Monitor System

The monitor system is used to assemble and debug programs. Whenever the monitor is ready to accept a command, the character ">" is printed in the first column of the input device.

Execution Parameters

The monitor is loaded via one of two commands under DOS.

*GO CASM
*GO DASM

The CASM file contains a version of the assembler which stores object code directly into memory whereas the DASM program produces a Hex Format object file for loading under the LOADER program. The user must take care in using CASM to insure that generated code does not overlay the assembler, its symbol table, or its buffer areas.

Monitor Commands

Monitor commands are 4 characters followed by the parameters necessary for the particular command. The entry of an unrecognizable command will cause the message "ILLEGAL COMMAND" to print.

If the debug commands are used to test out a program module, then the user must insure that the program being tested resides in memory somewhere above the monitor.

The following section describes the various monitor commands and their actions.

DUMP XXXX,YYYY — The contents of memory between the locations specified by XXXX and YYYY are printed 16 bytes per line. The last byte printed in each line corresponds to the memory address ending in 'F' (hex). If YYYY is not specified then only one byte is printed.

PATCH XXXX

The contents of memory from XXXX are patched according to the data on the following line. Each byte is placed in the next successive byte in memory. A carriage return terminates patch mode.

Example:

```
PATCH 124
15 27 16 32
DUMP 124,127
15 27 16 32
```

GOTO XXXX

Control is transferred to the address given by the GOTO command. No changes are made to insure that a proper address is specified.

MOVE XXXX,YYYY

This command will move the 256 byte block starting at XXXX to location YYYY through YYYY+255.

BRKP (XXXX)

A breakpoint is set at location XXXX or, if no parameter exists, all breakpoints are cleared If a memory location is specified as a breakpoint then an attempt to execute the instruction at that address will result in the message "XXXX BREAK" being printed. The continue command will proceed from a breakpoint with the environment intact. The breakpoint is cleared when it is hit. Up to 8 breaks may be specified at a time. The DUMP and PATCH commands can be used during a BREAK in order to determine the contents of memory. The environment may be examined by dumping memory location 1000 - 1007 which contain:

```
1000 A register
1001 state flags
1002 C register
1003 B register
1004 E register
1005 D register
1006 L register
1007 H register
```

CONT

The Continue command will proceed from a breakpoint, executing the instruction at which the breakpoint was set. A GOTO command may be used to return to a location other than the one at which the BREAK was detected.

DB,DW - Data definitions

The DB statement is used to define storage on the byte level while the DW statement reserves double byte storage. Literal strings, which are enclosed in single quote marks ('), always generate one byte of data per character regardless of the mode of the mnemonic. Multiple values may be specified on a line each separated by a comma. When the data definition statement is printed at assembly time, each field generated will be printed on a separate print line. Fields within a data definition statement may contain expressions which utilize variable names, constants (hex or decimal) and literals with combinations of the + and - operators. Literals used within arithmetic expressions must not exceed the byte length of the data definition statement.

Examples

```
DB  1,SYM,OFH,XV+3,'L'-1,'LITERAL'
DW  6,VAR,16H,XY-6,'AA'+1,'MESSAGE'
```

Macro Command

In order to simplify the creation of user programs a macro facility has been incorporated into the assembler. If the user finds a sequence of codes being used repeatedly, those codes could be generated by the use of a single statement rather than recopying the set each time. The assembler can optionally list or not list the expanded set of generated instructions within the assembly listing, the latter case saving on output print time. Expanded macros are printed on the assembly listing with a leading '+' character in the line. This is to remind the user that these lines do not exist in the actual source file.

Macro Format

Macro definition:

```
MNAME  MACRO  PARM1,PARM2,...,PARMn
       _____  _______
       _____  _______  }-- Body of Macro
       _____  _______
       MEND
```

Macro Call:

```
       MNAME    CALL-PARMs
```

where: MNAME: is the name under which the macro will be called. This name must not conflict with any variable names used within the program.

ASMF /NAME/	The 4 characters specified in the name field are used to specify a file name "NAME".ASM which is opened for assembler source input. A message will appear reporting the status of the attempted open.
OBJF /NAME/	The 4 characters specified in the name field are used to specify a filename "NAME".OBJ which is opened for assembler object output. This command is invalid for CASM.
CASM XXXX,(YYYY)	The input file is assembled with its base stating address as XXXX or the end of the mo monitor if the parameter is not specified. In the case of CASM, the output code can be specified to be stored at location YYYY and later moved to XXXX before execution (MOVE to GOTO commands).

Assembler

The macro-assembler is a program which reads symbolic input files containing assembly language (source) instructions, macro instructions and program directives, and generates either machine language code directly into memory (CASM) or INTEL 8080 hex format code into a disk file (DASM). Hex formatted data can be loaded into memory at a later time through use of the LOADER program.

Format

The assembler accepts free form input files or the user may optionally choose to use the tab capability to put the various fields used by the assembler into fixed columns. The format of an assembly statement is:

```
LABEL  OP  OPRND   COMMENTS
```

where: LABEL always starts in column 1 and is a 1-5 character sequence, optionally terminated by a ':' character.

OP is a standard INTEL mnemonic code or any of the psuedo ops: ORG, EQU, END, DS, DB, DW, MACRO or MEND. The OP field starts at least one column after the label field or in column 2 if the label field is not used.

OPRND

The operand field starts at least one column after the OP code field and contains the parameters specifying the registers, memory locations, immediate values, or data elements utilized by the various OP codes. The operand field may contain any arithmetic expressions consisting of combinations of variables, constants (hex and decimal), and literal strings (defined by the ' mark), along with either the + or - operators. The special symbol '$' refers to the current value of the location counter (address of the current location), and may also be used in expressions. For example, in order to define a variable whose value is the length of a given message, the following code may appear:

```
MSG    DB    'MESSAGE ONE'
MSGSZ  EQU   $-MSG
```

COMMENTS

The comments field is preceded by a ";" character. Comments may follow any source statement or they may stand alone by placing the ";" character into column one.

NOTE: An input line containing the tab character "†" would appear as follows:

```
LABEL:  OP  OPRND  COMMENTS
```

When it is listed by the assembler or text editor the fields will appear in columns: 1, 8, 14 and 25.

Psuedo Ops (Seembler Directives)

EQU

The equate psuedo is use to set a variable equal to the current contents of another variable. It may also be used to create a variable whose value is equivalent to a constant

For example:
```
AA EQU  BB
AD EQU 1012H
```

END

This is used to terminate the assembler. The lst of the two passes is completed and a 2nd pass over the source input file is done. Upon detection of the "end" after the final pass, disk files are closed and the current symbol table is printed if the option had been requested.

ORG — The set program origin directive allows the user to specify the starting memory location of the next instruction which will be generated by the assembler.

```
Example:  ORG  OFB3H
          ORG  146
          ORG  AXY+B-3
```

DS — The define storage psuedo op is used to reserve an area of memory. This area is not affected during program loading and it is therefore up to the user to initialize all DS areas.

```
Example:  TBLA  DS  1000
          TBLB  DS  TBLBX
```

Error Codes

The following error codes will appear in column one of the assembly output upon the detection of an assembly error.

A -- Argument Error
D -- Duplicate Label
L -- Label Error
M -- Missing Label
O -- Opcode Invalid
R -- Register Specification Error
S -- Syntax Unrecongnizable
U -- Undefined Symbol
V -- Value error - Argument field value overflow

Assembly Example

The following sample assembly program will convert a buffer of data coded in one character set (say ASCII) into a new buffer under a different character set (say EBCDIC). The program is not coded efficiently, it is presented in order to show the various assembler capabilities.

PARMs: are optional parameters which can be used within the macro. Upon calling a macro, the parameters passed in the call are substituted into the parameters used in the actual macro.

Example:

The following is an example of the creation and use of a macro to load a register pair from a memory location specified by the H and L registers.

```
LRPM  MACRO   RP
      MOV     A,M
      MOV     RP+1,A
      INX     H
      MOV     A,M
      MOV     RP,A
      INX     H
      MEND

    PROGRAM

      LRPM    B
      LRPM    D
```

Macros may also be used to change the names of certain instructions in oder to make programs more readable. For example:

```
JL  MACRO  ADR   ;JUMP IF LESS THAN
    JC     ADR
    MEND   ADR
```

The mnemonic JL might be more meaningful following a compare instruction than the condition code test statement.

```
;
;          CONVERT FROM ASCII TO EBCDIC
;
;
;          LOCATIONS:  ABUF = ADDRESS OF ASCII BUFFER
;                      ABUF = ADDRESS OF EBCDIC BUFFER
;
;          THIS SUBROUTINE RETURNS TO THE CALLER
;          UPON THE DETECTION OF AN ASCII
;          CARRIAGE RETURN.  THE SEQUENCE 'END'
;          WILL ALSO STOP THE SUBROUTINE.
;          THE CARRY IS SET UPON RETURN.
;
;
CNVRT:    LRP     D,ABUF      ;(D,E) = ADR OF ASCII BUFR
          XCHG                ;SAVE (D,E) IN (H,L)
          LRP     D,EBUF      ;(D,E) = ADR OF EBCDIC BUFR
LOOP:     MOV     A,M         ;GET ASCII CHAR
          CPI     ASCR        ;IS IT A CARRIAGE RETURN?
          JE      BYE         ;YES - DONE
          CPI     END         ;IS IT AN 'E'?
          JE      CHKND       ;YES - CHECK IF 'END'
PROC:     XCHG                ;NO - (H,L) = STORAGE LOCATION
          PUSH    H           ;SAVE ADR
          LXI     H,CNVTB     ;GET CONVERSION TBL ADR
          LXI     B,0
          MOV     C,A         ;MOVE CHR TO (B,C) PAIR
          DAD     B           ;POINT TO NEW CHR
          MOV     A,M         ;GET NEW CHR
          POP     H           ;GET STORAGE ADR OFF STACK
          MOV     M,A         ;STORE NEW CHR
          INX     H           ;POINT TO NXT STORAGE ADR
          XCHG                ;GET ASCII BUFR ADR IN (H,L)
          INX     H           ;POINT TO NXT CHR
          JMP     LOOP        ;PROCESS NXT

;
BYE:      STC                 ;SET CARRY
          RET                 ;RETURN TO CALLER

;
CHKND:    INX     H
          MOV     A,M         ;GET NXT CHR
          CPI     END+1       ;IS IT 'N'?
          JE      CHK2
PROCE:    DCX     H           ;NO - PROCESS THE 'E'
          MVI     A,'E'
          JMP     PROC

;
CHK2:     INX     H
          MOV     A,M         ;GET NXT
          CPI     END+2       ;IS IT 'D'?
          JE      BYE         ;YES - DONE
          DCX     H           ;ADJ PTR
          JMP     PROCE       ;NO PROCESS THE 'E'
```

```
#1        #8        #14           #25

          ORG       100 H         ;START AT HEX 100

;
LRP:      MACRO     RP,ADR        ;LOAD REG PAIR FROM ADR & ADR+1
          PUSH      H
          LXI       H,ADR
          MOV       RP+1,M
          INX       H
          MOV       RP,M
          POP       H
          MEND

;
JE        MACRO     ADR
          JZ        ADR
          MEND

;
ASCR:     DB        13
END:      DB        'END'
```

```
;
ABUF:     DS      2
ABUF:     DS      2
;
;         EBCDIC CONVERSION TABLE
;
CNVTB:    DS      192         ;SKIP OVER NON-ALPHAS
          DB      Ø,'ABCDEFGHI'
          DS      6
          DB      Ø,'JKLMNOPQR'
          DS      6
          DW      Ø,'STUVWXYZ'
          DS      6
          DB      'Ø123456789'
          DS      6
;
          END
```

Text Editor

The text editing program EDITOR is used to create and maintain data files on disk. The editor reduces textual storage by allowing the user to store a tabulation character (↓) which is converted to the appropriate number of spaces by all software in the system. (Tab stops are permanently set at columns 8, 14 and 25). Storage is further reduced by eliminating the storage requirements for end of line sequence replacing it with a flag on the last character in each data line.

The editor takes an input file and executes the commands specified by the user in order to produce the desired output file. If the input filename is not specified then the editor will only allow for the creation of a new data file (since no commands which would edit source input are meaningful). If an output file is not specified then only the end editing with listing command is acceptable since there is no purpose to editing the source file.

Execution Specifications

In order to execute the editor the following DOS commands are executed. (User commands are underlined).

```
*GO BASIC
READY
LOAD EDITOR
READY
RUN
```

The editor will startup by typing:

```
TEXT EDITOR - V1.0  10/77

INPUT FILENAME? _____
OUTPUT FILENAME?_____
```

The user enters the names of the disk files being edited or a carriage return if the particular file is not being used. The editor will respond with:

```
FILE  XXXXXXXX BEING OPENED.
FILE  YYYYYYYY BEING OPENED.

CMD? _____
```

At this time editing commands may be entered.

Editing Commands

An editing command consists of a single character. Any multiple character sequence is assumed to be a line of textual information. The editor maintains a "current line" of source data. Editing commands effect this current line points.

Command Code	Action
S (SKIP)	The editor asks the user for the number of lines to skip "#=?" and advances the current line pointer the specified number of lines. All lines skipped are written to the output file and are optional printed as they are bypassed. (See L. command).
D (DELETE)	The editor asks for the number of lines to be deleted "#=? The number of lines specified (including the current line) are deleted from the source file. Deleted lines are listed with an '*' preceding the line image.
F (FIND)	The user is asked to enter the desired string which is to be located "STR=?". All lines from the current line to the line containing the string are copied to the output file. Skipped lines are printed as controlled by the list mode.
L (LIST)	Skipped and delted lines are listed as they are processed. List mode is on by default when the editor is started.
N (NO LIST)	Only the current line is printed following the execution of a command.
E (END)	The editor is terminated. All lines from the current line up to the end of the source file are copied to the output file. Listing is controlled by the current listing indicators. If an output file is not specified the end command will cause the source file to be listed only.
Multiple Char Seq (INSERT)	The current line is written to the output file and the inputted string becomes the current line.
Carriage Return (SKIP-1)	The next line from the input file becomes the current line. The previous current line is written to the output file.

NOTE: Tab characters detected in the source file will cause the lines to be displayed with the appropriate spacing.

Upon termination of the editor, the number of sectors of data written to disk is printed so that the user can determined the current end of file.

Object File Sizes

This program is designed to calculate the size of the object code file in sectors.

Execution Parameters

The program is executed as follows:

```
*GO BASIC
READY
LOAD OBJSIZE
READY
RUN
OBJECT FILE SIZER - V1.0  10/77

OBJECT FILENAME (XXXX)?_____
OPENING FILE XXXX.OBJ

APPROXIMATE FILE SIZE = XXX SECTORS
NUMBER OF RECORDS = XX
```

Assembly Time Calculator

This program is designed to calculate the approximate printing time if the user is producing an assembly listing from the assembler.

Execution Parameters

The program is executed as follows:

```
*GO BASIC
READY
LOAD ASMTIME
READY
RUN

ASSEMBLY TIME CALCULATOR - V1.0  10/77

PRINTER SPEED  (CPS)?_______
FILENAME (XXXX)?     _______

FILE XXXX.ASM BEING OPENED.


NUMBER OF SOURCE LINES = XXXX LINES.
AVERAGE NUMBER OF CHARS/LINE = XX.X
MAXIMUM LINE SIZE = XX CHARS.
NUMBER OF OUTPUT CHARS = XXXXXX CHARS.
APPROXIMATE PRINTOUT TIME = XXX.X MINUTES
```

Object File Loader

The object file loader is designed to load INTEL Hex Format object code files from disk into memory.

Execution Parameters

The loader program is executed as follows:

```
*GO LOADER

OBJECT FILE LOADER -- V1.0  10/77

FILENAME (XXXX)?_____

FILE  XXXX.OBJ BEING OPENED
```

Checksum and illegal character error messages will be printed if the object file is in error. The block number (block starting address) will be printed to indicate the bad block.

NORTHSTAR DISK BASED MACRO ASSEMBLER

A PRODUCT OF INTERSYSTEMS SOFTWARE

VI.0A

Revision List

9/77	Original	Not Released
10/77	VI.0A	Release I

Memory Utilization - Standard Software (V1.0A)

To be supplied

Device Assignments

To be supplied

Special Commands

By using the CASM and DASM commands, an assembly listing and symbol table printout will be produced. Macros will not be expanded (printed in their entirety upon each call). The user may modify the assembly command to produce a specialized listing. This may be done by tagging an "option" character onto the CASM or DASM statement. The option characters are:

E	Only print lines with errors
L	Only produce an assembly listing
S	Only print a symbol table
X	Print an assembly listing with expanded macros

This document is an update to the original manual and describes:

- New Programs
- Assembler options
- Customizing I/O
- Fast Editor
- Debug modifications
- VI.0A Memory Map
- Original Documentation Errata

It also further explains various system capabilities.

NORTHSTAR DISK ASSEMBLER ADDITIONAL DOCUMENTATION

Monitor Start/Restart/Typing Errors

The starting address of the monitor is location Ø. The system will initialize its I/O by utilizing the standard or user defined initialization routine. After clearing breakpoints the monitor will wait for a command.

In order to restart the monitor without clearing breakpoints, the user can branch to location 3.

NOTE: If the program branches to location Ø due to a fault, the monitor will restart destroying all set breakpoints. The user must restore the breakpointed instructions before proceeding.

The "←" character is used to delete the last character which was input to the monitor or editor. The "@" character will delete the entire input line on which it appears.

Loader Program

The object file loader program has been rewritten in assembly language in order to minimize its memory requirements. The standard loader loads into locations 3C00-3EFF and locations 3F00-3FFF are used as a disk input buffer. The loader programs transfers object code from a disk file into memory and will branch back to OS upon loading, enter the users program directly, or enter the monitor debug routines.

The loader is executed by entering.

*GO LOADER

The loader will print a startup message and then wait for the 4 character filename of the object code file (XXXX.OBJ).

After loading the code into memory, if the user did not specify a starting address (as specified below) then the disk operating system is rebooted into memory by executing the PROM at location E900.

DASM/DASMX

The DASM file contains a disk assembler which produces INTEL loader format code.

DASMX contains a disk assembler which produces the compressed OBJECT FORMAT which is compatable with the LOADER program. This format for the object code is: byte count of record (1-255 bytes), 16 bit memory address, followed by the data bytes. A byte count of zero indicates end of file (EOF).

DASMX also contains the text editor.

NOTE: If an object file is not specified, then no object code is produced by the assembler.

EDITOR

The VI.0A editor has been incorporated into the DASMX package. The VI.0A editor is written in assembly language and executes many times faster than the VI.0 basic language text editor. It is invoked via the command:

EDIT or EDITN

The former command specifies that tab characters should cause proper formatting of all printed lines during editing while the latter command converts the tab character to a space in order to reduce print time.

The editor will then ask for the name of an input and output file. One or both may be specified. If the files are properly opened then the editor will prompt the user for editing commands by typing:

ED>

In order to branch to a specific memory location upon loading, the following code must appear in the users program:

```
ORG     3EFEH
DW      ADDRESS
```

where: ADDRESS is the desired execution address.

In order to utilize the memory dump and modification routines in the monitor as well as the program breakpoint feature, then the ADDRESS to be used for entry is Ø. (initiation of monitor) or 3 (continuation of monitor). The former address is used if the monitor had been load via an *LF command, while the latter is used if a *GO command loaded the monitor. For example:

Program with a start address of 3	address of Ø
*GO DASMX (or DASM OR CASM)	*LF DASMXØ
GO TO 2028 (restart DOS)	
*GO LOADER	*GO LOADER
OBJ LOADER --VI.0A -- 10/77	OBJ LOADER --V1 1Ø
?FIL1	?FIL1
> (automatic monitor start)	>

NEW PROGRAMS

Version 1.0A is released with four new basic programs.

COMPRESS - This program converts a file which is a core image of a "file" produced by The Self Contained Monitor System (which is provided with IMSAI 8080 microprocessing systems) into a file which is compatible with the Northstar Disk Based Assembler/Monitor Software.

FDUMP - This program will dump an image of a disk file in decimal and ASCII. The user enters a filename and EOF character when prompted. If the EOF character is unknown, then entering a number greater than 255 will cause all sectors up to the physical end of file to be printed.

All editing commands specified in the original documentation are permitted. The additional command specified has been added.

Command	Action
T	The editor asks the user to specify a character to be used as a tabulation character. If no character is specifie the default character "!" is utilized.

Whenever the specified tabulation character is found in a line being inserted into file, then it is converted into the internal representation for a tab "!". The "↑" used in the original implementation is no longer valid.

The delete command will list all deleted lines. The last line listed will be the new current line. If list mode is off, only the new current line will be shown.

Sector information will not be printed at the end of editing. FSIZER may be executed to determine the file size.

I/O Interface

The standard I/O routines which are delivered with the system are oriented towards an IMSAI 8080 microprocessor. If these routines are incompatable with a particular configuration, the steps in the next section describe how to add a customized I/O routine.

Standard I/O

Programmed Input Switch	Result
7 up	Input from device # 3, echo on #3
down	Input from device #5, echo on #3 and #5.
6 up	Output to device #3.
down	Output to device #3, and # 5.

FSIZER - The sizer program determines the number of sectors used within a file. This program is useful in determining the minimum size file needed to store a given data set.

FCOPY - The copy program will copy one file to another file. The file size of the new file must be large enough to hold the actual data contents of the source file, it does not have to be the same physical size of the source file.

The execution of the FSIZER program followed by the creation of a minimum length file, followed by the execution of FCOPY and subsequent deletion of the original file, will keep disk space utilization to a minimum.

Errata

This section corrects documentation errors in the original users manual.

DUMP - The proper format of the dump memory command is:

```
DMP XXXX YYYY
```

Where: XXXX - is a 1-4 hexadecimal digit start address

YYYY - is a 1-4 digit optional end address.

PATCH - The patch memory command is formatted as:

```
MOD XXXX
YY YY YY YY/
```

Where: XXXX - is a 1-4 hexadecimal digit start address

YY - is a 1-2 hexadecimal digit memory patch. The "/" character terminates patching. Carriage returns are ignored during processing.

MOVE - The proper format of the move command is:

```
MOVE XXXX YYYY
```

Where: XXXX - is the "from" address
YYYY - is the "to" address
One 256 byte block is moved

BRKP - Locations 1000 - 100B are set up at the time of a breakpoint as follows:

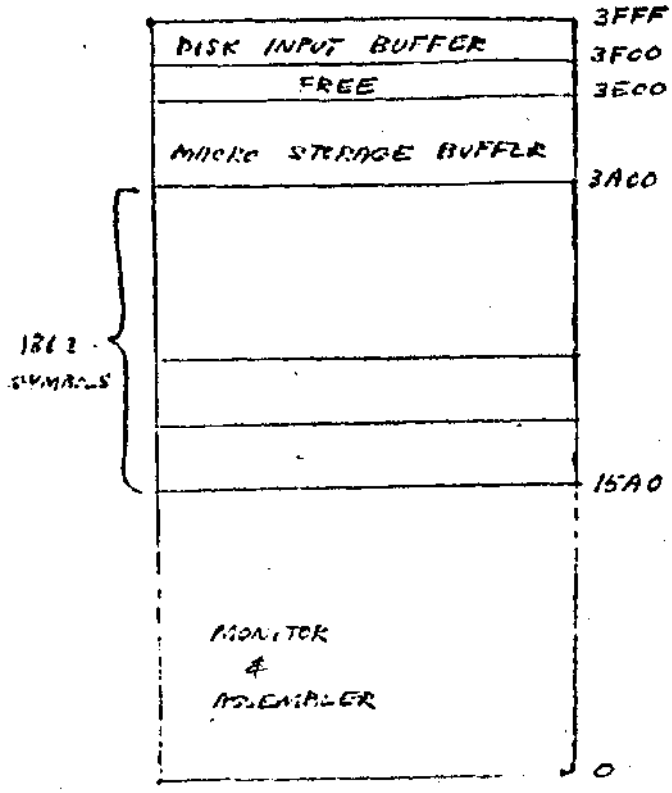
```
1000   PSW
1001   A Reg
1002   C Reg
1003   B Reg
1004   E Reg
1005   D Reg
1006  SP (Low)
1007  SP (High)
1008   L
1009   H
100A  PC (Low)
100B  PC (High)
```

CONT - To Proceed from a given address following a breakpoint the command:

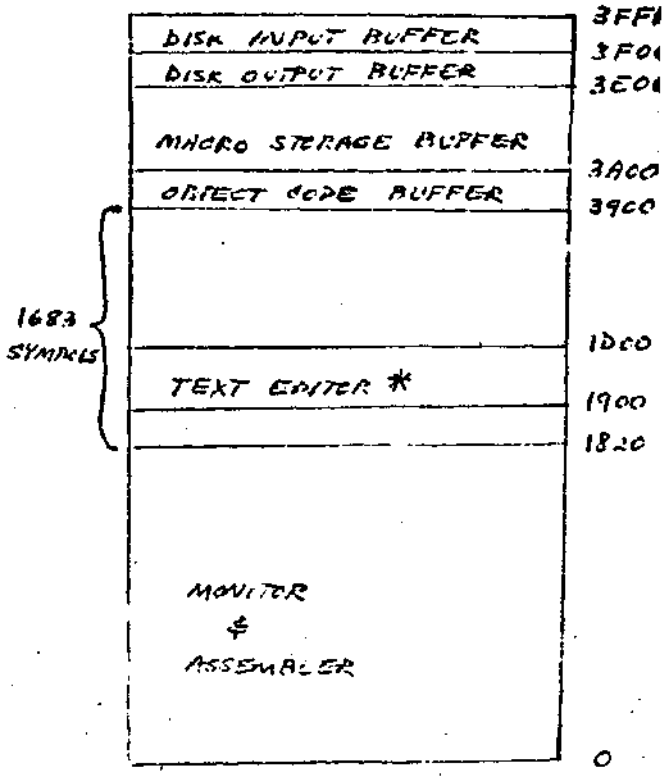
```
CONT XXXX
```

is used. The GOTO command does not restore the environment before entry to the program.

MEMORY MAP



CASM



DASM

* DASMX ONLY

Customized I/O

In order to add an I/O routine for your particular configuration the following steps are taken:

The I/O routines may be placed at locations IIFI - I24C and I256 - 1271 overlaying the standard I/O routines. The user may also choose to place the routines immediately above the end of the monitor (start of the symbol table). If the latter case is chosen, the last address of the I/O routines plus one becomes the new start of the symbol table and the new address must be specified to the monitor as described below.

The addresses of the three required I/O routines (I/O initialization, get a byte, put a byte) must also be patched into the system. The new monitor may then be written to disk using the procedures described below.

The addresses of the customized I/O routines must be patched into the following locations:

```
F73  C3  LL  HH   JMP   INIT
F76  C3  LL  HH   JMP   INPUT   ;  Put Byte in 'A' REG
F79  C3  LL  HH   JMP  OUTPT    ;  Write Byte in 'B' Reg
```

If the address of the symbol table changes:

```
B49  LL  HH      DW   SYMBOL - TABLE - START - ADDRESS
                      STANDARD:  CASM  15AO  (AO 15)
                                 DASM (X) 1820  (20 18)
```

After making the necessary memory modification, the customized package may be written to disk via the command sequence:

```
>GOTO 2028
*CR FILENAME SIZE*
*SF FILENAME Ø
*TY FILENAME 1  Ø
*GO FILENAME
```

The size must be large enough to hold all in core coding. Remember, DASMX contains the text editor up to location IDOO.

ASSEMBLY ERRORS

P - Phase Error

This new error code indicates that a failure occurred in addressing during pass 2 of the assembler. The addresses of labels in pass 2 should match those of pass 1 or the "P" error will appear. An invalid DB or DW psuedo op can cause a phase error.

FILENAMES

The following programs have been renamed in the VI.0A release.

Old Name	New Name
ASMTIME	ATIME
OBJSIZE	OSIZER

ASSEMBLER COMMANDS

The commands CASM and DASM have been removed and have been replaced as follows:

NOTE: The assembler reboots the DOS upon completion of the assembly phase.

ASMT 0 XXXX (YYYY) - Assemble a source file and expand tab characters properly if a listing is produced. The base address of the program is XXXX. If the core version of the assembler is used, then an optional YYYY parameter specifies where to store the generated coding if XXXX is not available. The file CASM will store output code into memory. DASM and DASMX will store object code in a disk file in loader format (see below). Optional assembly options{0} S,E,X, and L may be specified in the command.

ASMN {0} XXXX (YYYY) - Same as ASMT but tab characters will cause a single space to be printed thereby reducing print time.

NORTHSTAR MINI FLOPPY USERS

Now available in source form on mini-floppy diskette:

CORE IMAGE LOADER*

CORE IMAGE SAVER*

- **Allows the user to write and read core images from disk. Gets around the problem of loading modules which overlay the DOS image in memory.**
- **Available on diskette in source form compatable with the Minifloppy Disk-Based Macro-Assembler***

Supplied on diskette: $15.00
Written to users diskette: $10.50

* PRODUCTS OF: INTERSYSTEMS SOFTWARE INC.
(201) 871-4085