CHAPTER 4
ALPHABETIC LIST OF SYSTEM ROUTINES IN ROM

ACOS: ARCCOSINE FUNCTION

ACOS calculates the trigonometric arccosine of the floating point value in registers >75 through >7C. The result, in the angular units set in the indicator status byte, is returned in registers >75 through >7C. If the input value is less than -1 or greater than 1, a BAD ARGUMENT error code (>1D) is stored in A and the routine traps to the error handler. If any errors occur during the evaluation, the appropriate error code is returned in register >7F.

ENTRY CONDITIONS:	Floating point value in >75 through >7C
EXIT CONDITIONS:	Arccosine in >75 through >7C; Error status in >7F
CALL ADDRESS:	MOVD %>016C,B CALL @CALPAG
REGISTERS USED:	A, B, >58 through >7F

ASIN: ARCSINE FUNCTION

ASIN calculates the trigonometric arcsine of the floating point value in registers >75 through >7C. The result, in the angular units set in the indicator status byte, is returned in registers >75 through >7C. If the input value is less than -1 or greater than 1, a BAD ARGUMENT error code (>1D) is stored in A and the routine traps to the error handler. If any errors occur during the evaluation, the appropriate error code is returned in register >7F.

ENTRY CONDITIONS:	Floating point value in >75 through >7C
EXIT CONDITIONS:	Arcsine in >75 through >7C; Error status in >7F
CALL ADDRESS:	MOVD %>0169,B CALL @CALPAG
REGISTERS USED:	A, B, >58 through >7F

ATAN: ARCTANGENT FUNCTION

ATN calculates the tri gonometric arctangent of the floating point value in registers >75 through >7C. The result, in the angular units set in the indicator status byte, is returned in registers >75 through >7C. If any errors occur during the evaluation, the appropriate error code is returned in register >7F.

ENTRY CONDITIONS:	Floating point value in >75 through >7C
EXIT CONDITIONS:	Arctangent in >75 through >7C; Error status in >7F
CALL ADDRESS:	MOVD %>0166,@ CALL @CALPAG
REGISTERS USED:	A, B, >58 through >7F

ASSIGN: ASSIGN A VALUE TO A VARIABLE

ASSIGN assigns the value identified by the contents of registers >75 through >7C to the variable identified by the entry on top of the floating point stack. If the value and the variable are not the same type, ASSIGN stores the "String-number mismatch" error code >03 in register A and TRAPs to the error handler.

ENTRY CONDITIONS:	BASIC operating environment Variable information entry on top of floating point stack Value information in registers >75 through >7C
EXIT CONDITIONS:	Value assigned to variable
CALL ADDRESS:	MOVD %>0145,@ CALL @CALPAG
REGISTERS USED:	A, B, >54, >55, >5C through >67, >6B through >72, >77, >78
RAM USED:	>8E6, >8E7

BATCHK: CHECK BATTERY CONDITION

BATCHK checks the condition of the batteries and returns the status in RAM location >83C. This location contains the status of some of the display indicators including the LOW battery indicator (bit 1). If the batteries are low, bit 1 of location >83C is Set to ONE. If the batteries are not low, the indicator bit is Reset to ZERO.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	Battery condition in >83C
CALL ADDRESS:	MOVD %>0060,@ CALL @CALPAG
REGISTERS USED:	A, B
RAM USED:	>83C

BEEP: BEEP ROUTINE

BEEP sounds a single short tone through the beeper.

ENTRY CONDITIONS: None
EXIT CONDITIONS: None
CALL ADDRESS: MOVD %>005D,8
CALL @CALPAG
REGISTERS USED: A, B, >5C, >5D

BRKPT: BREAKPOINT HANDLER

BRKPT is a routine that is used within the BASIC operating environment to process breakpoints. This routine should be called when a program or subprogram detects that the BREAK key has been pressed.

If ON BREAK STOP (default) is in effect, BRKPT reports breakpoint message through the error handler, stacks the current BASIC program execution context on the floating point stack, and transfers control to the system command level. If the BASIC CONTINUE command is entered execution begins at the beginning of the line in which the BREAK key was detected. If ON BREAK NEXT is in effect, BRKPT simply returns control to the calling program. If ON BREAK ERROR is in effect, BRKPT sends the breakpoint to the error handler to be processed as an error.

ENTRY CONDITIONS: None
EXIT CONDITIONS: None
CALL ADDRESS: MOVD %>0130,8
CALL @CALPAG
REGISTERS USED: A, B

BRPAG: SYSTEM ROM PAGING ROUTINE

BRPAG transfers control to a specified branch table entry point in a specified system ROM page. The page number in register A is selected, the program number is used to calculate the entry point, and control is transferred to the calculated address.

ENTRY CONDITIONS: Page number in A
Program number in B
EXIT CONDITIONS: None
BRANCH ADDRESS: BR @>F839
REGISTERS USED: A, B

BTERP: CARTRIDGE MEMORY PAGING ROUTINE

BTERP transfers control to a specified branch table entry point on a specified cartridge-memory page. The page number in register A is selected, the program number is used to calculate the entry point, and control is transferred to the calculated address.

ENTRY CONDITIONS:	Page number in A Program number in B
EXIT CONDITIONS:	None
BRANCH ADDRESS:	BR @>F85A
REGISTERS USED:	A, B

BUFIN: ACCEPT AND DISPLAY KEYBOARD INPUT

BUFIN is used to accept input from the user. This routine uses >83E and the starting address. First, BUFIN displays the contents of the keyboard input buffer from the starting address >88D. The cursor is placed in the position indicated by register pair (>47,>48). After displaying the contents of the buffer, BUFIN accepts user input, modifies the contents of the buffer accordingly, and rewrites the display with the new data. User input is restricted to the input field defined by the parameters passed to the routine. While the data preceding the input field may be displayed, it cannot be edited. Editing and cursor-control keys are implemented within this routine (with the single exception of FORWARD TAB). Only the enter , shift-enter, and break keys are allowed to terminate the input. The enter and shift-enter keys return control to the calling program. See "Break Key" below for a discussion of the handling of this key.

BUFIN: ACCEPT AND DISPLAY KEYBOARD INPUT (CONT)

Registers >3A, >44, and >4C contain various flags which affect operation of the BUFIN routine as follows:

Register >3A

Bit	Meaning
5	0 no character validation
	1 validate input according to flags in register >44

All other flags ignored.

Register >44 (more than one option may be set)

Bit	Meaning
0	1= alpha; allow A-Z and a-z
1	1= alphanum; allow A-Z, a-z, and 0-9
2	1= ualpha; allow A-Z
3	1= ualphanum; allow A-Z and 0-9
4	1= digit; allow 0-9
5	1= numeric; allow 0-9, +, -, ., and E
6	1= User defined string; When this flag is set, register pair (>45,>46) must point to an ASCII string to be used for character validation. BUFIN will allow those characters which appear in the string. The first byte at the indicated address is the length of the string. The ASCII characters are in reverse order extending from the length byte to the lowest addressed character.

Register >4C

Bit	Meaning
0	This flag is used within BUFIN.
3	0= assignment (SHIFT-FN) keys and function (FN) keys are ignored by BUFIN 1= assignment keys assign contents of input buffer to indicated numeric key and then clear the input buffer. function keys display appropriate keyword or text assigned to numeric key. Assignment and function keys do not terminate BUFIN.

All others are ignored.

BUFIN: ACCEPT AND DISPLAY KEYBOARD INPUT (CONT)

Indicators

BUFIN turns on the left arrow when data is scrolled off the left side of the display and the right arrow when data is scrolled off the right side of the display. The SHIFT, FN, and CTL indicators are on when the keyboard is in shift, function, and control mode, respectively. The SHIFT and FN indicators are on simultaneously when the keyboard is in assignment mode. The UCL indicator is on when the keyboard is placed in upper case lock mode. BUFIN turns on the LOW battery indicator if the batteries are low.

Break Key

Two flags control the processing of the break key within BUFIN, the "program running" flag (bit 5 of register >48) and the "ON BREAK NEXT" flag (bit 1 of RAM location >8FB).

If both flags are set, the break key is ignored in BUFIN.

If the "program running" flag is reset, BUFIN stores the "BREAK" code >A5 in register A and TRAPs to the error handler.

If the "program running" flag is set, but the "ON BREAK NEXT" flag is reset, some operations necessary for a BASIC breakpoint are performed before storing the "BREAK" code and TRAPing to the error handler.

ENTRY CONDITIONS:	Start of input field in (>47,>48) End of input field in (>42,>43) Flag bits in registers >3A, >44, and >4C Address of validate string (if any) in (>43,>46)
EXIT CONDITIONS:	Input data in input field pointed to by (>3B,>3C) Keycode of terminating key in >5C
CALL ADDRESS:	MOVD %>0057,B CALL @CALPAG
REGISTERS USED:	A, B, >39, >3B, >3C, >3E, >3F, >47, >48, >4C, >4E, >4F, >53, >56 through >70, >75, >76
RAM USED:	>834 through >83B, >83E through >88D, >8B4 through >8C8, >8E8 through >8EB

CALPAG: SYSTEM ROM PAGING ROUTINE

CALPAG places the current system page number on the processor stack, selects the new system page, uses a routine number to calculate the address of the routine, and calls the indicated routine. When the called routine returns control to CALPAG, the saved system page number is popped from the stack and restored before control is returned to the original calling program.

ENTRY CONDITIONS:	Page number in A Routine number in B
EXIT CONDITIONS:	None
CALL ADDRESS:	CALL @>F836
REGISTERS USED:	A, B

CFI: CONVERT FLOATING POINT VALUE TO INTEGER VALUE

CFI converts the floating point value in registers >75 through >7C into a 16-bit signed integer from -32768 to 32767. If the floating point value (rounded to the nearest integer) does not fall in this range, CFI returns the "Bad value" code >04 in register >7F. Otherwise, the integer value is returned in register pair (>75,>76) and register >7F contains zero.

ENTRY CONDITIONS:	Floating point value in >75 through >7C
EXIT CONDITIONS:	Integer value in (>75,>76) Error status in >7F
CALL ADDRESS:	MOVD %>0115,B CALL @CALPAG
REGISTERS USED:	A, B, >73 through >7C, >7E, >7F

CFILNG: CONVERT FLOATING POINT TO UNSIGNED INTEGER

CFILNG converts the floating point value in registers >75 through >7C into a 16-bit unsigned integer from 0 to 65535. If the floating point value (rounded to the nearest integer) does not fall in this range, CFI places the "Bad argument" code >1D in the A register and TRAPs to the error handler. Otherwise, the integer value is returned in register pair (>75,>76).

ENTRY CONDITIONS:	Floating point value in >75 through >7C
EXIT CONDITIONS:	Integer value in (>75,>76)
CALL ADDRESS:	MOVD %>017E,B CALL @CALPAG
REGISTERS USED:	A, B, >5D through >60, >6B through >7F

CHKBRK: CHECK FOR THE BREAK KEY

CHKBRK directly checks the keyboard to determine whether the BREAK key is currently pressed. If the BREAK key is down, >FF is returned in the B register. If the BREAK key is not down, zero is returned in the B register.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	Break key status in B register
CALL ADDRESS:	MOVD %>0063,B CALL @CALPAG
REGISTERS USED:	A, B

CIF: CONVERT INTEGER TO FLOATING POINT

CIF converts the signed 16-bit integer value in register pair (>75,>76) into the corresponding floating point value in registers >75 through >7C.

ENTRY CONDITIONS:	Integer value in (>75,>76)
EXIT CONDITIONS:	Floating point value in >75 through >7C
CALL ADDRESS:	MOVD %>0127,B CALL @CALPAG
REGISTERS USED:	A, B, >5D, >5F, >60, >75 through >7E

CIS: CONVERT INTEGER VALUE TO ASCII STRING

CIS converts the signed 16-bit integer value in register pair (>75,>76) into the corresponding ASCII string representation in registers >6E through >74. The string is in reverse order beginning with the length byte in register >74 and the ASCII characters extending toward register >6E.

ENTRY CONDITIONS:	Integer value in (>75,>76)
EXIT CONDITIONS:	String in registers >6E through >74
CALL ADDRESS:	MOVD %>012A,B CALL @CALPAG
REGISTERS USED:	A, B, >5D, >6E through >7E

CLRARG: CLEAR SECOND ARGUMENT

CLRARG clears the second argument registers >6B through >72 by writing nulls into them.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	Nulls in >6B through 72
CALL ADDRESS:	CALL @>F83F
REGISTERS USED:	A, B, >6B through >72

CLRDSP: CLEAR LCD DISPLAY

CLRDSP writes blank characters throughout the display.

ENTRY CONDITIONS: None
EXIT CONDITIONS: Blanks in display
CALL ADDRESS: MOVD %>0045,B
CALL @CALPAG
REGISTERS USED: A, B, >53, >5C

CLRFAC: CLEAR FLOATING POINT ACCUMULATOR

CLRFAC stores nulls into registers >75 through >7C.

ENTRY CONDITIONS: None
EXIT CONDITIONS: Nulls in >75 through >7C
CALL ADDRESS: TRAP >10
REGISTERS USED: A, B, >75 through >7C

CLRINP: CLEAR THE KEYBOARD INPUT BUFFER

CLRINP places blanks in the 80 character keyboard input buffer from >83E to >88D in the RAM.

ENTRY CONDITIONS: None
EXIT CONDITIONS: Blanks in >83E through >88D
CALL ADDRESS: MOVD %>0021,B
CALL @CALPAG
REGISTERS USED: A, B
RAM USED: >83E through >88D

CLRRES: CLEAR NON-ARGUMENT PORTION OF FP AREA

CLRRES clears registers >5D through >6A, the non-argument portion of the floating-point area.

ENTRY CONDITIONS: None
EXIT CONDITIONS: Nulls in >5D through >6A
CALL ADDRESS: CALL @>F82A
REGISTERS USED: A, B, >5D through >6A

CLSALL: CLOSE ALL OPEN FILES

CLSALL closes all open files and devices included in the linked list of PABs. First, CLSALL calls CLSTMP to check whether there is a temporary PAB in registers >3A through >46, and if so it closes that file or device. Then, CLSALL steps through the linked list of PABs, closing each file or device and releasing the its buffer and PAB back to free memory space.

ENTRY CONDITIONS:	BASIC operating environment Address of linked list of peripheral access blocks in (>8EE,>8EF) Temporary PAB flag in >4B
EXIT CONDITIONS:	None
CALL ADDRESS:	MOVD %>021E,B CALL @CALPAG
REGISTERS USED:	A, B, >3A through >4B, >54, >55, >5C through >76, >79, >7A, >7D, >7E
RAM USED:	>8E6, >8E7, >8EE, >8EF

CLSTMP: CLOSE TEMPORARY PAB

CLSTMP tests the temporary PAB flag (bit 3) in register >4B to determine whether registers >3A through >46 contain a temporary PAB. If the flag is set, CLSTMP closes the file or device and resets the flag.

ENTRY CONDITIONS:	BASIC operating environment Temporary PAB in registers >3A through >46 Temporary PAB flag in >4B
EXIT CONDITIONS:	None
CALL ADDRESS:	MOVD %>0239,B CALL @CALPAG
REGISTERS USED:	A, B, >3A through >4B, >66 through >76

CNS: CONVERT FLOATING POINT TO STRING

CNS converts the floating point number in registers >75 through >7C to a corresponding ASCII representation based upon the format information specified in registers >5A through >5C. The register pair (>58,>59) points to the high end of the buffer where the resultant string is to be placed. The string is placed in this buffer in reverse order with the length byte first and the ASCII character codes extending toward the low end.

The values in registers >5A, >5B, and >5C determine the format of the resultant string. If all three registers are zero, CNS adheres to the standard BASIC format for numeric output. If the values are non-zero, they have the following effects.

>5A = number of character positions in the exponent including the "E" symbol and the algebraic sign of the exponent (range: 0, 4, or 5)
>5B = number of character positions to the left of the decimal point including the algebraic sign position (range: 0 to 14)
>5C = number of digit positions to the right of the decimal point including the decimal point (range: 0 to 15)

If the sum of the contents of registers >5B and >5C is greater than 15, CNS stores the "Error in image" code >17 in register A and TRAPs to the error handler.

The floating point value is rounded, if necessary to fit the specified format. If the value cannot be rounded to fit the format, CNS outputs a string of asterisks the length of the defined format.

ENTRY CONDITIONS: Floating point number in >75 through >7C
Destination address of string in (>58,>59)
Format information in >5A through >5C
EXIT CONDITIONS: Formatted ASCII representation of the floating point number
CALL ADDRESS: MOVD %>0242,8
CALL @CALPAG
REGISTERS USED: A, B, >58 through >7F

COPYFA: COPY FLOATING POINT ACCUMULATOR TO SECOND ARG

COPYFA copies the contents of registers >75 through >7C (the floating-point accumulator) into registers >6B through >72 (the second argument).

ENTRY CONDITIONS: None
EXIT CONDITIONS: Values in >75 - >7C also in >6B - >72
CALL ADDRESS: CALL @>F809
REGISTERS USED: >6B through >72
$$$$$$$

COS: COSINE FUNCTION

COS calculates the trigonometric cosine of the floating point value in registers >75 through >7C, which is interpreted in accordance with the current angular measure in the indicator status byte. result is returned in registers >75 through >7C. If a value outside the range of a valid angular argument is input, this routine stores a BAD ARGUMENT error (>1D) in the A register and traps to the error handler. If any errors occur during the evaluation, the appropriate error code is returned in register >7F.

ENTRY CONDITIONS:	Floating point value in >75 through >7C
EXIT CONDITIONS:	Cosine in >75 through >7C Error status in >7F
CALL ADDRESS:	MOVD %>0160,B CALL @CALPAG
REGISTERS USED:	A, B, >58 through >7F

CRTLST: SEARCH CARTRIDGE FOR PROGRAM OR SUBPROGRAM

CRTLST searches the linked list of programs and subprograms in cartridge ROM or RAM. The register pair (>5E,>5F) contains the address of the program-name string for which to search. This name is matched character-by-character, with a distinction made between upper-case and lower-case characters. After selecting page zero of the cartridge memory, CRTLST searches through the list beginning at the address set into RAM locations >8D8,>8DC in the powerup sequence. If the program is found, the address of the program header is returned in register pair (>3A,>3B). If the program is not found, zero is returned in register >3A.

ENTRY CONDITIONS:	Address of linked list of programs in (>8D8,>8DC) Address of program name in (>5E,>5F)
EXIT CONDITIONS:	Address of program header in (>3A,>3B)
CALL ADDRESS:	MOVD %>014E,B CALL @CALPAG
REGISTERS USED:	A, B, >3A through >3F, >6B through >6E

CSI: CONVERT STRING TO INTEGER

CSI converts the ASCII string which register pair R>58,R>59 points to the first byte of into a 16-bit integer value from 0 through 65535. The length of the string is in register R>5B, and register R>5A is null. Leading spaces are skipped prior to beginning the conversion. The conversion stops when any character other than a decimal digit (0-9) is encountered or when no more characters remain. The pointer and string length are adjusted as the conversion is performed. The pointer will return pointing to the character which terminated the conversion or beyond the end of the string. The length will be the remaining number of characters. If an error occurs, CSI returns with the most significant bit of register >7E set. Otherwise, the most significant bit of register >7E is reset.

ENTRY CONDITIONS:	Address of first ASCII character in (>58,>59) Null byte in >5A Length of string in (>5B)
EXIT CONDITIONS:	16-bit integer value in (>60,>61) Error status in >7E
CALL ADDRESS:	MOVD %>003C,B CALL @CALFAG
REGISTERS USED:	A, B, >58 through >5B, >5D through >61, >7E

CSN: CONVERT STRING TO FLOATING POINT

CSN converts the ASCII string which register pair >58,>59 points to the first byte of into a floating point number. The length of the string is in register R>5A, and register R>5A is null. . Leading spaces are skipped prior to beginning the conversion. The conversion stops when a non-numeric character (anything but 0-9, +, -, ., or E) is encountered or when no more characters remain. The pointer and string length are adjusted as the conversion is performed. The pointer will return pointing to the character which terminated the conversion or beyond the end of the string. The length will be the remaining number of characters. If an overflow occurs, the value >A1 is returned in register >7F and the largest possible floating point value is stored in registers >75 thrugh >7C with the appropriate algebraic sign.

ENTRY CONDITIONS:	Address of first ASCII character in (>58,>59) Null byte in >5A Length of ASCII string in >5B
EXIT CONDITIONS:	Floating point value in >75 through >7C Error status in >7F
CALL ADDRESS:	MOVD %>0018,B CALL @CALFAG
REGISTERS USED:	A, B, >39, >58 through >5B, >5D, >5F, >60, >61, >6E through >7F

CTERP: CARTRIDGE MEMORY PAGING CALL

CTERP places the current expansion ROM page number on the processor stack, selects the new cartridge ROM or RAM page, uses the routine number to calculate the address of the routine, and calls the indicated routine. When the called routine returns control to CTERP, the saved expansion ROM page number is popped from the stack and restored before control is returned to the original calling program.

ENTRY CONDITIONS:	Page number in A Routine number in B
EXIT CONDITIONS:	None
CALL ADDRESS:	CALL @>F857
REGISTERS USED:	A, B

DIVZER: DIVIDE BY ZERO WARNING

DIVZER places the "Division by zero" warning number >A2 in register >7F and places the largest possible floating point number in registers >75 through >7C. The most significant bit of register >5D is used to set the appropriate algebraic sign for the floating point number.

ENTRY CONDITIONS:	Algebraic sign in >5D
EXIT CONDITIONS:	Value >A2 in >7F "Infinity" in >75 through >7C
CALL ADDRESS:	CALL @>F830
REGISTERS USED:	A, B, >75 through >7C, >7F

DSPBUF: DISPLAY CONTENTS OF BUFFER

DSPBUF displays 31 characters starting at the address specified in register pair (>58,>59). The specified address is the high address of the buffer. The characters are in reverse order from the high address to the low address of the buffer. After displaying the characters, the cursor is set to the position indicated in register >53. DSPBUF does not turn the cursor on. The cursor is not seen unless the calling program turns it on.

ENTRY CONDITIONS:	Buffer address in (>58,>59) Final cursor position in >53
EXIT CONDITIONS:	None
CALL ADDRESS:	MOVB @>001E,B CALL @CALPAG
REGISTERS USED:	A, B, >4C, >58, >59

DSPCHR: DISPLAY A SINGLE CHARACTER

DSPCHR selects the display position indicated in register >53, writes the character code specified in register >5C to the selected display position, and moves the cursor to the next position if possible.

ENTRY CONDITIONS:	Cursor position (0 through 30) in >53 Character code in >5C
EXIT CONDITIONS:	Character is displayed in position specified
CALL ADDRESS:	MOVD %>0036,B CALL @CALPAG
REGISTERS USED:	A, >53

DSPINT: INITIALIZE THE DISPLAY HARDWARE

DSPINT initializes the display hardware to the defined default state.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	None
CALL ADDRESS:	MOVD %>0066,B CALL @CALPAG
REGISTERS USED:	A, B

ERROR: ERROR HANDLER

ERROR is a routine which is used within the BASIC operating environment to process breakpoints, warnings, and errors. This routine is accessed by storing a break code (>A5), warning code (negative value) or an error code (positive value) in the A register and executing a TRAP >08 instruction. TRAP >08 transfers control to RAM location >81C. In the BASIC operating envirpnment this location contains a BR(anch) instruction to the ERROR routine. An assembly language program may perform its own error handling by storing an appropriate BR(anch) instruction in this three byte location (>81C through >81E).

The ERROR routine is written specifically to function in the BASIC operating environment. The actions taken by the routine depend upon the type of input value (break, warning, or error) and the current status of the BASIC ON BREAK, ON WARNING, and ON ERROR statements. The following figures show the three types of input values and the corresponding ERROR routine processing.

ERROR: ERROR HANDLER (CONT)

ON BREAK

If the break error number (>A5) is sent to the error handler, the result depends on the status of the ON BREAK basic statement.

The default (condition if no ON BREAK statement is used) ON BREAK statement is ON BREAK STOP, this causes the error handler to display a message and then transfer control to the basic command level.

If ON BREAK NEXT is in effect then the error handler will do nothing but return to the calling program. The only adresses changed by this will be the registers A,B,>4C,>7F. No system ram will be changed.

If ON BREAK ERROR is in effect then the error handler will treat the break command as an error (see below under ON ERROR).

ON WARNING

If the error code is negative (the most significant bit set) the result depends on the status of the ON WARNING statement.

The default (condition if no ON WARNING statement is used) ON WARNING statement is ON WARNING PRINT, this causes a message to be displayed and then the error handler to return to the calling program. The registers and ram described at the end are altered by this path through the error handler.

If ON WARNING NEXT is in effect then the error handler will do nothing but to the calling program. The only adresses changed by this will be the registers A,B,>4C,>7F. No system ram will be changed

If ON BREAK ERROR is in effect then the error handler will process the warning as an error (see below under ON ERROR).

ERROR: ERROR HANDLER (CONT)

ON ERROR

If the error code is positive (the most significant bit reset) the result depends on the status of the ON ERROR statement. If an ON BREAK/ERROR or ON WARNING ERROR is usec then the code is treated as an error (positive error code) and the result is dependent on the ON ERROR statement.

The default (condition if no ON ERROR statement is used) ON ERROR statement is ON ERROR STOP, this causes the error handler to display the error message and transfer control to the basic command level.

IF ON ERROR line-number is in effect then the error handler transfers control to a specified basic error processing subroutine.

Any condition which causes the error handler to transfer control to the basic command level, will alter the contents of most of the registers and many ram locations.

ENTRY CONDITIONS:	BASIC operating environment Break, warning, or error code in register A A BR(anch) instruction to the appropriate error handling routine in RAM locations (>81C through >81E)
EXIT CONDITIONS:	None
CALL ADDRESS:	TRAP 08
REGISTERS USED:	A, B, >4B, >4C, >53, >7F
RAM USED:	>834 through >83B, >83D, >8B0 through >8B3, >925 through >948

Note: When the error handler is waiting for an acknowleddgement of a warning, the user may terminate the program by pressing the BREAK key or OFF key.

When a RETURN statement is used to terminate a basic error-processing subroutine (called by an ON ERROR line-number statement), the statement which caused the error is re-executed. Thus, if an assembly language subprogram reported the error then the subprogram CALL will be re-executed.

EXP: EXPONENTIAL FUNCTION

EXP calculates the exponent (the value of x in e to the x power) value for the floating point value in registers >73 through >7C. The result is returned in registers >73 through >7C. If an overflow error occurs during the evaluation, the code >A1 is returned in register >7F.

ENTRY CONDITIONS:	Floating point value in >73 through >7C
EXIT CONDITIONS:	Exponential value in >73 through >7C
	Error status in >7F
CALL ADDRESS:	MOVD @>0172,8
	CALL @CALPAG
REGISTERS USED:	A, B, >58 through >7F

FADD: FLOATING-POINT ADDITION

FADD adds the floating point value in registers >68 through >72 to the floating point value in registers >73 through >7C. If the addition causes an overflow, the value >A1 is returned in register >7F and the largest possible floating point value with the appropriate algebraic sign is placed in registers >73 through >7C.

ENTRY CONDITIONS:	Floating point value #1 in >73 through >7C
	Floating point value #2 in >68 through >72
EXIT CONDITIONS:	Sum of #1 and #2 in >73 through >7C
	Error status in >7F
CALL ADDRESS:	CALL @>F818
REGISTERS USED:	A, B, >5D through >60, >68 through >7F

FDIV: FLOATING-POINT DIVISION

FDIV divides the floating point value in registers >68 through >72 by the floating point value in registers >73 through >7C and places the result in registers >73 through >7C. If the division results in an overflow, the value >A1 is returned in register >7F. If the divisor is zero, the value >A2 is returned in register >7F. For either error the largest possible floating point number with the appropriate algebraic sign is returned in registers >73 through >7C.

ENTRY CONDITIONS:	Floating point value #1 in >73 through >7C
	Floating point value #2 in >68 through >72
EXIT CONDITIONS:	Value #2 divided by value #1 in >73 through >7C
	Error status in >7F
CALL ADDRESS:	CALL @>F821
REGISTERS USED:	A, B, >5A through >7F

FLTCMP FLOATING-POINT VALUE COMPARISON

FLTCMP compares the floating point value in registers >6B through >72 to the floating point value in registers >75 through >7C and returns the appropriate status. The CALL can be followed by conditional jump instructions (JHE,JL,JEQ,JNE) to transfer control based upon the result of the comparison.

ENTRY CONDITIONS:	Floating point value #1 in >75 through >7C Floating point value #2 in >6B through >72
EXIT CONDITIONS:	C (carry) status if #2 is higher than or equal to #1 N (negative) status if #2 is lower than #1 Z (zero) status if #2 is equal to #1
CALL ADDRESS:	CALL @>F815
REGISTERS USED:	A, B

FMUL FLOATING-POINT MULTIPLICATION

FMUL multiplies the floating point value in registers >6B through >72 times the floating point value in registers >75 through >7C and places the result in registers >75 through >7C. If the multiplication results in an overflow, the value >A1 is returned in register >7F, and the largest possible floating point number with the appropriate algebraic sign is returned in registers >75 through >7C.

ENTRY CONDITIONS:	Floating point value #1 in >75 through >7C Floating point value #2 in >6B through >72
EXIT CONDITIONS:	Product of #1 and #2 in >75 through >7C Error status in >7F
CALL ADDRESS:	CALL @>F81E
REGISTERS USED:	A, B, >5A through >6A, >73 through >7F

FPPOP: FLOATING-POINT POP

FPPOP moves the top entry from the floating point stack to registers >75 through >7C. First, FPPOP checks whether the stack is empty. If the stack pointer (register pair (>56,>57)) is lower than or equal to the base address contained in RAM locations (>8EA,>8EB), FPPOP reports a "Stack underflow" by storing the error code in the A register and TRAPing to the error handler. If the stack pointer is higher than the base address, the top eight bytes of the stack are copied into registers >75 through >7C, and the pointer is adjusted to point to the next lower stack entry.

FPPOP: FLOATING-POINT POP (CONT)

ENTRY CONDITIONS: Address of top of floating point stack in registers (>56,>57)
Base address of floating point stack in (>8EA,>8EB)
EXIT CONDITIONS: Top entry of floating point stack popped into registers >75 through >7C
CALL ADDRESS: TRAP 23
REGISTERS USED: A, B, >56, >57, >75 through >7C

FPPUSH FLOATING-POINT PUSH

FPPUSH copies the contents of register >75 through >7C to the top of the floating point stack. First, FPPUSH checks whether sufficient spaces exists for the eight byte entry. If the stack pointer (register pair (>56,>57)) is within eight bytes of the end of the dynamic area (register pair (>54,>55)) or if the stack pointer is higher than the end of the dynamic area, FPPUSH reports "Memory full" by storing the error code in register A and TRAPing to the error handler. If sufficient space is available between the current top of stack and the end of the dynamic area, the contents of registers >75 through >7C are copied above the current top of stack and the stack pointer is adjusted to point to the new top of stack.

ENTRY CONDITIONS: Address of top of floating point stack in registers (>56,>57)
Address of end of dynamic area in registers (>54,>55)
EXIT CONDITIONS: Contents of registers >75 through >7C copied to top of stack
CALL ADDRESS: TRAP 22
REGISTERS USED: A, B, >56, >57

FRE: FREE MEMORY FUNCTION

FRE returns information about the current use of RAM. The input parameter determines the information to be returned, as follows.

0: Total RAM space not required by the system
1: Total space used by the program in memory
2: Total amount of free space and temporarily allocated space
3: Size of the largest free block in memory
4: Total amount of free memory
5: Number of blocks of free memory

ENTRY CONDITIONS: BASIC operating environment
Input parameter in >73,>74
EXIT CONDITIONS: Output value in >73,>74

FRE:	FREE MEMORY FUNCTION (CONT)
CALL ADDRESS:	MOVD %>0227,B CALL @CALFAG
REGISTERS USED:	A, B, >75 through >7E

FSUB — FLOATING-POINT SUBTRACTION

FSUB subtracts the floating point value in registers >75 through >7C from the floating point value in registers >6B through >72 and leaves the result in registers >75 through >7C. If the subtraction causes an overflow, the value >A1 is returned in register >7F and the largest possible floating point value with the appropriate algebraic sign is placed in registers >75 through >7C.

ENTRY CONDITIONS:	Floating point value #1 in >75 through >7C Floating point value #2 in >6B through >72
EXIT CONDITIONS:	Value #2 - value #1 in >75 through >7C Error status in >7F
CALL ADDRESS:	CALL @>F81B
REGISTERS USED:	A, B, >5D through >60, >6B through >7F

GCONA — LOAD FLOATING-POINT ARGUMENT

GCONA copies eight bytes from the address indicated in register pair (>58,>59) into registers >6B through >72. The eight bytes copied are the byte pointed to and the previous seven bytes. The order is maintained. In other words the first byte is copied to register >72, the next previous byte is copied to register >71, the next to register >70, and so on.

ENTRY CONDITION:	Address of 8-byte value in registers (>58,>59)
EXIT CONDITIONS:	Value copied into registers >6B through >72
CALL ADDRESS:	TRAP 15
REGISTERS USED:	A, B, >58, >59, >6B through >72

GETADR: BUILD SYMBOL INFORMATION ENTRY

GETADR builds a symbol information entry of the following form about the variable token passed in register >4D.

Registers	Information
>75	If numeric entry, byte unused. If string entry, the bit flags: Bit 7: Set to indicate that string is assigned to a variable Bit 6: Set if string value is in dynamic memory area; Reset if string value is within the program image
>76	>00 if numeric entry >AA if string entry
>77,>78	Pointer to value
>79,>7A	Pointer to symbol table entry
>7B,>7C	Pointer to string pointer if string entry

The entry is pushed on the floating point stack and returns control to the calling program.

ENTRY CONDITIONS:	BASIC operating environment Variable token in register >4D Address of next character in program pointer (>4E,>4F)
EXIT CONDITIONS:	Symbol information entry in registers >75 through >7C and on top of the floating point stack
CALL ADDRESS:	MOVD %>0148,8 CALL @CALFAG
REGISTERS USED:	A, B, >4D through >51, >54 through >7F

GETCHR: GET PROGRAM CHARACTER

GETCHR loads the character pointed to by register pair (>4E,>4F) into the A register. The pointer is decremented to point to the next character. Finally, the character is copied to register >4D consequently setting status based on the character.

ENTRY CONDITIONS:	BASIC operating environment Address of character in program pointer (>4E,>4F)
EXIT CONDITIONS:	Program character in register A and in register >4D Address of next program character in program pointer (>4E,>4F) Status based on contents of A register
CALL ADDRESS:	TRAP 18
REGISTERS USED:	A, >4D through >4F

GETNUM PARSE A NUMERIC EXPRESSION

GETNUM parses the numeric expression with a first character in register >4D and a next-character address in register pair (>4E,>4F). The resulting floating point value is returned in registers >75 through >7C. If the expression yields a string result instead of a number, GETNUM pushes the string information entry on the floating point stack, stores the "String-number mismatch" error code >03 in register A, and TRAPs to the error handler.

ENTRY CONDITIONS: BASIC operating environment
First character of expression in register >4D
Address of next character in program pointer (>4E,>4F)

EXIT CONDITIONS: Floating point number in registers >75 through >7C

CALL ADDRESS: MOVD %>0136,B
CALL @CALPAG

REGISTERS USED: A, B, >4D through >51, >54 through >7F

GETSTR: PARSE A STRING EXPRESSION

GETSTR parses the string expression indicated by the contents of register >4D and the address in register pair (>4E,>4F). Information about the resulting string is returned in registers >75 through >78 as follows.

Registers	Information
>75	Bit 7: Set if string value is already assigned to a variable; Reset if string value is temporary (result of expression evaluation) or is constant (resides within program image)
	Bit 6: Set if string resides in dynamic memory area; Reset if string resides in program image
>76	>AA -> String ID byte
>77,>78	Pointer to string value

If the expression yields a numeric result instead of a string, GETSTR pushes the number on the floating point stack, stores the "String-number mismatch" error code >03 in register A, and TRAPs to the error handler.

GETSTR: PARSE A STRING EXPRESSION (CONT)

ENTRY CONDITIONS: BASIC operating environment
First character of expression in register >4D
Address of next character in program pointer (>4E,>4F)

EXIT CONDITIONS: String information entry in registers >75 through >78

CALL ADDRESS: MOVD %>013F,B
CALL @CALPAG

REGISTERS USED: A, B, >4D through >51, >54 through >7F

GRINT: GREATEST INTEGER FUNCTION

GRINT returns the largest floating point integer which is less than or equal to the value supplied in registers >75 through >7C. The result is also returned in registers >75 through >7C.

ENTRY CONDITIONS: Floating point value in >75 through >7C
EXIT CONDITIONS: Result in >75 through >7C
CALL ADDRESS: MOVD %>014B,B
CALL @CALPAG
REGISTERS USED: A, B, >5D, >75 through >7E

IOS: I/O SUBSYSTEM

IOS performs the I/O operation specified in the indicated peripheral access block. When IOS is used, both the PAB and the associated I/O buffer must be in the system RAM or register file. See chapter 2 for additional information about using the I/O subsystem.

ENTRY CONDITIONS: Address of Peripheral Access Block (PAB) in (>75,>7
Null in A register
EXIT CONDITIONS: I/O operation dependent
CALL ADDRESS: CALL >F84B
REGISTERS USED: A, B, >66 through >76
RAM USED: >808 through >80C, >88E through >8A5

KEYIN: WAIT FOR A SINGLE KEY ENTRY

KEYIN waits for the input of a single keystroke and then returns the corresponding keycode in >5C. If KEYIN determines that the same key has been held down for a specified length of time, the keycode is returned again as if it were a new keystroke, generating the auto-repeat feature of the keyboard. When a key which is held down is first recognized by KEYIN, the routine outputs the corresponding character and delays for approximately one second before it outputs it for a second time. Thereafter it delays for only one-twelfth to one-fourteenth second before putting it out subsequent times.

KEYIN also tests the automatic power-down byte at RAM address >830. If the value is non-zero, automatic power down is disabled (as it is when the ALDS is running). If the parameter is zero and no key is pressed for approximately ten minutes, KEYIN returns the "OFF" keycode as if [OFF] has been pressed. If any machine-language program changes this byte to a non-zero value before returning to BASIC, automatic power down remains disabled, resulting in a possible decrease in console battery life.

ENTRY CONDITIONS:	Auto-Power-Down parameter in >830
EXIT CONDITIONS:	Keycode in >5C
CALL ADDRESS:	MOVD %>000C,B CALL @CALPAG
REGISTERS USED:	A, B, >5C through >67
RAM USED:	>834 through >83B

KSTAT: READ KEYBOARD STATUS

KSTAT scans the keyboard to determine the status and returns the requested information in the B register. The B register contains >FF if no key is down, >00 if a new key is down, and >01 if the same key is down (that is the current key was also down on the last call to KSTAT). The code for the key pressed is returned in >5C.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	Keyboard status in B Key code in >5C
CALL ADDRESS:	MOVD %>0039,B CALL @CALPAG
REGISTERS USED:	A, B, >5C through >62
RAM USED:	>834 through >83B

LINEIN: ACCEPT KEYBOARD INPUT AND ECHO IN DISPLAY

LINEIN is used to accept input from the user. This routine first displays the contents of the keyboard input buffer (RAM >83E through >88D). The data must be in reverse order with the first byte of data at the highest address (>88D). The display offset in register >52 is subtracted from the buffer address (>88D) to locate the first character to be displayed. This character and the next 30 lower addressed characters are output to the display. Next, the cursor is displayed in the display relative position indicated in register >53.

After displaying the specified contents of the buffer, LINEIN accepts user input, modifies the contents of the buffer accordingly, and rewrites the display with the new data. The edit and cursor control keys as described in the User's Guide are implemented within this routine.

Register pair (>4B,>4C) contains various flags which affect operation of the LINEIN routine as follows:

Register >48

Bit	Meaning
0	0= any key which is not a displayable character, an edit key, or a cursor control character terminates execution; [FN] and [SHIFT][FN] keys are ignored. 1= only the BREAK, ENTER, and SHIFT-ENTER keys terminate execution of the routine
2	0= normal operation with blinking cursor 1= pause mode; display underline cursor instead of blinking cursor; LEFT, RIGHT, BACK TAB, FORWARD TAB, and HOME keys can be used to move the cursor; CLR, BREAK, ENTER, and SHIFT-ENTER can be used to terminate display; ALL other keys are ignored.

All other flag bits in this byte are used by other parts of the system and should not be modified. The two flags mentioned are also used elsewhere in the system and should be cleared after using them with the LINEIN routine.

LINEIN: ACCEPT KB INPUT AND ECHO IN DISPLAY (CONT)

Register >4C

Bit	Meaning
0	This flag is used within LINEIN.
1	0= normal operation 1= calculator mode; clear the input buffer if the first key pressed is NOT one of the following: *, +, -, /, ^, LEFT, BACK TAB, HOME, PLAYBACK, ENTER, SHIFT-ENTER, INSERT, or assignment of text to a numeric key. If the first key pressed is INSERT, the cursor is homed before beginning insertion. This flag is used by the system when displaying the result of an immediate calculation.
2	This flag is used within LINEIN
3	0= assignment [SHIFT][FN] keys and function [FN] keys terminate execution of LINEIN (But if bit 0 of R>4B is set to ONE, these keys are ignored). 1= assignment keys assign contents of input buffer to indicated numeric key and then clear the input buffer. function keys display appropriate keyword or text assigned to numeric key. Assignment and function keys do not terminate LINEIN.
4	This flag is set upon return from LINEIN if the contents of the display were modified. The calling program should clear the flag before calling LINEIN.
5	0= normal operation 1 = This flag is used in conjunction with bit 0 of register >4B. If both bits are set and the first key pressed is [ENTER] or [SHIFT][ENTER], then the input buffer is cleared before returning to the calling program.

Bits 6 and 7 are used elsewhere in the system and should not be modified.

LINEIN turns on the left arrow when data is scrolled off the left side of the display and the right arrow when data is scrolled off the right side of the display. The SHIFT, FN, and CTL indicators are on when the keyboard is in shift, function, and control mode. respectively. The SHIFT and FN indicators are on simultaneously when the keyboard is in assignment mode. The UCL inditator is on when the keyboard is placed in upper case lock mode. LINEIN turns on the LOW battery indicator if the batteries are low.

LINEIN:	ACCEPT KB INPUT AND ECHO IN DISPLAY (CONT)
ENTRY CONDITIONS:	Offset to start of display in >52 Offset from start of display to cursor in >53 Appropriate flag bits in (>4B,>4C)
EXIT CONDITIONS:	Input data in >83E through >88D Bit 4 of register >4C is set if the input buffer was modified. (NOTE: The contents of the input buffer may still be the same as when the routine was called.) It is the responsibility of the calling program to clear this flag bit prior to the call. Keycode of terminating key in >5C The low battery indicator is turned on if the batteries are low.
CALL ADDRESS:	MOVD %>0006,B CALL @CALPAG
REGISTERS USED:	A, B, >3B, >3C, >4C, >52, >53, >56 through >6A
RAM USED:	>834 through >83C, >83E through >88D, >884 through >8C8, >8E8 through >8EB

LN: NATURAL LOGARITHM

LN calculates the natural logarithm of the floating point value in registers >75 through >7C. The result is returned in registers >75 through >7C. If the input value is less than or equal to zero, the routines stores a BAD ARGUMENT code (>1D) into the A register and traps the the error handler. If any errors occur during the evaluation, the appropriate error code is returned in register >7F.

ENTRY CONDITIONS:	Floating point value in >75 through >7C
EXIT CONDITIONS:	Natural logarithm in >75 through >7C Error status in >7F
CALL ADDRESS:	MOVD %>0175,B CALL @CALPAG
REGISTERS USED:	A, B, >58 through >7F

LOG: COMMON LOGARITHM

LOG calculates the common logarithm of the floating point value in registers >75 through >7C. The result is returned in registers >75 through >7C. If the input value is less than or equal to zero, the routines stores a BAD ARGUMENT code (>1D) into the A register and traps the the error handler. If any errors occur during the evaluation, the appropriate error code is returned in register >7F.

ENTRY CONDITIONS:	Floating point value in >75 through >7C
EXIT CONDITIONS:	Common logarithm in >75 through >7C Error status in >7F
CALL ADDRESS:	MOVD %>0178,B CALL @CALPAG
REGISTERS USED:	A, B, >58 through >7F

MDISP: DYNAMIC MEMORY DEALLOCATION

MDISP deallocates a block of memory which was previously allocated by MNEW. The address of the block (returned by MNEW) must be provided in register pair (>6B,>6C). Calling MDISP with any address other than one specified by MNEW will cause destruction of the dynamic memory area. MDISP returns the specified block to the appropriate position in the linked list of free space. If the deallocated block is adjacent to another free block, the two blocks are combined into one free block.

ENTRY CONDITIONS:	Address of allocation in (>6B,>6C) Free space pointer in (>54,>55) Floating point stack pointer in (>56,>57) Pointer to first block in free space list in (>8E6,>8E7)
EXIT CONDITIONS:	None
CALL ADDRESS:	MOVD %>0100,B CALL @CALPAG
REGISTERS USED:	A, B, >54, >55, >5C through >67, >6B, >6C
RAM USED:	>8E6, >8E7

MNEW: DYNAMIC MEMORY ALLOCATION

MNEW allocates a block of memory from the dynamic memory area. The size of the block is specified in register pair (>6D,>6E). If the requested allocation is zero or greater than 32767, MNEW stores the "System error" code >7E in register A and TRAPs to the error handler. If the requested block cannot be allocated, MNEW stores the "Memory full" code >7F in register A and TRAPs to the error handler. If the block is successfully allocated, the address of the high end of the block is returned in register pair (>6B,>6C).

ENTRY CONDITIONS:	Size of allocation in (>6D,>6E) Free space pointer in (>54,>55) Floating point stack pointer in (>56,>57) Pointer to first block in free space list in (>8E6,>8E7)
EXIT CONDITIONS:	Address of allocation in (>6B,>6C)
CALL ADDRESS:	MOVD %>0103,B CALL @CALPAG
REGISTERS USED:	A, B, >54, >55, >5C through >66, >6B through >6E
RAM USED:	>8E6, >8E7

MONBEG: DEBUG MONITOR

MONBEG is the assembly language DEBUG Monitor. See chapter 8 of the CC-40 Editor/Assembler User's Guide for a discussion of the use of the DEBUG Monitor.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	None
CALL ADDRESS:	MOVD %>0300,B CALL @CALPAG

MOVDWN: MOVE MEMORY BLOCK FROM HIGH ADDRESS TO LOW

MOVDWN copies the contents of a block of memory pointed to by register pair (>63,>66) to a block pointed to by register pair (>67,>68). The address pointers must point to the lowest address in the respective blocks. The number of bytes moved is determined by the contents of register pair (>69,>6A).

ENTRY CONDITIONS:	Source address in (>63,>66) Destination address in (>67,>68) Byte count in (>69,>6A)
EXIT CONDITIONS:	The block is copied to the new address
CALL ADDRESS:	CALL @>F80F
REGISTERS USED:	A, >63 through >6A

MOVUP: MOVE MEMORY BLOCK FROM LOW ADDRESS TO HIGH

MOVUP copies the contents of a block of memory pointed to by register pair (>65,>66) to a block pointed to by register pair (>67,>68). The address pointers must point to the highest address in the respective blocks. The number of bytes moved is determined by the contents of register pair (>69,>6A).

ENTRY CONDITIONS:	Source address in (>65,>66) Destination address in (>67,>68) Byte count in (>69,>6A)
EXIT CONDITIONS:	The block is copied to the new address
CALL ADDRESS:	CALL @>F80C
REGISTERS USED:	A, >65 through >6A

MVINPB: MOVE INPUT BUFFER TO PLAYBACK BUFFER

MVINPB copies the contents of the keyboard input buffer (RAM locations >83E through >88D) to the playback buffer (RAM locations >928 through >977). In addition the RAM locations >926 and >927 are cleared.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	The buffer is copied and the word cleared
CALL ADDRESS:	MOVD %>0024,B CALL @CALPAG
REGISTERS USED:	A, B
RAM USED:	>926 through >977

NEWPRG: INITIALIZE NEW PROGRAM IN MEMORY

NEWPRO deletes the current program in memory by overwriting it with a "null" program. In addition many of the associated pointers are initialized or cleared. The routine sets the cartridge page to zero; if it is called from cartridge, therefore, it must be called from page zero. This routine is used by the BASIC "NEW" command to initialize the program space.

ENTRY CONDITIONS:	Highest RAM address in (>800,>801) Floating point stack base address in (>8EA, >8E5)
EXIT CONDITIONS:	Null program Cartridge page set to zero
CALL ADDRESS:	MOVD %>0030,B CALL @CALPAG
REGISTERS USED:	A, B, >4B, >4C, >50, >51, >54 through >5B, >65 through >6A, >7B through >7E
RAM USED:	>828, >82A, >82E, >832, >833, >8B0, >8B1, >8CB through >8DA, >8DF through >8E2, >8E4, >8E6, >8E7, >8EC, >8ED, >8F2, >8F3, >8FA, >8FB, >8FE

NORMAL: FLOATING POINT NORMALIZATION

NORMAL normalizes a floating point value in registers >75 through >7D so that the representation does not have leading zeros in the mantissa. The floating point value must be positive with the appropriate algebraic sign in the most significant bit of register >5D (0=positive,1=negative). The most significant byte of the mantissa must be >00, or control will be transferred to the ROUND routine. The sign bit in register >5D is used to set the appropriate algebraic sign for the value in >75 through >7C. For each byte that the mantissa is shifted to the left in NORMAL, the exponent is decremented by one.

ENTRY CONDITIONS:	Floating point value in >75 through >7D (with a most-significant byte of >00) Algebraic sign in >5D
EXIT CONDITIONS:	Normalized and sign adjusted value in >75 through >7C Error status in >7F
CALL ADDRESS:	CALL @>F827
REGISTERS USED:	A, B, >5F, >60, >75 through >7D, >7F

OFFCRS: TURN OFF THE CURSOR

OFFCRS turns off the cursor regardless of which one is currently displayed.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	The cursor is turned off.
CALL ADDRESS:	MOVD %>0048,B CALL @CALPAG
REGISTERS USED:	A, B

ONCRS TURN ON THE FLASHING CURSOR

ONCRS turns on the flashing cursor.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	The flashing cursor is turned on.
CALL ADDRESS:	MOVD %>0048,B CALL @CALPAG
REGISTERS USED:	A, B

OPEN: OPEN A FILE OR DEVICE

OPEN is used to open a peripheral file or a device. The open attributes are passed to OPEN in register >3A as follows.

Bit	Meaning
7,6	0,0= Append access mode 0,1= Input access mode 1,0= Output access mode 1,1= Update access mode
5	0= Sequential file structure 1= Relative file structure
4	0= Variable length records 1= Fixed length records
3	0= Display type data 1= Internal type data
2-0	0

The requested I/O buffer length must be provided in register pair (>40,>41). If the requested buffer length is zero, it will be replaced by the default buffer length returned by the peripheral during the open operation. The unique file number (logical unit number or LUNO) to be assigned to the file or device is passed to OPEN in register >44. The device/file specification is passed to OPEN as a BASIC string entry on the floating point stack.

OPEN uses the above information to build a temporary peripheral access block in registers >3A through >46. Then, a temporary buffer is allocated for any device dependent information which follows the device number in the device/file specification. This information is copied into the buffer with all lower-case alphabetic characters translated to upper-case. Next, IOS is called to open the device or file. If the open is successful, OPEN places the buffer length and record number returned by the peripheral in the PAB, releases the temporary buffer, sets the "temporary PAB" flag (bit 3 of register >4B), and returns control to the calling program. If an error occurred during the open operation, OPEN releases the temporary buffer, stores the "I/O error" code >00 in register A, and TRAPs to the error handler.

ENTRY CONDITIONS:	BASIC operating environment Open attributes in >3A Requested I/O buffer length in (>40,>41) LUNO in >44 String entry on floating point stack containing device/file specification
EXIT CONDITIONS:	Temporary PAB flag in >4B
CALL ADDRESS:	MOVD %>022D,B CALL @CALPAG
REGISTERS USED:	A, B, >3A through >43, >45, >46, >4B, >54 through >7C, >7E
RAM USED:	>8E6, >8E7

OVFLOW: OVERFLOW WARNING

OVFLOW places the "Overflow" warning number >A1 in register >7F and places the largest possible floating point number in registers >75 through >7C. The most significant bit of register >5D is used to set the appropriate algebraic sign for the floating point number.

ENTRY CONDITIONS:	Algebraic sign in >5D
EXIT CONDITIONS:	Value >A1 in >7F
	"Infinity" in >75 through >7C
CALL ADDRESS:	CALL @>F833
REGISTERS USED:	A, B, >75 through >7C, >7F

POLYX: SINGLE-VARIABLE POLYNOMIAL EVALUATION

POLYX evaluates a single-variable polynomial defined by a set of coefficients. The coefficients are floating point numbers listed in DATA (or BYTE) statements with the lowest order coefficient first and the highest order coefficient last. The list must be preceded by the one byte value >FF. For example, the following polynomial requires the coefficients listed beneath it.

$$3.59\,X^{3} - 53.798\,X^{2} + 104\,X - 6$$

```
BYTE >FF
BYTE >BF,>06,>00,>00,>00,>00,>00,>00          -6
BYTE >41,>01,>04,>00,>00,>00,>00,>00         104
BYTE >BF,>53,>79,>80,>00,>00,>00,>00      -53.798
BYTE >40,>03,>59,>00,>00,>00,>00,>00        3.59
```

The address passed in registers >58 and >59 is the address of the last byte of the list (in this case the address of the last >00 in the floating point representation of 3.59).

The value of the variable for which the polynomial is to be evaluated must be on top of the floating point stack.

The polynomial is evaluated using a recursive process called nested multiplication. The byte >FF at the top of the list is necessary to stop the recursion. The answer is left in the floating point accumulator (>75 - >7C) and the value of the variable is left on top of the floating point stack.

This routine is used to evaluate the intrinsic functions built into the system, e.g. SIN, COS, LN, EXP, and so on. Further information about this method of function evaluation can be found in the following two references.

Cody, W.J. Jr. and W. Waite [1980]. Software Manual for the Elementary Functions. Prentice Hall, Englewood Cliffs, N.J.

POLYX: SINGLE-VARIABLE POLYNOMIAL EVALUATION (CONT)

Hart, J.F., E.W. Cheney, C.L. Lawson, H.J. Maehly, C.K. Mesztenyi, J.R. Rice, H.C. Thacher, Jr., and C. Witzgall [1968]. Computer Approximations. Wiley, New York.

ENTRY CONDITIONS:	List of polynomial coefficients Address of polynomial coefficients in register pair (>58,>59) Value of variable on top of floating point stack
EXIT CONDITIONS:	Value of polynomial in registers >75 through >7C Value of variable still on top of floating point stack
CALL ADDRESS:	CALL @>F806
REGISTERS USED:	A, B, and >58 through >7F

POPARG: POP VALUE FROM FP STACK INTO SECOND ARGUMENT

POPARG compares the top and base stack addresses. If the base address is higher than or equal to the top address, POPARG stores the "Stack underflow" value >05 in the A register and traps to the error handler. Otherwise, the eight byte value pointed to by register pair (>56,>57) is moved into registers >6B through >72 and the address of the top of the stack is adjusted to point to the next lower stack entry.

ENTRY CONDITIONS:	Address of top of floating point stack in (>56,>57) Base address of floating point stack in (>8EA,>8EB)
EXIT CONDITIONS:	Floating point value in >6B through >72 Address of next lower stack entry in (>56,>57)
CALL ADDRESS:	CALL @>F83C
REGISTERS USED:	A, B, >56, >57, >6B through >72

POWDWN: POWER DOWN

POWDWN closes all open files in the linked list of PABs and devices, initializes some important memory pointers in RAM, resets the I/O bus, calculates and stores an exclusive-or (XOR) checksum of the contents of RAM emory, and turns the computer off. There is no return from a call to POWDWN.

ENTRY CONDITIONS: None
EXIT CONDITIONS: Computer is turned off.
BRANCH ADDRESS:
```
MOVD %>005A,B
BR @BRPAG
```
RAM USED: >804, >805, >808, >809, >828, >82A, >82E, >832, >833, >8A6, >8A7, >8B0, >8B1, >8CB through >8DA, >8E1, >8E2, >8E4, >8E6, >8E7, >8EE, >8F2, >8F3, >8FA, >8FB, >8FE

POWUP: POWER-UP ENTRY ADDRESS

POWUP is the power-up entry point where execution begins when the computer is turned on. When a program transfers control to POWUP, the effect is the same as pressing the RESET button. There is no return from POWUP.

ENTRY CONDITIONS: None
EXIT CONDITIONS: Computer is reset
BRANCH ADDRESS:
```
MOVD %>0003,B
BR @BRPAG
```
REGISTERS USED: All

RETINT: RETURN TO SYSTEM WITH INITIALIZATION

RETINT performs a cold start of the system and thus, initializes the entire operating environment. This return point should be used to terminate any assembly language program which has destroyed the BASIC operating environment.

ENTRY CONDITIONS: None
EXIT CONDITIONS: None
CALL ADDRESS:
```
MOVD %>0009,B
BR   @BRPAG
```
REGISTERS USED: All

RETNEW: SYSTEM RETURN WITH INITIALIZATION OF PROGRAM

RETNEW is used to terminate any assembly language program which has not destroyed the BASIC operating environment, but which has destroyed any BASIC program in memory. The initializes the system as though the BASIC "NEW" command were executed.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	Main program area is initialized
CALL ADDRESS:	MOVD %>0248,B
	BR @BRPAG

RETSYS RETURN TO SYSTEM COMMAND LEVEL

RETSYS is used to terminate any assembly language program which has not destroyed the BASIC operating environment. This entry point resets the cartridge page to zero, resets the register file stack pointer to one, resets the dynamic memory management subsystem, and enters the command level of BASIC.

ENTRY CONDITIONS:	The BASIC operating environment is intact.
EXIT CONDITIONS:	As listed above
CALL ADDRESS:	MOVD %>0042,B
	BR @BRPAG

RND: GENERATE A RANDOM NUMBER BETWEEN 0 AND 1

Each time it is called this routine generates the next value in the current sequence of pseudo-random numbers in registers >75 through >7C. The sequence is determined by the two floating-point values maintained by this routine in RAM locations >8CB through >8DA.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	A random floating point number in >75 through >7C
CALL ADDRESS:	MOVD %>0327,B
	CALL @CALPAG
REGISTERS USED:	A, B, >58 through >7F
RAM USED:	>8CB through >8DA

RNDBYT: GENERATE 8 AND 16 BIT RANDOM INTEGERS

RNDBYT reads the seed value in RAM locations (>8A6,>8A7), uses this value to calculate the next pseudo-random integer in the sequence, and stores that value back in RAM locations (>8A6,>8A7). The most significant byte of the new value is returned as a binary 8-bit random integer in register >75. This value modulo 100 is converted to BCD format and returned in register >76.

ENTRY CONDITIONS:	Seed value in (>8A6,>8A7)
EXIT CONDITIONS:	Binary 8-bit pseudo-random integer in >75 BCD 8-bit pseudo-random integer in >76 Binary 16-bit pseudo-random integer in (>8A6,>8A7)
CALL ADDRESS:	MOVD %>032A,B CALL @CALPAG
REGISTERS USED:	A, B, >75, >76
RAM USED:	>8A6, >8A7

ROUND: FLOATING POINT ROUNDING

ROUND rounds the floating point value in registers >75 through >7C to the byte position indicated in register B. The floating point value must be positive with the appropriate algebraic sign stored in the most significant bit of register >5D (0=positive, 1=negative). The routine adds one to the byte indicated by the contents of the B register. If a carry results, B is decremented and one is added to the next lower byte, and so on. If the rounding results in an overflow, the value >A1 is returned in register >7F and the largest possible floating point value is returned in registers >75 through >7C. In any case, the sign bit in register >5D is used to set the apprcpriate algebraic sign for the value in >75 through >7C.

ENTRY CONDITIONS:	Floating point value in >75 through >7C Rounding position in B Algebraic sign in >5D
EXIT CONDITIONS:	Rounded and sign adjusted value in >75 through >7C Error status in >7F
CALL ADDRESS:	CALL @>F824
REGISTERS USED:	A, B, >75 through >7C, >7F

ROUND2 FLOATING POINT ROUNDING

ROUND2 rounds the floating point value in registers >75 through >7C based on the value in register >7D. If the value in >7D is higher than or equal >50, the floating point value is rounded up. The floating point value must be positive with the appropriate algebraic sign stored in the most significant bit of register >5D (0=positive, 1=negative). If the rounding results in an overflow, the value >A1 is returned in register >7F and the largest possible floating point value is returned in registers >75 through >7C. In any case, the sign bit in register >5D is used to set the appropriate algebraic sign for the value in >75 through >7C.

ENTRY CONDITIONS:	Floating point value in >75 through >7D Algebraic sign in >5D
EXIT CONDITIONS:	Rounded and sign adjusted value in >75 through >7C Error status in >7F
CALL ADDRESS:	CALL @>F851
REGISTERS USED:	A, B, >75 through >7D, >7F

RSTRND: RESET THE RANDOM NUMBER GENERATOR

RSTRND places the default random number seeds in the RAM locations >8CB through >8D2 and >8D3 through >8DA.

ENTRY CONDITIONS:	None
EXIT CONDITIONS:	Random number generator reset
CALL ADDRESS:	MOVD %>0306,B CALL @CALPAG
REGISTERS USED:	A, B, >58 through >5B
RAM USED:	>8CB through >8DA

SETCRS: SET THE CURSOR POSITION

SETCRS selects the cursor position indicated in register >53. This routine does not turn the cursor on. If the indicated position is beyond the last display position, the cursor is set to the last display position.

ENTRY CONDITIONS:	Cursor position in >53
EXIT CONDITIONS:	Cursor at position specified.
CALL ADDRESS:	MOVD %>0051 CALL @CALPAG
REGISTERS USED:	A, B

SIN: SINE FUNCTION

SIN calculates the trigonometric sine of the floating point value in registers >73 through >7C, which is interpreted in accordance with the current angular measure in the indicator status byte. The result is returned in registers >73 through >7C. If the input value is outside the range of angular measurement, this routine stores a BAD ARGUMENT error code in the A ragister and traps to the error handler. If any errors occur during the evaluation, the appropriate error code is returned in register >7F.

ENTRY CONDITIONS: Floating point value in >73 through >7C
EXIT CONDITIONS: Sine in >73 through >7C
Error status in >7F
CALL ADDRESS: MOVD %>015D,8
CALL @CALPAG
REGISTERS USED: A, B, >58 through >7F

SQR: SQUARE ROOT FUNCTION

SQR calculates the square root of the floating point value in registers >73 through >7C. The result is returned in registers >73 through >7C. If any errors occur during the evaluation, the appropriate error code is returned in register >7F.

ENTRY CONDITIONS: Floating point value in >73 through >7C
EXIT CONDITIONS: Square root in >73 through >7C
Error status in >7F
CALL ADDRESS: MOVD %>016F,8
CALL @CALPAG
REGISTERS USED: A, B, >58 through >7F

STGCL2: RELEASE STRING VALUE

STGCL2 checks whether the string entry in registers >68 through >72 is temporary and residing in the dynamic area, and if so, releases the string back to free memory space. If the content of registers >68 through >72 is not a string entry, if the string value is not temporary, or if the string value is not in the dynamic area, STGCL2 simply returns to the calling program.

ENTRY CONDITIONS: BASIC operating environment
String entry in registers >68 through >72
EXIT CONDITIONS: None
CALL ADDRESS: MOVD %>012D,8
CALL @CALPAG
REGISTERS USED: A, B, >54, >55, >5C through >67, >6B, >6C
RAM USED: >8E6, >967

STGCLR: RELEASE STRING VALUE

STGCLR pops the top entry from the floating point stack into registers >6B through >72, checks whether the string is temporary and residing in the dynamic area, and if so, releases the string back to free memory space. If the stack entry is not a string entry, if the string value is not temporary, or if the string value is not in the dynamic area, STGCLR simply returns to the calling program.

ENTRY CONDITIONS:	BASIC operating environment String entry on floating point stack
EXIT CONDITIONS:	None
CALL ADDRESS:	MOVD %>0124,B CALL @CALPAG
REGISTERS USED:	A, B, >54, >55, >5C through >67, >6B through >72
RAM USED:	>8E6, >8E7

STGDSP: DYNAMIC MEMORY DEALLOCATION

STGDSP deallocates a block of memory which was previously allocated by STGNEW. The address of the block (returned by STGNEW) must be provided in register pair (>6B,>6C). Calling STGDSP with any address other than one specified by STGNEW will cause destruction of the dynamic memory area. STGDSP returns the specified block to the appropriate position in the linked list of free space. If the deallocated block is adjacent to another free block, the two blocks are combined into one free block.

ENTRY CONDITIONS:	Address of allocation in (>6B,>6C) Free space pointer in (>54.>55) Floating point stack pointer in (>56,>57) Pointer to first block in free space list in (>8E6,>8E7)
EXIT CONDITIONS:	None
CALL ADDRESS:	MOVD %>0121,B CALL @CALPAG
REGISTERS USED:	A, B, >54, >55, >5C through >67, >6B, >6C
RAM USED:	>8E6, >8E7

STGNEW: DYNAMIC MEMORY ALLOCATION

STGNEW allocates of block of memory from the dynamic memory area. The size of the block is specified in register >6E. If the requested allocation is zero, STGNEW stores the "System error" code >7E in register A and TRAPs to the error handler. If the requested block cannot be allocated, STGNEW stores the "Memory full" code >7F in register A and TRAPs to the error handler. If the block is successfully allocated, the address of the length byte at the high end of the block is returned in register pair (>6B,>6C). This length byte must remain intact for the purpose of deallocation. The first available byte is at the address beneath the length byte (the total number of bytes allocated is one greater than the number requested in order to account for the length byte). This memory allocation routine is most often used to allocate memory for the values of string variables. However, it can be used for any allocation for which the length is less than or equal to 255.

ENTRY CONDITIONS:	Size of allocation in >6E Free space pointer in (>54,>55) Floating point stack pointer in (>56,>57) Pointer to first block in free space list in (>8E6,>8E7)
EXIT CONDITIONS:	Address of allocation in (>6B,>6C)
CALL ADDRESS:	MOVD %>011E,B CALL @CALPAG
REGISTERS USED:	A, B, >54, >55, >5C through >66, >6B through >6E
RAM USED:	>6E6, >8E7

STRCMP: ASCII STRING COMPARISON

STRCMP compares the string pointed to by register pair (>6D,>6E) to the string pointed to by register pair (>6B,>6C) and returns the appropriate status. The first byte pointed to by each pointer is the length of each string. Then, the string representations are in reverse order moving from high address to low address. The CALL can be followed by conditional jump instructions (JHS, JL, JEQ, JNE) to transfer control based upon the result of the comparison.

ENTRY CONDITIONS:	Address of string #1 in (>6B,>6C) Address of string #2 in (>6D,>6E).
EXIT CONDITIONS:	C (carry) status if string #2 higher or equal string N (negative) status if string #2 lower than string Z (zero) status if string #2 equal to string #1
CALL ADDRESS:	CALL @>F542
REGISTERS USED:	A, B, >6B through >6E

SWAPFA: SWAP FP ACCUMULATOR AND SECOND ARGUMENT

SWAPFA exchanges the contents of registers >68 through >72 with the contents of registers >75 through >7C.

ENTRY CONDITIONS: None
EXIT CONDITIONS: Contents of blocks are swapped.
CALL ADDRESS: CALL @>FB2D
REGISTERS USED: A, B, >68 through >72, >75 through >7C

SYM: SEARCH BASIC SYMBOL TABLE

SYM searches the current BASIC program symbol table. The variable token is specified in register >4D. When the value provided in >4D is less than >20 or greater than >7E, SYM stores an ILLEGAL SYNTAX error code (>01) in the A register and traps to the error handler. If the symbol is not found in the symbol table, the imperative symbol table is searched.

If the symbol is not found, a Z (zero) status condition is returned. If the symbol is found, a NZ (non-zero) status condition is returned along with the address of the symbol table entry in register pair (>79,>7A).

ENTRY CONDITIONS: BASIC operating environment
Previous call to GETCHR
Variable token in >4D
EXIT CONDITIONS: Z status if symbol not found
NZ status if symbol found
Address of symbol table entry in >79, >7A
CALL ADDRESS: CALL @>F812
REGISTERS USED: A, B, >75 through >7C

TAN: TANGENT FUNCTION

TAN calculates the trigonometric tangent of the floating point value in registers >75 through >7C, which is interpreted in accordance with the indicator status byte. The result is returned in registers >75 through >7C. If the value provided to TAN is outside the valid range of angular measurement, the routine stores a BAD ARGUMENT error code (>1D) in the A register and traps to the error handler. If any errors occur during the evaluation, the appropriate error code is returned in register >7F.

ENTRY CONDITIONS: Floating point value in >75 through >7C
EXIT CONDITIONS: Tangent in >75 through >7C
Error status in >7F
CALL ADDRESS: MOVD %>0163,B
CALL @CALPAG
REGISTERS USED: A, B,—>58 through >7F

UNCRS: TURN ON UNDERLINE CURSOR

UNCRS turns on the non-blinking underline cursor.

ENTRY CONDITIONS: None
EXIT CONDITIONS: The underline cursor is displayed.
CALL ADDRESS: MOVD %>002A,B
CALL @CALPAG
REGISTERS USED: A, B

VERSION: GET INTERPRETER VERSION NUMBER

VERSION returns a value which identifies the version of BASIC in ROM. The BASIC interpreter in CC-40 ROM returns the value 10.

ENTRY CONDITIONS: None
EXIT CONDITIONS: BASIC interpreter version number in B register
CALL ADDRESS: MOVD %>007E,B
CALL @CALPAG
REGISTERS USED: A, B

WRTIND: WRITE STATUS OF INDICATORS TO DISPLAY

WRTIND writes the current status of the indicators as defined by the contents of RAM addresses >83A through >83D to the display. ONE bits turn indicators on; ZERO bits turn them off. The siginificance of the bits within the four bytes is as follows.

Bit	>83A	>83B	>83C	>83D
MSB 7	unused	unused	unused	unused
6	unused	unused	unused	unused
5	unused	unused	unused	unused
4	RAD	<-	user 4	ERROR
3	GRAD	SHIFT	user 5	user 1
2	I/O	CTRL	user 6	user 2
1	UCL	FN	[LOW]	user 3
LSB 0	->	DEG	unused	unused

ENTRY CONDITIONS: Status of indicators in >83A through >83D
EXIT CONDITIONS: Indicators on or off
CALL ADDRESS: MOVD %>0000,B
CALL @CALPAG
REGISTERS USED: A, B

CHAPTER 5
ASSEMBLY LANGUAGE PROGRAMMING EXAMPLES

This chapter contains extended examples of programs written for the CC-40 in assembly language. The examples demonstrate various techniques for using system routines in ROM and programming in the BASIC environment.

Two subprograms, DIR and COPY demonstrate communication on the HEX-BUS-tm. To illustrate parameter passing in detail, a subprogram called NUMHEX converts a numeric value passed to it by BASIC into a string to be returned to BASIC. A subprogram called PRMYN demonstrates keyboard and display management, as well as data-format conversion.

This chapter also contains a group of five subroutines which illustrate the handling of peripheral access blocks used in input/output operations. These subroutines may be used as a "library" routines to simplify and speed up programming.

DIR: A WAFERTAPE DIRECTORY SUBPROGRAM

A subprogram called DIR lists the directory of a WAFERTAPE cartridge to the display of the CC-40. DIR is a subprogram designed for level 1 memory usage: it verifies that that a program is not running.

The subprogram illustrates illustrates the following assembly language programming techniques.

GENERAL PROGRAMMING TECHNIQUES

* How to use equates for registers and other constants in a proc
* How to pass a numeric parameter to a subroutine.
* How to set up a subroutine header.
* How to find out if a subprogram is called from a running program.
* How to store text messages in a program.

HEX-BUS-tm COMMUNICATION TECHNIQUES

* How to create and use a PAB in the statement temporary area.
* How to issue the HEX-BUS catalog command.
* How to dynamically allocate a buffer needed for the catalog command.
* How to close all open files and reset the dynamic area.

DATA FORMAT CONVERSION

* How to convert a floating point number to an integer.
* How to convert a floating point number to a string.

ERROR CHECKING AND REPORTING

* How to check the syntax of a passed parameter.
* How to use the ERROR firmware routine to display errors and return to the basic environment.

KEYBOARD-DISPLAY I/O

* How to display messages on the LCD display.
* How to wait till a key is pressed and check for the BREAK key.

The following syntax is used to call DIR from BASIC.

CALL DIR(N)

where N is the device number of the WAFERTAPE drive containing files for the directory

```
 TITL 'Tape Directory Utility'
 LIST  60              Do 60 lines/page in listing
 IDT  'DIR'
 OPTION X
*********************************
*                               *
* W a f e r t a p e  D i r e c t o r y  U t i l i t y *
*                               *
*                           tm  *
* This progras reads the directory of a WAFERTAPE  wafer.*
* It is a level 1 assembly language subroutine.         *
* To load and call it, use the following procedure.     *
*                                                       *
*    CALL LOAD("DK.DIR")                                *
*    CALL DIR(DRIVE) where DRIVE is the drive number to *
*                    get the directory of.              *
*                                                       *
*********************************

*
* General system equates
KBINPT EQU  >088D             * Keyboard input buffer
PAUSMD EQU  >04               * Pause mode set for linein
RUNMOD EQU  >20               * The basic running flag (1 if program running)
KSBRK  EQU  >E7               * The key code for the break key
ESBARG EQU  >1D               * Bad argument error code
ESILP  EQU  >1C               * Illegal in program error code
ESIO   EQU  >00               * I/O error code
ESSYN  EQU  >01               * Illegal syntax error code
TSRPAR EQU  >AF               * Token for right parentheses
TSLPAR EQU  >C0               * Token for left parentheses

* BASIC ROM Routine equates
GETNUM EQU  >0136             * GETNUM
IOS    EQU  >F84B             * IOS
CALPAG EQU  >F836             * CALPAG
CIS    EQU  >012A             * CIS
STRNEW EQU  >011E             * STRNEW
STRDSP EQU  >0121             * STRDSP
CLRINP EQU  >0021             * CLRINP
LINEIN EQU  >0006             * LINEIN
TRSHDY EQU  >0072             * TRSHDY
CFI    EQU  >0115             * CFI
DSPBUF EQU  >001E             * DSPBUF
UNCRS  EQU  >002A             * UNCRS
KEYIN  EQU  >000C             * KEYIN
ERROR  EQU  6                 * ERROR trap number
GETCHR EQU  18                * GETCHR trap number
```

```
* Register equates
TEMP0  EQU  >3A          * Statement temporary equate
TEMPF  EQU  >49          * Statement temporary equate
FLAGS  EQU  >4B          * System flag register
CURCHR EQU  >4D          * Current program character (byte)
DSPST  EQU  >52          * Display offset from start of buffer (byte)
CRSPOS EQU  >53          * Cursor position (byte)
IOBUFP EQU  >59          * Pointer to I/O BUFFER (word)
FPAREA EQU  >5C          * Beginning of the floating point area
ARG    EQU  >68          * Second floating point accumulator
FAC    EQU  >75          * Floating point accumulator
FPSTAT EQU  >7F          * Floating point status (byte)
DYNADR EQU  ARG+1        * Dynamic memory allocation pointer (word)
DYNLEN EQU  ARG+3        * Dynamic memory allocation length  (word)
CATBUF EQU  TEMPF        * Catalog buffer

* PAB statement temporary register equates
BUFADR EQU  TEMP0+2      * Address of the data buffer
STATUS EQU  TEMP0+3      * Returned I/O status
DATLEN EQU  TEMP0+5      * Data length of PAB
BUFLEN EQU  TEMP0+7      * Buffer length
RECORD EQU  TEMP0+9      * Record number/File number
LUNO   EQU  TEMP0+10     * Logical unit number
COMAND EQU  TEMP0+11     * Hex-bus command code
DEVICE EQU  TEMP0+12     * Device number


*****************************************************************
*                  THE PROGRAM ENTRY POINT
*
START
  BTJO @RUNMOD,FLAGS,RUNERR * If running program, then ILLEGAL IN PROGRAM
  MOVB @TRSHDY,8            * Close all open files, empty dynamic area
  CALL @CALPAG              * CALL TRSHDY
  CMP  @TKLPAR,CURCHR       * Is current character a '('
  JNE  SYNTAX               * If not then SYNTAX ERROR
  TRAP GETCHR               * Get the next program character
  MOVB @GETNUM,8            * Parse the parameters for a number
  CALL @CALPAG              * CALL GETNUM  It will issue BAD ARGUMENT if a
*                           * number is not found
  MOVB @CFI,8               * Convert the floating point number to an integer
  CALL @CALPAG              * CALL CFI
  BTJO @>FF,FPSTAT,BARG     * Was there a bad conversion? IF so BAD ARGUMENT
  BTJO @>FF,FAC,BARG        * Was the device number greater than >255?
  BTJO @>FF,FAC+1,OK        * Was the device number zero?  yes, then BAD
*                           * ARGUMENT ERROR
BARG
  MOV  @KE$BARG,A           * Move BAD ARGUMENT error in to A
  TRAP ERROR                * TRAP to ERROR  for BAD ARGUMENT
```

```
OK
  CMP  ZT$RPAR,CURCHR      * IS current character ")"
  JNE  SYNTAX              * If not then SYNTAX ERROR
  TRAP GETCHR              * Get next program character
  JNZ  SYNTAX              * Is it a EOL character(>00) no, then ILLEGAL SYNTAX
  MOV  FAC+1,DEVICE        * Put device number in the PAB for the catalog
  MOVD Z>0000,RECORD       * Put zero in the record number in PAB
  MOV  Z>1E,DYNLEN         * Establish the length of the buffer for catalog
  MOVD ZSTGNEW,B           * Dynamically allocate that space as a string
  CALL @CALPAB             * CALL STGNEW  don't return if MEMORY FULL error
  MOVD DYNADR,CATBUF       * CATBUF points to the length byte of allocation
  DECD CATBUF              * CATBUF points to the first byte of allocation
  CLR  LUNO                * Use logical unit number of 0 in PAB so device
*                          * is printed by I/O error
  MOV  Z>0E,COMAND         * Move catalog command in to PAB
TOP
  MOVD Z>0018,BUFLEN       * Establish buffer length of 24 in PAB
  MOVD Z>0000,DATLEN       * Move zero in to data length of the PAB
  MOVD CATBUF,BUFADR       * Move CATBUF in to buffer pointer in PAB
  MOVD IDEVICE,FAC+1       * Move the address of PAB in to FAC and FAC+1
  CALL @IOS                * CALL IOS to do the catalog command
  BTJO Z>FF,STATUS,IOERR   * Was there a non-zero status, Issue I/O error
  INC  RECORD              * Increment LSB of file number in PAB
  ADC  Z>00,RECORD-1       * Add any carry to the MSB of file number in PAB
  MOVD ZCLRINP,B           * Clear the display buffer
  CALL @CALPAB             * CALL CLRINP
  LDA  *BUFADR             * Get the file number from the buffer
  CMP  Z10.A               * Is the File number > 9?
  JL   NOTTEN              * If file number is lower than 10 only print 1 digit
  PUSH A                   * Save the file number
  MOV  Z'1',A              * Use  '1' as digit
  STA  @KBINPT             * Store it in the display buffer
  POP  A                   * Restore the file number
  SUB  Z10,A               * Subtract the 10's digit
NOTTEN
  OR   Z>30,A              * Create ASCII numeric character
  STA  @KBINPT-1           * Store it as second digit of display buffer
ENDFN
  MOV  Z':',A              * Move the : separator in to the A register
  STA  @KBINPT-2           * Store it in the next buffer location
  MOV  Z' ',A              * Move a blank in to the A register
  STA  @KBINPT-3           * Store it in the next buffer location
  MOV  Z12,B               * Move the 12 character filename length in to B  8
FNAMLP
  DECD BUFADR              * Skip the file number byte
  LDA  *BUFADR             * Get the first character of the filename
  STA  @KBINPT-16(B)       * Store it in the proper buffer location        12
  DJNZ B,FNAMLP            * Decrement the file length Jump while non-zero
  MOV  Z7,B                * Put length of MAXLEN message in to B          2
MLNLP1
  LDA  @MAXLEN-1(B)        * Get a character of the message
  STA  @KBINPT-25(B)       * Store it in to the display buffer             15
  DJNZ B,MLNLP1            * Decrement the message length jump while non-zero
```

```
*
  CALL @GET16B          * Get a 16 bit maximum record length
  MOVB @KBINPT-26,@FAC+1  * Move address of the Display buffer in to FAC+1
  CALL @OUTBUF          * Move the number in to the display buffer
  MOV  @7,@3            * Move the length of record number message
RCSLP1
  LDA  @MAXREC-1(@3)    * Get character of "# RECS:"
  STA  @KBINPT-40(@3)   * Store it in the display buffer
  DJNZ @,RCSLP1         * Decrement message length, Jump while non-zero
  CALL @GET16B          * Get a 16 bit number of records
  MOVB @KBINPT-41,@FAC+1  * Move address of the display buffer in to FAC+1
  CALL @OUTBUF          * Move number of records string to display buffer
  DECD @BUFADR          * Point at next byte of input buffer
  MOV  @'*',@3          * Default to inactive file character
  LDA  *@BUFADR         * Get flag information from input buffer
  PUSH @                * Save the byte
  JPZ  NOACTI           * If negative flag then file is active
  MOV  @'A',@3          * Set active file character
NOACTI
  XCHB @                * Move flag to B and file character to A
  STA  @KBINPT-50       * Store it in the display buffer
  MOV  @':',@           * move an ":" in the A register
  STA  @KBINPT-51       * Store it in the display buffer
  MOV  @'D',@           * Default to DISPLAY type file
  BTJZ @>10,@,DISPT     * If bit is zero then it is display type file
  MOV  @'I',@           * Move "I" to a to indicate INTERNAL type file
DISPT
  STA  @KBINPT-52       * Store the file type character in display buffer
  CLR  @CRSPOS          * Set cursor position to zero
  CLR  @DSPST           * Set display start offset to zero
  OR   @PAUSMD,@FLAGS   * Set Pause mode flag
  MOVB @LINEIN,@3       * Display it and wait for any pressed character
  CALL @CALPAG          * CALL LINEIN
  AND  @>FF-PAUSMD,@FLAGS  * Clear pause mode flag
  POP  @                * Get back flag information
  CMP  @KSBRK,@FPAREA   * Was the break key pressed
  JNE  CONTIN           * If not then continue with the catalog
DONE
  INC  @CATBUF          * Increment CATBUF to point at length byte of
  ADC  @>00,@CATBUF-1   * the string so it can be deallocated
  MOVB @CATBUF,@DYNADR  * Restore the dynamically allocated area
  MOVB @STRDSP,@3       * deallocate the string
  CALL @CALPAG          * CALL STRDSP
  RETS                  * Return basic system
CONTIN
  BTJO @>40,@A,@DONE    * If this is last file then leave
  BR   @TOP             * repeat this for next file
```

```
;*****************************************************************
;                        GET16B
; This routine takes a 16 bit number in the buffer
; and converts it to a string with the length byte in fac-1

GET16B
  DECW BUFADR             ; Point to the LSB of the number
  LDA  @BUFADR            ; Get the LSB of the number
  MOV  A,FAC+1            ; Store the LSB of the number in FAC+1
  DECW BUFADR             ; Point to the MSB of the number
  LDA  @BUFADR            ; Get the MSB of te number
  MOV  A,FAC              ; Store the MSB of the number in FAC
CIEZ
  MOVD #CIS,B             ; Convert the number in FAC,FAC+1 to a string
  CALL @CALPAG            ; CALL CIS
  RETS                    ; Return to calling instruction


;*****************************************************************
;                        OUTBUF
; Takes a string with a length byte in FAC-1 and move it to
; the display buffer.
OUTBUF
  MOVD #FAC-2,FAC+3       ; Set pointer to first ASCII character
OUTLOP
  LDA  @FAC+3             ; Get a character of the ASCII number
  STA  @FAC+1             ; Store it in to the display buffer
  DECW FAC+3              ; Point at the next character of the string
  DECW FAC+1              ; Point at next display buffer location
  DJNZ FAC-1,OUTLOP       ; Decrement string length byte, jump if non-zero
  RETS                    ; Return to calling instruction

;*****************************************************************
;                     TAPE IS EMPTY MESSAGE
BADONE
  MOVD #EMPTY,IOBUFP      ; Display tape empty message
  MOV  #>00,CRSPOS        ; Set cursor position to 0
  MOVD #DSPBUF,B          ; Display the message
  CALL @CALPAG            ; CALL DSPBUF
  MOVD #UNCRS,B           ; Turn on the underline cursor
  CALL @CALPAG            ; CALL UNCRS
  MOVD #KEYIN,B           ; Wait for any key
  CALL @CALPAG            ; CALL KEYIN
  RETS                    ; Return to basic system


;*****************************************************************
;                     ILLEGAL IN PROGRAM ERROR
RUNERR
  MOV  #ERRIIP,A          ; Move ILLEGAL IN PROGRAM error code to A
  TRAP ERROR              ; TRAP TO ERROR for ILLEGAL IN PROGRAM
```

```
*****************************************************************
*                       ILLEGAL SYNTAX ERROR
SYNTAX
  MOV  @E$SYN,A           * Move ILLEGAL SYNTAX error code in to A
  TRAP ERROR              * TRAP to ERROR for ILLEGAL SYNTAX


*****************************************************************
*                       I/O ERROR
IOERR
  CALL @DONE              * Deallocate dynamic memory space used
  CMP  @>03,STATUS        * Was the File not found?
  JEQ  GODONE             * IF yes then Tape was empty
  MOV  @E$IO,A            * No then move I/O error code in to A
  TRAP ERROR              * TRAP to ERROR for I/O error


*****************************************************************
*                       MESSAGE STRINGS
*
MAXLEN
  TEXT ':NELXAM'          * 'MAXLEN' in reverse
MAXREC
  TEXT ':SCER #'          * '# RECS:' in reverse
*
  TEXT '                   ytpmE epaT'     * 'Tape Empty' in reverse
EMPTY  EQU  $-1


*****************************************************************
*                       END OF THE PROGRAM HEADER

  TEXT 'DIR'              * Subroutine name 'DIR' in reverse capitals
  BYTE 3                  * Length byte
NAME   EQU  $-1
  DATA START              * Pointer to subroutine entry point
  DATA >0000              * Offset to next header: zero since no other header
  DATA NAME-$+1           * Offset to subroutine name
  BYTE >00                * Reserved byte
  BYTE >A4                * Flag information; relocatable assembly subroutine
HEADER EQU  $-1
*                       BEGINNING OF THE PROGRAM HEADER
*****************************************************************
  END
```

COPY: A FILE COPY SUBPROGRAM

The following subprogram copys one file to another. The subprogram illustrates several techniques not used in the DIR subprogram.

GENERAL PROGRAMMING TECHNIQUES

* How to pass a string parameter.

HEX-BUS COMMUNICATION TECHNIQUES (FILES)

* How to open a file using the OPEN firmware routine.
* How to put a PAB in the dynamic memory area into the linked list of PABs.
* How to copy a PAB in the linked list into the statement temporary area for an I/O operation.
* How to indicate that a PAB is no longer in the statement temporary area.
* How to use a data buffer in read and write operations.
* How to close all files in the linked list of PABs.

The syntax used to call COPY from basic is as follows.

```
CALL COPY("WW.XXXXXXX","YY.ZZZZZZ")
```

Where WW.XXXXXXX comprises the device number and name of the source file and YY.ZZZZZZ comprises the device number and name of the destination file.

COPY is a level 1 subprogram which makes sure a program is not running before it starts copying files.

```
  TITL 'File copy utility'
  IDT 'COPY'
  OPTION XREF
*********************************************
*       F i l e   c o p y   u t i l i t y   *
*                                           *
*  This progrsm copies a file from one mass storage device  *
*  to another device.                       *
*                                           *
*  To load and call this subprogrss, use the following      *
*  procedure.                               *
*                                           *
*  CALL LOAD("DN.FILENAME")                 *
*  CALL COPY("DN.FILE1","DN.FILE2") where file1 is the file  *
*  to be copied to file2.                   *
*********************************************

*
* General systss equates
PABTMP EQU  >08       * BASIC PAB in statsent temp flag (R)4B)
RUNMOD EQU  >20       * BASIC progrss running flag 1 if run

* Register equates
TEMP0  EQU  >3A       * Beginning of statsent tsmporary area
TEMPF  EQU  >49       * 2nd to last byts in stat temp area
FLAGS  EQU  >4B       * Systss flag register
CURCHR EQU  >4D       * Current progrss character (byts)
IOBUFP EQU  >59       * Pointer to i/o buffer
FPAREA EQU  >5C       * Beginning of the floating point area
ARG    EQU  >6B       * Second floating point accumulator
FAC    EQU  >75       * Floating point accumulator
FPSTAT EQU  >7F       * Floating point status register
DYNADR EQU  ARG+1     * Dynamic sesory allocation pointer
DYNLEN EQU  ARG+3     * Dynamic sesory allocation length
TEMP   EQU  FAC+9     * Temporary register pair
IPAB   EQU  TEMP0+14  * Pointer to the input PAB
OPAB   EQU  TEMPF+1   * Pointer to the output PAB
PABADR EQU  TEMPF+1   * Temporary use as PAB address
IPAB2  EQU  TEMP0+1   * Pointer to index into input pab
OPAB2  EQU  TEMP0+3   * Pointer to index into output pab
IOLEN  EQU  ARG+1     * Temporary data length
IOPTR  EQU  ARG+3     * Temporary pointer to buffer

* Error code equates
E%SYN  EQU  >01       * Error code for SYNTAX ERROR
E%IIP  EQU  >13       * Error code for ILLEGAL IN PROGRAM
E%IO   EQU  >00       * Error code for I/O ERROR
```

```
* BASIC ROM routine equates:
CLSALL EQU  >021E
GETSTR EQU  >013F
MNEW   EQU  >0103
OPEN   EQU  >0220
TRSHDY EQU  >0072
CALPAG EQU  >FB36
IDG    EQU  >F848
ERROR  EQU  8          * ERROR trap number
GETCHR EQU  18         * GETCHR trap number

* PAB statement temporary register equates
ATTR   EQU  TEMP0              * Device attributes
BUFADR EQU  TEMP0+2            * Address of the data buffer
STATUS EQU  TEMP0+3            * Returned I/O status
DATLEN EQU  TEMP0+5            * Data length of PAB
BUFLEN EQU  TEMP0+7            * Buffer length
RECORD EQU  TEMP0+9            * Record number/File number
LUNO   EQU  TEMP0+10           * Logical unit number
COMAND EQU  TEMP0+11           * Hex-bus command code
DEVICE EQU  TEMP0+12           * Device number
CURBUF EQU  TEMP0+14           * Current pointer in the buffer

* Token equates
T$COMA EQU  >AD          * Token for comma
T$LPAR EQU  >C0          * Token for left parentheses
T$RPAR EQU  >AF          * Token for right parentheses

* I/O command codes
OPENF  EQU  >00          * Command code for open
CLOSE  EQU  >01          * Command code for close
READ   EQU  >03          * Command code for read
WRITE  EQU  >04          * Command code for write

* RAM equates
FSTPAB EQU  >8E2

*****************************************************************
*                       ENTRY POINT
COPY
  BTJZ XRUNMOD,FLAG$,OK  * If running program, then ILLEGAL IN
*                        * PROGRAM error

*****************************************************************
*                       ILLEGAL IN PROGRAM error
  MOV  IE$IIP,A          * ILLEGAL IN PROGRAM error code in A
  TRAP ERROR             * TRAP ERROR for ILLEGAL IN PROGRAM
```

```
OK
  MOVB %TRSHDY,B        * Close all open files, empty dynasic
  CALL @CALPAG          * CALL TRSHDY                 area
  CMP  %T%LPAR,CURCHR   * Is current character a "("
  JNE  SYNTAX           * If not then  syntax error TRAP
  TRAP GETCHR           * Get the next program character
  MOVB %GETSTR,B        * Get file-name string from parameters
  CALL @CALPAG          * CALL GETSTR, it will issue the error
*                       * BAD ARGUMENT if string is not found
  MOV  %>01,LUNO        * Set luno to 1
  MOV  %>40,ATTR        * Attributes input,display,sequential
  MOVB %>0000,BUFLEN    * Set buffer length to zero to
*                       * determine the needed buffer size
  MOVB %OPEN,B          * Open the file with the file name
  CALL @CALPAG          * CALL OPEN

  CALL @STOPAB          * Store the PAB in statement temporary
*                       * area dynamically and create buffer.
*                       * also allocate the i/o buffer and
*                       * put it in the input PAB
  MOVB PABADR,IPAB      * Save the pointer to input PAB
  SUB  %>04,IPAB        * Point at the command byte of the
  SBB  %>00,IPAB-1      * PAB instead of the ptr to next one
  CMP  %T%COMA,CURCHR   * Is it a comma?
  JNE  SYNTAX           * If not then jump to SYNTAX ERROR

* open the second file for output
  TRAP GETCHR           * Get the next program character
  MOVB %GETSTR,B        * Get file-name string from parameters
  CALL @CALPAG          * CALL GETSTR, it will issue the error
*                       * BAD ARGUMENT if string is not found
  CMP  %T%RPAR,CURCHR   * Is it a left parenthesis
  JEQ  CONT             * If it is a left parenthesis jump

*****************************************************************
*         Syntax error code
SYNTAX
  MOV  %E%SYN,A
  TRAP ERROR

* open the second file for output
CONT
  TRAP GETCHR           * Check for the EOS token
  MOV  CURCHR,A         * If not End Of Statement token >00
  JNZ  SYNTAX           * Issue syntax error
  MOV  %>02,LUNO        * Set luno to two
  MOV  %>80,ATTR        * Attributes output,display,sequential
*                       * Buflen is still in existance from
*                       * the previous OPEN of input
  MOVB %OPEN,B          * Open the file with the file name
  CALL @CALPAG          * CALL OPEN
  CALL @STOPAB          * Store the PAB in statement temporary
```

```
* Output file is now open and the pab is dynamically allocated
* so the statement temporary area is now available
  SUB  %>04,OPAB        * Since PABADR = OPAB adjust it to
  SBB  %>00,OPAB-1      * Point at device number

* Set up for the read of the first record
RDREC
  MOVD IPAB,FAC+1       * Point to the input PAB
  CALL @LOADIT          * Bring the PAB in to temporary area
  MOV  %READ,COMAND     * Put a read command in input pab

* Read a record from the input file
  MOVD %>0000,DATLEN    * Clear data length in input PAB
  MOVD %DEVICE,FAC+1    * Point at temporary PAB for the read
  CALL @IOS             * Do the read operation
  MOV  STATUS,A         * Check the status
  JNZ  DONE             * If I/O error then it is done
  INC  RECORD           * Update the record number for random
  ADC  %>00,RECORD-1    *  access devices
  MOVD DATLEN,TDLEN     * Save datlen in temporary space
  MOVD BUFADR,TBPTR     * Save buflen in temporary space
  MOVD IPAB,FAC+1       * Point to input PAB
  CALL @SAVEIT          * Store the PAB back to linked PAB
  MOVD OPAB,FAC+1       * Point to the output PAB
  CALL @LOADIT          * Load the output temporary PAB
  MOVD TDLEN,DATLEN     * Restore the data length
  MOVD TBPTR,BUFADR     * Restore the buffer pointer
  MOV  %WRITE,COMAND    * Write out the record
  MOVD %DEVICE,FAC+1    * Point at the PAB for IOS
  CALL @IOS             * Do the write
  INC  RECORD           * Update the record number for
  ADC  %>00,RECORD-1    *  random access devices
  MOV  STATUS,A         * Check the status on the write
  JNZ  IOERR            * Jump if I/O error has occured
  MOVD %>0000,BUFADR    * Clear pointer to Buffer FIRST
  MOVD OPAB,FAC+1       * Point at the PAB To save it
  CALL @SAVEIT          * Save updated PAB
  JMP  RDREC            * Get the next record
IOERR
  MOV  %>10,A           * Issue  i/o error
  TRAP ERROR            * Issue I/O error
DONE
  CMP  %>07,STATUS      * Any message other than EOF error
  JNE  IOERR            * means copy fails issue I/O error.
  MOVB %CLSALL,B        * Close all open files and deallocate
  CALL @CALPAB          * the buffers and PAB's
  RETS
```

```
*****************************************************************
*                       LOADIT
* (Copy the linked PAB into the statement temporary for easier access.)

LOADIT
  MOV  @>0D,B           * Copy 12 of the PAB entries
LOOP1
  LDA  *FAC+1           * Get a byte from the linked PAB
  STA  @DEVICE->0D(B)   * Store it as a temporary PAB
  DECD FAC+1            * Point at the next byte
  DJNZ B,LOOP1          * Repeat for 12 bytes
  RETS                  * Return from the subroutine

*****************************************************************
*                       SAVEIT
* (Copy a temporary PAB into the linked list of PAB'S so another one can be
* used.)

SAVEIT
  MOV  @>0D,B           * Copy 12 of the PAB entries
LOOP
  LDA  @DEVICE->0D(B)   * Get a byte of the temporary PAB
  STA  *FAC+1           * Store it in the linked PAB
  DECD FAC+1            * point at next space in linked PAB
  DJNZ B,LOOP           * Repeat for 12 bytes
  RETS                  * Return from the subroutine


*****************************************************************
*                       STOPAB
* (Store temporary PAB into dynamic memory area, set up a buffer for it,
* and link it into the PAB list.)

STOPAB
*               ALLOCATE SPACE FOR PAB
  MOVB @17,@DYNLEN      * Request 17 bytes
  MOVB @MNEM,B          * Request 17 bytes for the PAB
  CALL @CALPAB          * CALL MNEM
*   If MNEM couldn't allocate the ram, it won't return here
  MOVB @DYNADR,@PABADR  * Save address for PAB chaining

  MOVB @>0000,@BUFADR   * Clear out buffer pointer
  MOV  @13,B            * Store the 13 bytes of PAB
  MOVB @PABADR,@TEMP    * Temp is first byte of allocated area
  DECD TEMP             * Skip the top 2 bytes of the linked
  DECD TEMP             * PAB list for the pointer to next PAB
  CLR  A                * Clear pointer to current buffer
  STA  *TEMP            * Write LSB out as zero
  DECD TEMP             * Point at MSB
  STA  *TEMP            * Write MSB out as zero
  DECD TEMP             * Point at beginning of real PAB
```

```
STPAB1
  LDA  @TEMP0-1(B)       * Get a byte of the PAB to store
  STA  *TEMP             * Store a byte in the linked PAB
  DECD TEMP              * Point at next place to store a byte
  DJNZ B,STPAB1          * B = B-1 jump if it is not zero
*       Now the PAB is out in dynamic memory
  CMP  @>01,LUNO         * Is it input file?
  JNE  NEXT              * If not then don't allocate buffer
  MOVB BUFLEN,DYNLEN     * Allocate buffer for input/output
  MOVB @>NEW,B           * Request buflen bytes for the PAB
  CALL @VCALPAB          * CALL MNEW
*       Will not return if the request is too large
  MOVB PABADR,TEMP       * Put the buffer address in the PAB
  SUB  @>0E,TEMP         * Point at the buffer address in the
  SBB  @>00,TEMP-1       * linked PAB
  MOV  DYNADR,A          * Put LSB of buffer address in A
  STA  *TEMP             * Store LSB in linked PAB
  DECD TEMP              * Point to linked PAB's MSB
  MOV  DYNADR-1,A        * Put MSB of buffer address in A
  STA  *TEMP             * Store MSB in linked PAB
NEXT
*          DO PAB CHAINING
  MOVB PABADR,FAC+1      * FAC+1 points to top of linked PAB
  LDA  @FSTPAB           * Get system pointer to first linked
*                        * PAB'S LSB
  STA  *FAC+1            * Put pointer LSB into linked PAB
  DECD FAC+1             * Point at linked PAB's second byte
  LDA  @FSTPAB-1         * Get system pointer to first linked
*                        * PAB'S MSB
  STA  *FAC+1            * Put pointer MSB into linked PAB
  MOV  PABADR,A          * Put LSB of new PAB's address in A
  STA  @FSTPAB           * Store it in system pointer to
*                        * Linked PAB's
  MOV  PABADR-1,A        * Put MSB of new PAB's address in A
  STA  @FSTPAB-1         * Store it in system pointer to
*                        * linked PAB's
  ANDB @>FF-PABTMP,FLAGS * Turn off PAB in stat temp flag
*         The PAB is now moved and linked in to the system
  RETS

********************************************************************************
*                    END OF THE PROGRAM HEADER
  TEXT 'YPOC'              * Subroutine name 'COPY' in reverse capitals
  BYTE 4                   * Length byte
NAME  EQU  $-1
  DATA COPY                * Pointer to subroutine entry point
  DATA >0000               * Offset to next header; zero since no other header
  DATA NAME-$+1            * Offset to subroutine name
  BYTE >00                 * Reserved byte
  BYTE >A4                 * Flag information; relocatable assembly subroutine
HEADER EQU  $-1
*                  BEGINNING OF THE PROGRAM HEADER
********************************************************************************
```

NUMHEX:NUMBER TO HEXIDECIMAL STRING SUBPROGRAM

A subprogram called NUMHEX converts a numeric argument to a ASCII hexadecimal string. The subprogram illustrates the following techniques not used in the DIR and COPY subprograms.

* How to get the address of a variable.
* How to check a variable to see if it is a string variable.
* How to assign a value to a string variable.
* How to dynamically allocate a string variable.
* How to get a numeric parameter.
* How to convert a floating point number to a positive integer in the range 0 to 65535.
* How to convert a number to a ASCII hexadecimal string.

The syntax used to call NUMHEX is as follows.

```
CALL NUMHEX(NUMBER,STRING$)
```

Where NUMBER is a variable or expression in the range 0 to 65535, and STRING$ is a string variable to which the hexadecimal string is to be assigned. The character string generated will be from one to four characters in length.

NUMHEX is designed for level 0 memory usage; it is callable from a BASIC program.

```
  TITL 'Number to HEX String'
  LIST  60              So 60 lines/page in listing
  IDT  'NUMHEX'
  OPTION X
*********************************
*                               *
* Number To HEX String Utility *
*                               *
*                          to   *
* This subprogram takes a number as input and creates a *
* HEXADECIMAL string. It converts numbers from 0 to     *
* 65535.  To call it use the following procedure:       *
*                                                       *
*   CALL LOAD("DK.NUMHEX")                              *
*   CALL NUMHEX(NUMBER,STRING$) where NUMBER is a       *
*     variable or expression in the range 0 to 65535, and*
*     STRING$ is a string variable to which the         *
*     HEXADECIMAL string is to be assigned. The         *
*     character string generated will be from one       *
*     to four characters in length.                     *
*********************************************************

*
* General system equates
ESBARG EQU  >10             * Bad argument error code
ESILIP EQU  >13             * Illegal in program error code
ESSYN  EQU  >01             * Illegal syntax error code
TSCOMA EQU  >AB             * Token for comma
TSRPAR EQU  >AF             * Token for right parentheses
TSLPAR EQU  >C0             * Token for left parentheses

* Register equates
TEMP0  EQU  >3A             * Statement temporary equate
TEMPF  EQU  >49             * Statement temporary equate
CURCHR EQU  >4D             * Current program character (byte)
ARG    EQU  >63             * Second floating point accumulator
FAC    EQU  >75             * Floating point accumulator
FPSTAT EQU  >7F             * Floating point status (byte)
TMPBUF EQU FAC+4            * Temporary character buffer
DYNADR EQU  ARG+1           * Dynamic memory allocation pointer (word)
DYNLEN EQU  ARG+3           * Dynamic memory allocation length  (word)
NUMBER EQU  TEMP0+1         * Number to be converted
MEMAD  EQU  TEMP0+3         * Address of dynamic allocated
```

```
* System ROM equates
GETNUM EQU  >0136          * GETNUM
CALPAG EQU  >F836          * CALPAG
ASSIGN EQU  >0145          * ASSIGN
CFILNG EQU  >017E          * CFI
GETADR EQU  >0148          * GETADR
STGNEW EQU  >011E          * STGNEW
ERROR  EQU  8              * ERROR trap number
GETCHR EQU  18             * GETCHR trap number

NUMHEX
  CMP  @T$LPAR,@CURCHR * Is current character a "("
  JNE  SERR            * If not then SYNTAX ERROR
  TRAP GETCHR          * Get the next program character
  MOVB @GETNUM,B       * Parse the parameters for a number
  CALL @CALPAG         * CALL GETNUM  It will issue BAD ARGUMENT
*                      * if a number is not found
  MOVB @CFILNG,B       * Convert the floating point number to
  CALL @CALPAG         * CALL CFILNG         an integer
  MOVB FAC+1,NUMBER    * Save away the number to be converted
  CMP  @T$COMA,@CURCHR * Is there a comma separator?
  JNE  SERR            * If not then syntax error
  TRAP GETCHR          * Get the next program character
  MOVB @GETADR,B       * Get information about the string
  CALL @CALPAG         * CALL GETADR           variable
  CMP  @>AA,FAC+1      * Check the variable type
  JEQ  OK              * Is it a string variable?
*                      * No then bad argument error
BARG
  MOV  @E$BARG,A       * Move BAD ARGUMENT error code to A
  TRAP ERROR           * Issue BAD ARGUMENT error
OK
  CMP  @T$RPAR,@CURCHR * IS current character ")"
  JNE  SERR            * If not then SYNTAX ERROR
  TRAP GETCHR          * Get next program character
  JZ   HEXV            * Is it a EOL character(>00) no, then
*                      *                          ILLEGAL SYNTAX
SERR
  MOV  @E$SYN,A        * ILLEGAL SYNTAX error
  TRAP ERROR           * Issue ILLEGAL SYNTAX error
HEXV
  MOVB NUMBER,FAC+3    * Restore the number to be converted
  CLR  B               * Haven't found any digits yet
  MOV  FAC+2,A         * Put the MSB in to A register
  SWAP A               * Convert Most Significant Nibble
  JZ   HEXV08          * Are 1st two digits are zero?
  CALL @CNVCHR         * If it is >0 then make a digit
HEXV04
  MOV  FAC+2,A         * Put the MSB back in the A register
  CALL @CNVCHR         * Convert Least Significant Nibble
HEXV08
  MOV  FAC+3,A         * Put the LSB in to A register
  SWAP A               * Convert Most Significant Nibble
  CALL @CNVCHR         * If it is >0 then make a digit
```

```
HEXV12
  MOV  FAC+3,A        * Convert Least Significant Nibble
  AND  >0F,A          * Get rid of MSN so we can force "0"
  CALL @CNV04         * Convert LSN of number

  PUSH B              * Save length of the string
  MOV  B,@DYNLEN      * Establish dynamic strings length
  MOVB @STRNEW,B      * Allocate dynamic string to hold it
  CALL @CALPAG        * CALL STRNEW
  MOVB @DYNADR,@NEWAD * Store string pointer
  POP  FAC            * Retrieve string length
  CLR  B              * Clear counter of characters
DONE06
  DECB @NEWAD         * Point at next character position
  LDA  @TMPBUF(B)     * Get character from temporary buffer
  STA  *NEWAD         * Store character in string
  INC  B              * Continue till entire string stored
  CMP  B,FAC          * Check if last character done
  JNE  DONE06         * If not repeat

*     BUILD UP A STRING VALUE INFORMATION ENTRY
  MOVB @>40AA,FAC+1   * Notation for Temporary String
  MOVB @DYNADR,FAC+3  * Establish Pointer to the string
  MOVB @>8B3B,B       * Assign the hex string to variable
  CALL @CALPAG        * CALL ASSIGN
  RETS                * RETURN to calling code

*****************************************************************
*                        CNVCHR
* Convert a nibble in A to a ASCII hexadecimal digit and store it if it
* is <> 0 or if a previous digit has been found.  Increment the B register
* if a digit is found.

CNVCHR
  AND  @>0F,A         * Clear out the MSN of the A register
  JNZ  CNV04          * Is digit >0? If so convert it
  TSTB                * Check for previous numbers
  JZ   CNVOUT         * If none then don't convert the zero

*****************************************************************
*                        CNV04
* Convert the least significant nibble in A to an ASCII hexadecimal digit
* and store it.  Increment the B register.
CNV04
  CMP  @>A,A          * Was digit > 9?
  JHE  MAKCHR         * Yes then convert it to alphabetic
  OR   @>30,A         * Ascii 0 to 9 = >30 to >39
  JMP  SAVCHR         * Go and save the character
MAKCHR
  ADD  @'A'->0A,A     * ASCII A TO F = >41 to >46
SAVCHR
  STA  @TMPBUF(B)     * Save the character temporarily
  INC  B              * Update number of characters
CNVOUT
  RETS                * Return to calling code
```

```
*****************************************************************
*                  END OF THE PROGRAM HEADER

  TEXT 'XEHMUN'       * Subroutine name in reverse capitals
  BYTE 6              * Length byte
NAME  EQU  $-1
  DATA NUMHEX         * Pointer to subroutine entry point
  DATA >0000          * Offset to next header zero no other header
  DATA NAME-6+1       * Offset to subroutine name
  BYTE >00            * Reserved byte
  BYTE >A4            * Flag byte relocatable  subroutine
HEADER EQU  $-1
*                  BEGINNING OF THE PROGRAM HEADER
*****************************************************************
  END
```

PRMYN: PROMPT FOR Y(ES) OR N(O)

A subprogram called PRMYN displays the first 29 characters of a string variable and then waits for a Y or an N to be pressed. If any other character is pressed it beeps and waits for another key. The PRMYN subprogram demonstrates the following techniques.

DISPLAY HANDLING TECHNIQUES

* How to clear the display
* How to display a string character by character
* How to turn the cursor on and off
* How to set the cursor position

KEYBOARD INPUT TECHNIQUES

* How to input a key from the keyboard
* How to program for the BREAK key

NUMERIC VARIABLE HANDLING

* How to check if a variable is numeric
* How to assign a numeric value
* How to convert from integer to floating point format

The syntax used to PRMYN is as follows.

CALL PRMYN(STRING$,NUMBER)

Where STRING$ is a string variable to which the hexadecimal string is to be assigned and NUMBER is a variable in which the status is returned 1 for Y and 0 for N.

The PRMYN subprogram is designed for level 0 memory usage; it is callable from a BASIC program.

```
 TITL 'Yes or No prompt'
 LIST 60              So 60 lines/page in listing
 IBT  'PRMYN'
 OPTION I
++++++++++++++++++++++++++++++++++++++
+                                     +
+ Yes or No Prompt                    +
+                                     +
+                                     +
+                                     +
+ This subprogram displays the first 29 characters of a +
+ string on the LCD and waits for a Y or N response     +
+ it then returns 1 for Y and N for 0 in a numeric      +
+ variable.                           +
+                                     +
+ The subprogram is loaded and called as follows.       +
+                                     +
+   CALL LOAD("DN.PRMYN")             +
+   CALL PRMYN(A$,STATUS) where A$ is the string to be  +
+   displayed and STATUS is a numeric variable which is +
+   returned either 0 or 1 depending on the Key pressed +
+                                     +
++++++++++++++++++++++++++++++++++++++
```

```
*
* Register equates
TEMP0  EQU  >3A      * Statement temporary equate
TEMPF  EQU  >49      * Statement temporary equate
FLAGS  EQU  >48      * System flag register
CURCHR EQU  >40      * Current program character byte
DSPST  EQU  >52      * Display offset byte
CRSPOS EQU  >53      * Cursor position byte
IOBUFP EQU  >59      * Pointer to I/O BUFFER (word)
FPAREA EQU  >5C      * Beginning of floating point area
ARG    EQU  >6B      * Second floating point accumulator
FAC    EQU  >75      * Floating point accumulator
STRPTR EQU  FAC+3    * Pointer to temporary string entry
PTR    EQU  TEMP0+1  * Temporary pointer
TLEN   EQU  TEMP0+2  * Temporary length of string byte
CURSOR EQU  TEMP0+3  * Temporary cursor position
STATUS EQU  TEMP0+5  * Integer to return in numeric variable
CHAR   EQU  FPAREA   * Character to put in to display

* System ROM equates
GETSTR EQU  >013F    * GETSTR
GETADR EQU  >0148    * GETADR
BRKPT  EQU  >0130    * BRKPT
CALPAG EQU  >F8B6    * CALPAG
BEEP   EQU  >005D    * BEEP
CLRDSP EQU  >0045    * CLRDSP
DSPCHR EQU  >0036    * DSPCHR
SETCRS EQU  >0051    * SETCRS
ASSIGN EQU  >0145    * ASSIGN
CIF    EQU  >0127    * CIF
OFFCRS EQU  >004B    * OFFCRS
ONCRS  EQU  >004B    * ONCRS
KEYIN  EQU  >000C    * KEYIN
ERROR  EQU  8        * ERROR trap number
GETCHR EQU  18       * GETCHR trap number

* General system equates
E$BARG EQU  >1D          * Bad argument error code
E$SYN  EQU  >01          * Illegal syntax error code
K$BRK  EQU  >E7          * Key code for BREAK key
T$COMA EQU  >AD          * Token for comma
T$RPAR EQU  >AF          * Token for right parentheses
T$LPAR EQU  >C0          * Token for left parentheses

*********************************************
PRNYN
  CMP  @T$LPAR,@CURCHR * Is current character a "("
  JEQ  OK
  BR   @SERR           * If not then SYNTAX ERROR
OK
  TRAP GETCHR          * Get the next program character
  MOVB @GETSTR,8       * Parse the parameters for a string
  CALL @CALPAG         * CALL GETSTR  It will issue BAD ARGUMENT
*                      *              if a string not found
```

```
  MOVD %CLRDSP,B      * Clear the display
  CALL @CALPAG        * CALL CLRDSP
  MOVD STRPTR,PTR     * Get pointer to the string
  CLR  CURSOR         * Start at the beginning of the display
  CLR  STATUS-1       * Clear MSB of floating point number
  LDA  *PTR           * Get the length byte
  JZ   NULSTR
  CMP  %29,A          * Make sure that it is only 29 characters
  JL   OK1
  MOV  %29,A
OK1
  MOV  A,TLEN         * Save the length
  DECD PTR            * Point at first byte of the string
LOOP
  MOV  CURSOR,CRSPOS  * Use the first free character position
  LDA  *PTR           * Get the character to be displayed
  MOV  A,CHAR         * Put character to be displayed in CHAR
  DECD PTR            * Point at next byte of string
  INC  CURSOR         * Increment to next cursor position
  MOVD %DSPCHR,B      * Display character at position CURSOR
  CALL @CALPAG        * CALL CALPAG
  DJNZ TLEN,LOOP      * Repeat if characters remain in string
NULSTR
  MOV  CURSOR,CRSPOS  * Establish the cursor position
  MOVD %SETCRS,B      * Set the cursor position for call
  CALL @CALPAG        * CALL SETCRS
  MOVD %ONCRS,B       * Turn on the cursor
  CALL @CALPAG        * CALL ONCRS
  MOVD %KEYIN,B       * Return the first key pressed in CHAR
  CALL @CALPAG        * CALL KEYIN
  MOVD %OFFCRS,B      * Turn off the cursor
  CALL @CALPAG        * CALL OFFCRS
  CMP  %KSBRK,CHAR    * Was it the break key?
  JNE  DSPLY          * Continue
  MOVD %BRKPT,B       * Check ON BREAK status and
  CALL @CALPAG        * CALL BRKPT
  JMP  NULSTR         * Try again if ON BREAK NEXT
DSPLY
  MOV  CURSOR,CRSPOS  * Restore the cursor positon
  MOVD %DSPCHR,B      * Display the pressed character
  CALL @CALPAG        * CALL DSPCHR
  OR   %>20,CHAR      * Convert it to lower case
  CMP  %'y',CHAR      * Is it a yes response?
  JEQ  YES            * Jump if YES
  CMP  %'n',CHAR      * Is it a no response?
  JEQ  NO             * Jump if NO
* Invalid key beep then try again
  MOVD %BEEP,B        * Beep the beeper
  CALL @CALPAG        * CALL BEEP
  MOV  CURSOR,CRSPOS  * Restore the cursor positon
  MOV  %' ',CHAR      * Use a blank as the display character
  MOVD %DSPCHR,B      * Display a blank over input character
  CALL @CALPAG        * CALL DSPCHR
  JMP  NULSTR         * Try again
```

```
YES
  MOV  >01,STATUS     * Indicate YES
  JMP  CONT           * Continue to assign the variable
NO
  CLR  STATUS         * Indicate NO
CONT
  CMP  >T$COMA,CURCHR * Is there a comma separator?
  JEQ  OK2            * If not then syntax error
SERR
  MOV  >E$SYN,A       * Issue SYNTAX ERROR
  TRAP ERROR          * CALL ERROR
OK2
  TRAP GETCHR         * Get the next program character
  MOVB >GETADR,8      * Build symbol info to of the argument
  CALL @CALPAG        * CALL GETADR
  MOV  FAC+1,A        * Check if numberic variable is 00
  JEQ  OK3            * Jump if it is a numeric variable
  MOV  >E$BARG,A      * Issue BAD ARGUMENT error
  TRAP ERROR          * Go to the error handler
OK3
  MOVD STATUS,FAC+1   * Convert status to floating point
  MOVD >CIF,8         * Convert integer to floating point
  CALL @CALPAG        * CALL CIF
  MOVD >ASSIGN,8      * Assign the value to the number
  CALL @CALPAG        * CALL CALPAG
  MOVD >CLRDSP,8      * Clear the display upon exit
  CALL @CALPAG        * CALL CLRDSP
  RETS                * Return to calling code


*****************************************************************
*                    END OF THE PROGRAM HEADER

  TEXT 'MYMRP'          * Subroutine name in reverse CAPS
  BYTE 5                * Length byte
NAME  EQU  $-1
  DATA PRMYN            * Pointer to entry point
  DATA >0000            * No other header so zero offset
  DATA NAME-$+1         * Offset to subroutine name
  BYTE >00              * Reserved byte
  BYTE >44              * Flag information
HEADER EQU  $-1
*                    BEGINNING OF THE PROGRAM HEADER
*****************************************************************
```

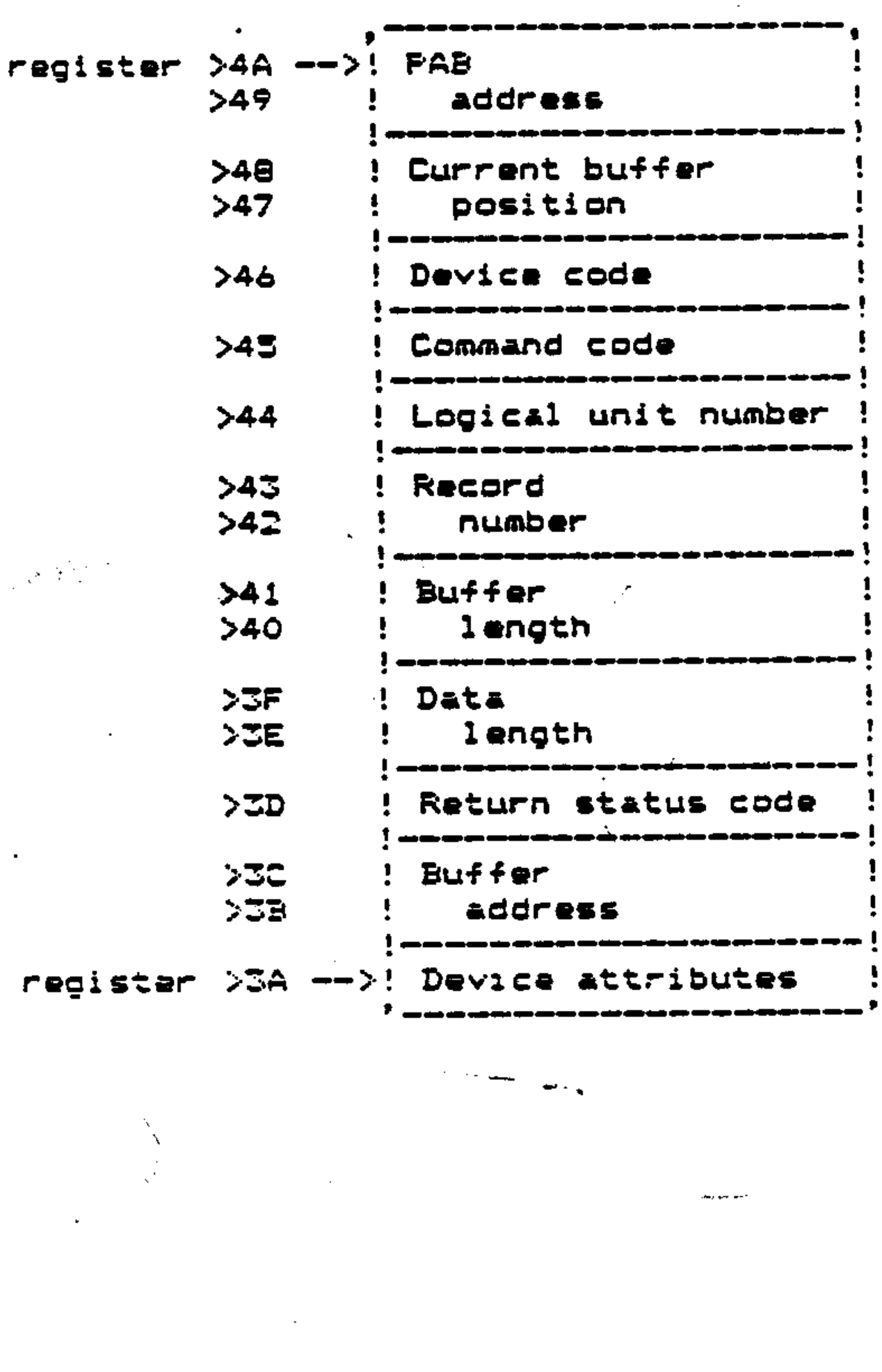
SUBROUTINES FOR PERIPHERAL ACCESS

The five subroutines which follow show how to program for use of peripheral access blocks (PABs) within the BASIC operating

environment. They show how a PAB can be added or deleted from the linked list of PABs that is maintained in the dynamic memory area. They also show how to find a specific PAB, copy it into the statement level temporary area, or copy it from the statement level temporary area into the dynamic memory area.

ADDING A PAB INTO THE LINKED LIST

The following subroutine can be used to add a PAB to the linked list in the dynamic memory area. The PAB must currently reside in the statement level temporary area as shown in the following figure.



The buffer length and record number must reflect the information returned by the open operation, but the I/O buffer has not been allocated. Thus, the PAB address, the current buffer position, and the buffer address are not yet initialized. The subroutine verifies that the I/O buffer length is non-zero, allocates the I/O buffer, initializes the buffer pointer, allocates space for the PAB, links the PAB space into the list, and calls another subroutine to copy the PAB into the allocated space.

```
* Equates
PABTMP EQU >08          "Temporary PAB" flag bit in register >48
FLAGS  EQU >48          System flag register
*
BUFADR EQU >3C          Buffer address
BUFLEN EQU >41          Buffer length
CURBUF EQU >48          Current buffer position
PABADR EQU >4A          PAB address
*
DYNADR EQU >6C          Address of allocated memory block
DYNLEN EQU >6E          Size of requested memory allocation
*
FSTPAB EQU >6EF         Pointer to first PAB in linked list
*
MMEM   EQU >0103        Dynamic memory allocation routine
CALPAG EQU >F33A        System paged call routine
*
LNKPAB
   MOVB BUFLEN,DYNLEN   Prepare to allocate the I/O buffer
   JNZ  LNKP10          Jump if valid buffer length
   MOV  BUFLEN,A        Get here if MSB=0, now check LSB
   JZ   FERR            Report file error if buffer length = 0
LNKP10
   MOVB @MMEM,8         Allocate the I/O buffer
   CALL @CALPAG
   MOVB DYNADR,BUFADR   Initialize the buffer address in the PAB
   MOVB DYNADR,CURBUF   Initialize the current buffer position
   MOVB @17,DYNLEN      Prepare to allocate space for the PAB
   MOVB @MMEM,8         Allocate space for the PAB
   CALL @CALPAG
   MOVB DYNADR,PABADR   Save the PAB address
```

```
* Now link the PAB into the front of the linked list
  LDA  @FSTPAB          Fetch LSB of pointer to first PAB in list
  STA  *DYNADR          Store in PAB as link pointer
  DECD DYNADR           Point to next PAB location
  LDA  @FSTPAB-1        Fetch MSB of pointer to first PAB in list
  STA  *DYNADR          Store in PAB as link pointer
  MOV  PABADR,A         Fetch LSB of pointer to new PAB
  STA  @FSTPAB          Store as pointer to first PAB in list
  MOV  PABADR-1,A       Fetch MSB of pointer to new PAB
  STA  @FSTPAB-1        Store as pointer to first PAB in list
  CALL @STOPAB          Call subroutine to store the PAB (see below)
  AND  >FF-PABTMF,FLAGS Turn off the "temporary PAB" flag
  RETS                  Return to calling program
*
FERR
  MOV  @>8FILE,A        Prepare to report a file error
  TRAP ERROR            Report the error
```

STORING A PAB INTO DYNAMIC MEMORY

The following subroutine copies a PAB from the statement level temporary area to its designated position in the dynamic memory area for access by the linked list of PABs. The pointer to the address in the area is in the PABADR register pair. A temporary pointer called PTR is also used.

```
* Equates
TMPPAB EQU >3A          Low end of PAB in statement temporary area
PABADR EQU >4A          PAB address
*
STOPAB
  MOVB PABADR,PTR       Copy PAB address
  DECD PTR              Skip the link pointer
  DECD PTR
  MOV  >15,B            Copy the remaining 15 bytes of the PAB
STPAB1 EQU $
  LDA  @TMPPAB-1(B)     Fetch a byte from the temporary area
  STA  *PTR             Store the byte in the PAB
  DECD PTR              Point to the next position in the PAB
  DJNZ B,STPAB1         Decrement counter/offset, loop until done
  RETS                  Return to calling program
```

FINDING A PAB IN THE LINKED LIST

The following subroutine searches the linked list of PABs for the one containing a specific logical unit number (LUNO). The target LUNO is passed to the routine in a register called LUNO. The routine requires two other register pairs, one called PABADR that is used to point to each PAB in turn, and another called PRVPAB that is used to save the previous PAB address.

The routine returns three pieces of information. The status register reflects whether or not the LUNO has been found. If the CARRY bit is set, the LUNO was found. If it is reset, the LUNO was not found. If the LUNO was found, the address of the PAB is in PABADR and the address of the previous PAB is in PRVPAB. The address of the previous PAB is not required but can be quite useful when linking a PAB into the list or removing a PAB from the list. If the LUNO was not found, PABADR and PRVPAB both contain the address of the last PAB in the list.

FSTPAB is a pointer in the system reserved area of the system RAM that indicates the first PAB in the linked list. If the most significant byte of this pointer is zero, the list is empty.

Since the paging routines do not preserve the status register, this routine cannot be called across page boundaries unless some means other than the status register is used to indicate whether or not the LUNO was found.

```
* Equates
PABADR EQU >4A          PAB address
*
FSTPAB EQU >8EF    First PAB pointer
*
FNDPAB
  MOVB @FSTPAB,@PABADR    Set up pointer to the first PAB pointer
FNDP10 EQU $
  MOVB @PABADR,@PRVPAB    Save address in previous PAB pointer
  LDA  *PABADR            Fetch the least significant byte
  MOV  A,B                Save it in the B register
  DECT @PABADR            Point to the most significant byte
  LDA  *PABADR            Fetch the MSB
  JZ   FNDP20             Jump if at end of linked list
  MOVB B,@PABADR          Copy the pointer to the PAB
  SUB  @>6,B              Adjust pointer to the LUNO location
  SBB  @0,A
  LDA  *B                 Fetch the LUNO from the PAB
  CMP  @LUNO,A            Compare to the target LUNO
  JNE  FNDP10             Jump if this is not the correct LUNO
  SETC                    Indicate that the LUNO was found
FNDP20
  RETS                    Return to the calling program
```

LOADING A PAB INTO THE TEMPORARY AREA

The following subroutine can be used to load a PAB from the linked list into the statement level temporary area. The PAB must have been located and the location pointed to by the contents of the PABADR register pair. A temporary pointer PTR is also used.

```
* Equates
TMPPAB EQU >3A          Low end of PAB in statement temporary area
PABADR EQU >4A          PAB address
*
```

```
LDPAB
  MOVB PABADR,PTR       Copy the PAB address
  DECD PTR              Skip over the link pointer
  DECD PTR
  MOV  %15,B            Copy the remaining 15 bytes of information
LDPAB1
  LDA  *PTR             Fetch a byte
  STA  @TMPPAB-1(B)     Store it in temporary area
  DECD PTR              Point to next byte of PAB
  DJNZ B,LDPAB1         Decrement counter/offset, loop until done
  RETS                  Return to calling program
```

DELETING A PAB

The following subroutine can be used to delete a PAB from the linked list and release the allocated blocks of dynamic memory. This code assumes that the PAB has been copied to the statement level temporary area and that a register pair called PRVPAB points to the previous PAB in the list (as returned by the FNDPAB subroutine).

```
* Equates
BUFADR EQU >3C          Buffer address
PABADR EQU >4A          PAB address
*
DYNADR EQU >6C          Address of allocated memory block
*
MDISP  EQU >0100        Dynamic memory deallocation routine
CALPAG EQU >F334        System paged call routine
*
```

```
DELPAB
  MOVD PABADR,DYNADR    Save the PAB address for later release
  LDA  *PABADR          Fetch the LSB of the link to the next PAB
  STA  *PRVPAB          Store as the link in the previous PAB
  DECD PABADR           Point to MSB
  DECD PRVPAB           Point to MSB
  LDA  *PABADR          Fetch the MSB of the link to the next PAB
  STA  *PRVPAB          Store as the link in the previous PAB
  MOVD @MDISP,B         Release the allocated PAB space
  CALL @CALPAB
  MOVD BUFADR,DYNADR    Prepare to release the I/O buffer
  MOVD @MDISP,B         Release the I/O buffer
  CALL @CALPAB
  RETS                  Return to the calling program
```

CHAPTER 6
CC-40 SYSTEM HARDWARE DESCRIPTION

The CC-40 system is designed around a TMS70C20 eight-bit CMOS microprocessor with a clock rate of 2.5 megahertz. The processor integrated-circuit chip contains both random-access memory (RAM) and read-only memory (ROM). Additionally, the processor addresses internal and external registers in a 256-byte address block at >100-1FF for "peripheral" registers which are accessed by processor input/output commands. During normal CC-40 operation, the first eight addresses of this block access registers which are in the processor chip. The remaining addresses access devices external to the chip.

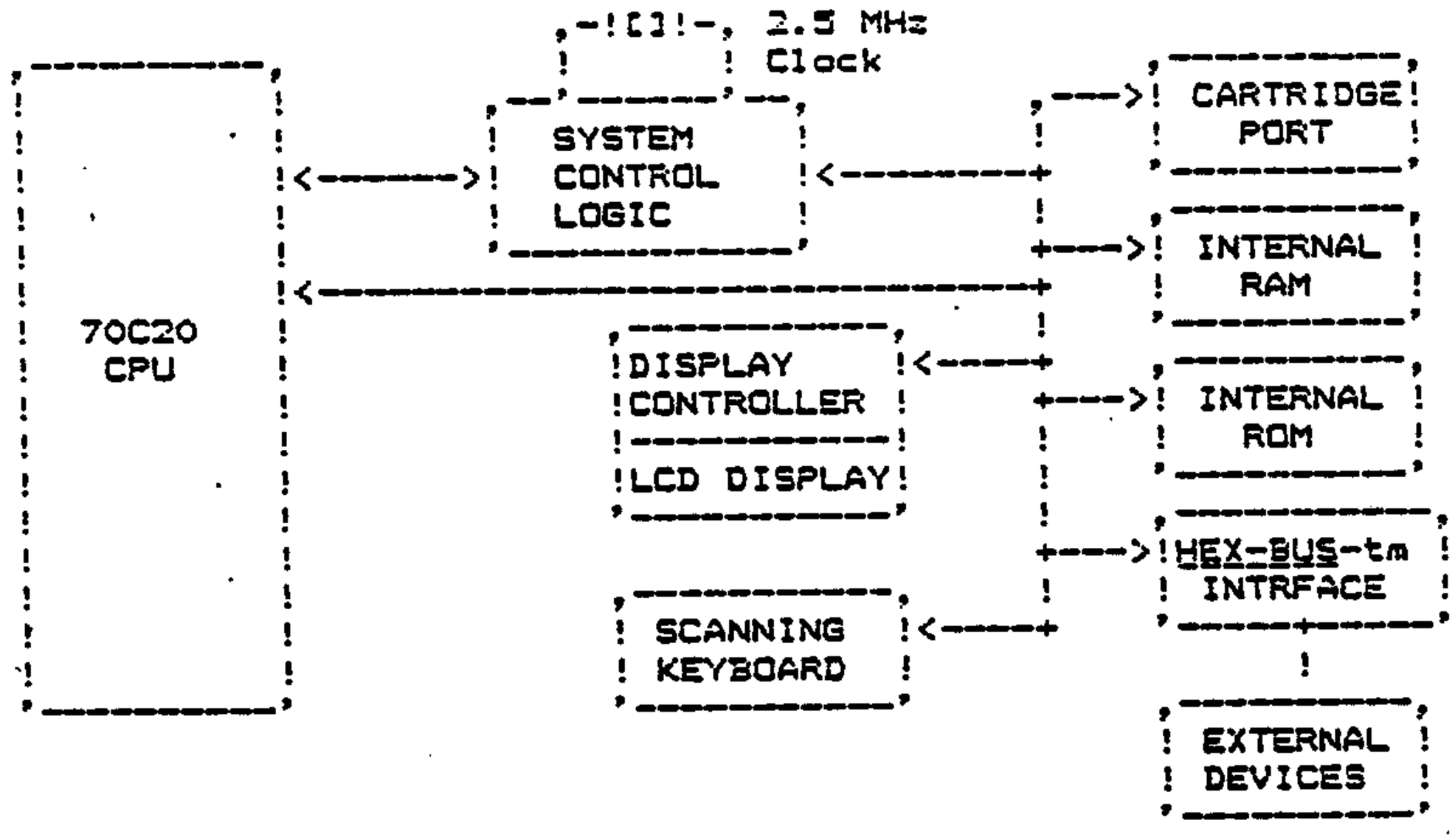
The processor chip is designed for interrupt operation in conjunction with a built-in timer and event counter.

System elements external to the 70C20 chip but inside the CC-40 console include the following.

* 32K bytes of ROM containing system software (code for BASIC and a debug monitor)
* 6K bytes of RAM (with the capability of internal expansion to 18K)
* A scanning-type keyboard
* A display controller and a liquid crystal display (LCD)
* A cartridge port through which external RAM or ROM may be added to the system by the user
* A HEX-BUS-tm Intelligent Peripheral Interface for communication with devices external to the CC-40 console
* A power supply

System elements outside the the CC-40 console are intelligent-peripheral devices controlled by software-generated

command codes issued over the HEX-BUS-tm interface. These devices include RS232 interface devices and Wafertape-tm devices. A block diagram of the CC-40 system is shown below.



THE 70C20 PROCESSOR

The 70C20 processor, as it is used in the CC-40, has two modes of operation. It operates in single-chip mode during powerup. After powerup, it operates full-expansion mode with full interface to other CC-40 system elements.

Control, data, and memory-address signals to and from the processor, as listed below, are grouped into four "ports," some lines of which function differently during full-expansion and single-chip operation.

PORT A: In either full-expansion or single-chip operation, a group of eight input lines accessed at address >104 (P4).

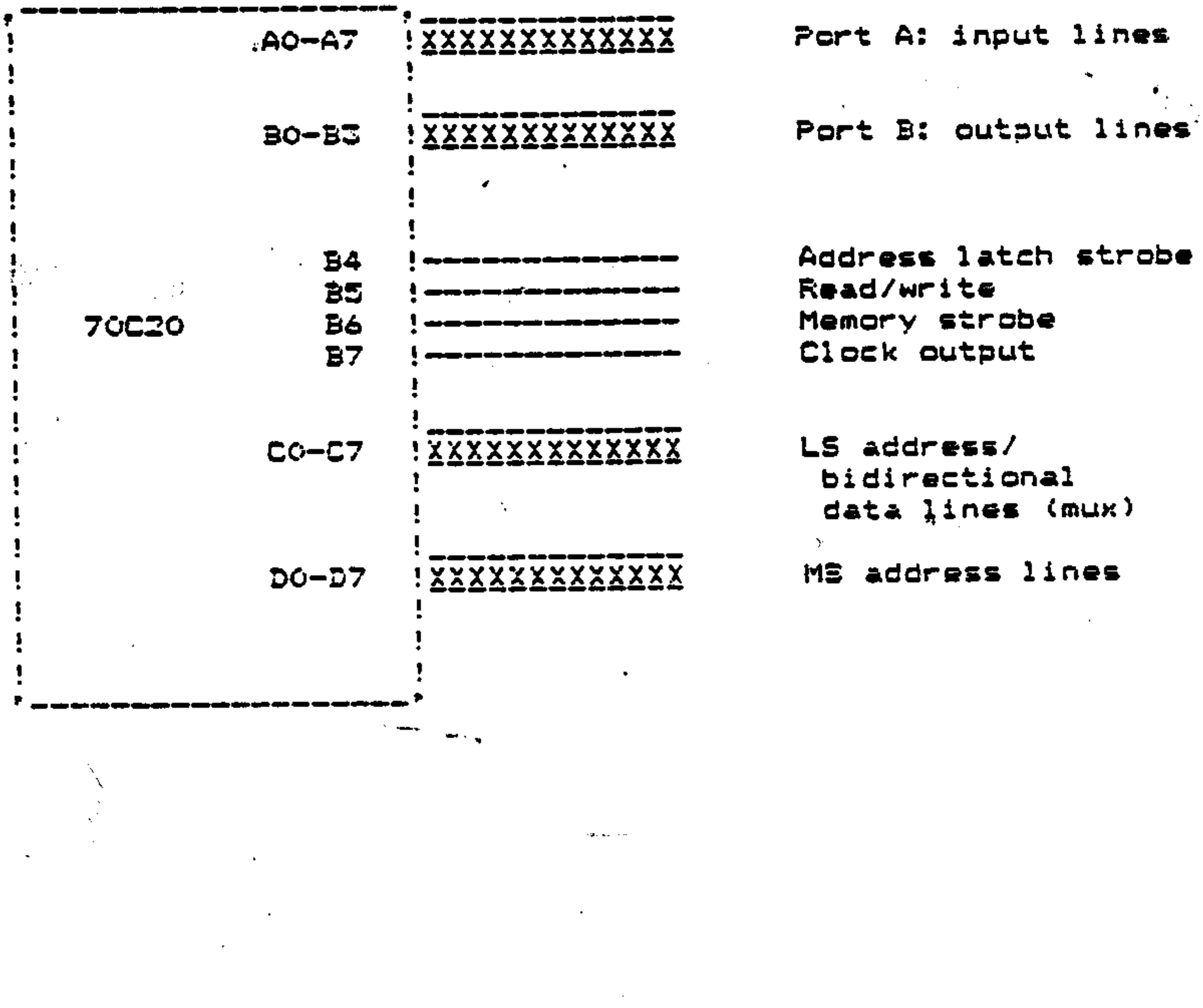
PORT B: In full-expansion operation, a four-bit group of bidirectional I/O lines and a group of four processor control and status lines. In single-chip operation, an eight-bit latched output port.

PORT C: In full-expansion operation, an eight-bit group of lines with multiplexed bidirectional data-bus signals and low-order address-output signals. In single-chip operation, an eight-bit bidirectional I/O port.

PORT D: In full-expansion operation, a group of eight high-order address lines. In single-chip operation, an eight-bit bidirectional I/O port.

The 70C20 is initialized during reset to single-chip operation in which only the 2K bytes of internal ROM and the register and peripheral files can be accessed. When firmware in the 70C20 chip begins execution, the operation of the processor is switched to full-expansion mode so that all 64K bytes of processor addressing capability can be used in CC-40 operation.

In full-expansion operation, the lines of the 70C20 appear as shown in the figure below.



Further information on the 70C20 processor is included in TMS7000 Data Manual, available from Texas Instruments XXX Division, 123 X Street, Somewhere.

MEMORY ORGANIZATION

In the CC-40, the 70C20 microprocessor addresses 64K bytes of memory. This memory is mapped into the following blocks.

* The 128-byte general-purpose on-chip RAM register file
* The peripheral-register file implemented in both on-chip and off-chip registers
* System RAM in the CC-40 console
* ROM or RAM in a cartridge plugged into the console cartridge port
* System ROM (firmware) within the CC-40 console
* System ROM within the 70C20 processor chip

Each of these blocks is addressed at a specific area in the CC-40 memory map as shown below.

CC-40 MEMORY MAP

HEX	Description	
>0000- >007F	Register file	(128 bytes)
>0080- >00FF	Unused	(128 bytes)
>0100- >01FF	Peripheral File	(256 bytes)
>0200- >07FF	Unused	(1.5K bytes)
>0800- >4FFF	System or Cartridge RAM (up to 18K bytes)	
>5000- >CFFF	Cartridge ROM or RAM (maximum 4 pages of 32K bytes/page)	
>D000- >EFFF	System ROM (maximum four pages of 8K bytes/page)	
>F000- >F7FF	Unused	(2K bytes)
>F800- >FFFF	Processor ROM	(2K bytes)

GENERAL-PURPOSE REGISTER FILE

The register file contains the following general-purpose registers implemented in on-chip RAM.

1. A register designated "A" or R0 at address >0000
2. A register designated "B" or R1 at address >0001
3. Registers designated by numbers R2-R127 (or R>2 through R>7F) at addresses >0002-007F

PERIPHERAL-REGISTER FILE

The 70C20 processor interprets Input/Output commands by reading from and writing to addresses >100-1FF with register names P0 through P255 (or P>0 through P>FF). These registers, and their functions, are listed below.

MEMORY IN THE CONSOLE

Memory in the console consists of three areas: (1) system RAM from >800 through (as high as) >4FFF, (2) system ROM from >D000 through >EFFF, and (3) processor ROM from >F800 through >FFFF.

PERIPHERAL-FILE REGISTERS

REG NO.	HEX ADR	Description
P0	>100	I/O control
P1	>101	Processor use only
P2	>102	Timer data
P3	>103	Timer control
P4	>104	Byte from keyboard
P5	>105	Processor use only
P6	>106	Byte to keyboard

"CONTINUED ON FOLLOWING PAGE

PERIPHERAL-FILE REGISTERS (CONTINUED)

REG NO.	HEX ADR	Description
P7	>107	Processor use only
P16	>110	Address. speed ctrl
P17	>111	Power on hold latch
P18	>112	I/O bus data
P19	>113	I/O bus available
P20	>114	I/O bus handshake
P21	>115	Beeper control
P22	>116	Low battery sense
P26	>11A	System clock control
P30	>11E	LCD commands / status
P31	>11F	LCD data

System RAM

Depending on the size of the memory chips installed during manufacture, system RAM within the console extends to as high as >4FFF.

System RAM installed in the CC-40 console is always mapped to begin at >800, with 2k bytes of CC-40 system RAM always present at >800 through >FFF. During manufacture, additional RAM chips are installed in the console to provide RAM beginning at >1000. These chips may be 2k-byte chips, 8k-byte chips, or one of each. Bits 2 and 3 of peripheral-file register P16 are set to provide for contiguous addressing of the two sizes of RAM chips, as shown in the following table.

CONSOLE 2K AND 8K RAM EXPANSION

P16		RAM CHIP SIZE			
b3	b2	RAM 1	RAM 2	Adr RAM 1	Adr RAM 1
0	0	8K	8K	1000-2FFF	3000-4FFF
0	1	8K	2K	1000-2FFF	3000-37FF
1	0	2K	8K	1000-17FF	1800-37FF
1	1	2K	2K	1000-17FF	1800-1FFF

System ROM

System ROM addresses begin at >D000 and extend through >EFFF. This 8k-byte block of memory is actually one of four distinct "pages" of a 32k-byte ROM. The page (from page 0 through page 3) which is active at a particular time is controlled by bits 0 and 1 of P25, as shown in the following table.

SYSTEM ROM PAGE SELECTION BY P25

b1	b0	PAGE
0	0	0
0	1	1
1	0	2
1	1	3

At system reset, both bits affecting system ROM paging are initialized to 0. Until changed by software, page 0 of system ROM is active.

Processor ROM

Processor ROM contains firmware in a single 2K byte page beginning at >F800.

CARTRIDGE MEMORY

Cartridge memory may be RAM or ROM addressed from >5000 through >CFFF. As in system RAM, peripheral-register control provides for addressing of different-size ROM or RAM chips. As in system ROM, cartridge memory may be selected from among four pages.

Cartridge memory addressing within the >5000-CFFF address block is set up under control of bits 5 and 6 or P16 so that 2k-, 8k-, or 16k-byte ROM chips may be used in contiguous address blocks.

Additionally, cartridge memory may be mapped to be contiguous with the first two kilobytes of system RAM. When so mapped, cartridge memory containing 8K byte chips appears from >1000 through >4FFF, and all system RAM chips installed in the console at or above >1000 are disabled.

CARTRIDGE MEMORY ADDRESSING BITS IN P16

CHIP SIZE	b6	b5	ROM 1 ADR	ROM 2 ADR
2k bytes	0	0	5000-57FF	5800-5FFF
8k bytes	0	1	5000-6FFF	7000-8FFF
16k bytes	1	0	5000-8FFF	9000-CFFF
8k bytes	1	1	1000-2FFF	3000-4FFF

The same peripheral-file register which controls paging for system ROM also controls paging of cartridge memory. Bits 2 and 3 of register P25 (>119) select the page, as shown in the following table.

CARTRIDGE PAGE SELECTION BY P25

b3	b2	SELECTED PAGE
0	0	Page 0
0	1	Page 1
1	0	Page 2
1	1	Page 3

At system reset, both bits affecting cartridge memory paging are initialized to 0. Until changed by software, page 0 of cartridge memory is active.

Cartridge Memory Speed Control

The speed of the CC-40 system clock is normally 2.5 MHz, resulting in a memory access time of 300 nanoseconds. The clock may be slowed by any odd-numbered factor between 3 and 17. Slow clock speed permits the use of ROM chips with access times greater than 300 nanoseconds (and it permits RAM to be used at slow speed to simulate execution in slow ROM). The factor by which the clock is slowed, a divisor of the clock speed, is placed in P26, bits 0, 1, and 2 in accordance with the values in the following table.

SLOW-CLOCK BITS

Register P26 b2	b1	b0	Division factor	Slow clock Speed (MHz)	Access Time (Average, usec)
0	0	0	17	0.147	9.8
0	0	1	15	0.167	8.6
0	1	0	13	0.192	7.4
0	1	1	11	0.227	6.2
1	0	0	9	0.278	5.0
1	0	1	7	0.357	3.8
1	1	0	5	0.500	2.6
1	1	1	3	0.833	1.4

CAUTION: Setting the division factor to 9 or greater may impair processor operation.

Setting bits in P26 for slow system-clock operation only defines the speed of slow-clock operation. Clock speed is unaffected until certain conditions, set under either hardware or software control, are in effect.

The clock is slowed under software control by writing ONE into Bit 3 of P26. Until a ZERO is written into the bit, the clock thereafter runs at the frequency set by bits 0, 1, and 2.

The clock may be automatically slowed for selected blocks of memory under hardware control. For hardware control of clock frequency, bits must be set in P16 to identify system memory, cartridge memory, or both for slow-clock access. Setting bit 6 of P16 identifies system ROM (>0000 through >E000) for slow operation. Setting bits 0 and 1 of P16, respectively, identify

the first and second chips in cartridge memory for slow operation.

PROCESSOR INTERRUPTS AND TIMER CONTROL

TMS7000-family processors have three maskable interrupts, two of which are inputs from off-chip devices and a third of which is an input from an on-chip timer and event counter. Within the CC-40, only the timer interrupt is of practical use: the off-chip interrupts must be reserved for control of system functions by firmware.

USE OF INTERRUPTS

Interrupts are initiated by an interrupt request, which is a Low signal appearing on a processor interrupt-input line or a similar signal from the timer in the processor chip. If the processor is set to recognize an interrupt request, it responds to the request (after it completes execution of its current instruction) by pushing the status register, the most-significant program counter byte, and the least-significant program counter byte onto the stack. The program counter is then loaded indirectly with addresses stored in firmware for the TRAP 1, TRAP 2, and TRAP 3 (TRAP 0 is reserved for use by the processor upon receiving a RESET signal).

The three interrupts are assigned priority levels in accordance with the order of processor recognition when they occur simultaneously. Interrupt 1, which branches to the TRAP 1 address stored at >FFFC and >FFFD, has highest priority. Interruot 2, which branches to the TRAP 2 address stored at >FFFA

and >FFFB. has intermediate priority: interrupt 3, which branches to the TRAP 3 address at >FFFB and >FFF9, has lowest priority.

Within the CC-40, interrupts 1 and 3 and the routines which service them are inaccessable except to system devices and system firmware (through the HEX-BUS interface, and the like). Interrupt 2 is accessable through software control of the timer and event counter (see below).

For the timer and event-counter interrupt to be recognized, two conditions must be present. Processor interrupt logic must be enabled with an EINT instruction. Also, an individual bit corresponding to the interrupt must be set in the TMS70C20 input/output control register (IOCR), which is accessed at P0 of the peripheral-register file.

The IOCR is an eight-bit register which provides software control of the use of input-output pins on the TMS70C20 chip. The lower six bits of the register control interrupt recognition. The upper two bits control whether the processor is in single-chip or full-expansion mode. Bit 7 must remain ONE and bit 6 must remain ZERO 10 during normal CC-40 program execution. Altering these bits results in loss of data and program control.

The lower six bits of the IOCR are written to in order to enable ("mask") each of the three interrupts, and they are read in order to determine whether an interrupt is enabled and whether an interrupt request has occurred. For a write operation, these bits appear as in the following table.

BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
INT 3 CLEAR	INT 3 ENABLE	INT 2 CLEAR	INT 2 ENABLE	INT 1 CLEAR	INT 1 ENABLE

Writing a ONE to an interrupt-enable bit masks the corresponding interrupt input for processor response. After this bit is set to ONE and an EINT instruction is in effect, the processor responds to the interrupt request as described above. After it is set to ZERO (or a DINT instruction is in effect), the processor does not respond to an interrupt request.

The interrupt-clear bits reset interrupt requests that have been latched upon receipt of interrupt requests, as described below. Interrupts may not be cleared and enabled simultaneously.

The lower six bits of the IOCR are read in order to determine whether interrupts are enabled and whether an interrupt request has been received. Input bits of the IOCR appear as in the following table.

BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
INT 3 FLAG	INT 3 ENABLE	INT 2 FLAG	INT 2 ENABLE	INT 1 FLAG	INT 1 ENABLE

If interrupt 1 has been enabled by writing ONE into bit 0, a ONE bit is present for reading from bit 0.

The interrupt-flag bits are set to ONE by interrupt requests regardless of whether interrupts are enabled or an EINT instruction is in effect. They remain set until the processor

value with the current count and store the result as the count modulus.

CONSOLE INPUT/OUTPUT DEVICES

Input/output devices in the CC-40 console include the compact keyboard, the liquid crystal display, and the beeper. These units communicate directly with the 70C20 processor via registers in the peripheral-register file. Peripheral units external to the console rely on the HEX-BUS-tm, while the HEX-BUS-tm itself is implemented via registers in the peripheral-register file.

CONSOLE KEYBOARD

Keyboard contact closure is detected by scanning columns and rows of keyswitches in an eight-column by eight-row matrix. Successive bytes output by the 70C20 processor write a single latched ONE bit (High signal level) to each of the eight columns of the keyswitch matrix. While each individual column lines is latched High, a byte input to the 70C20 from the eight rows of the matrix is tested for a High signal (ONE bit). If a High signal is detected, the keyswitch corresponding to the column (to which the ONE is written) and the row (from which the ONE is read) is uniquely identified.

Column bytes are written through peripheral-file register P6 and latched, and row bytes are read through the unlatched Port A (P4) inputs.

LIQUID-CRYSTAL DISPLAY (LCD) AND CONTROLLER

The liquid-crystal display shows 31 of the 80 alphanumeric or user-defined characters in display RAM. Each character is formed from a 5-by-7 dot matrix, generation of which is performed in a 190 character ROM or a 7-character RAM.

The display unit is addressed at P30 and P31 of the peripheral-register file. It is controlled with instruction bytes and RAM addresses written to P30. Bytes for display RAM or character-generator RAM are written into P31. The current RAM address (automatically incremented after each read or write to the display) is available from reading bits 0-6 of P30 and the controller status (ready/busy) is available from bit 7. The currently addressed display- or character-generator RAM character is read from P31.

Instructions provided to the display control the following operations.

(1) display clearing and scrolling
(2) cursor positioning, blinking, and disable (off)
(3) display disable/enable (on/off)

BEEPER

The beeper is driven by a transistor amplifier from the latched bit 0 output of P21. Alternately writing ONEs and ZEROs to this register produces an amplified square wave which causes the beeper to emit sound.

acknowledges the corresponding interrupts or until they are cleared under program control by writing ONEs to corresponding interrupt-clear bits in the IOCR.

USE OF THE TIMER AND EVENT COUNTER

The timer and event counter consists of an eight-bit decrementing counter and a count-frequency prescaler. When the counter decrements past >00, it provides the processor with a level 2 interrupt pulse.

The counter is loaded with a count modulus through peripheral-file register P2. After the modulus value is stored and the counter is enabled, the counter decrements at the counter-clock frequency until it decrements past >00. The counter then issues an interrupt 2 to the processor. After issuing the interrupt, the counter is once again loaded with the count modulus, and it continues decrementing.

The counter-clock is either internal to or external to the processor. A ZERO written to bit 6 of P3 sets the clock input to be internal to the processor at one sixteenth the processor clock frequency. A ONE bit sets clock input to be external to the processor through bit 7 of Port A, enabling the clock to count events. The event counter decrements on the positive-going edge of an input signal.

The counter is loaded with the modulus and enabled when ONE is written to bit 7 of P3. When ZERO is written to the bit, the counter is disabled and the count is frozen until it is once again enabled. If a ONE is written to the counter-enable bit

when the counter is already enabled, the count modulus is restored once again.

The prescaler divides the frequency of incoming pulses by any value up to 32. This prescaling divisor is one plus the value written into the least significant five bits of P3.

Bit 5 of P3 must always be ONE. Writing a ZERO to this bit, which is unrelated to counter operation, powers down some of the elements of the TMS70C20 and, when an IDLE instruction is executed, results in a loss of data and program control.

A value of >A3 written into P3, for example, sets the clock to the processor clock divided by 16, starts the clock, and sets the prescaler divisor to 4. The clock therefore decrements the counter at 1/64th the processor clock rate.

Information written (output) to P2 and P3 is not the same as information which may be read (input) from the registers. Whereas the count modulus is written to P2, the current value in the counter is read from the register. Whereas counter-control information is written to P3, the latched value of the counter when the last interrupt 3 pulse occurred is read from the register.

Differences in information written to and read from the counter registers make it very important that only the MOVP RN,PN and the STA instructions are used to output the count modulus and counter-control byte. Other output instructions, which read a peripheral register before writing to it, produce undesirable results. Using any of the peripheral-file logical instructions (ANDP, ORP, or XORP), for example, will logically operate on a

HEX-BUS-tm INTELLIGENT PERIPHERAL INTERFACE

The HEX-BUS interface is a four-bit, medium-speed I/O bus which transfers data at speeds up to 6000 bytes/second. The bus communicates over an eight-line cable which contains the following groups of lines.

(1) Four bidirectional data lines interfaced to the lower four bits of peripheral-file register P18

(2) A bus-available line interfaced to the least-significant bit of P19

(3) A handshake line interfaced to the least-significant bit of P20

(4) A ground line

(5) A line reserved for for future expansion of the bus

Signals for the data lines are clocked out of and into processor address and data lines 0 through 3 (bits 0 through 3 of Port C at P18). On output, the nibble of data is latched and provided through open-drain buffers to the bus lines. On input, tri-state buffers strobe data into bits 0 through 3 of P18).

Data transfers on the bus are controlled by the bidirectional bus-available and handshake lines. The bus-available signal controls the start and finish of a transmission: the signal is pulled low by the CC-40 console or a peripheral to indicate that a message is about to be transmitted over the bus. The handshake signal indicates, when low, that a nibble is available on the bus.

Message transmission on the bus consists of the console transmitting a bus command message along with any required data to a particular device, and the device transmitting a response

message with any returned data and the status of the operation.

Program-controlled operation of the HEX-BUS-tm through the Input/Output Subsystem is described in chapter 2 of this manual.

POWER-ON HOLD REGISTER OUTPUT

To provide for program-control of CC-40 power down, power to the unit is controlled by register P17 in the peripheral file. Bit 0, is set to ONE by hardware during power up to latch the power on. Power continues to be provided to the unit until a ZERO is written to the bit as the last step of a power-down sequence.

LOW BATTERY VOLTAGE INPUT

When battery voltage to the CC-40 becomes marginal, bit 0 of P22 is ZERO. Programs developed with the ALDS may use this bit to detect impending battery failure.