APPLESOFT

BASIC

INTERPRETER

INTERNAL ENTRY POINTS
AND
SUBROUTINE DESCRIPTION

A guide to interfacing to the Applesoft Basic Interpreter for the Apple Computer (or compatible) 6502 Assembly Language programmer.

APPLESOFT

BASIC

INTERPRETER

INTERNAL ENTRY POINTS
AND
SUBROUTINE DESCRIPTION

A guide to interfacing to the Applesoft Basic Interpreter for the Apple Computer (or compatible) 6502 Assembly Language programmer.

APPLESOFT BASIC INTERPRETER

INTERNAL ENTRY POINTS AND SUBROUTINE DESCRIPTION

INTRODUCTION

This is a guide for the 6502 machine language programmer who wants to take advantage of the various subroutines in Applesoft. The addresses included assume that the user has an Apple // series computer (or compatible), with Applesoft in firmware or Language Card. This list is believed to be correct, but be warned that it was a spare time project. If you find errors, contact your user group.

This data is meant for the experienced programmer, *NOT THE BEGINNER*. This compilation of data is taken mostly from the article published in APPLE ORCHARD written by John Crossley. However, many other sources were quoted. Many errors have been corrected and additional data has been included. However the possibility for errors still exists. Let caution be your guide. Remember that Applesoft uses an incredible amount of Zero Page locations therefore making it very difficult to write programs that must coexist with it in a 6502 environment. You would be well advised to have a assembly source code listing to compliment this document.

Read your Applesoft Reference manual for more information.

Take special note of **CHRGET**. This subroutine is the heart of Applesoft. When Applesoft wants the next character or an instruction it points **TXTPTR** at the program or the input buffer and JSRs to **CHRGET**. When Applesoft **READS DATA**, **TXTPTR** is temporarily set to the last used **DATA** statement.

The COMMAND TABLE from which Applesoft parses is located at $D0D0. The format is: each character has it's high order bit off except the last.

The VECTOR TABLE is at the beginning of Applesoft at $D000. The format is: each two bytes constitutes an address in lo-hi order form, This address points to one less than the location of the routine that handles the associated command. The ERROR TEXT TABLE is at $D260. The format is the same as for the COMMAND TABLE.

$D000--$D07F	Statement Entry Table
$D080--$D0B0	Function Entry Table
$D0B2--$D0CF	Operator Tag Entry Table
$D0D0-$D25F	Keyword Token Table
$D260--$D364	Error Text Table

ABBREVIATIONS

A the 6502 accumulator
X the 6502 X register
Y the 6502 Y register
Z the zero flag of the 6502 status register
C the carry flag of the 6502 status register

(A,X) is a 16 bit number where A has the most significant byte and X the least significant byte.
(Y,A) is the number or string whose address is in Y and A with the msb in Y and the lsb in A.

FAC the Floating point ACcumulator
ARG the ARGument register
msb most significant bit or byte
lsb least significant bit or byte
eol end of line token ($00)

APPLESOFT VARIABLES

LABELS	HEX ADR	LABELS
A1	3C,3D	Apple monitor pointer for cassette routines
A2	3E,3F	Apple monitor pointer for cassette routines
ARYTAB	6B,6C	Start of array storage
BUF	200-2FF	Line input buffer
CH	24	Horizontal cursor position.
CV	25	Vertical cursor position.
CHARAC	0D	Used by **STRLT2**
CHRGET	B1-C8	Main Applesoft Input routine.
CMASK	1C	Color masking byte shifted.
COLD	00-02	Cold start vector.
COLLSN	EA	Collision counter for HiRes graphics.
COMSTAT	16	Status for compare routine.
CURINP	7F,80	Current source of input address.
CURLIN	75,76	The current line number. (=$FF if in direct mode.)
DATAFLG	13	Used in **Parse**
DATLIN	7B,7C	Line number of current **DATA** statement
DATPTR	7D,7E	The address where the next **DATA** is to be read from.
DIMFLG	10	Also **PTRGET** scratch.
DSCTMP	9D,9E,9F	Temp string descriptor
ENDCHR	0E	Used by **STRLT2**
ERRFLG	D8	$80 if ONERR active
ERRLIN	DA,DB	Line number where error occurred
ERRNUM	DE	Which error occurred
ERRPOS	DC,DD	**TXTPTR** save for **HNDLERR**
ERRSTK	DF	Stack pointer value before error
FBUFFR	100-110	**FOUT** buffer
FIRST	F0	Used by **PLOTFNS**
FNPNTR	90	Vector to invoke a defined function.
FORPNT	85,86	General pointer, see **COPY**
FRESPC	71,72	Temp pointer for string storage routines
FRETOP	6F,70	Bottom of string storage
GARFLG	13	Used in **GARBAG**
GRBSCR	8F	Scratch pointer for garbage collection.
H2	2C	Used by **PLOTFNS**
HBYTE	26,27	Internal cursor vertical.
HCOLOR	E4	HiRes plotting color.
HIBLOK	D0-D5	Hires scratch pointers.
HIGHDS	94,95	Used by **BLTU**
HIGHTR	96,97	Used by **BLTU**
HIRES	E0,E1,E2	HiRes X and Y coordinates.
HMASK	30	Bit position mask.
HNDX	E5	Internal cursor horizontal.
HPAG	E6	HIRES page to plot on. ($20 for **HGR**, $40 for **HGR2**)
HSCR	1D	HIRES scratch.

INDEX	5E,5F	Temp pointer for moving strings. Also an indirect JMP location.
INVFLG	32	Mask for inverse output
LASTFOP	12	Result of last **FAC** operation. $FF=Integer.
LASTPT	53	Last used temp string pointer
LASTVAR	81,82	Last accessed variable name (ASCII).
LINNUM	50,51	General purpose 16 bit number location
LOWTR	9B,9C	General purpose register. **GETARYPT**, **FINDLN**, **BLTU**
MEMSIZ	73,74	**HIMEM** value.
MULDIV	62-66	Result of last multiply or divide.
NUMDIM	0F	Used in array returns.
OLDLIN	77,78	Last line executed.
OLDTXT	79,7A	Old text pointer save area.
ONERR	F4-F8	Onerr pointers.
ONERLN	F4,F5	Onerr line# to goto if error occurs.
ORMASK	F3	Mask for flashing output
PRGEND	AF,B0	The end of the program text
PNTR	0F,10	General purpose pointer.
READPTR	87,88	**READ** pointer.
PTRGT	10	**PTRGET** scratch.
READPTR	87,88	**READ DATA** pointer.
REMSTK	F8	Stack pointer saved before each statement
ROT	F9	Rotation factor for HiRes graphics.
SCALE	E7	Scale factor for HiRes graphics.
SECBYT	D6	Security byte.
SGNFLG	AB	Sign flag used by math routines.
SHPPNT	1A,1B	Shape pointer during **DRAW**.
SHPTAB	E8,E9	Pointer to current shape table.
SPDBYT	F1	Speed=delay number
STREND	6D,6E	The top of array storage
STRNG1	AB,AC	Pointer to a string. See **MOVINS**
STRNG2	AD,AE	Pointer to a string. See **STRLT2**
STRPTR	60,61	Used by **FIN** to index through input string.
SUBFLG	14	$00 subscripts allowed, $80=no subscripts
TMPFRM	89	Temporary storage for **FRMEVL**.
TEMPPT	52	Last used temporary string descriptor
TEMPST	55-5D	Temporary string descriptor storage.
TRACE	F2	**TRACE** byte. $80=ON.
TXTTAB	67,68	Start of program text
TXTPTR	B8,B9	Main pointer for **CHRGET** subroutine.
USER	0A,0B,0C	USR instruction JMP vector.
V2	2D	Used by **PLOTFNS**
VALTYP	11	Flags last FAC operation $00=number, $FF=string
VARPNT	83,84	Used by **PTRGET**
VARTAB	69,6A	Start of variable storage
WARM	03-05	Warm start vector. Actually JMP's to **STROUT**.

TXTPTR INPUT ROUTINES

CHRGET $00B1 Fetch character in A. Increment **TXTPTR**.
CHRGOT $00B7 Fetch character in A. No increment.

These routines load A from **TXTPTR** and set certain 6502 status flags. X and Y are not changed.

On exit:

A=the character
Z is set if A= ':' ($3A) or eol ($00).
C is clear if A is an ASCII number ('0' to '9').

CHRSET $F150 This is the routine which copies **CHRGET** into zero page. The image is stored at $F10B.

CHARACTER GET/GOT

```
00B1- E6 B8    CHRGET INC   TXTPTR
00B3- D0 02           BNE   CHRGOT
00B5- E6 B9           INC   TXTPTR+1
00B7- AD xx xx CHRGOT LDA   TXTPTR     <--Text Pointer here
00BA- C9 3A           CMP   #$3A       ;A colon
00BC- B0 0A           BCS   RETURN
00BE- C9 20           CMP   #$20       ;A blank
00C0- F0 EF           BEQ   CHRGET
00C2- 38              SEC
00C3- E9 30           SBC   #$30
00C5- 38              SEC
00C6- E9 D0           SBC   #$D0
00C8- 60       RETURN RTS
```

NOTE: This routine is self modifying as the pointer being incremented is contained within the routine.

TXTPTR TO INTEGER

LINGET $DAOC Read a line number (integer 0 to 63999) from **TXTPTR** into **LINNUM**. **LINGET** assumes that the 6502 registers and A have been set up by the **CHRGET** that fetched the first digit. Normally exits through **CHRGET** which fetches the character after the number. If the number is greater than 63999 then **LINGET** exits via SYNTAX ERROR. **LINNUM** is zero if there is no number at **TXTPTR**.

GTBYTC $E6F5 JSR to **CHRGET** to gobble a character and fall into **GETBYT**.

GETBYT $E6F8 Evaluates the formula at **TXTPTR**, leaves the result in **FAC**, and falls into **CONINT**. On entry **TXTPTR** points to the first character of the formula for the first number.

PLOTFNS $F1EC Get 2 LORES plotting coordinates (0-47,0-47) from **TXTPTR** separated by a comma. On entry **TXTPTR** points to the first character of the formula for the first number. **PLOTFNS** puts the first number in **FIRST** and the second number in **H2** and **V2**.

HFNS $F6B9 Get HIRES plotting coordinates (0-279,0-191) from **TXTPTR**. On entry **TXTPTR** points to the first character of the formula for the first number. Leaves the 6502 registers set up for **HPOSN**.
On exit:

A=vertical coordinate
X=lsb of horizontal coordinate
Y=msb of horizontal coordinate.

FLOATING POINT MATH PACKAGE INTRODUCTION

This is the number format used throughout Applesoft:

The exponent is a single byte signed number (**EXP**) in excess $80 form (the signed value has $80 added to it). The mantissa is 4 bytes (**HO**,**MOH**,**MO**,**LO**). The binary point is assumed to be to the right of the most significant bit. Since in binary floating point notation the msb is always 1, the number's sign is kept there when the number is stored in packed form in memory. While in the math package the sign is kept in a separate byte (**SGN**) where only bit 7 is significant. If the exponent is zero then the number is zero although the mantissa isn't necessarily zero. These routines require that A and Z reflect **FACEXP** prior to entry and **SGNFLG** must be set up as well. To insure this:

```
LDA   FACSGN    ($A2)
EOR   ARGSGN    ($AA)
STA   SGNFLG    ($AB)
LDA   FACEXP    ($9D)
JSR   .....     (SUBROUTINE CALL)
```

Examples:

Packed format - 5 bytes each.

	EXP	HO	MOH	MO	LO	SGN
(-10)	84	A0	00	00	00	
(10)	84	20	00	00	00	

FAC format - 6 bytes each.

	EXP	HO	MOH	MO	LO	SGN
(-10)	84	A0	00	00	00	FF
(10)	84	A0	00	00	00	00

Arithmetic routine calling conventions:

For SINGLE argument functions:
The argument is in **FAC**.
The result is left in **FAC**.

For TWO argument functions:
The first argument is in **ARG** (see **CONUPK**).
The second argument is in **FAC**.
The result is left in **FAC**.

FLOATING POINT REGISTERS

NOTE: many of the following locations are used for other things when not being used by the floating point math package.

	FAC	ARG	TEMP1	TEMP2	TEMP3	RND
	7	6	5	5	5	5
EXP	9D	A5	93	98	8A	C9
HO	9E	A6	94	99	8B	CA
MOH	9F	A7	95	9A	8C	CB
MO	A0	A8	96	9B	8D	CC
LO	A1	A9	97	9C	8E	CD
SGN	A2	AA	(packed format)			
EXT	AC		(FAC extension byte)			

REG	RANGE
TEMP3	8A-8E
TEMP1	93-97
TEMP2	98-9C
FAC	9D-A2 and AC
ARG	A5-AA
RND	C9-CD

FLOATING POINT OPERATORS

FMULT	$E97F	Move the number in memory pointed to by Y,A into **ARG** and fall into...
FMULTT	$E982	Multiply **FAC** and **ARG**. On entry A and Z reflect **FACEXP**.
FDIV	$EA66	Move the number in memory pointed to by Y,A into **ARG** and fall into...
FDIVT	$EA69	Divide **ARG** by **FAC**. On entry A and Z reflect **FACEXP**.
FADD	$E7BE	Move the number in memory pointed to by Y,A into **ARG** and fall into...
FADDT	$E7C1	Add **FAC** and **ARG**. On entry A and Z reflect **FACEXP**.
FSUB	$E7A7	Move the number in memory pointed to by Y,A into **ARG** and fall into...
FSUBT	$E7AA	Subtract **FAC** from **ARG**. On entry A and Z reflect **FACEXP**.
FPWRT	$EE97	Exponentiation (**ARG** to the **FAC**'th power). On entry A and Z reflect **FACEXP**.
FAND	$DF55	Logical AND the **FAC** with the **ARG** and leave the result in **FAC**. **FAC**=1 if **FAC** AND **ARG** both non-zero, else **FAC**=0.
FOR	$DF4F	Logical OR the **FAC** with **ARG**. **FAC**=0 if **FAC** and **ARG** both zero, else **FAC**=1.

NOTE: Most move routines in Applesoft set up A and Z to reflect **FACEXP** but a LDA $9D will insure the proper values in the flag.

FLOATING POINT FUNCTIONS

SGN $EB90 Calls **SIGN** and floats the result in the **FAC**.
On Exit:
FAC= 1 if **FAC** was GREATER than 0.
FAC= 0 if **FAC** was EQUAL to 0.
FAC=-1 if **FAC** was LESS than 0.

FROUND $EB72 Round of the **FAC** by one bit.

NOTFAC $DE98 Returns a one if **FAC**=0, else returns a zero.

ABS $EBAF Absolute value of **FAC**. Result remains in the **FAC**.

INT $EC23 Greatest integer value of **FAC**. Uses **QINT** and floats the result and leaves it in the **FAC**.

SQR $EE8D Computes the square root of the number contained in the **FAC**, and leaves it in the **FAC**.

LOG $E941 Log base e of **FAC**. Result remains in **FAC**.

EXP $EF09 Raise e to the **FAC**'th power. Result remains in the **FAC**.

RND $EFAE Form a random number in the **FAC**.

COS $EFEA Takes the cosine of the **FAC**.

SIN $EFF1 Takes the sine of the **FAC**.

TAN $F03A Takes the tangent of the **FAC**.

ATAN $F09E Takes the arctangent of the **FAC**.

FLOATING POINT NUMBER MOVE ROUTINES

MOVFM $EAF9 Move the floating point number in memory pointed to by Y,A (msb,lsb) into the **FAC**. On exit A and Z reflect **FACEXP**.

MOVFP $EAFD Move the real variable in memory pointed to by (**INDEX**, **INDEX**+1) into the **FAC**.

MOVPF $EB27 Pack the **FAC** and move it into memory pointed to by (**FORPNT**,**FORPNT**+1).

MOV2F $EB1E Pack the **FAC** and move it into Temporary register #2. Uses **MOVMF**. On exit A and Z reflect **FACEXP**.

MOV1F $EB21 Pack the **FAC** and move it into Temporary register #1. Uses **MOVMF**. On exit A and Z reflect **FACEXP**.

MOVML $EB23 Pack the **FAC** and move it into the Zero Page area pointed to by X. Uses **MOVMF**. On exit A and Z reflect **FACEXP**.

MOVMF $EB2B Pack the **FAC** and move it into memory pointed to by Y,X (msb,lsb). On exit A and Z reflect **FACEXP**.

MOVFA $EB53 Move the contents of the **ARG** register into the **FAC**, and set **FACEXT** to zero. On exit A=**FACEXP** and Z is set.

MOVAF $EB63 Pack **FACEXT** into **FAC** and move the contents of the **FAC** into the **ARG** register. On exit A=**FACEXP** and Z is set.

TRANS $EB66 Move the **FAC** into the **ARG** without the **FACEXT** byte.

CONUPK $E9E3 Load the **ARG** register from memory pointed to by Y,A (msb,lsb). On exit A and Z reflect **FACEXP**.

ARGPR $E9E7 Load **ARG** from memory pointed to by (**INDEX**, **INDEX**+1).

FACSTK $DE10 Pack the **FAC** and stack it. (6 BYTES).

STKARG $DE47 Pop six bytes off the STACK and store them into the **ARG** register, in ascending order. Then EOR the **ARG** SIGN with **FACSGN** and store it at $AB. A returns containing **FACEXP**.

SUMMARY OF MOVE ROUTINES

FAC	--> (Y,X)	**MOVMF**---(EB2B)
FAC	--> (**FORPNT**)	**MOVPF**---(EB27)
(**INDEX**)	--> **FAC**	**MOVFP**---(EAFD)
FAC	--> (0,X)	**MOVML**---(EB23)
FAC	--> **TEMP1**	**MOV1F**---(EB21)
FAC	--> **TEMP2**	**MOV2F**---(EB1E)
FAC	--> **ARG**	**MOVAF**---(EB63)
(Y,A)	--> **FAC**	**MOVFM**---(EAF9)
(Y,A)	--> **ARG**	**CONUPK**-(E9E3)
ARG	--> **FAC**	**MOVFA**---(EB53)
(**INDEX**)	--> **ARG**	**ARGPR**---(E9E7)
FAC	--> STK	**FACSTK**-(DE10)
STK	--> **ARG**	**STKARG**-(DE47)

FLOATING POINT CONSTANTS

NOTE: The following addresses point to numbers in packed form, suitable for use by **CONUPK** and **MOVMF**.

RND.....$00C9
1/4.....$F070
1/2.....$EE64
-1/2.....$E937
1.....$E913
10.....$EA50
SQR(.5).....$E92D
SQR(2).....$E932
LN(2).....$E93C
LOG(e)2.....$EEDB
π/2.....$F063
π*2.....$F06B
-32768.....$E0FE
1000000000...$ED14

FLOATING POINT UTILITIES

SIGN $EB82 Set A according to the value of the **FAC**.
On exit:
A=$01 if **FAC** is POSITIVE (+).
A=$00 if **FAC** is ZERO.
A=$FF if **FAC** is NEGATIVE (-).

FOUT $ED34 Creates a string in **FBUFFR** equivalent to the value in the **FAC**. The string ends in a zero byte. The **FAC** is scrambled during this operation. On exit Y,A points to the string. You may use **STROUT** to then print the number.

COMPARE $DF6A Compare the **FAC** with **ARG** according to the status flag **COMSTAT**($16). Result is left in the **FAC**. 1=TRUE 0=FALSE.

TEST			COMSTAT
ARG	>	**FAC**	1
ARG	=	**FAC**	2
ARG	>=	**FAC**	3
ARG	<	**FAC**	4
ARG	<>	**FAC**	5
ARG	<=	**FAC**	6

FCOMP $EBB2 Compare the **FAC** with a number in memory, in packed form, pointed to by Y,A.
On exit:
A=$01 if (Y,A)<**FAC**
A=$00 if (Y,A)=**FAC**
A=$FF if (Y,A)>**FAC**

NEGOP $EED0 The **FAC** is negated. (**FAC** = -**FAC**)

FACNRM $E82E Normalize the **FAC**.

FACPCK $EB72 Pack the **FAC**.

FADDH $E7A0 Add 1/2 to the **FAC** contents.

DIV10 $EA55 Divide the **FAC** by 10. Returns positive numbers only.

MUL10 $EA39 Multiply the **FAC** by 10. Works for both positive and negative numbers.

INTEGER TO FLOATING POINT CONVERSION

SNGFLT $E301 Float the unsigned integer in Y.

GIVAYF $E2F2 Float the signed integer in A,Y

FLOAT $EB93 Float the signed integer in A.

FLOATING POINT TO INTEGER CONVERSION

CONINT $E6FB Convert the **FAC** into a single byte number in both X and **FACLO**. Normally exits through **CHRGET**. If **FAC** is greater than 255 or less than 0 then exit via **ERROR** with an ILLEGAL QUANTITY ERROR.

CONDBL $E108 Same as **AYINT** but **FAC** must be positive for unsigned conversion.

AYINT $E10C If **FAC** is less than +32767 and greater than -32767 then perform **QINT**.

QINT $EBF2 Quick greatest integer function. Leaves INT(**FAC**) in **FACHO**, **FACMO**, **FACLO** in signed form. Assumes that **FAC** < 2 to the 23rd power. (8388608 in decimal).

FACINT $DA65 Pack the **FAC** and convert it to a two byte integer and store the integer in memory pointed to by (**FORPNT**, **FORPNT**+1). Integer is stored Hi,Lo order form.

GETADR $E752 Convert the number in the **FAC** into a 2 byte integer (0-65535) in **LINNUM** and in A,Y. If **FAC** < -65535 or > 65535 then ILLEGAL QUANTITY ERROR.

GETINT $DEE9 Load an integer variable pointed to by (**FACMO**,**FACLO**) into the **FAC** and float it.

GETNUM $E746 Read a 2 byte integer number into **LINNUM** from **TXTPTR**, check for a comma ',' and get a single byte number in X. On entry **TXTPTR** points to the first character of the formula for the first number. Uses **FRMNUM**, **GETADR**, **CHKCOM**, **GETBYT**.

COMBYT $E74C Check for a comma ',' and get a byte in the X register. Uses **CHKCOM**, **GETBYT**. On entry **TXTPTR** points to the comma.

TXTPTR TO FAC

FRMEVL $DD7B Evaluate the formula at **TXTPTR** using the **CHRGET** routine and leave the result in the **FAC**. On entry the **TXTPTR** points to the first character of the formula. This is the main subroutine for the commands that use formulas and works for both STRINGS and FORMULAS. This routine expects the formula in tokenized form else SYNTAX ERROR results. If the formula is a string literal then **FRMEVL** gobbles the opening quote and executes **STRLIT** and **STRTXT** which leaves a pointer to the generated string's descriptor in **FACMO**, **FACLO**. Precede calls to this routine with a JSR CHRGOT to insure proper operation. If **FRMEVL** encounters string formulas it will generate temporary strings, and will increment **TEMPPT** for each one. You must reset this pointer or else FORMULA TOO COMPLEX wiil occur. You may use **FREFAC** to do this.

FRMNUM $DD67 Evaluate the formula at **TXTPTR** and leave it in the **FAC**, and make sure that it is a number. On entry **TXTPTR** points to the first character of the formula. If the formula is a STRING then **FRMNUM** exits with a TYPE MISMATCH ERROR.

FIN $EC4A Input a floating point number into the **FAC** from **CHRGET**. **FIN** assumes that the 6502 registers and A have been set up by the **CHRGET** call that fetched the first digit. Returns $FF in X if there is nothing to convert.

FRMINT $E105 Evaluate the formula at **TXTPTR** using **FRMNUM** (which must be non-negative) and convert it to a 2 byte integer in **FACMO**, **FACLO**.

STRING UTILITIES

In APPLESOFT strings have three parts: the DESCRIPTOR, the PQINTER to the descriptor, and the ASCII STRING. A string descriptor contains the the length of the string it describes and the address of it's first character Lo-Order, Hi-Order form. Through most of the routines the descriptor is kept in memory while a pointer to the descriptor is kept in the **FAC**. The pointer is the address of the descriptor. The actual string could be anywhere in memory. In a program, the text: A$="HELLO" will leave a descriptor with the length showing 5 and a pointer into the program text where the statement was encountered.

CAT $E597 Concatenate two strings. On entry **FACMO**, **FACLO** points to the first string, and **TXTPR** points to the + (plus sign) prior to the name of the second string.

STRINI $E3D5 Get space for the creation of a string and create a descriptor for it in **DSCTMP**. On entry A=length of the string. On exit Y,X points to the allocated area in memory for the string.

STRSPA $E3DD Calls the **GETSPA** routine and then stores the pointer in **DSCTMP**.

COPY $DAB7 Free the string temporary pointed to by Y,A and move it into memory pointed to by **FORPNT**.

MOVINS $E5D4 Move a string whose descriptor is pointed to by **STRNG1** into memory pointed to by **FRESPA**.

SAVSTR $DAA1 Save string in memory whose descriptor is in **DSCTMP**.

SAVDSC $DA9A Save descriptor in **DSCTMP** into memory pointed to by **FORPNT**,**FORPNT**+1.

MOVSTR $E5E2 Move the string pointed to by Y,X with a length of A into memory pointed to by **FRESPA**.

ST2TXT $E73D Move string pointer contained in **STRNG2** into **TXTPTR**.

STRTXT $DE81 Sets Y,A equal to **TXTPTR** plus C and falls into **STRLIT**.

STRLIT $E3E7 Store a quote in **ENDCHR** and in **CHARAC** so that **STRLT2** will stop on it.

STRLT2 $E3ED Take a string literal whose first character is pointed to by Y,A and build a descriptor for it. The descriptor is built in **DSCTMP**, then **PUTNEW** transfers it into a temporary and leaves a pointer to it in **FACMO** (A0=lsb), **FACLO**

(A1=msb). Characters other than zero that terminate the string should be saved in **CHARAC** and **ENDCHR**. Leading quotes should be skipped before **STRLT2**. On exit the character just after the string literal is pointed to by **STRNG2**. Falls into **PUTNEW**.

PUTNEW $E42A Some string function is returning with a result in **DSCTMP**. Move **DSCTMP** to a temporarary descriptor, put a pointer to the descriptor in **FACMO**, **FACLO**. And flag the result as a string.

GETSPA $E452 Get space for the creation of a string. May force garbage collection to occur. **GETSPA** moves **FRESPC** and **FRETOP** down enough to store the string. On entry A=number of characters in the string. **GETSPA** returns with A unaffected and a pointer to the freed up space in Y,X. **FRESPC** and **FRETOP** also point to the freed up space. If there is no space available then exits with an OUT OF MEMORY ERROR.

FRESTR $E5FD Make sure that the last **FAC** operation result was a string and fall into **FREFAC**.

FREFAC $E600 Frees up the temporary descriptor pointed to by **FACMO**, **FACLO**.

FRETMP $E604 Free up a temporary string. On entry the pointer to the descriptor is in Y,A. A check is made to see if the desciptor is a temporary one allocated by **PUTNEW**. If so, the temporary is freed up by updating **TEMPPT**. If a temporary is freed up then a further check is made to see if the string is the lowest in memory. If so, that area in memory is also freed up by updating the **FRETOP** pointer. On exit the address of the string is in Y,X and in **INDEX**, and the string length is in A.

FRETMS $E635 Free up the descriptor without freeing up the string. On entry Y,A points to the descriptor to be freed. On exit Z is set if anything was freed.

DEVICE INPUT ROUTINES

INLIN $D52C Issue no prompt and fall into...

INLIN+2 $D52E Use character in X as prompt character and input a line of text from the current input device into the input buffer (**BUF**) and fall into **GDBUFS**...

GDBUFS $D539 Puts a zero at the end of the input buffer and masks off the msb on all the bytes contained in **BUF**.
On entry:
X=the end of the input line.
On exit:
A=$00
X=$FF
Y=$01

INCHR $D553 Get one character from the current input device in A and mask off the msb. **INCHR** uses the main APPLE input routines through **KSWL** and supports normal handshaking.

DEVICE OUTPUT ROUTINES

STROUT $DB3A Print string pointed to by Y,A. The string must end with a zero or a quote.

STRPRT $DB3D Print a string whose descriptor is pointed to by **FACMO**, **FACLO**.

OUTDO $DB5C Print the character in A. INVERSE, FLASH, and NORMAL in effect.

CRDO $DAFB Print a carriage return.

OUTSPC $DB57 Print a space.

OUTQST $DB5A Print a question mark.

INPRT $ED19 Print "IN" and the current line number from **CURLIN**. Uses **LINPRT**.

LINPRT $ED24 Prints the 2 byte unsigned number in X,A.

PRNTFAC $ED2E Prints the current value of **FAC**. **FAC** is destroyed. Uses **FOUT** and **STROUT**.

INTERNAL LOCATOR ROUTINES

PTRGET $DFE3 Read a variable name from **CHRGET** and find it in memory. On entry **TXTPTR** points to the first character of the variable name. On exit the address to the value of the variable is in **VARPNT** and Y,A (Y=msb,A=lsb). If **PTRGET** can't find a simple variable it creates one. If it can't find an array it creates one dimensioned to 10 and set all elements equal to zero. **PTRGET** also stores the variable name ASCII characters in **LSTVAR** and **LSTVAR**+1. **PTRGET** also sets up **VALTYP** as follows: 0 if pointer fetched was for a numeric and $FF for a string. This means that **CHKSTR**, **CHKNUM**, and **CHKVAL** will work for this function. **PTRGET** also stores the pointer to the start of the variable table or array storage area in **LOWTR**,**LOWTR**+1. If an array variable was found then **HIGHDS**,**HIGHDS**+1 will point to the first element of the array.

RESULTING FLAGS:

	LASTVAR		VALTYP	LASTFOP
Variable type	**$81**	**$82**	**$11**	**$12**
REAL	POS	POS	$00	$00
STRING($)	POS	NEG	$FF	$00
INTEGER(%)	NEG	NEG	$00	$80

GETARYPT $F7D9 Read a variable name from **CHRGET** and find it in memory. On entry **TXTPTR** points to the first character of the variable name. This routine leaves **LOWTR** pointing to the name of the variable array. If the array can't be found the result is an OUT OF DATA ERROR. **GETARYPT** leaves **TXTPTR** pointing at the indice inside the parentheses. This routine also sets up the flags as in **PTRGET**.

FNDLIN $D61A Searches the program for the line whose number is in **LINNUM**.

On exit:

1. If C set **LOWTR** points to the link field of the desired line.
2. If C clear then line not found. **LOWTR** points to the next higher line.

DATA $D995 Move **TXTPTR** to the end of the statement. Looks for **':'** or eol ($00).

DATAN $D9A3 Calculate the offset in Y from **TXTPTR** to the next **':'** or eol ($00).

REMN $D9A6 Calculate the offset in Y from **TXTPTR** to the next eol ($00).

ADDON $D998 Add Y to **TXTPTR**.

INITIALIZATION ROUTINES

SCRTCH	$D64B	The 'NEW' command. Clears the program, variables, and STACK.
CLEARC	$D66C	The 'CLEAR' command. Clears the variables and STACK.
LOMEM	$F2AC	Checks value passed in **LINNUM** to make sure that it is not greater than **MEMSIZ** and not less than **TXTTAB** and copies **LINNUM** into **VARTAB**.
HIMEM	$F28C	Checks value in **LINNUM** to make sure it is not less than **STREND** and sets **FRETOP** and **MEMSIZ** from **LINNUM**.
STKINI	$D683	Clears the STACK.
CHKSTK	$D3D6	This routine takes the current value in A and multiplies it by 2. It then checks the STACK to see if there is enough room remaining to store that many bytes. Exits with OUT OF MEMORY ERROR if not enough room, else it simply returns.
RESTOR	$D849	Sets the DATA pointer, **DATPTR**, to the beginning of the program.
STXTPT	$D697	Set **TXTPTR** to the beginning of the program.
FPINIT	$F128	Sets up Zero Page pointers for Applesoft.
TOKEN	$D559	Tokenizes the input buffer by identifying and replacing keywords with tokens, and removing blanks. It then places two zero bytes at end of the input line.

STORAGE MANAGEMENT ROUTINES

BLTU $D393 Block transfer makes room by moving everything forward.
On entry:
Y,A and **HIGHDS**=destination of high address + 1
LOWTR=lowest address to be moved
HIGHTR=highest address to be moved +1
On exit:
LOWTR is unchanged
HIGHTR=LOWTR - $100
HIGHDS=lowest address transferred - $100

REASON $D3E3 Makes sure there's enough room in memory, checks to be sure that the address Y,A is less than **FRETOP**. May cause garbage collection. Causes OUT OF MEMORY ERROR if there's no room.

GARBAG $E484 Move all currently used strings up in memory as far as possible. This maximizes the free area, and takes a hell of a long time to execute.

MEMFRE $E2E8 Determine amount of free memory left but does not cause garbage collection. Floats the result in the **FAC**.

MISCELLANEOUS BASIC COMMANDS

Note that many commands are not documented because they jump into the new statement fetcher and cannot be used as a subroutine.

CONT $D898 Moves **OLDTXT** and **OLDLIN** into **TXTPTR** and **CURLIN**.

MAIN $D805 This is the main execution loop from which almost all the other routines are called. While **CHRGET** may be the eyes of Applesoft, this routine is it's heart.

NEWSTT $D7D2 Execute a new statement. On entry **TXTPTR** points to the **':'** preceding the statement or the zero at the end of the previous line. Use **NEWSTT** to restart the program with **CONT**. *THIS ROUTINE DOES NOT RETURN!*

RUN $D566 Run the program in memory. *THIS ROUTINE DOES NOT RETURN!*

GOTO $D93E Uses **LINGET** and **FNDLIN** to update **TXTPTR**. **GOTO** assumes that the 6502 registers and A have been set up by the **CHRGET** that fetched the first digit.

LET $DA46 Uses **CHRGET** to get address of the variable, then the equal sign, evaluate the formula, and store it in the target variable. On entry **TXTPTR** points to the first character of the variable name.

HIRES GRAPHICS ROUTINES

There are 4 data in 5 memory locations that specify a point on the hires screen (whether the screen is displayed or not, is irrelevant). These are the EXTERNAL CURSOR data. The five locations are as follows:

X0L $E0: Low order bits of horizontal screen coordinate.
X0H $E1: High order bits of horizontal screen coordinate.
Y0 $E2: Vertical screen coordinate.
HCOLORZ $E4: Color masking byte from the color table ($F6F6-$F6FD).
HPAG $E6: Page indicator ($20=Page1 $40=Page2).

The above set of data was termed EXTERNAL CURSOR, since the actual point plotting is performed by the following five instructions:

```
LDA   $1C
EOR   ($26),Y
AND   $30
EOR   ($26),Y
STA   ($26),Y
```

We may call the data in the following five locations the INTERNAL CURSOR data:

$1C:.....The color masking byte shifted for odd addresses and none for black or white, unchanged otherwise.
$26,$27:.....(Low, High order) address of the byte, corresponding to the page,vertical coordinate and left-most seven points of the screen.
$E5:.....(Register Y) The integer part of the horizontal screen coordinate divided by seven.
$30:.....The bit position taken from the BIT POSITION TABLE corresponding to the remainder of the horizontal coordinate divided by seven.

These two sets of data are equivalent in the sense that given one, the other may be derived from it.

NOTE: Regardless of which screen is being displayed, **HPAG** (location $E6) determines which screen is drawn on. ($20 for **HGR**, $40 for **HGR2**)

HGR2 $F3DB Initialize and clear Page 2 HIRES. Displays Page 2 in all graphics mode, sets **HPAG** to $40, clears Page 2 to black and sets **CMASK** ($1C) to zero (black 1).

HGR $F3E2 Initialize and clear Page 1 HIRES. Displays Page 1 in mixed mode, sets **HPAG** to $20, clears Page 1 to black and sets **CMASK** ($1C) to zero (black 1).

HCLR $F3F2 Clear the HIRES screen to black by loading A with zero and falling into **BKGND**...

BKGND $F3F4 Clear the HIRES screen to last plotted color. Clears the Page defined by **HPAG** to the color defined by the contents of the A register, which should be one from the Color Masking Table. Also stores A in **CMASK** ($1C).

HPOSN $F411 Position the HIRES cursor without plotting, **HPAG** determines which Page the cursor is pointed at. Assumes the registers have been set up as follows: X=lo order bits of the horizontal screen coordinate, Y=hi order bit of the horizontal screen coordinate, A=vertical screen coordinate. This routine makes the INTERNAL CURSOR equivalent to the external one.
On entry:
Horizontal=Y,X
Vertical=A

HPLOT $F457 Call **HPOSN** then try to plot a dot at the cursor's position. No dot may be plotted if plotting non-white at a complementary color X coordinate. This routine expects the registers to be set up exactly as in **HPOSN** which it then calls. Then this routine goes to **PLOT**.

PLOT $F45A This routine executes the five instructions listed at the beginning of this section, which plots a point using the INTERNAL CURSOR data. If this entry is used directly, then the user should be sure that Y is set up from $E5.

INTX $F465 Modifies the INTERNAL CURSOR, in effect incrementing or decrementing the horizontal screen coordinate by one. On entry if the the N flag is zero (positive) then it increments. If the N flag if set (negative) then it decrements. The routine incorporates a wrap-around feature if at the extreme ends of the screen. Uses **DECRX** or **INCRX**.

DECRX $F467 Routine to decrement the horizontal screen coordinate by 1.

INCRX $F48A Routine to increment the horizontal screen coordinate by 1.

INTY $F4D3 Modifies the internal cusor data in $26, $27 to reflect an increment or decrement of the vertical screen coordinate by 1. On entry the N flag is checked and if it is set

(negative) then jump to **INCRY**. If the N flag is not set (positive) then go to **DECRY**. Note that the sign convention is the OPPOSITE of that used by **INTX**. This routine also incorporates a wrap-around feature.

DECRY $F4D5 Routine to decrement the vertical screen coordinate by 1.

INCRY $F504 Routine to increment the vertical screen coordinate by 1.

HLIN $F53A Draws a line from the last plotted point or line destination to the coordinate in the 6502 registers. The routine draws a line from the INTERNAL CURSOR to the point defined by the input.
On entry:
Horizontal=X,A
Vertical=Y

HFIND $F5CB Converts the HIRES cursor's position to X-Y coordinates. Used after SHAPE to find where you've been left. Sets the EXTERNAL CURSOR data equal to the INTERNAL CURSOR data.
On exit:
$E0=horizontal lsb
$E1=horizontal msb
$E2=vertical

DRAW $F601 Draw the shape pointed to be Y,X by ploting in the existing color. On entry A=rotation factor.

XDRAW $F65D Draw the shape pointed to by Y,X by inverting the existing color of the dots the shape draws over. On entry, A=rotation factor.

SETHCOL $F6EC Set the HIRES color to X. X must be less than 8.

SHLOAD $F775 Loads a shape table into memory from tape above **MEMSIZ** (HIMEM) and sets up the pointer at $E8.

HCOLOR $F6F0 Assumes register X contains the color index (0-7). This routine picks up the appropriate color code from the Color Masking Table and stroes it in $E4.

BIT POSITION TABLE

```
$F5B2:  $81 = 10000001
$F5B4:  $84 = 10000100
$F5B5:  $88 = 10001000
$F5B6:  $90 = 10010000
$F5B7:  $A0 = 10100000
$F5B8:  $C0 = 11000000
```

COLOR MASKING TABLE

```
$F6F6:  $00 = 00000000  (BLACK 1)
$F6F7:  $2A = 00101010
$F6F8:  $55 = 01010101
$F6F9:  $7F = 01111111  (WHITE 1)
$F6FA:  $80 = 10000000  (BLACK 2)
$F6FB:  $AA = 10101010
$F6FC:  $D5 = 11010101
$F6FD:  $FF = 11111111  (WHITE 2)
```

ERROR PROCESSOR ROUTINES

ERROR $D412 Checks **ERRFLG** and jumps to **HNDLERR** if ONERR is active. Otherwise it prints '?' <error message &X> '**ERROR**'. If this is during program execution then it also prints 'IN' and the **CURLIN**.

HANDLERR $F2E9 Saves **CURLIN** in **ERRLIN**, **TXTPTR** in **ERRPOS**, X in **ERRNUM**, and **REMSTK** in **ERRSTK**. **REMSTK** is equal to the 6502 stack pointer and is set up at the start of each statement. X containes the error code. This may be used to interrupt the execution of a BASIC program.

SNERR $DEC9 Exit with SYNTAX ERROR.

UNDERR $D97C Exit with UNDEFINED STATEMENT ERROR.

OVFERR $E8D5 Exit with OVERFLOW ERROR.

FTCERR $E430 Exit with FORMULA TOO COMPLEX ERROR.

DIVERR $EAE1 Exit with DIVISION BY ZERO ERROR.

TYPERR $DD76 Exit with TYPE MISMATCH ERROR.

MEMERR $D410 Exit with OUT OF MEMORY ERROR.

IQERR $E199 Exit with ILLEGAL QUANTITY ERROR.

SUBERR $E196 Exit with BAD SUBSCRIPT ERROR.

NXTERR $DD0B Exit with NEXT WITHOUT FOR ERROR.

ILDERR $E306 Exit with ILLEGAL DIRECT ERROR.

RETERR $D979 Exit with RETURN WITHOUT GOSUB ERROR.

RESUME $F317 Restores **CURLIN** from **ERRLIN** and **TXTPTR** from **ERRPOS** and transfers **ERRSTK** into the 6502 stack pointer.

ERROR TABLE

#	ERROR TEXT
0	**NEXT** without **FOR**
1	Language Not Available †
2	Range Error †
3	Range Error †
4	Write protected Disk †
5	End of Data †
6	File Not Found †
7	Volume Mismatch †
8	I/O Error †
9	Disk Full †
10	File Locked †
11	Syntax Error †
12	No Buffers Available †
13	File Type Mismatch †
14	Program Too Large †
15	Not Direct Command †
16	Syntax Error
22	**RETURN** without **GOSUB**
42	Out of Data
53	Illegal Quantity
69	Overflow
77	Out of Memory
90	Undefined Statement
107	Bad Subscript
120	Redimensioned Array
133	Division by Zero
163	Type Mismatch
176	String Too Long
191	Formula Too Complex
224	Undefined Function
254	Bad Response to INPUT Statement
255	Control C Interrupt Attempted

NOTE: Error codes marked with † are DOS 3.3 ERRORS, and were included here to make this list complete.

Error codes are found by `PEEK(222)`. Also, `LINE NUMBER= PEEK(218) + (PEEK(219) * 256)`. `POKE 216,0` will clear `ONERR` status.

The following is a short machine code routine that is used to clean up the STACK if an ONERR statement is used without the matching RESUME. Simply CALL it from within the BASIC code that handles errors before doing a GOTO, RETURN, etc.

ERROR HANDLER:

```
768:104 168 104 166 223 154  72 152  72  96
```

CALL 768 within routine to activate.

NOTE: The previous routine is also known as "ONERR FIX" and can be relocated to any other convenient (and available) location in memory. The following is the same information in hexidecimal.

$0300: 68 A8 68 A6 DF 9A 48 98 48 60

SYNTAX CHECKING ROUTINES

ISCNTC $D858 Checks the Apple keyboard for a control - C ($83). Executes the BREAK routine if there is a control - C.

CHKNUM $DD6A Make sure **FAC** is numeric. See **CHKVAL**.

CHKSTR $DD6C Make sure **FAC** is a string. See **CHKVAL**.

CHKVAL $DD6D Checks the result of the most recent **FAC** operation to see if it is a string or numeric variable. A TYPE MISMATCH ERROR results if **FAC** and C don't agree.
On entry:
C set checks for strings
C clear checks for numerics

ERRDIR $E306 Causes ILLEGAL DIRECT ERROR if the program isn't running. X is modified.

ISLETC $E07D Checks A for an ASCII letter ('A' to 'Z'). On exit C set if A is a letter.

PARCHK $DEB2 Checks for '(', evaluates a formula, and checks for ')'. Uses **CHKOPN** and **FRMEVL** then falls into **CHKCLS**.

CHKCLS $DEB8 Checks at **TXTPTR** for ')'. Uses **SYNCHR**.

CHKOPN $DEBB Checks at **TXTPTR** for '(', Uses **SYNCHR**.

CHKCOM $DEBE Checks at **TXTPTR** for ','. Uses **SYNCHR**.

SYNCHR $DEC0 Checks at **TXTPTR** for the character in A. **TXTPTR** is not modified. Normally exits through **CHRGET**. Exits with SYNTAX ERROR if they don't match.

MISC ROUTINES

MAIN $D43C Main command loop for Applesoft.

PARSER $D56C This is the main input parser routine. More will be added later.

LIST $D6A5 This is the routine that parses the range of linenumbers to be used and lists out the program utilizing the command table text at $D0D0.

GETFAC $D72C This routine uses the **FAC** as a zero page indirect pointer and fetches bytes in A from it. Set up starting pointers in **FACEXP** and **FACHO**, then set up Y. Call this routine as necessary. **GETFAC** will change **FACHO** as it goes.

EXTRA $DCD7 Prints out ?EXTRA IGNORED and returns. The text for this message is at $DCDF followed by the text for ?REENTER at $DCEF.

PADDLE $DFCD Read a paddle input. Expects the **FAC** to contain the number of the paddle to be read. Result is floated in the **FAC**.

CASSETTE ROUTINES

SAVE $D8B0 Save the program in memory to tape.

LOAD $D8C9 Load a program from tape..

VARTIO $D8F0 Set up **A1** and **A2** to save 3 bytes ($50-$52) for the length.

PROGIO $D901 Set up **A1** and **A2** to save the program text.

VARIABLE STORAGE FORMATS

SIMPLE INTEGER VARIABLE

BYTE	Description
0	VNAME CHAR #1 (NEG)
1	VNAME CHAR #2 (NEG)
2	HIGH BYTE VALUE
3	LOW BYTE VALUE
4	Unused
5	Unused
6	Unused

SIMPLE REAL VARAIBLE

BYTE	Description
0	VNAME CHAR #1 (POS)
1	VNAME CHAR #2 (POS)
2	EXPONENT
3	MANTISSA (MSB)
4	MANTISSA
5	MANTISSA
6	MANTISSA (LSB)

SIMPLE STRING VARIABLE

BYTE	Description
0	VNAME CHAR #1 (POS)
1	VNAME CHAR #2 (NEG)
2	STRING LENGTH
3	LOW BYTE ADDRESS POINTER TO STRING
4	HIGH BYTE ADDRESS
5	Unused
6	Unused

DEFINED FUNCTION ENTRY

BYTE	Description
0	VNAME CHAR #1 (NEG)
1	VNAME CHAR #2 (POS)
2	LOW BYTE ADDRESS OF DEFINITION
3	HIGH BYTE ADDRESS OF DEFINITION
4	LOW BYTE ADDRESS OF TARGET VARIABLE
5	HIGH BYTE ADDRESS OF TARGET VARIABLE
6	POS ASCII representation of char after "="

ARRAY INTEGER VARIABLE

BYTE	Description
0	VNAME CHAR #1 (NEG)
1	VNAME CHAR #2 (NEG)
2	LOW BYTE OFFSET POINTER TO NEXT ENTRY
3	HIGH BYTE OFFSET
4	NUMBER OF DIMENSIONS
5	HIGH BYTE SIZE LAST DIMENSION
6	LOW BYTE SIZE LAST DIMENSION
7	HIGH BYTE SIZE FIRST DIMENSION
8	LOW BYTE SIZE FIRST DIMENSION
	INTEGER (0,0,..,0)
n	HIGH BYTE VALUE
n+1	LOW BYTE VALUE

NEXT VALUE...

ARRAY REAL VARIABLE

BYTE	Description
0	VNAME CHAR #1 (POS)
1	VNAME CHAR #2 (POS)
2	LOW BYTE OFFSET POINTER TO NEXT ENTRY
3	HIGH BYTE OFFSET
4	NUMBER DIMENSIONS
5	HIGH BYTE SIZE LAST DIMENSION
6	LOW BYTE SIZE LAST DIMENSION
7	HIGH BYTE SIZE FIRST DIMENSION
8	LOW BYTE SIZE FIRST DIMENSION
	REAL (0,0,..,0)
n	EXPONENT
n+1	MANTISSA (MSB)
n+2	...
n+3	...
n+4	MANTISSA (LSB)

NEXT VALUE...

ARRAY STRING VARIABLE

BYTE	Description
0	VNAME CHAR #1 (POS)
1	VNAME CHAR #2 (NEG)
2	LOW BYTE OFFSET POINTER TO NEXT ENTRY
3	HIGH BYTE OFFSET
4	NUMBER DIMENSIONS
5	HIGH BYTE SIZE LAST DIMENSION
6	LOW BYTE SIZE LAST DIMENSION
7	HIGH BYTE SIZE FIRST DIMENSION
8	LOW BYTE SIZE FIRST DIMENSION
	STRING (0,0,..,0)
n	STRING LENGTH
n+1	LOW BYTE POINTER TO STRING TEXT
n+2	HIGH BYTE POINTER TO STRING TEXT

NEXT STRING POINTER

STACKING FORMATS

FOR/NEXT (18 bytes)

1-2	The begin address of the statement following the FOR statement.
3-4	The current line number.
5-9	The loop limit.(5 bytes)
10-15	The STEP increment. (6 bytes)
16-17	A pointer to the loop control variable.
18	Hex code $81 (code for loops).

GOSUB/RETURN (7 bytes)

1-2	An internal subroutine return address.
3-4	The begin address of the statement following the GOSUB statement.
5-6	The current line number.
7	Hex code $B0 (code for GOSUBs).