# APPENDIX B
# ASSIST09 MONITOR PROGRAM

## B.1 GENERAL DESCRIPTION

The M6809 is a high-performance microprocessor which supports modern programming techniques such as position-independent, reentrancy, and modular programming. For a software monitor to take advantage of such capabilities demands a more refined and sophisticated user interface than that provided by previous monitors. ASSIST09 is a monitor which supports the advanced features that the M6809 makes possible. ASSIST09 features include the following:

- Coded in a position (address) independent manner. Will execute anywhere in the 64K address space.
- Multiple means available for installing user modifications and extensions.
- Full complement of commands for program development including breakpoint and trace.
- Sophisticated monitor calls for completely address-independent user program services.
- RAM work area is located relative to the ASSIST09 ROM, not at a fixed address as with other monitors.
- Easily adapted to real-time environments.
- Hooks for user command tables, I/O handlers, and default specifications.
- A complete user interface with services normally only seen in full disk operating systems.

The concise instruction set of the M6809 allows all of these functions and more to be contained in only 2048 bytes.

The ASSIST09 monitor is easily adapted to run under control of a real-time operating system. A special function is available which allows voluntary time-slicing, as well as forced time-slicing upon the use of several service routines by a user program.

## B.2 IMPLEMENTATION REQUIREMENTS

Since ASSIST09 was coded in an address-independent manner, it will properly execute anywhere in the 64K address space of the M6809. However, an assumption must be made regarding the location of a work area needed to hold miscellaneous variables and the default stack location. This work area is called the page work area and it is addressed within ASSIST09 by use of the direct page register. It is located relative to the start of the

ASSIST09 ROM by an offset of −1900 hexadecimal. Assuming ASSIST09 resides at the top of the memory address space for direct control of the hardware interrupt vectors, the memory map would appear as shown in Figure B-1.
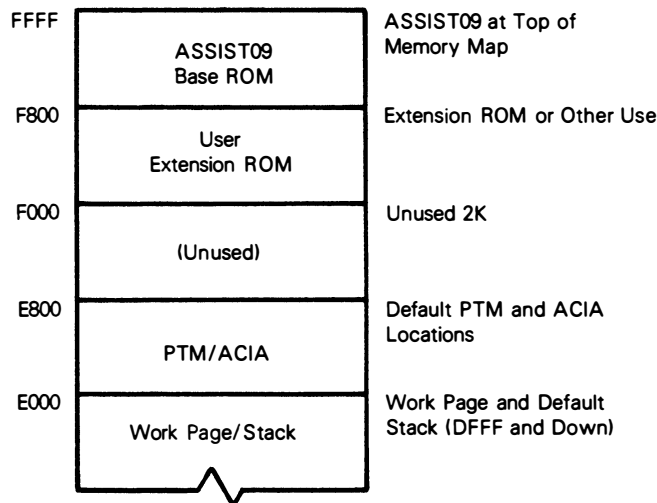


**Figure B-1. Memory Map**

If F800 is not the start of the monitor ROM the addresses would change, but the relative locations would remain the same except for the programmable timer module (PTM) and asynchronous communications interface adapter (ACIA) default addresses which are fixed.

The default console input/output handlers access an ACIA located at E008. For trace commands, a PTM with default address E000 is used to force an $\overline{\text{NMI}}$ so that single instructions may be executed. These default addresses may easily be changed using one of several methods. The console I/O handlers may also be replaced by user routines. The PTM is initialized during the MONITR service call (see Paragraph B.9 SERVICES) to fireup the monitor unless its default address has been changed to zero, in which case no PTM references will occur.

## B.3 INTERRUPT CONTROL

Upon reset, a vector table is created which contains, among other things, default interrupt vector handler appendage addresses. These routines may easily be replaced by user appendages with the vector swap service described later. The default actions taken by the appendages are as follows:

$\overline{\text{RESET}}$ — Build the ASSIST09 vector table and setup monitor defaults, then invoke the monitor startup routine.

SWI — Request a service from ASSIST09.

$\overline{\text{FIRQ}}$ — An immediate RTI is done.

SWI2, SWI3, $\overline{\text{IRQ}}$, Reserved, $\overline{\text{NMI}}$ — Force a breakpoint and enter the command processor.

The use of $\overline{\text{IRQ}}$ is recommended as an abort function during program debugging sessions, as breakpoints and other ASSIST09 defaults are reinitialized upon $\overline{\text{RESET}}$. Only the primary software interrupt instruction (SWI) is used, not the SWI2 or SWI3. This avoids page fault problems which would otherwise occur with a memory management unit as the SWI2 and SWI3 instructions do not disable interrupts.

Counter number one of the PTM is used to cause an $\overline{\text{NMI}}$ interrupt for the trace and breakpoint commands. At $\overline{\text{RESET}}$ the control register for timer one is initialized for tracing purposes. If no tracing or breakpointing is done then the entire PTM is available to the user. Otherwise, only counters two and three are available. Although control register two must be used to initialize control register one, ASSIST09 returns control register two to the same value it has after a $\overline{\text{RESET}}$ occurs. Therefore, the only condition imposed on a user program is that if the "operate/preset" bit in control register one must be turned on, $A7 should be stored, $A6 should be stored if it must be turned off.


## B.4 INITIALIZATION

During ASSIST09 execution, a vector table is used to address certain service routines and default values. This table is generated to provide easily changed control information for user modifications. The first byte of the ASSIST09 ROM contains the start of a subroutine which initializes the vector table along with setting up certain default values before returning to the caller.

If the ASSIST09 $\overline{\text{RESET}}$ vector receives control, it does three things:
1. Assigns a default stack in the work space,
2. Calls the aforementioned subroutine to initialize the vector table, and
3. Fires up the ASSIST09 monitor proper with a MONITR SWI service request.

However, a user routine can perform the same functions with a bonus. After calling the vector intitialization subroutine, it may examine or alter any of the vector table values before starting normal ASSIST09 processing. Thus, a user routine may "bootstrap" ASSIST09 and alter the default standard values.

Another method of inserting user modifications is to have a user routine reside at an extension ROM location 2K below the start of the ASSIST09 ROM. The vector table initialization routine mentioned above, looks for a "BRA*" flag ($20FE) at this address, and if found calls the location following the flag as a subroutine with the U register pointing to the vector table. Since this is done after vector table initialization, any or all defaults may be altered at this time. A big advantage to using this method is that the modifications are "automatic" in that upon a $\overline{\text{RESET}}$ condition the changes are made without overt action required such as the execution of a memory change command.

No special stack is used during ASSIST09 processing. This means that the stack pointer must be valid at all interruptable times and should contain enough room for the stacking of at least 21 bytes of information. The stack in use during the initial MONITR service call to start up ASSIST09 processing becomes the "official" stack. If any later stack validity checks occur, this same stack will be re-based before entering the command handler.

ASSIST09 uses a work area which is addressed at an offset from the start of the ASSIST09 ROM. The offset value is −1900 hexadecimal. This points to the base page used during monitor execution and contains the vector table as well as the start of the default stack. If the default stack is used and it exceeds 81 bytes in size, then contiguous RAM must exist below this base work page for proper extension of the stack.

## B5. INPUT/OUTPUT CONTROL

Output generated by use of the ASSIST09 services may be halted by pressing any key, causing a 'FREEZE' mode to be entered. The next keyboard entry will release this condition allowing normal output to continue. Commands which generate large amounts of output may be aborted by entering CANCEL (CONTROL-X). User programs may also monitor for CANCEL along with the 'FREEZE' condition even when not performing console I/O (PAUSE service).

## B.6 COMMAND FORMAT

There are three possible formats for a command:

<Command> CR

<Command> <Expression1> CR

<Command> <Expression1> <Expression2> CR

The space character is used as the delimiter between the command and all arguments. Two special quick commands need no carriage return, "." and "/". To re-enter a command once a mistake is made, type the CANCEL (CONTROL-X) key.

Each "expression" above consists of one or more values separated by an operator. Values can be hex strings, the letters "P", "M", and "W", or the result of a function. Each hexadecimal string is converted internally to a 16-bit binary number. The letter "P" stands for the current program counter, "M" for the last memory examine/change address, and "W" for the window value. The window value is set by using the WINDOW command.

One function exists and it is the INDIRECT function. The character "@" following a value replaces that value with the 16-bit number obtained by using that value as an address.

Two operators are allowed, "+" and "−" which cause addition and subtraction. Values are operated on in a left-to-right order.

Examples:

480 — hexadecimal 480

W+3 — value of window plus three

P-200 — current program counter minus 200 hexadecimal

M−W — current memory pointer minus window value

100@ — value of word addressed by the two bytes at 100 hexadecimal

P+1@ — value addressed by the word located one byte up from the current program counter

## B.7 COMMAND LIST

Table B-1 lists the commands available in the ASSIST09 monitor.

### Table B-1. Command List

| Command Name | Description | Command Entry |
|---|---|---|
| Breakpoint | Set, clear, display, or delete breakpoints | B |
| Call | Call program as subroutine | C |
| Display | Display memory block in hex and ASCII | D |
| Encode | Return indexed postbyte value | E |
| Go | Start or resume program execution | G |
| Load | Load memory from tape | L |
| Memory | Examine or alter memory | M |
| | Memory change or examine last referenced | / |
| | Memory change or examine | hex/ |
| Null | Set new character and new line padding | N |
| Offset | Compute branch offsets | O |
| Punch | Punch memory on tape | P |
| Registers | Display or alter registers | R |
| Stlevel | Alter stack trace level value | S |
| Trace | Trace number of instructions | T |
| | Trace one instruction | . |
| Verify | Verify tape to memory load | V |
| Window | Set a window value | W |

## B.8 COMMANDS

Each of the commands are explained on the following pages. They are arranged in alphabetical order by the command name used in the command list. The command name appears at each margin and in slightly larger type for easy reference.

# BREAKPOINT BREAKPOINT

**Format:**   Breakpoint
Breakpoint –
Breakpoint <Address>
Breakpoint – <Address>

**Operation:** Set or change the breakpoint table. The first format displays all breakpoints. The second clears the breakpoint table. The third enters an address into the table. The fourth deletes an address from the table. At reset, all breakpoints are deleted. Only instructions in RAM may be breakpointed.

# CALL CALL

**Format:**   Call
Call <Address>

**Operation:** Call and execute a user routine as a subroutine. The current program counter will be used unless the address is specified. The user routine should eventually terminate with a "RTS" instruction. When this occurs, a breakpoint will ensue and the program counter will point into the monitor.

# DISPLAY <span style="float:right">DISPLAY</span>

**Format:**   Display <From>
Display <From> <Length>
Display <From> <To>

**Operation:** Display contents of memory in hexadecimal and ASCII characters. The second argument, when entered, is taken to be a length if it is less than the first, otherwise it is the ending address. A default length of 16 decimal is assumed for the first format. The addresses are adjusted to include all bytes within the surrounding modulo 16 address byte boundary. The CANCEL (CONTROL-X) key may be entered to abort the display. Care must be exercised when the last 15 bytes of memory are to be displayed. The <Length> option should always be used in this case to assure proper termination: D FFE0 40

Examples:

D    M    10   — Display 16 bytes surrounding the last memory location examined.

D    E000   F000   — Display memory from E000 to F000 hex.

# ENCODE <span style="float:right">ENCODE</span>

**Format:**   Encode <Indexed operand>

**Operation:** The encode command will return the indexing instruction mode postbyte value from the entered assembler-like syntax operand. This is useful when hand coding instructions. The letter "H" is used to indicate the number of hex digits needed in the expression as shown in the following examples:

E ,Y              — Return zero offset to Y register postbyte.

E [HHHH,PCR] — Return two byte PCR offset using indirection.

E [,S+ +]        — Return autoincrement S by two indirect.

E H,X            — Return 5-bit offset from X.

Note that one "H" specifies a 5-bit offset, and that the result given will have zeros in the offset value position. This comand does not detect all incorrectly specified syntax or illegal indexing modes.

# GO                                    GO

**Format:** Go
Go <Address>

**Operation:** Execute starting from the address given. The first format will continue from the current program counter setting. If it is a breakpoint no break will be taken. This allows continuation from a breakpoint. The second format will breakpoint if the address specified is in the breakpoint list.


# LOAD                                  LOAD

**Format:** Load
Load <Offset>

**Operation:** Load a tape file created using the S1-S9 format. The offset option, if used, is added to the address on the tape to specify the actual load address. All offsets are positive, but wrap around memory modulo 64K. Depending on the equipment involved, after the load is complete a few spurious characters may still be sent by the input device and interpreted as command characters. If this happens, a CANCEL (CONTROL-X) should be entered to cause such characters to be ignored. If the load was not successful a "?" is displayed.

**Format:**   MEMORY <Address>/
<Address>/
/

**Operation:** Initiate the memory examine/change function. The second format will not accept an expression for the address, only a hex string. The third format defaults to the address displayed during the last memory change/examine function. (The same value is obtained in expressions by use of the letter "M".) After activation, the following actions may be taken until a carriage return is entered:

| | |
|---|---|
| <Expr> | Replaces the byte with the specified value. The value may be an expression. |
| SPACE | Go to next address and print the byte value. |
| , | (Comma) Go to next address without printing the byte value. |
| LF | (Line feed) Go to next address and print it along with the byte value on the next line. |
| ∧ | (Circumflex or Up arrow) Go the previous address and print it along with the byte value on the next line. |
| / | Print the current address with the byte value on the next line. |
| CR | (Carriage return) Terminate the command. |
| '<Text>' | Replace succeeding bytes with ASCII characters until the second apostrophe is entered. |

If a change attempt fails (i.e., the location is not valid RAM) then a question mark will appear and the next location displayed.

# NULL                                    NULL

**Format:**   Null < Specification >

**Operation:** Set the new line and character padding count values. The expression value is treated as two values. The upper two hex represent the character pad count, and the lower two the new line pad count (triggered by a carriage return). An expression of less than three hex digits will set the character pad count to zero. The values must range from zero to 7F hexadecimal (127 decimal).

Example:

   N        3   — Set the character count to zero and new line count to three.

   N        207  — Set character padding count to two and new line count to seven.

Settings for TI Silent 700 terminals are:

| Baud | Setting |
|------|---------|
| 100  | 0       |
| 300  | 4       |
| 1200 | 317     |
| 2400 | 72F     |

# OFFSET                                  OFFSET

**Format:**   Offset < Offset addr >  < To instruction >

**Operation:** Print the one and two byte offsets needed to perform a branch from the first expression to the instruction. Thus, offsets for branches as well as indexed mode instructions which use offsets may be obtained. If only a four byte value is printed, then a short branch count cannot be done between the two addresses.

Example:

   0     P + 2   A000  — Compute offsets needed from the current program counter plus two to A000.

# PUNCH

**Format:**   Punch < From >  < To >

**Operation:** Punch or record formatted binary object tape in S1-S9 (MIKBUG) format.


# REGISTER

**Format:**   Register

**Operation:** Print the register set and prompt for a change. At each prompt the following may be entered.

| | |
|---|---|
| SPACE | Skip to the next register prompt |
| < Expr > SPACE | Replace with the specified value and prompt for the next register. |
| < Expr > CR | (carriage return) Replace with the specified value and terminate the command. |
| CR | Terminate the command. |

---

MIKBUG is a trademark of Motorola Inc.

# STLEVEL

**Format:**    Stlevel
Stlevel < Address >

**Operation:** Set the stack trace level for inhibiting tracing information. As long as the stack is at or above the stack level address, the trace display will continue. However, when lower than the address it is inhibited. This allows tracing of a routine without including all subroutine and lower level calls in the trace information. Note that tracing through a ASSIST09 "SWI" service request may also temporarily supress trace output as explained in the description of the trace command. The first format sets the stack trace level to the current program stack value.

# TRACE

**Format:**    Trace < Count >
. (period)

**Operation:** Trace the specified number of instructions. At each trace, the opcode just executed will be shown along with the register set. The program counter in the register display points to the NEXT instruction to be executed. A CANCEL (CONTROL-X) will prematurely halt tracing. The second format (period) will cause a single trace to occur. Breakpoints have no effect during the trace. Selected portions of a trace may be disabled using the STLEVEL command. Instructions in ROM and RAM may be traced, whereas breakpoints may be done only in RAM. When tracing through a ASSIST09 service request, the trace display will be supressed starting two instructions into the monitor until shortly before control is returned to the user program. This is done to avoid an inordinate amount of displaying because ASSIST09, at times, performs a sizeable amount of processing to provide the requested services.

# VERIFY <span style="float:right">VERIFY</span>

**Format:**    Verify
              Verify < Offset >

**Operation:** Verify or compare the contents of memory to the tape file. This command has the same format and operation as a LOAD command except the file is compared to memory. If the verify fails for any reason a "?" is displayed.

# WINDOW <span style="float:right">WINDOW</span>

**Format:**    Window < Value >

**Operation:** Set the window to a value. This value may be referred to when entering expressions by use of the letter "W". The window may be set to any 16-bit value.

## B.9 SERVICES

The following describes services provided by the ASSIST09 monitor. These services are invoked by using the "SWI" instruction followed by a one byte function code. All services are designed to allow complete address independence both in invocation and operation. Unless specified otherwise, all registers are transparent over the "SWI" call. In the following descriptions, the terms "input handler" and "output handler" are used to refer to appendage routines which may be replaced by the user. The default routines perform standard I/O through an ACIA for console operations to a terminal. The ASCII CANCEL code can be entered on most terminals by depressing the CONTROL and X keys simultaneously. A list of services is given in Table B-2.

### Table B-2. Services

| Service | Entry | Code | Description |
|---------|-------|------|-------------|
| Obtain input character | INCHP | 0 | Obtain the input character in register A from the input handler |
| Output a character | OUTCH | 1 | Send the character in the register A to the output handler |
| Send string | PDATA1 | 2 | Send a string of characters to the output handler |
| Send new line and string | PDATA | 3 | Send a carriage return, line feed, and string of characters to the output handler |
| Convert byte to hex | OUT2HS | 4 | Display the byte pointed to by the X register in hex |
| Convert word to hex | OUT4HS | 5 | Display the word pointed to by the X register in hex |
| Output to next line | PCRLF | 6 | Send a carriage return and line feed to the output handler |
| Send space | SPACE | 7 | Send a blank to the output handler |
| Fireup ASSIST09 | MONITR | 8 | Enter the ASSIST09 monitor |
| Vector swap | VCTRSW | 9 | Examine or exchange a vector table entry |
| User breakpoint | BRKPT | 10 | Display registers and enter the command handler |
| Program break and check | PAUSE | 11 | Stop processing and check for a freeze or cancel condition |

# BRKPT

**User Breakpoint**

# BRKPT

**Code:**      10

**Arguments:**  None

**Result:**     A disabled breakpoint is taken. The registers are displayed and the command handler of ASSIST09 is entered.

**Description:**  Establishes user breakpoints. Both SWI2 and SWI3 default appendages cause a breakpoint as well, but do not set the I and F mask bits. However, since they may both be replaced by user routines the breakpoint service always ensures breakpoint availability. These user breakpoints have nothing to do with system breakpoints which are handled differently by the ASSIST09 monitor.

**Example:**    BRKPT    EQU  10          INPUT CODE FOR BRKPT

                     SWI              REQUEST SERVICE
                     FCB   BRKPT     FUNCTION CODE BYTE


# INCHP

**Obtain Input Character**

# INCHP

**Code:**      0

**Arguments:**  None

**Result:**     Register A contains a character obtained from the input handler.

**Description:**  Control is not returned until a valid input character is received from the input handler. The input character will have its parity bit (bit 7) stripped and forced to a zero. All NULL ($00) and RUBOUT ($7F) characters are ignored and not returned to the caller. The ECHO flag, which may be changed by the vector SWAP service, determines whether or not the input character is echoed to the output handler (full duplex operation). The default at reset is to echo input. When a carriage return ($0D) is received, line feed ($A0) is automatically sent back to the output handler.

**Example:**    INCHNP        EQU 0        INPUT CODE FOR INCHP

                     SWI                   PERFORM SERVICE CALL
                     FCB          INCHNP     FUNCTION FOR INCHNP

            A REGISTER NOW CONTAINS NEXT CHARACTER

**Code:**        8

**Arguments:**   S→Stack to become the "official" stack
DP→Direct page default for executed user programs
A = 0 Call input and output console initialization handlers and give the "ASSIST09" startup message
A#0 Go directly to the command handler

**Result:**      ASSIST09 is entered and the comand handler given control

**Description:**  The purpose for this function is to enter ASSIST09, either after a system reset, or when a user program desires to terminate. Control is not returned unless a "GO" or "CALL" command is done without altering the program counter. ASSIST09 runs on the passed stack, and if a stack error is detected during user program execution this is the stack that is rebased. The direct page register value in use remains the default for user program execution.

The ASSIST09 restart vector routine uses this function to startup monitor processing after calling the vector build subroutine as explained in IN-ITIALIZATION.

If indicated by the A register, the input and output initialization handlers are called followed by the sending of the string "ASSIST09" to the output handler. The programmable timer (PTM) is initialized, if its address is not zero, such that register 1 can be used for causing an NMI during trace commands. The command handler is then entered to perform the command request prompt.

**Example:**
```
MONITR   EQU   8               INPUT CODE FOR MONITR

LOOP     CLRA                  PREPARE ZERO PAGE REGISTER AND
*                              INITIALIZATION PARAMETER
         TFR   A,DP            SET DEFAULT PAGE VALUE
         LEAS  STACK, PCR      SETUP DEFAULT STACK VALUE
         SWI                   REQUEST SERVICE
         FCB   MONITR          FUNCTION CODE BYTE
         BRA   LOOP            REENTER IF FALLOUT OCCURS
```

# OUTCH

**Output a Character**

# OUTCH

**Code:**        1

**Arguments:**   Register A contains the byte to transmit.

**Result:**      The character is sent to the output handler

The character is set as follows ONLY if a LINEFEED was the character to transmit:
  CC = 0 if normal output occurred.
  CC = 1 if CANCEL was entered during output.

**Description:** If a FREEZE Occurs (any input character is received) then control is not returned to the user routine until the condition is released. The FREEZE condition is checked for only when a linefeed is being sent. Padding null characters ($00) may be sent following the outputted character depending on the current setting of the NULLS command. For DLE (Data Link Escape), character nulls are never sent. Otherwise, carriage returns ($00) receive the new line count of nulls, all other characters the character count of nulls.

**Example:**     OUTCH   EQU   1              INPUT CODE FOR OUTCH

                 LDA   #'0              LOAD CHARACTER "0"
                 SWI                    SEND OUT WITH MONITOR CODE
                 FCB   OUTCH            SERVICE CODE BYTE

# OUT2HS

**Convert Byte to Hex**

# OUT2HS

**Code:**        4

**Arguments:**   Register X points to a byte to display in hex.

**Result:**      The byte is converted to two hex digits and sent to the output handler followed by a blank.

**Example:**     OUT2HS   EQU 4             INPUT CODE FOR OUT2HS

                 LEAX   DATA, PCR        POINT TO 'DATA' TO DECODE
                 SWI                     REQUEST SERVICE
                 FCB   OUT2HS            SERVICE CODE BYTE

# OUT4HS
**Convert Word to Hex**
# OUT4HS

**Code:** 5

**Arguments:** Register X points to a word (two bytes) to display in hex.

**Result:** The word is converted to four hex digits and sent to the output handler followed by a blank.

**Example:**

```
OUT4HS  EQU 5              INPUT CODE FOR OUT4HS

        LEAX  DATA, PCR    LOAD 'DATA' ADDRESS TO DECODE
        SWI                REQUEST ASSIST09 SERVICE
        FCB   OUT4HS       SERVICE CODE BYTE
```

# PAUSE
**Program Break and Check**
# PAUSE

**Code:** 11

**Arguments:** None

**Result:** CC = 0  For a normal return.
CC = 1  If a CANCEL was entered during the interim.

**Description:** The PAUSE service should be used whenever a significant amount of processing is done by a program without any external interaction (such as console I/O). Another use of the PAUSE service is for the monitoring of FREEZE or CANCEL requests from the input handler. This allows multi-tasking operating systems to receive control and possibly re-dispatch other programs in a timeslice-like fashion. Testing for FREEZE and CANCEL conditions is performed before return. Return may be after other tasks have had a chance to execute, or after a FREEZE condition is lifted. In a one task system, return is always immediate unless a FREEZE occurs.

# PCRLF     Output to Next Line     # PCRLF

**Code:**     6

**Arguments:** None

**Result:** A carriage return and line feed are sent to the output handler.
     C = 1 if normal output occurred.
     C = 1 if CONTROL-X was entered during output.

**Description:** If a FREEZE occurs (any input character is received), then control is not returned to the user routine until the condition is released. The string is completely sent regardless of any FREEZE or CANCEL events occurring. Padding characters may be sent as described under the OUTCH service.

**Example:**

| | | | |
|---|---|---|---|
| PCRLF | EQU | 6 | INPUT CODE PCRLF |
| | SWI | | REQUEST SERVICE |
| | FCB | PCRLF | SERVICE CODE BYTE |


# PDATA     Send New Line and String     # PDATA

**Code:**     3

**Arguments:** Register X points to an output string terminated with an ASCII EOT ($04).

**Result:** The string is sent to the output handler following a carriage return and line feed.
     CC = 0 if normal output occurred.
     CC = 1 if CONTROL-X was entered during output.

**Description:** The output string may contain embedded carriage returns and line feeds thus allowing several lines of data to be sent with one function call. If a FREEZE occurs (any input character is received), then control is not returned to the user routine until the condition is released. The string is completely sent regardless of any FREEZE or CANCEL events occurring. Padding characters may be sent as described by the OUTCH function.

**Example:**

```
PDATA   EQU   3                  INPUT CODE FOR PDATA

MSGOUT  FCC   'THIS IS A MULTIPLE LINE MESSAGE.'
        FCB   $0A, $0D          LINE FEED, CARRIAGE RETURN
        FCC   'THIS IS THE SECOND LINE.'
        FCB   $04               STRING TERMINATOR

        LEAX  MSGOUT, PCR       LOAD MESSAGE ADDRESS
        SWI                     REQUEST A SERVICE
        FCB   PDATA             SERVICE CODE BYTE
```

# PDATA1

**Send String**

# PDATA1

**Code:**       2

**Arguments:**  Register X points to an output string terminated with an ASCII EOT ($04).

**Result:**     The string is sent to the output handler.
CC = 0 if normal output occurred.
CC = 1 if CONTROL-X was entered during output.

**Description:** The output string may contain embedded carriage returns and line feeds thus allowing several lines of data to be sent with one function call. If a FREEZE occurs (any input character is received), then control is not returned to the user routine until the condition is released. The string is completely sent regardless of any FREEZE or CANCEL events occurring. Padding characters may be sent as described by the OUTCH function.

**Example:**

```
PDATA   EQU   2                  INPUT CODE FOR PDATA1

MSG     FCC   'THIS IS AN OUTPUT STRING'
        FCB   $04               STRING TERMINATOR

        LEAX  MSG, PCR          LOAD 'MSG' STRING ADDRESS
        SWI                     REQUEST A SERVICE
        FCB   PDATA1            SERVICE CODE BYTE
```

# SPACE <span style="float:right">SPACE</span>

<div align="center">Single Space Output</div>

**Code:**     7

**Arguments:**  None

**Result:**    A space is sent to the output handler.

**Description:** Padding characters may be sent as described under the OUTCH service.

**Example:**

```
SPACE    EQU  7          INPUT CODE SPACE
         SWI             REQUEST ASSIST09 SERVICE
         FCB  SPACE      SERVICE CODE BYTE
```

# VCTRSW <span style="float:right">VCTRSW</span>

<div align="center">Vector Swap</div>

**Code:**     9

**Arguments:**  Register A contains the vector swap input code.
Register X contains zero or a replacement value.

**Result:**    Register X contains the previous value for the vector.

**Description:** The vector swap service examines/alters a word entry in the ASSIST09 vector table. This table contains pointers and default values used during monitor processing. The entry is replaced with the value contained in the X register unless it is zero. The codes available are listed in Table B-3.

**Example:**

```
VCTRSW   EQU  9             INPUT CODE VCTRSW
.IRQ     EQU  12            IRQ APPENDAGE SWAP FUNCTION
                            CODE

         LEAX  MYIRQH,PCR   LOAD NEW IRQ HANDLER ADDRESS
         LDA   #.IRQ        LOAD SUBCODE FOR VECTOR SWAP
         SWI                REQUEST SERVICE
         FCB   VCTRSW       SERVICE CODE BYTE
      X NOW HAS THE PREVIOUS APPENDAGE ADDRESS
```

## B.10 VECTOR SWAP SERVICE

The vector swap service allows user modifications of the vector table to be easily installed. Each vector handler, including the one for SWI, performs a validity check on the stack before any other processing. If the stack is not pointing to valid RAM, it is reset to the initial value passed to the MONITR request which fired-up ASSIST09 after RESET. Also, the current register set is printed following a "?" (question mark) and then the command handler is entered. A list of each entry in the vector table is given in Table B-3.

### Table B-3. Vector Table Entries

| Entry | Code | Description |
|-------|------|-------------|
| .AVTBL | 0 | Returns address of vector table |
| .CMDL1 | 2 | Primary command list |
| .RSVD | 4 | Reserved MC6809 interrupt vector appendage |
| .SWI3 | 6 | Software interrupt 3 interrupt vector appendage |
| .SWI2 | 8 | Software interrupt 2 interrupt vector appendage |
| .FIRQ | 10 | Fast interrupt request vector appendage |
| .IRQ | 12 | Interrupt request vector appendage |
| .SWI | 14 | Software interrupt vector appendage |
| .NMI | 16 | Non-maskable interrupt vector appendage |
| .RESET | 18 | Reset interrupt vector appendage |
| .CION | 20 | Input console intiialization routine |
| .CIDTA | 22 | Input data byte from console routine |
| .CIOFF | 24 | Input console shutdown routine |
| .COON | 26 | Output console initialization routine |
| .CODTA | 28 | Output/data byte to console routine |
| .COOFF | 30 | Output console shutdown routine |
| .HSDTA | 32 | High speed display handler routine |
| .BSON | 34 | Punch/load initialization routine |
| .BSDTA | 36 | Punch/load handler routine |
| .BSOFF | 38 | Punch/load shutdown routine |
| .PAUSE | 40 | Processing pause routine |
| .CMDL2 | 44 | Secondary command list |
| .ACIA | 46 | Address of ACIA |
| .PAD | 48 | Character and new line pad counts |
| .ECHO | 50 | Echo flag |
| .PTM | 52 | Programmable timer module address |

The following pages describe the purpose of each entry and the requirements which must be met for a user replaceable value or routine to be successfully substituted.

# .ACIA        ACIA Address        .ACIA

**Code:**      46

**Description:** This entry contains the address of the ACIA used by the default console in-
put and output device handlers. Standard ASSIST09 initialization sets this
value to hexadecimal E008. If this must be altered, then it must be done
before the MONITR startup service is invoked, since that service calls the
.COON and .COIN input and output device initialization routines which in-
itialize the ACIA pointed to by this vector slot.


# .AVTBL      Return Address of Vector Table      .AVTBL

**Code:**      0

**Description:** The address of the vector table is returned with this code. This allows mass
changes to the table without individual calls to the vector swap service.
The code values are identical to the offsets in the vector table. This entry
should never be changed, only examined.

# .BSDTA          Punch/Load Handler Routine          .BSDTA

**Code:**          36

**Description:** This entry contains the address of a routine which performs punch, load, and verify operations. The .BSON routine is always executed before the routine is given control. This routine is given the same parameter list documented for .BSON. The default handler uses the .CODTA routine to punch or the .CIDTA routine to read data in S1/S9 (MIKBUG) format. The function code byte must be examined to determine the type request being handled.

A return code must be given which reflects the final processing disposition:

    Z = 1 Successful completion

or

    Z = 0 Unsuccessful completion.

The .BSOFF routine will be called after this routine is completed.


# .BSOFF          Punch/Load Shutdown Routine          .BSOFF

**Code:**          38

**Description:** This entry points to a subroutine which is designated to terminate device processing for the punch, load, and verify handler .BSDTA. The stack contains a parameter list as documented for the .BSON entry. The default ASSIST09 routine issues DC4 ($14 or stop) and DC3 ($13 or x-off) followed by a one second delay to give the reader/punch time to stop. Also, an internally used flag by the INCHP service routine is cleared to reverse the effect caused by its setting in the .BSON handler. See that description for an explanation of the proper use of this flag.

# .BSON     Punch/Load Initialization Routine     .BSON

**Code:**     34

**Description:** This entry points to a subroutine with the assigned task of turning on the device used for punch, load, and verify processing. The stack contains a parameter list describing which function is requested. The default routine sends an ASCII "reader on" or "punch on" code of DC1 ($11) or DC2 ($12) respectively to the output handler (.CODTA). A flag is also set which disables test for FREEZE conditions during INCHNP processing. This is done so characters are not lost by being interpreted as FREEZE mode indicators. If a user replacement routine also uses the INCHNP service, then it also should set this same byte non-zero and clear it in the .BSOFF routine. The ASSIST09 source listing should be consulted for the location of this byte.

The stack is setup as follows:

      S + 6 = Code byte, VERIFY ( − 1), PUNCH (0), LOAD (1)

      S + 4 = Start address for punch only

      S + 2 = End address for punch, or offset for READ/LOAD

      S + 0 = Return address

# .CIDTA     Input Data Byte from Console Routine     .CIDTA

**Code:**     22

**Description:** This entry determines the console input handler appendage. The responsibility of this routine is to furnish the requested next input character in the A register, if available, and return with a condition code. The INCHP service routine calls this appendage to supply the next character. Also, a "FREEZE" mode routine calls at various times to test for a FREEZE condition or determine if the CANCEL key has been entered. Processing for this appendage must abide by the following conventions:

| | |
|---|---|
| **Input:** | PC → ASSIST09 work page |
| | S → Return address |
| **Output:** | C = 0, A = input character |
| | C = 1 if no input character is yet available |
| **Volatile Registers:** | U, B |

The handler should always pass control back immediately even if no character is yet available. This enables other tasks to do productive work while input is unavailable. The default routine reads an ACIA as explained in Paragraph B.2 Implementation Requirements.

# .CIOFF

**Input Console Shutdown Routine**

**Code:** 24

**Description:** This entry points to a routine which is called to terminate input processing. It is not called by ASSIST09 at any time, but is included for consistency. The default routine merely does an "RTS". The environment is as follows:

    **Input:** None
    **Output:** Input device terminated
    **Volatile Registers:** None


# .CION

**Input Console Initialization Routine**

**Code:** 20

**Description:** This entry is called to initiate the input device. It is called once during the MONITR service which initializes the monitor so the command processor may obtain commands to process. The default handler resets the ACIA used for standard input and output and sets up the following default conditions: 8-bit word length, no parity checking, 2 stop bits, divide-by-16 counter ratio. The effect of an 8-bit word with no parity checking is to accept 7-bit ASCII and ignore the parity bit.

    **Input:** .ACIA Memory address of the ACIA
    **Output:** The output device is initialized
    **Volatile Registers:** A, X

**Code:**     2

**Description:** User supplied command tables may either substitute or replace the ASSIST09 standard tables. The command handler scans two lists, the primary table first followed by the secondary table. The primary table is pointed to by this entry and contains, as a default, the ASSIST09 command table. The secondary table defaults to a null list. A user may insert their own table into either position. If a user list is installed in the secondary table position, then the ASSIST09 list will be searched first. The default ASSIST09 list contains all one character command names. Thus, a user command "PRINT" would be matched if the letters "PR" are typed, but not just a "P" since the system command list would match first. A user may replace the primary system list if desired. A command is chosen on a first match basis comparing only the character(s) entered. This means that two or more commands may have the same initial characters and that if only that much is entered then the first one in the list(s) is chosen.

Each entry in the users command list must have the following format:

| | | | |
|---|---|---|---|
| +0 | FCB | L | Where "L" is the size of the entry including this byte |
| +1 | FCC | '<string>' | Where "<string>" is the command name |
| +N | FDB | EP − * | Where "EP" represents the symbol defining the start of the command routine |

The first byte is an entry length byte and is always three more than the length of the command string (one for the length itself plus two for the routine offset). The command string must contain only ASCII alphanumeric characters, no special characters. An offset to the start of the command routine is used instead of an absolute address so that position-independent programs may contain command tables. The end of the command table is a one byte flag. A − 1 ($FF) specifies that the secondary table is to be searched, or a −2 ($FE) that command list searching is to be terminated. The table represented as the secondary command list must end with −2. The first list must end with a −1 if both lists are to be searched, or a −2 if only one list is to be used.

A command routine is entered with the following registers set:

    DPR→     ASSIST09 page work area.

    S→     A return address to the command processor.

    Z = 1     A carriage return terminated the command name.

    Z = 0     A space delimiter followed the command name.

**Primary Command List
(Continued)**

A command routine is entered after the delimiter following the command name is typed in. This means that a carriage return may be the delimiter entered with the input device resting on the next line. For this reason the Z bit in the condition code is set so the command routine may determine the current position of the input device. The command routine should ensure that the console device is left on a new line before returning to the command handler.

**Secondary Command List**

**Code:**  44

**Description:** This entry points to the second list table. The default is a null list followed by a byte of − 2. A complete explanation of the use for this entry is provided under the description of the .CMDL1 entry.

**Output Data Byte to Console Routine**

**Code:**  28

**Description:** The responsibility of this handler is to send the character in the A register to the output device. The default routine also follows with padding characters as explained in the description of the OUTCH service. If the output device is not ready to accept a character, then the "pause" subroutine should be called repeatedly while this condition lasts. The address of the pause routine is obtained from the .PAUSE entry in the vector table. The character counts for padding are obtained from the .PAD entry in the table. All ASSIST09 output is done with a call to this appendage. This includes punch processing as well. The default routine sends the character to an ACIA as explained in Paragraph B.2 Implementation Requirements. The operating environment is as follows:

> **Input:**  A = Character to send
> DP = ASSIST09 work page
> .PAD = Character and new line padding counts (in vector table)
> .PAUSE = Pause routine (in vector table)
>
> **Output:**  Character sent to the output device
> **Volatile Registers:**  None. All work registers must be restored

# .COOFF          Output Console Shutdown Routine          .COOFF

**Code:**     30

**Description:** This entry addresses the routine to terminate output device processing. ASSIST09 does not call this routine. It is included for completeness. The default routine is an "RTS".

      **Input:**                       DP→ASSIST09 work page
      **Output:**                   The output device is terminated
      **Volatile Registers:**  None


# .COON          Output Console Initialization Routine          .COON

**Code:**     26

**Description:** This entry points to a routine to initialize the standard output device. The default routine initializes an ACIA and is the very same one described under the .CION vector swap definition.

      **Input:**                       .ACIA vector entry for the ACIA address
      **Output:**                   The output device is initialized
      **Volatile Registers:**  A, X

# .ECHO

**.ECHO**       Echo Flag       **.ECHO**

**Code:**      50

**Description:** The first byte of this word is used as a flag for the INCHP service routine to determine the requirement of echoing input received from the input handler. A non-zero value means to echo the input; zero not to echo. The echoing will take place even if user handlers are substituted for the default .CIDTA handler as the INCHP service routine performs the echo.


**.FIRQ**      Fast Interrupt Request Vector Appendage      **.FIRQ**

**Code:**      10

**Description:** The fast interrupt request routine is located via this pointer. The MC6809 addresses hexadecimal FFF6 to locate the handler when processing a FIRQ. The stack and machine status is as defined for the FIRQ interrupt upon entry to this appendage. It should be noted that this routine is "jumped" to with an indirect jump instruction which adds eleven cycles to the interrupt time before the handler actually receives control. The default handler does an immediate "RTI" which, in essence, ignores the interrupt.

# .HSDTA <span>High Speed Display Handler Routine</span> .HSDTA

**Code:**     32

**Description:** This entry is invoked as a subroutine by the DISPLAY command and passed a parameter list containing the "TO" and "FROM" addresses. The from value is rounded down to a 16 byte address boundary. The default routine displays memory in both hexadecimal and ASCII representations, with a title produced on every 128 byte boundary. The purpose for this vector table entry is for easy implementation of a user routine for special purpose handling of a block of data. (The data could, for example, be sent to a high speed printer for later analysis.) The parameters are all passed on the stack. The environment is as follows:

**Input:**                    S + 4 = Start address
                               S + 2 = Stop address
                               S + 0 = Return Address
                               DP→ASSIST09 work page
**Output:**                    Any purpose desired
**Volatile Registers:**  X, D

# .IRQ <span>Interrupt Request Vector Appendage</span> .IRQ

**Code:**     12

**Description:** All interrupt requests are passed to the routine pointed to by this vector. Hexadecimal FFF8 is the MC6809 location where this interrupt vector is fetched. The stack and processor status is that defined for the $\overline{IRQ}$ interrupt upon entry to the handler. Since the routine's address is in the vector table, an indirect jump must be done to invoke it. This adds eleven cycles to the interrupt time before the $\overline{IRQ}$ handler receives control. The default $\overline{IRQ}$ handler prints the registers and enters the ASSIST09 command handler.

# .NMI          Non-Maskable Interrupt Vector Appendage          .NMI

**Code:**      16

**Description:**  This entry points to the non-maskable interrupt handler to receive control whenever the processor branches to the address at hexadecimal FFFC. Since ASSIST09 uses the $\overline{NMI}$ interrupt during trace and breakpoint processing, such commands should not be used if a user handler is in control. This is true unless the user handler has the intelligence to forward control to the default handler if the $\overline{NMI}$ interrupt has not been generated due to user facilities. The $\overline{NMI}$ handler given control will have an eleven cycle overhead as its address must be fetched from the vector table.


# .PAD          Character and New Line Pad Count          .PAD

**Code:**      48

**Description:**  This entry contains the pad count for characters and new lines. The first of the two bytes is the count of nulls for other characters, and the second is the number of nulls ($00) to send out after any line feed is transmitted. The ASCII Escape character ($10) never has nulls sent following it. The default .CODTA handler is responsible for transmitting these nulls. A user handler may or may not use these counts as required.

The "NULLS" command also sets these two bytes with user specified values.

# .PAUSE        Processing Pause Routine        .PAUSE

**Code:**        40

**Description:**  In order to support real-time (also known as multi-tasking) environments
ASSIST09 calls a dead-time routine whenever processing must wait for
some external change of state. An example would be when the OUTCH ser-
vice routine attempts the sending of a character to the ACIA through the
default .CODTA handler and the ACIA status registers shows that it cannot
yet be accepted. The default dead-time routine resides in a reserved four
byte area which contains the single instruction, "RTS". The .PAUSE vector
entry points to this routine after standard initialization. This pointer may be
changed to point to a user routine which dispatches other programs so that
the MC6809 may be utilized more efficiently. Another example of use would
be to increment a counter so that dead-time cycle counts may be ac-
cumulated for statistical or debugging purposes. The reason for the four
byte reserved area (which exists in the ASSIST09 work page) is so other
code may be overlayed without the need for another space in the address
map to be assigned. For example, a master monitor may be using a memory
management unit to assign a complete 64K block of memory to ASSIST09
and the programs being executed/tested under ASSIST09 control. The
master monitor wishes, or course, to be reentered when any "dead time"
occurs, so it overlays the default routine ("RTS") with its own "SWI". Since
the master monitor would be "front ending" all "SWI's" anyway, it knows
when a "pause" call is being performed and can redispatch other systems
on a time-slice basis.

All registers must be transparent across the pause handler. Along with
selected points in ASSIST09 user service processing, there is a special ser-
vice call specifically for user programs to invoke the pause routine. It may
be suggested that if no services are being requested for a given time period
(say 10 ms) user programs should call the .PAUSE service routine so that
fair-task dispatching can be guaranteed.

# .PTM        Programmable Timer Module Address        .PTM

**Code:**        53

**Description:**  This entry contains the address of the MC6840 programmable timer module
(PTM). Alteration of this slot should occur before the MONITR startup ser-
vice is called as explained in Paragraph B.4 Initialization. If no PTM is
available, then the address should be changed to a zero so that no in-
itialization attempt will take place. Note that if a zero is supplied, ASSIST09
Breakpoint and Trace commands should not be issued.

# .RESET

**Reset Interrupt Vector Appendage**

# .RESET

**Code:** 18

**Description:** This entry returns the address of the RESET routine which initializes ASSIST09. Changing it has no effect, but it is included in the vector table in case a user program wishes to determine where the ASSIST09 restart code resides. For example, if ASSIST09 resides in the memory map such that it does not control the MC6809 hardware vectors, a user routine may wish to start it up and thus need to obtain the standard RESET vector code address. The ASSIST09 reset code assigns the default in the work page, calls the vector build subroutine, and then starts ASSIST09 proper with the MONITR service call.

# .RSVD

**Reserved MC6809 Interrupt Vector Appendage**

# .RSVD

**Code:** 4

**Description:** This is a pointer to the reserved interrupt vector routine addressed at hexadecimal FFF0. This MC6809 hardware vector is not defined as yet. The default routine setup by ASSIST09 will cause a register display and entrance to the command handler.

# .SWI
### Softare Interrupt Vector Appendage
# .SWI

**Code:**     14

**Description:** This vector entry contains the address of the Software Interrupt routine. Normally, ASSIST09 handles these interrupts to provide services for user programs. If a user handler is in place, however, these facilities cannot be used unless the user routine "passes on" such requests to the ASSIST09 default handler. This is easy to do, since the vector swap function passes back the address of the default handler when the switch is made by the user. This "front ending" allows a user routine to examine all serivce calls, or alter/replace/extend them to his requirements. Of course, the registers must be transparent across the transfer of control from the user to the standard handler. A "JMP" instruction branches directly to the routine pointed to by this vector entry when a SWI occurs. Therefore, the environment is that as defined for the "SWI" interrupt.

# .SWI2
### Software Interrupt 2 Vector Appendage
# .SWI2

**Code:**     8

**Description:** This entry contains a pointer to the SWI2 handler entered whenever that instruction is executed. The status of the stack and machine are those defined for the SWI2 interrupt which has its interrupt vector address at FFF4 hexadecimal. The default handler prints the registers and enters the ASSIST09 command handler.

# .SWI3      Software Interrupt 3 Vector Appendage      .SWI3

**Code:**      6

**Description:** This entry contains a pointer to the SWI3 handler entered whenever that instruction is executed. The status of the stack and machine are those defined for the SWI3 interurpt which has its interrupt vector address located at hexadecimal FFF2. The default handler prints the registers and enters the ASSIST09 command handler.