUTPUT
          LDX     #2       ;CHANNEL 2
          BRK
          .BYTE   4        ;SVC 4 = OUTPUT BYTE
          JMP     THERE
          ...
```

This program segment outputs $09 over channel 2.

SVC #5 ($05)

PURPOSE: Input line of text from channel.

ARGUMENTES:

X = Channel number to read from.

U5 = Address of desired input buffer for line.

ARGUMENTS RETURNED:

A = Count of characters in line.

Y = 0.

Flags: CY = 1 if End-of-File encountered.

DESCRIPTION:

SVC 5 inputs a line of text from the file or device assigned to channel 5. The text will be deposited in a buffer whose address is specified in U5. The line of text will be terminated by a $00 byte. After the SVC is processed, the Carry will be set only if no characters could be read from the channel because End-of-File was encountered. The A register will contain a character count for the input line. This count does not include the $00 terminator. The Y register is always returned as 0 to facilitate user processing of the line using Indirect, Y addressing. If the channel selected is assigned to a device, then End-of-Line is defined as the first carriage return ($0D) encountered. This carriage return is converted to the $00 terminator in the buffer, and is not included in the character count in A.

EXAMPLE:

```
SVCENB    =       $A0       ;LOCATION OF SVC ENABLE FLAG
U5        =       $0A       ;P-REGISTER U5 LOCATION
          ...
          LDA     #$80
          STA     SVCENB    ;ENABLE SVCS
          LDA     #$00
          STA     U5
          LDA     #$10
          STA     U5+1      ;DEFINE BUFFER ADDRESS AS $1000.
          ...
          LDX     #5        ;CHANNEL 5
          BRK
          .BYTE   5         ;SVC 5 = INPUT LINE FROM CHAN. (X).
          BCS     EOFHI     ;BRANCH IF END-OF-FILE
          STA     NCHLN     ;ELSE SAVE COUNT OF CHARACTERS IN LINE
          ...
```

This program segment inputs a line of text from channel 5 and places it in a buffer starting at address $1000.

NOTES:

1. The system maintains a "Maximum Input Record Length" parameter for text input, which has a default value of 80 ($50) characters. If an SVC 5 attempts to input a line with more than 80 characters, then the system will automatically add an end-of-line character after 80 characters are read. This is to prevent SVC 5 from wiping out all of memory if the channel is inadvertently assigned to a non-text file which does not contain end-of-line terminators. The value of the Maximum Record Length parameter can be altered if it is necessary to read lines of greater than 80 characters. The system buffers are only 80 characters long, however, so the user will have to provide a buffer elsewhere and not use SVC 12 to define the buffer address.

2. The following editing characters are recognized by SVC 5. These editing characters are not returned in the line, but instead perform the function indicated:

BACKSPACE, DEL, or RUBOUT ($08 or $7F): Backspace one character. Will not backspace beyond beginning-of-buffer.

CNTRL-X or CAN ($18): Delete entire line (start over).

RETURN ($0D): End-of-line.

CNTRL-Z ($16): End-of-File (applies only if entered from device, not in file; must follow carriage return).

SVC #6 ($06)

PURPOSE: Output line of text on channel.

ARGUMENTS:

X = Channel desired.

Y = Number of characters in line.

U6 = Starting address of line of text.

ARGUMENTS RETURNED: None.

DESCRIPTION:

SVC 6 outputs a line of text over a channel which is assigned to a valid file or device. U6 must contain the address of a buffer containing the text to be sent. The Y register must hold the number of characters to be sent, not including the line terminator (which is added by the system).

(continued)

EXAMPLE:

```
SVCENB      =       $A0       ;LOCATION OF SVC-ENABLE FLAG
U6          =       $0C       ;LOCATION OF P-REGISTER U6
            ...
            LDA     #$80
            STA     SVCENB    ;ENABLE SVCS
            ...
            LDA     PROD
            STA     U6        ;DEFINE ADDRESS OF TEXT TO BE SENT
            LDA     PROD/256
            STA     U6+1
            LDX     #6        ;CHANNEL 6
            LDY     #11       ;11 CHARACTERS IN LINE
            BRK
            .BYTE   6         ;SVC 6 = OUTPUT LINE
            ...
PROD        .BYTE   'DISK SYSTEM'
            ...
```

This program segment will output "DISK SYSTEM" followed by an end-of-line character on channel 6.

NOTES:

1. The line to be output cannot exceed 254 characters.

SVC #7 ($07)

PURPOSE: Output string of text on channel.

ARGUMENTS:

X = Channel desired.

Y = Number of characters in string.

U6 = Starting address of string of text.

ARGUMENTS RETURNED: None.

DESCRIPTION:

SVC 7 outputs a string of text over a channel which is assigned to a valid file or device. U6 must contain the address of a buffer containing the text to be sent. The Y register must hold the number of characters to be sent.

EXAMPLE:

```
SVCENB      =       $A0       ;LOCATION OF SVC-ENABLE FLAG
U6          =       $0C       ;LOCATION OF P-REGISTER U6
            ...
            LDA     #$80
            STA     SVCENB    ;ENABLE SVCS
            ...
            LDA     PROD
            STA     U6        ;DEFINE ADDRESS OF TEXT TO BE SENT
            LDA     PROD/256
            STA     U6+1
            LDX     #6        ;CHANNEL 6
            LDY     #11       ;11 CHARACTERS IN LINE
            BRK
            .BYTE   7         ;SVC 7 = OUTPUT STRING
            ...
PROD        .BYTE   'DISK SYSTEM'
            ...
```

This program segment will output "DISK SYSTEM". NO End-of-line character will be added by the system.

NOTES:

1. The text to be output cannot exceed 254 characters.

INTERIM MEMORY MAP

(AIM-65 VERSION)

NOTE: THIS IS A PRELIMINARY MEMORY MAP. ALL ADDRESSES ARE SUBJECT TO CHANGE WITHOUT NOTICE.

Address	Description
$0000-0010	Pseudo registers U0 through U7. Used by system only if SVCs are enabled; otherwise, user-RAM.
$00A0	SVCENB. SVC-enable flag, settable by user programs. See section on SVCs.
$00A1-00CF	System zero-page ram. Reserved for use by APEX-65.
$0100-01FF	Reserved for stack.
$0200-47FF	User RAM; however, Utility programs such as FORMAT, DUP, and COPYF also use some of this RAM during execution.
$4800-50FF	Interim System Ram for future "overlays". Reserved for system but not protected in any way! Do not overwrite.
$5100-5BFF	User Ram.
$5C00-5CFF	Default location for system input and output line buffers (can be altered).
$5D00-5FFF	Optional DMA buffers if more than default number of simultaneously-open files is used.
$6000-7FFF	Recommended location for visible memory board, if used.
$8000-85FF	Protected DMA buffers for System.
$8600-9EFF	Protected APEX system nucleus and RAM.
$9FF0-9FE7	Future Bootstrap ROM.
$9FE8-9FFF	Disk Controller I-O ports.
$A000-FFFF	AIM-65 ROMs.

INTERIM SYSTEM GENERATION AND CUSTOMIZING

(AIM VERSION)

In order to make effective use of APEX-65, certain modifications need to be made for any given target computer. In particular, the operating system needs to know the number of disk drives present in the system, and the types of I-O devices which need to be supported. A conversational SYSGEN program is planned for later release to perform these functions easily. For now, however, it will be necessary to perform the System "customizing" using the method described below.

One feature of APEX that greatly facilitates the customization process it the startup procedure. When "booted" up, APEX-65 will read a list of commands on a file called STARTUP.J and executes them before accepting any keyboard commands. Therefore, if you have any special needs for your system, these can be appended to the STARTUP.J file. For example, if your system needs to load I-O driver routines into memory, this can be easily accomplished without user intervention. If certain patches need to be made, they can be accomplished by simply entering the appropriate SET commands onto the STARTUP.J file, or a program could be executed to perform more complex patches.

***CAUTION: WHEN USING THE STARTUP.J FILE, YOU MAY ADD ANY COMMANDS YOU WISH TO THE END OF THE FILE. YOU MAY NOT DELETE OR REPLACE THE EXISITNG COMMANDS OR THE SYSTEM WILL NOT COME UP PROPERLY WHEN BOOTED! ***

You can ascertain what is on the distribution STARTUP.J file by simply giving a TYPE STARTUP.J command.

Table 1 lists some of the most important addresses you might wish to patch in the present interim release. These locations are subject to change. Dont forget to have the UNPROTECT command preceed any commands that alter system memory in the STARTUP.J file.

As an example, the following STARTUP.J file will define a new Ouptut device called "P" (Printer), which has an output driver subroutine which must be loaded from disk.

```
GET OVL.Z ;EXISITNG COMMAND ON STARTUP FILE
GET PRINTDRIVER.X ; LOAD PRINTER DRIVER ROUTINE
UNPROTECT ;TEMPORARILY REMOVE SYSTEM PROTECT
SET 8617="P" 8631=00 02 ;ADD P DEVICE, DRIVER ADDR TO TBL.
PROTECT ;RESTORE SYSTEM TO PROTECTED MODE
```

SYSTEM PARAMETER LOCATIONS

(AIM VERSION)

Address	Default	Name/Meaning
$8607		Address of Console Character-In driver routine. Return character in A, bit 7=0
$860A		Address of Console Character-Out driver routine. Character is in A.
$860D		Address of Console key-down test routine. Returns bit 7 of A=1 if no key pressed, A=character, bit 7=0 if pressed
$8615		Device Name Table. See listing provided in following section.
$87FE	02	Number of drives (1 or 2).
$882A	00	Flag. If bit 7 = 1 then ignore any irrecoverable CRC disk I-O errors and continue reading/writing.
$8830	00	Flag. If bit 7 = 1 then Console input will not be echoed to console output.
$8837	1A	End-of-file character for console input. Default = CNTRL-Z.
$8845	08	Number of bytes to display per line on DUMP command.
$8842	51	Maximum Input record length.
$8855	5C00	Address of default system input buffer.
$8857	5C53	Address of default system output buf.
$8859	890B	Address of user interrupt service routine.

Notes:

1. The current version only supports one or two drives.

```
X   VERSION 0.2 :AIM
ICE DRIVER TABLES

                                .PAGE 'DEVICE DRIVER TABLES'

88                      ;
89                      ;       DNT: DEVICE NAME TABLE.
90                      ;
91 8615 4E              DNT:    .BYTE 'N'       ;"N" = NULL DEVICE DRIVER
92 8616 43                      .BYTE 'C'       ;"C" = CONSOLE DEVICE
93 8617 00                      .BYTE 0
94 8618 00                      .BYTE 0         ;RESERVED FOR CUSTOM DEVICES...
95 8619 00                      .BYTE 0
96 861A 00                      .BYTE 0
97 861B 00                      .BYTE 0
98 861C 00                      .BYTE 0
99                      ;
00                      ;       DDTI: DEVICE DRIVER DISPATCH TABLE FOR INPUT.
01                      ;
02 861D C395            DDTI:   .WORD NULDVR    ;NULL DRIVER DEVICE (DTI=X'80)
03 861F 0C9D                    .WORD CIN       ;CONSOLE INPUT ROUTINE (DTI=$82)
04 8621 0000                    .WORD 0
05 8623 0000                    .WORD 0         ;CUSTOM DRIVER ADDRESSES...
06 8625 0000                    .WORD 0
07 8627 0000                    .WORD 0
08 8629 0000                    .WORD 0
09 862B 0000                    .WORD 0
10                      ;
11                      ;       DDTO: DEVICE DRIVER DISPATCH TABLE FOR OUTPUT.
12                      ;
13 862D C395            DDTO:   .WORD NULDVR    ;NULL DRIVER (DTI=X'80)
14 862F 1D9D                    .WORD COUT      ;CONSOLE OUTPUT ROUTINE (DTI = $82)
15 8631 0000                    .WORD 0
16 8633 0000                    .WORD 0         ;CUSTOM DEVICE DRIVERS...
17 8635 0000                    .WORD 0
18 8637 0000                    .WORD 0
19 8639 0000                    .WORD 0
20 863B 0000                    .WORD 0
21
```