HDS

HYBRID DEVELOPMENT SYSTEM

FOR

NORTH STAR SYSTEMS

INCLUDING:	INTERACTIVE ASSEMBLER/EDITOR EXTENSIONS TO NORTH STAR BASIC
FEATURING:	CO-RESIDENT ASSEMBLER/EDITOR FULL Z-80 CAPABILITY OPERATIONAL ON Z-80 OR 8080 MACHINES INTEL MNEMONICS AUTOMATIC FILE HANDLING FLEXIBLE INTERFACE TO ASSEMBLY ROUTINES PARAMETERS PASSED BY ADDRESS OR VALUE
REQUIRES:	RAM AT LOW MEMORY MINIMUM 24K SYSTEM

READY TO RUN ON DISKETTE

COMPLETE DOCUMENTATION

FULL USER SUPPORT

$40

Allen Ashley
395 Sierra Madre Villa
Pasadena, CA 91107

(213) 793-5748

HYBRID DEVELOPMENT SYSTEM

The HDS development system enhances communication between North Star BASIC and assembly language routines. Now critical portions of your BASIC programs may be executed in assembly language while retaining ease of program development. A BASIC program in which portions are performed by assembly language routines is referred to as a hybrid program. Such hybrid programs may be attractive for several reasons:

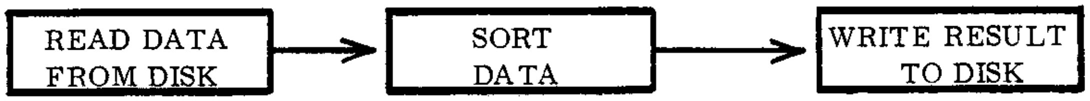
1. In speed-critical applications, a hybrid program may be very much faster than BASIC.
2. Proprietary program segments can be coded in assembly language for protection.
3. Hybrid programs ease the transition from BASIC to assembly language programming.
4. Certain operations are more easily performed in assembly language than BASIC.

HDS includes modifications to North Star BASIC to enable hybrid program development as well as a co-resident assembler/editor -- ASMB -- which includes all the features necessary for the creation, modification and disk storage of assembly language source files for Z-80 or 8080 computers. ASMB is a very fast assembler which, together with the co-resident editor, is structured for a very rapid assemble/execute/modify cycle. The instruction set of ASMB is designed to be a logical and syntactical extension of the widely familiar INTEL instruction set for the 8080. Users already familiar with 8080 assembly language will readily acquire the extended instruction set of the Z-80 processor.

Modifications are provided for North Star BASIC to enable variables to be accessed by address as well as value. The BASIC CALL operation has been modified to allow an unlimited number of parameters to be passed to the assembly language routine. Program variables and strings may be passed to an assembly language routine, modified, and passed back to BASIC.

HDS is an exceptionally powerful development system combining the execution speed advantage of assembly language while retaining the ease of BASIC program development.

It has been established that the major portion of execution time is accounted for by a rather small portion of typical computer programs. It follows that significant reduction in execution time can be achieved by coding critical program segments in assembly language. A program which illustrates the hybrid potential is diagrammed below.



The execution times of the first and last of these blocks, disk read/write, are not significantly influenced by programming. The central block may be performed by a simple exchange sort which might be coded in BASIC as

```
100  REM N IS THE NUMBER OF POINTS
110  FOR J = 1 TO N-1
120  FOR K = J+1 TO N
130  IF A(K) =>A(J) THEN 150
140  A1 = A(K): A(K) = A(J): A(J) = A1
150  NEXT K
160  NEXT J
```

The outer loop of this program (110-160) is performed some N times, for which the inner loop (120-150) requires an average of N/2 repetitions. Line 130 is executed some N*N/2 times. For any reasonable value of N this program will consume most of its time executing lines 130 and 140. If the inner loop of this program were coded in assembly language the overall execution time would be dramatically reduced. Such a hybrid program using the simple exchange sort algorithm could compete favorably with a more elegant sort coded entirely in BASIC. Typical program segments run 50 times faster in assembly language than BASIC. Owing to the unavoidable speed dilution resulting from disk operations, the overall speed improvement factor would be expected to be on the order of 20.

Such hybrid programs lie outside the original intent of BASIC, and existing interpreters do not provide adequate facilities for communication between BASIC and assembly language.

1. BRINGING UP HDS

1. Write protect the HDS diskette before attempting to use it.

2. Make a working copy of the HDS diskette using the RD and WR commands of the DOS.

3. Store the original diskette as a master backup copy.

4. Read the entire documentation.

2. INTERFACING HDS TO NORTH STAR DOS

The components of HDS utilize the standard entry points to the North Star Disk Operating System:

DOS + ØDH	Character out
DOS + 1ØH	Character in
DOS + 16H	Control/C
DOS + 28H	Warm start entry

File names communicated to HDS are terminated by a carriage return. The file name may be suffixed by an optional unit number. The unit number, if present, must be separated from the file name by a comma. File names not suffixed by a unit number default to drive 1.

Components of HDS which generate disk output request an output file name. The output file must be found in the directory. HDS will examine the size of the output file. A zero-length output file is treated as a new file and HDS will update the directory entry to reflect the completed disk operations.

If a required file is not found in the directory, HDS issues a "?" prompt and awaits re-entry of the file name. HDS will automatically size the output file if the user creates (under the DOS) an output file of length Ø before entering the program. As an example:

```
CR OFILE Ø
GO ASMB4Ø
```

Respond to the FILE query with OFILE. HDS will update the directory entry.

It is generally not possible for HDS to predict the required output file size before disk operations commence. If the user elects to direct disk output to an existing file, he must ensure that the file size is sufficient to contain the output. HDS will cease disk operations with a "NO ROOM" message when the existing output file is full.

3. MODIFYING BASIC

Perform the following steps to modify your copy of BASIC. Carriage return is indicated by ↵ .

	Release 4.0		Release 5.0
1.	LF BASIC 2A∅∅ ↵	load BASIC for modification	2D∅∅
2.	GO ASMB4∅ ↵		ASMB4∅.5
3.	After sign-on, type		
	F /PATCH/6∅∅∅ ↵	create memory file	
	After response, type		
	R ↵		
	Respond to the FILE query with		
	PATCH4.∅ ↵		PATCH5.∅
	After memory allocation response, type		
	A ∅ ↵		
	At completion of the assembly, type		
	B ↵	return to DOS	
4.	SF BASIC 2A∅∅ ↵	save BASIC	2D∅∅

4. BASIC MODIFICATIONS

Modified BASIC interprets a variable enclosed in square brackets as a reference to the address or location of the variable rather than to the current value of the variable. The address of floating point variables refers to the sign/exponent byte in standard North Star floating point form. The address of string variables refers to the first character in the string storage area. Addresses passed to assembly language routines allow these routines to operate on any BASIC variable.

The modified BASIC CALL to an assembly language routine imposes no limit to the number of such parameters. If there are exactly two parameters in the CALL, the first parameter is the destination address while the second parameter is passed in the DE registers to the assembly routine. When more than two parameters are present, the last parameter is passed in DE, while all preceding parameters are passed on the stack. Upon RETurn from the assembly routine the value present in HL is assigned to the value of the CALL. Generally a dummy assignment statement is used to invoke the CALL.

When parameters are passed on the stack it is the responsibility of the assembly language routine to POP the correct number of items off the stack to ensure proper RETurn to BASIC.

5. FLOATING POINT STORAGE MODE

North Star BASIC stores all numeric values in BCD floating point mode. Standard BASIC (8-digit) allocates 4 bytes for the mantissa and one byte for the characteristic and sign. The 8 digits of the mantissa are packed two BCD digits per byte in four consecutive memory bytes. The exponent byte follows the four bytes of the mantissa. The sign is the most significant bit of the exponent byte (Ø implies positive). The characteristic is stored excess 64, which means that the value 64 is added to the characteristic. A few examples should clarify:

$12345678 = .12345678 * 1Ø^{8}$

The mantissa is stored as

```
12 34 56 78
```

in four consecutive hex bytes. The characteristic is stored as

$64 + 8 = 48$ hex

The complete representation in memory is

```
12 34 56 78 48
```

The number -12345678 is stored as above, except that the sign bit is 1.

```
12 34 56 78 C8      (C8 = 48H + 8ØH)
```

The number .ØØ1 is written as

$.ØØ1 = .1 * 1Ø^{-2}$

```
1Ø ØØ ØØ ØØ 3E
```

Pointers to floating point numbers point to the sign/exponent byte.

6. STRING STORAGE MODE

Pointers to string variables point to the first character of the text area. The two bytes preceding the text represent the number of defined characters in the text. The two bytes preceding that contain the total dimension of the string variable.

7. ELEMENTARY OPERATIONS

The elementary arithmetic operations are performed by pointing register pair BC to the leading (first) operand, DE to the secondary operand, and CALLing the appropriate routine. The operation overwrites the leading operand. Thus to perform 12/3, point BC to 12, DE to 3, and CALL the DIVIDE entry. The representation of 12 is overwritten by the answer 4.

8. FUNCTIONAL OPERATIONS

Functions in BASIC are invoked by pointing the DE register pair to the argument and CALLing the appropriate functional routine. The argument is overwritten by the result.

9. ROADMAP OF BASIC OPERATIONS

OPERATION	RELEASE 4.0 ENTRY POINT	RELEASE 5.0 ENTRY POINT
+	4B32	4ED8
-	4B1B	4EC1
*	4A1Ø	4DB6
/	4C4Ø	4FE6
↑	3FB6	4349
SQRT	3F46	42D9
INT	3E51	41E1
SGN	3DFF	418F
SIN	59F4	5EØØ
COS	59EA	5DF6
ATN	5AE5	5EF1
ABS	3DFA	418A
LOG	58FF	5DØB
EXP	57B2	5BBE
COMPARE	3D8D	411D

10. HYBRID UTILITY ROUTINES

The following routines serve as examples of hybrid program development, and perform certain useful functions. In the following,

addr	represents the address at which the assembly routine is located.
xxyy	represents an arbitrary address.

OVERLAY LOADER FOR ASSEMBLY ROUTINES

The overlay loader allows BASIC programs to load an assembly routine into memory, prior to invoking that routine within the BASIC program. This routine is of such general use that it may prove desirable to incorporate it as part of BASIC. (IN DOS ~~1/0~~ AREA WITH JUMP @ $2901_H = 10497_{DEC}$)

In BASIC the loader is invoked by the sequence

```
P$ = "OBJFILE"                   ;P$ is the file to be loaded
Z9 = CALL (addr, xxyy, [P$] )
                       ► 10497₁₀
```

The Z9 is a dummy assignment. addr is the location of the loader, xxyy is the location at which the routine P$ is to be loaded. The loader itself is:

```
DOS:EQU 2ØØØH
ORG ADDR
MVI A,1              ;DRIVE NUMBER
XCHG                 ;HL POINTS TO FILE NAME
CALL DOS+1CH         ;DLOOK
JC  DOS+28H          ;FILE ERROR

ORI  8ØH             ;FOR DOUBLE DENSITY ONLY
MOV  C,A             ;DRIVE NUMBER
MVI  B,1             ;READ COMMAND
MOV  E,M
INX  H
MOV  D,M             ;DE HAS DISK ADDRESS
INX  H
MOV  A,M             ;FILE SIZE
POP  H               ;LOAD ADDRESS
XCHG
JMP  DOS+22H         ;DISK READ
```

The overlay loader is provided as file OVLOADR (single density).

REPLACING FOR/NEXT LOOPS

Simple FOR/NEXT loops are easily replaced by assembly routines, often with a dramatic improvement in speed. Consider the following BASIC segment to sum the N elements of an array A.

```
S=0
FOR J=1 TO N
S=S+A(J)
NEXT J
```

Replace this segment with

```
Z9=CALL(addr, [A(1)], [S], N)
```

The summation is performed by the assembly routine:

```
        ORG   ADDR
        XCHG
        SHLD  COUNT          ;N
        POP   B              ;POINTS TO S
        XRA   A
        STAX  B              ;S=Ø
        POP   D              ;POINTS TO ARRAY
SUMLP :LHLD  COUNT
        MOV   A, H
        ORA   L
        DCX   H
        SHLD  COUNT
        RZ                   ;RETURN IF DONE
        PUSH  B
        PUSH  D              ;SAVE POINTERS
        CALL  FPADD          ;FLOATING POINT ADD IN BASIC
        POP   D
        POP   B              ;RECOVER POINTERS
        LXI   H, 5           ;BYTES PER FLOATING POINT
        DAD   D              ;ADVANCE TO NEXT
        XCHG
        JMP   SUMLP
 COUNT :DW    Ø
```

NUMERICAL COMPARISON

The numerical comparison routine sets the flags according to a numerical comparison n between the elements pointed to by the BC and DE register pairs. The flags are affected as follows:

No flags	if @B > @D
Z	if @B = @D
C	if @B < @D

in which @B, @D refer to the floating point numbers addressed by BC and DE respectively. The flag status reflects the operation @BC-@DE.

The following routine uses the COMPARE routine to find the minimum of an array and return its value in B:

```
       ORG   ADDR
       XCHG
       SHLD  COUNT          ;NUMBER OF POINTS
       POP   D              ;POINTER TO ARRAY
       MOV   B,D
       MOV   C,E
MINLP: LHLD  COUNT
       MOV   A,H
       ORA   L
       DCX   H
       SHLD  COUNT
       JZ    COPY           ;SAVE MIN WHEN DONE
       PUSH  B
       PUSH  D              ;SAVE POINTERS
       CALL  COMPARE
       POP   D
       POP   B
       JC    BIGR
       MOV   B,D
       MOV   C,E            ;FOUND SMALLER
 BIGR: LXI   H,5            ;FLOATING POINT LENGTH
       DAD   D
       XCHG
       JMP   MINLP
 COPY: POP   H
       MVI   E,5            ;BYTES TO MOVE
COPLP: LDAX  B
```

(over)

```
        MOV    M,A
        DCX    B
        DCX    H                 ;POINTERS BACKWARD
        DCR    E
        JNZ    COPLP
        RET
COUNT:DW       Ø
```

This routine in invoked from BASIC as

```
    Z9 = CALL (ADDR,[B],[A(1)],N)
```

A S M B

A disk-based assembler/editor
for the development of small to medium size
assembly language programs

Allen Ashley
395 Sierra Madre Villa
Pasadena, CA 91107

(213) 793-5748

INTRODUCTION

ASMB is a powerful disk-based editor/assembler system for program development on a Z80 microcomputer. Structurally and operationally similar to the program development packages SP-1 and ESP-1, ASMB offers more extensive editing and assembling features while extending the instruction assembly to the entire Z80 instruction set.

ASMB includes all the features necessary for the creation, modification and storage of assembly language programs. Departing from the cumbersome ZILOG assembly language, ASMB features instructions mnemonics similar to the more widely familiar INTEL set. Indeed, mnemonics for the 8080 subset of the Z80 instruction set are identical to the standard INTEL format. Users familiar with INTEL assembly language will appreciate the treatment of the Z80 instruction superset as a logical and syntactical extension of the INTEL instructions.

ASMB is itself written entirely in the 8080 instruction subset, and is therefore operational on either 8080 or Z-80 machines. ASMB can thus serve as a two-way cross assembler, assembling 8080 source programs on a Z-80 machine, or Z-80 object programs on an 8080 machine. The versatility and power of ASMB make it an ideal program development system for either those presently owning a Z-80 machine or those anticipating a future expansion of their present 8080 machine to the more powerful Z-80 processor.

ASMB ORGANIZATION

The ASMB program development system consists of a combination text editor, assembler, and system executive for the creation and modification of Z80 assembly language programs.

The system executive is responsible for handling all input/output operations, invoking the editor or assembler, and dealing with the disposition of source and object files in central memory.

The text editor is responsible for the creation and modification of source programs within the memory file area. The text editor is line-oriented in that editing consists of entering or deleting source lines identified by ascending line numbers. The editor features automatic line numbering, line renumbering, moderately free-form source input, well-formatted source output, and a unique mini-editor for the modification of source code lines.

The assembler performs a two-pass translation of source to object code. The assembler includes the powerful feature of conditional assembly. Instruction mnemonics are logically and syntactically identical to the INTEL assembly language. The assembler is file-oriented with up to six source files simultaneously residing in memory. Optional symbol communication between files enables a moderate block structure development.

The concept and structure of ASMB were strongly influenced by Software Package #1. Assembly language source programs are maintained in source files under control of the system executive. Source files are created and deleted by commands to the system executive. Source code is entered into the source files under control of the editor, and the assembler can be directed to translate the source file to object code anywhere in memory.

Available space for the ASMB symbol table limits the size to approximately 200 labels for any single assembly.

CHANGES TO ASMB40

BACKSPACE CONTROL (USE BS=Ø8H)

@ ØØDD CHANGE 5F TO Ø8
@ ØØEC CHANGE 5F TO Ø8
@ Ø5E3 CHANGE 5F TO Ø8

V COMMAND (VECTOR TO F8[illegible])

@ Ø367 CHANGE ØØ TO Ø6
@ Ø368 CHANGE DØ TO F8

EXECUTIVE COMMANDS

COMMAND FORMAT

Executive commands consist of a single letter identifier, together with an optional modifier character, and one or two hexadecimal parameters. The command character(s) must be separated from any numerical parameters by a single blank. Numerical parameters are likewise separated by a blank.

In the following, hexadecimal parameters are indicated by the sequence nnnn or mmmm while an optional character modifier is indicated by a lower-case c. Unless otherwise noted, the modifier c is a device control character (Ø-7) which will be present in the accumulator for all subsequent console I/O.

All command lines are terminated by a carriage return.

COMMAND LIST

Fc /NAME/	File control command. The file control command enables the user to create or destroy source files. Each source file is identified by a file NAME of up to five characters. The file name must be delimited by slashes. The opening slash must be separated by a blank from the command characters. The hexadecimal parameter nnnn and the modifier character are optional.
F /NAME/nnnn	Opens a source file NAME, starting at location nnnn, making NAME the active file. Any previously active files are maintained.
F /OTHER/	Recall previously active file, OTHER, making it the currently active file. Note the hexadecimal parameter is absent.
F /ERASE/Ø	Delete file named ERASE, freeing memory space for a new source file.
F	Display the currently active file parameters, file name, starting and ending memory locations.
FS	Display the file parameters of all memory files.
W	Write the currently active source file to disk. The executive will respond with the query FILE. The user must then type the disk file to receive the source.
R	Read source code from disk into the currently active memory file. The executive responds with the FILE query.
C n	Append a disk file to the currently active memory file, renumbering all source code lines by the increment n.
	Improperly formed disk operations, disk read errors, or insufficient disk file capacity result in the DISK ERROR diagnostic.

D nnnn mmmm	Delete lines numbered nnnn up to and including mmmm from the source file. If mmmm is omitted only nnnn is deleted.
B	(BYE) Return to disk operating system.
I	Initialize the system, clearing all source files. The initialization is automatically performed upon initial entry. No lines of source code can be entered until a new source file has been defined.
Pc nnnn Pc /STRNG/	Print a formatted listing of the current source file, starting at line number nnnn.
Lc nnnn Lc /STRNG/	Print an unformatted listing, suppressing line numbers, of the current source file. The optional modifying character, c, can be an ASCII digit in the range Ø - 7. The numerical value of this modifier will be present in the accumulator for all subsequent I/O, or until redefined by the user. The value is initialized to zero.
G nnnn	Execute at location nnnn. A user program may return to the system executive by a simple return statement.
~~U~~	~~Execute at location DØØØ. This command is reserved for entry to the DEBUG control system~~.
U	JUMP TO F8Ø6 (IMSAI SMP/80.0)
A nnnn mmmm	Assemble the current source file using implied origin (ORG) nnnn and place resulting object code into memory starting at location mmmm. The second parameter is optional; if absent, the object code is placed into memory at nnnn.
AS	Mark existing symbol table for future global reference. (Save symbol table resulting from last assembly.) This command must follow an assembly: a symbol table must have been generated.
AE nnnn mmmm	Assemble, as above, displaying only source code lines containing an assembler diagnostic.
AK	Release (kill) the global symbol table.

E nnnn	Enter the mini-editor to edit the currently active source file beginning at line nnnn. The mini-editor enables the user to scroll through the source file, changing source lines on the fly. Upon entry, the mini-editor displays source line nnnn or the first source line if nnnn is omitted. The mini-editor then awaits keyboard input. Depressing any key except ESCAPE (1BH) advances the file pointer to display the next successive line. The escape key allows the user to re-enter the source line starting at character position two. (At the label field, no line number is required.) The user-entered line, terminated by carriage return, then overlays the old line. The mini-editor cannot insert new source lines into the file. Return to system executive via Control C.
E /STRNG/	Enter the mini-editor to edit the currently active source file beginning at the first occurrence of character string STRNG. The string may be at most five characters long and may contain no blanks. The string search is operable for the P and L commands as well.
N nnnn	Renumber source lines, starting at nnnn and incrementing by nnnn.

EDITOR

Source lines are entered into the currently active source file under control of the file editor. The system executive recognizes a source line by a four-digit decimal line number, which must precede every line in the source file. Modifications to the source file consist of one or more whole lines. Lines may be deleted by the D control command. Lines may be modified by retyping the line number and entering the new source line. The editor adjusts the source file to accommodate line length without any wasted file space.

Source program lines consist of a four-digit line number followed by a terminating blank. The first character of the source line may contain identifiers '*' or ';'. These identifiers proclaim the entire line to be a comment. The label field of the source line must be separated by exactly one blank from the line number. Identifying labels can be from one to five characters long and may contain no special characters. The operation field must be separated from the label field by one or more blanks. The operand field, if present, must be separated from the operation by a single blank. Two blanks following the last operand separate the comment field, which should start with a semicolon. Source lines may be up to 72 characters in length.

The user can invoke automatic line numbering for lines entered into the source file. In the automatic mode, line numbers are incremented by one from the starting value. Automatic line numbering is initiated by entering the starting line number followed by > (greater than). Subsequent entries begin in character position two. The automatic mode is exited by typing < (less than) following the carriage return for the last source line. Failure to properly exit the automatic mode can result in erroneous source lines. Lengthy insertions can be made into an existing source file by renumbering the file before entering the automatic mode.

The mini-editor allows text lines in the source file to be modified. When under control of the mini-editor, typing the Escape key switches from the scroll mode to the modify mode. Editing of the source line begins at the first character of the label field. Characters typed in under the modify mode are used to build the new source line. The old source line can be used as a model for generating the new source line: characters can be retrieved from the old line and placed in the new line. In the modify mode, the following control characters are recognized:

CONTROL-A	Fetch the next character from the old line and place it in the new line.
CONTROL-Z	Delete the next character from the old line.
CONTROL-Q	Back up one character in both the old and new lines.
CONTROL-G	Transfer the remainder of the old line to the new line.
CONTROL-S	Reads a character from the console, and transfers all characters from the old line up to, but not including, the input character.
CONTROL-Y	An insert toggle. Between successive toggles, input characters are inserted into the new line.

Any other characters typed in under the modify mode are entered into the new line, overriding the corresponding character from the old line.

ASSEMBLER OPERATION

The assembler operates upon the currently active source file only. The source file consists of a sequence of source lines composed of the four fields: label, operation, operand, and comment.

The label field, if present, must start in the second character position after the line number. Entries present in the label field are maintained in a symbol table. These entries are assigned a value equal to the program counter at the time of assembly, except that for the SET and EQU pseudo operations the variable defined by the label field is assigned the value of the operand field. The variables defined by the label field can be used in the operand field of other instructions either as data constants or locations.

The operation field, separated from the label field by one or more blanks or a colon, cannot appear before the third character following the line number. Entries in the operation field must consist of either a valid Z80 instruction or one of the several pseudo-operations.

The operand field, separated by a blank from the operation field, consists of an arithmetic expression containing one or more program variables, constants, or the special character $ connected by the operators + or -. Evaluation of the operand field is limited to a left to right scan of the expression, using 16 bit integer arithmetic. Operations requiring multiple operands (e.g., MOV A,B or BIT 3,IX,4) expect the operands to be separated by a comma.

The special operand $ refers to the program counter at the start of the instruction being assembled.* The program variable $ can be used as any other program variable except that its value changes constantly throughout assembly. The location counter $ allows the user to employ program relative computations.

Assembler constants may be either decimal or hexadecimal character strings. Valid hexadecimal constants must begin with a decimal digit, possibly Ø, and be terminated by the suffix H.

* NOTE: Some assemblers interpret $ as the start of the next instruction.

REGISTER MNEMONICS

All of the Z-80 registers are listed below. The predefined register set is defined as:

Register	Definition
A	Accumulator
B	8 or 16 bit
C	8 bit
D	8 or 16 bit
E	8 bit
H	8 or 16 bit
L	8 bit
M	Memory Indirect (HL)
SP	Stack Pointer
PSW	Program Status Word
IX	16 bit Index
IY	16 bit Index
RF	Refresh Register
IV	Interrupt Vector

INSTRUCTION SET ARRANGED BY GROUP

The complete Z-80 instruction set is listed below. In the instruction mnemonics which follow:

pp qq	refers to an arbitrary 16 bit datum;
yy	refers to an arbitrary 8 bit datum;
d	refers to a Z80 displacement except for relative jumps;
R	refers to an 8 bit register (A, B, C, D, E, H, L, M)
RP	refers to a 16 bit register pair (B, D, H, SP)
QP	refers to a 16 bit register pair (PSW, B, D, H)

MNEMONIC	ZILOG	REMARKS
8 BIT LOAD		
MOV R,R	LD R,R	Register to register (to, from) (R≠M)
MOV R,IX,d	LD R,(IX+d)	Register indirect
MOV R,IY,d	LD R,(IY+d)	"
MOV IX,d,R	LD (IX+d),R	Memory indirect (R≠M)
MOV IY,d,R	LD (IY+d),R	
MOV A,IV	LD A,I	Fetch interrupt vector
MOV A,RF	LD A,R	Fetch refresh register
MOV IV,A	LD I,A	Load interrupt vector
MOV RF,A	LD R,A	Load refresh register
ACCUMULATOR LOAD/STORE		
LDA pp qq	LD A,(nn)	Accumulator direct
LDAX B	LD A,(BC)	Accumulator extended
LDAX D	LD A,(DE)	
STA pp qq	LD (nn),A	Accumulator direct
STAX B	LD (BC),A	Accumulator extended
STAX D	LD (DE),A	
8 BIT LOAD IMMEDIATE		
MVI R,yy	LD R,n	Register immediate
MVI IX,d,yy	LD (IX+d),n	Memory indirect immediate
MVI IY,d,yy	LD (IY+d),n	

16 BIT LOAD/STORE RP = B, D, H, SP QP = PSW, B, D, H

MNEMONIC	ZILOG	REMARKS
LXI RP,pp qq	LD RP,nn	Extended immediate
LXI IX,pp qq	LD IX,nn	
LXI IY,pp qq	LD IY,nn	
LHLD pp qq	LD HL,(nn)	Extended indirect load
LBCD pp qq	LD BC,(nn)	
LDED pp qq	LD DE,(nn)	
LIXD pp qq	LD IX,(nn)	
LIYD pp qq	LD IY,(nn)	
LSPD pp qq	LD SP,(nn)	
SHLD pp qq	LD (nn),HL	Extended indirect store
SBCD pp qq	LD (nn),BC	
SDED pp qq	LD (nn),DE	
SIXD pp qq	LD (nn),IX	
SIYD pp qq	LD (nn),IY	
SSPD pp qq	LD (nn),SP	
SPHL	LD SP,HL	Set stack pointer
SPIX	LD SP,IX	
SPIY	LD SP,IY	
PUSH QP	PUSH QP	To stack
PUSH IX	PUSH IX	
PUSH IY	PUSH IY	
POP QP	POP QP	From stack
POP IX	POP IX	
POP IY	POP IY	

EXCHANGE, BLOCK TRANSFER, AND SEARCH

MNEMONIC	ZILOG	REMARKS
XCHG	EX DE,HL	Exchange
EX	EX AF,AF'	
EXX	EXX	
XTHL	EX (SP),HL	
XTIX	EX (SP),IX	
XTIY	EX (SP),IY	
LDI	LDI	Transfer
LDIR	LDIR	
LDD	LDD	
LDDR	LDDR	
CPD	CPD	Search
CPDR	CPDR	
CPII	CPI	
CPIR	CPIR	

8 BIT ARITHMETIC AND LOGICAL

MNEMONIC	ZILOG	REMARKS
ADD R	ADD R	Add register
ADI yy	ADD A,yy	Add immediate
ADD IX,d	ADD (IX+d)	Add indirect
ADD IY,d	ADD (IY+d)	
ADC R	ADC R	Register with carry
ADC IX,d	ADC (IX+d)	Memory indirect with carry
ADC IY,d	ADC (IY+d)	
ACI yy	ADC n	Immediate with carry
SUB R	SUB R	Subtract Register
SUB IX,d	SUB (IX+d)	Subtract memory indirect
SUB IY,d	SUB (IY+d)	
SBB R	SBC R	Register with carry
SBB IX,d	SBC (IX+d)	Memory indirect with carry
SBB IY,d	SBC (IY+d)	
ANA R	AND R	Logical and register
ANA IX,d	AND (IX+d)	Memory indirect
ANA IY,d	AND (IX+d)	
ORA R	OR R	Logical OR register
ORA IX,d	OR (IX+d)	Memory indirect
ORA IY,d	OR (IY+d)	
XRA R	XOR R	Exclusive OR register
XRA IX,d	XOR (IX+d)	Memory indirect
XRA IY,d	XOR (IY+d)	
CMP R	CP R	Register compare
CMP IX,d	CP (IX+d)	Memory indirect
CMP IY,d	CP (IY+d)	
INR R	INC R	Register increment
INR IX,d	INC (IX+d)	
INR IY,d	INC (IY+d)	
DCR R	DEC R	Register decrement
DCR IX,d	DEC (IX+d)	
DCR IY,d	DEC (IY+d)	
ANI yy	AND yy	Accumulator immediate
XRI yy	XOR yy	
CPI yy	CP yy	
ORI yy	OR yy	
SUI yy	SUB yy	
SBI yy	SBC A,yy	

MNEMONIC	ZILOG	REMARKS
GENERAL PURPOSE ARITHMETIC AND CPU CONTROL		
DAA	DAA	Decimal adjust accumulator
CMA	CPL	Complement accumulator logical
NEG	NEG	Negate accumulator
CMC	CCF	Complement carry flag
STC	SCF	Set carry flag
NOP	NOP	No operation
HLT	HALT	HALT CPU
DI	DI	Disable interrupts
EI	EI	Enable interrupts
IM Ø IM 1 IM 2	IM Ø IM 1 IM 2	Set interrupt mode
16 BIT ARITHMETIC GROUP RP = B, D, H, SP		
DAD RP	ADD HL,RP	16 bit add (RP≠H or IY)
CAD RP	ADC HL,RP	16 bit add with carry (RP≠H or IX)
SBC RP	SBC HL,RP	16 bit subtract with carry
DAD IX,RP	ADD IX,RP	16 bit add register pair to IX
DAD IY,RP	ADD IY,RP	16 bit add register pair to IY
INX RP INX IX INX IY	INC RP INC IX INC IY	16 bit increment
DCX RP DCX IX DCX IY	DEC RP DEC IX DEC IY	16 bit decrement

ROTATE AND SHIFT GROUP   R = B, C, D, E, H, L, M, IX+d, IY+d

MNEMONIC	ZILOG	REMARKS
RLC	RLCA	Accumulator left circular
RAL	RLA	Left circular through carry
RRC	RRCA	Accumulator right circular
RAR	RRA	Right circular through carry
SLC R	RLC R	Register left circular
SLC M	RLC (HL)	Memory left circular
SLC IX,d SLC IY,d	RLC (IX+d) RLC (IY+d)	Left circular memory indirect
RL R	RL R	Register left through carry
SRC R	RRC R	Register right circular
RR R	RR R	Register right through carry
SLA R	SLA R	Left linear bit Ø = Ø
SRA R	SRA R	Right linear bit 7 = extended
SRL R	SRL R	Right linear bit 7 = Ø
RLD	RLD	Left decimal
RRD	RRD	Right decimal

MNEMONIC	ZILOG	REMARKS
BIT MANIPULATION	b = bit number	Ø ≤ b ≤ 7
BIT b,R	BIT b,R	Zero flag = bit b of register R
BIT b,M BIT b,IX,d BIT b,IY,d	BIT b,(HL) BIT b,(IX+d) BIT b,(IY+d)	

STB b,R
STB b,M
STB b,IX,d
STB b,IY,d
	SET b,R
	SET b,(HL)
	SET b,(IX+d)
	SET b,(IY+d)
		Set (1) bit b of register or memory

RES b,R RES b,M RES b,IX,d RES b,IY,d	RES b,R RES b,(HL) RES b,(IX+d) RES b,(IY+d)	Reset (Ø) bit b of register or memory
INPUT/OUTPUT GROUP	P = port number	R = register
IN P	IN A,(P)	Input to accumulator
CIN R	IN R,(C)	Register R from port (C)
INI	INI	Input and increment
INIR	INIR	Repeated input and increment
IND	IND	Input and decrement
INDR	INDR	Repeated input and decrement
OUT P	OUT (P),A	Output accumulator
COUT R	OUT (C),R	Register R to port (C)
OUTI	OUTI	Output and increment
OUTIR	OUTIR	Repeated output and increment
OUTD	OUTD	Output and decrement
OUTDR	OUTDR	Repeated output and decrement

JUMP GROUP V = location (16 bit) dest = destination (±128 bytes displacement)

MNEMONIC	ZILOG	REMARKS
JMP V	JP V	Jump
JNC V	JP NC,V	No carry
JC V	JP C,V	Carry
JNZ V	JP NZ,V	Not zero
JZ V	JP Z,V	Zero
JPO V	JP PO,V	Parity odd
JPE V	JP PE,V	Parity even
JP V	JP P,V	Positive
JM V	JP M,V	Negative
JR dest	JR d	Jump relative
JRC dest	JR C,d	Carry
JRNC dest	JR NC,d	No carry
JRZ dest	JR Z,d	Zero
JRNZ dest	JR NZ,d	Not zero
PCHL	JP (HL)	Branch to location in HL
PCIX	JP (IX)	Branch to IX
PCIY	JP (IY)	Branch to IY
DJNZ dest	DJNZ,d	Decrement and jump relative if not zero

CALL AND RETURN GROUP V = address

MNEMONIC	ZILOG	REMARKS
CALL V	CALL V	Subroutine transfer
CNC V	CALL NC,V	No carry
CC V	CALL C,V	Carry
CNZ V	CALL NZ,V	Not zero
CZ V	CALL Z,V	Zero
CPE V	CALL PE,V	Parity even
CPO V	CALL PO,V	Parity odd
CP V	CALL P,V	Positive
CM V	CALL M,V	Negative
RET	RET	Return
RNC	RET NC	No carry
RC	RET C	Carry
RNZ	RET NZ	Not zero
RZ	RET Z	Zero
RPE	RET PE	Parity even
RPO	RET PO	Parity odd
RP	RET P	Positive
RM	RET M	Negative
RETI	RETI	Return from interrupt
RETN	RETN	Return from non-maskable interrupt
RST n	RST n	Restart

PSEUDO OPERATIONS

ASSEMBLER	PSEUDO OPERATIONS expr = arithmetic expression
ORG expr	Define program counter to nnnn
DS expr	Reserve n bytes of storage
DW expr	16 bit datum definition
DB expr	8 bit datum or ASCII character string definition. The operand may be an ASCII character string enclosed in single quotation marks. ASMB allows only a single entry per line. Examples: DB 5 DB 'ASCII STRING'
EQU	The operand defined by the label field is set equal to the expression defined by the operand field. This operation is performed in pass one of the assembler and the variable definition is fixed by the first such definition encountered.
SET	The operand defined by the label is set equal to the expression defined by the operand field. This operation is performed in both pass 1 and pass 2 and the replacement is effected upon every encounter.
IF expr	expr is evaluated. If the result is zero the scanner skips to the next ENDIF, END, or end of file before resuming assembly. If the expression evaluates to any non-zero value, assembly proceeds. Operation is performed in both passes.
ENDIF	Identifies the end of a conditional assembly block.
END	Terminates assembly.
USE operand	Allows program assembly to proceed with multiple location counters. The operation is skipped if the operand has not previously been defined; however, the definition can appear after the reference, to be used by pass 2. The USE operation is best explained by example.

```
AORG   SET   ØAØØØH
BORG   SET   ØBØØØH
       USE   AORG;     SET code origin to AORG
      { code at ØAØØØH }
       USE   BORG;     SET value of AORG to PC
                       SET PC to BORG
      { code at ØBØØØH }
```

```
        USE   AORG;    Resume code at end of previous
                        block which started at AØØØ.

        { code }

        USE   BORG;    Resume code at END of block
                       which started at BØØØ.
```

The USE instruction can be used to insert program data at the end of instruction code.

```
AFTR    SET   LAST;    Not known on pass 1.
        ORG   START;   Somewhere.

        { code }

RESUM   SET   $;       Remember where we are.
        USE   AFTR
STRING: DB    'CHARACTERS'
        USE   RESUM;   Resume in line coding.

        { code }

        USE   AFTR

        { more data }

        USE   RESUM;   Continue.

 LAST   SET   $
        END
```

ASSEMBLER ERRORS/DIAGNOSTICS

Assembler error and diagnostic messages consist of single character identifiers which flag some irregularity discovered either during pass 1 or pass 2 of the assembly. The single character precedes the line number of the formatted assembly listing.

P	Phase error: the value of the label has changed between the two assembly passes.
L	Label error: label contains illegal or too many characters, e.g., LB#1:
U	Undefined program variable.
V	Value error: the evaluated operand is not consistent with the operation e.g., MVI A, 1ØØØH (not a valid 8 bit operand).
S	Syntax error e.g., MOV A+B
O	Opcode error, e.g. DCS B
M	Missing label field.
A	Argument error.
R	Register error.
D	Duplicate label error.

EXISTING SOURCE FILES

ASMB is compatible with programs generated under SP#1 or its many descendents, SCS 1,2, ESP-1, ALS-8, etc. These related source programs can be included in the ASMB disk system by the following procedure:

1. Load ASMB and create a memory file at a convenient memory location.
2. Exit from ASMB and load the existing source file into memory starting at the memory location defined in step 1.
3. Re-enter ASMB and examine the file with the P command.
4. Delete and re-enter the last line of the source code.
5. Save the memory file on disk via the W command.
6. EDIT will re-format the source file for MAKRO via the N command.

While all such files are compatible with ASMB, EDIT may be unable to effect the reformat. A failure may arise if EDIT does not encounter the ASMB end-of-file Ø1 (catastrophic).

SAMPLE ASMB OPERATION

```
>GO ASMB
ASMB DEVELOPMENT SYSTEM
F /TEST/6000                                  Create memory file
TEST  6000    6000
0010LABEL:INX H                               > typed after line number, but not echoed
 DAD B
 ORA A                                        Auto line mode
 END                                          < typed after carriage return
P                                             Print formatted listing
0010 LABEL   INX    H
0011         DAD    B
0012         ORA    A
0013         END
A F000                                        Assemble file
F000 23                              0010 LABEL   INX    H       Assembly listing
F001 09                              0011         DAD    B
F002 B7                              0012         ORA    A
F003                                 0013         END
SYMBOL TABLE
LABEL F000
W                                             Write source to disk
FILE
SAVE WRITTEN                                  Disk operation completed
B>LI
DOS         4   10    0
MAKRO      14   32    1 2A00
EDIT       46   11    1 2A00
END       168    0    0
SAVE      170    1    0                       Source file
ASMB       57   25    1 2A00
DEBUG      82   55    0
KWIKABS   137    3    1 2A00
KWIK      140   15    0
LINKED    155   13    0
```