ASMB-Z8

Z80/8080 CROSS ASSEMBLER FOR THE ZILOG Z8 MICROPROCESSOR

A disk-based assembler/editor compatible with the
ZILOG Z8 Instruction Set

$75

Allen Ashley
395 Sierra Madre Villa
Pasadena, CA 91107

(213) 793-5748

ALLEN ASHLEY

395 SIERRA MADRE VILLA • PASADENA, CALIFORNIA 91107 • (213) 793-5748

Dear User:

I regret being unable to include a personal note. However, there are a few points which could not be covered in the documentation.

First, I want you to be happy with this software package. If you have any difficulty -- however slight -- with either the documentation or the program, please contact me. I prefer to interact by telephone, but as time allows I will correspond by mail.

Should program errors arise they will be repaired at no charge. I ask only that you return your original disk or cassette with proper packaging and a return manila envelope with sufficient return postage.

Many of the best features of this software were suggested by users, and your comments and suggestions on the documentation or the program are welcome. Let's keep in touch.

Sincerely,

Allen Ashley

Allen Ashley

CONTENTS

BRINGING UP ASMB

1. Write protect the ASMB disk or cassette.

2. Make a working copy of the master program; store the original as backup. The ASMB cassette loads at the 500 baud SYSTEM rate.

3. Read the ASMB documentation.

4. ASMB resides in memory immediately after the DOS. In the standard configuration the memory region from 52ØØH to 78ØØH is reserved for ASMB and assembler tables. Neither source nor object files can be located within this region without damage to the programs.

5. <u>Cassette Load Sequence:</u>

   a. Enter ROM BASIC, with cassette 'L' (500 baud rate).

   b. Execute SYSTEM command.

   c. Respond to "*?" with

      AXnnnn where nnnn is the appropriate file name:

      AXCOP4
      AX8Ø48
      AXZ8
      AX1802
      AX2021
      AX2224
      AX3870

   d. If the assembler is <u>not</u> to be saved on disk then you may branch directly:

      *? /

   e. If the assembler is to be saved on disk for later, more rapid access, use the TAPE utility. (The cassett load sequence over-writes the DOS.) Follow this sequence:

      1. TAPE (S=T, D=D)

      2. CASS? L

   The program will be saved.

INTRODUCTION

ASMB is a powerful disk/tape based editor/assembler system for target processor program development on a TRS-80 microcomputer.

ASMB includes all the features necessary for the creation, modification and storage of assembly language programs for the target processor. With minor exceptions, ASMB features instruction mnemonics identical to the manufacturer's instruction set.

Programs developed with ASMB must be off-loaded for execution by the target processor.

INTERFACE TO TRS DOS

File names communicated to ASMB 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 Ø.

DISKFILE or
DISKFILE,Ø refer to file DISKFILE on drive Ø.

If a required file is not found in the directory, the file will be created; otherwise it will be overwritten.

Assembly source files are automatically assigned an extension ASM.

All programs use backspace (Ø8) as character delete and BREAK (Ø1) as abort. The Model I BREAK key may return to TRS DOS. In that event, ASMB must be patched to use an alternate ABORT key. Change locations 6714H and 5724H from Ø1 to your desired ABORT key.* One suggestion might be to change that value to 1F and thereby use the CLEAR key as an abort.

* For later versions of cross-assemblers ASMB-8051, -8070 and -TMS7, these two locations are 573C and 6809 respectively.

ASMB ORGANIZATION

The ASMB program development system consists of a combination text editor, assembler, and system executive for the creation and modification of 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, and well-formatted source output.

The assembler performs a two-pass translation of source to object code. The assembler includes the powerful feature of conditional assembly. Instruction mnemonics are generally logically and syntactically identical to the manufacturer's instruction set. 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.

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.

The ASMB editor/assembler resides in memory immediately after the DOS. In the standard configuration, the memory region from 52ØØH up to 75ØØH is reserved for ASMB and assembler tables. Neither source nor object files can be located within this region without damage to the programs.

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), of which only Ø (CRT) and 1 (printer) are supported.

COMMAND LIST

F /NAME/

(Generic command; specific examples below.)

Generic 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. There is no relation between memory file NAME and any disk file.

F /NAME/nnnn

Opens a source file NAME, starting at memory location nnnn, making NAME the active file. Any previously active files are maintained. NOTE: no spaces after the /.

F /OTHER/

Recall previously active file OTHER, making it the currently active file. Note that 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.

WT Write currently active source file to tape (500 baud).*

WD Write 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. *

RT Read source code from tape.*

RD Read source code from disk into the currently active memory file. The executive responds with the FILE query. *

CT n Append a source file from tape, renumbering source lines by increment n.*

CD n Append a disk file to the currently active memory file, renumbering all source code lines by the increment n.*

* Improperly formed operations, read errors, or insufficient disk file capacity result in the DISK ERROR or TAPE ERROR diagnostics.

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 *** Print a formatted listing of the current source file, starting at line number nnnn. **

Lc nnnn *** Print an unformatted listing (suppressing line numbers) of the current source file. **

** The optional modifying character, when present, can be the digit 1 to direct output to list device.

G nnnn Execute at location nnnn; used to enter an auxiliary program, such as a PROM burner.

A nnnn mmmm*** Assemble the current source file using implied origin (ORG) nnnn and place the 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.

If there is no ORG in your program, the first parameter acts as ORG nnnn in your program. The code will be assembled as if it is to run at location nnnn. Most applications, however, require an execution address in low memory, in conflict with the ROM of the TRS-80. The second parameter mmmm allows the code to be repositioned to available RAM. Thus

```
A Ø BØØØ
```

will assemble the code for execution at location Ø (first parameter), and place the object code in memory at BØØØ (second parameter).

Note that the source file address given in a previous F command does not appear in the A (assemble) command.

AS — Mark existing symbol table for future global reference. (Save symbol table resulting from last assembly.) This command, if used, 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.

AT — Print symbol table resulting from previous assembly.

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 up-arrow (5BH) advances the file pointer to display the next successive line. The up-arrow 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 a carriage return, then overlays the old line. The mini-editor cannot insert new source lines into the file. Return to system executive via BREAK.

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 nn — Renumber source lines, starting at nn and incrementing by nn. The value nn is a decimal parameter.

*** P, L and A command examples: AØ nnnn or Ab̸ nnnn will send the output to the CRT.
A1 nnnn or A2 nnnn will send the output to a printer.

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. Character deletion is accomplished by the DELETE (←) key.

Source program lines consist of a four-digit 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.

SCROLLING PROGRAM OUTPUT

The assembler allows the output to be scrolled. Pressing the space bar will freeze the display; any other key will resume scroll. Holding the space bar down progresses output at the repeat rate.

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 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 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.

The individual bytes of a 16-bit operand may be accessed as 8-bit operands:

```
VALUE!H is the high order byte
VALUE!L is the low order byte
```

where VALUE is a 16-bit quantity and ! is the ASCII exclamation character with value 21H.

Arithmetic expressions involving string operands must not begin with the string. Example:

```
80H + 'A'   is valid
'A' + 80H   is invalid
```

A presentation of the target processor assembly language may be found in the appropriate programming manual.

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

PSEUDO OPERATIONS

ASSEMBLER

PSEUDO OPERATIONS (expr = arithmetic expression)

ORG expr

Define program counter to nnnn.

DS expr

Reserve expr 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. Examples:

```
DB 5,6,7
DB 'ASCII STRING',ØDH,ØAH
```

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 assembly and the variable definition is fixed by the last 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. The USE operation is best explained by example:

* Neither the IF nor NIF blocks preceding the ENDIF may contain comments containing the END or ENDI character sequences.

USE operand
(cont'd)

Example:

```
AORG   SET    1ØØH
BORG   SET    2ØØH
       USE    AORG;      SET code origin to AORG

       { code at 1ØØH }

       USE    BORG;      SET value of AORG to PC
                         SET PC to BORG

       { code at 2ØØH }

       USE    AORG;      Resume code at end of
                         previous block which started
                         at 1ØØH.

       { code }

       USE    BORG;      Resume code at END of
                         block which started at 2ØØH.
```

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

U Undefined program variable

V Value error: the evaluated operand is not consistent with the operation

S Syntax error

O Opcode error

M Missing label field

A Argument error

R Register error

D Duplicate label error

SAMPLE ASMB OPERATION

```
.............................................................ASMB

ASMB DEVELOPMENT SYSTEM
F /TEST/ 75ØØ                                     Create memory file at 75ØØH
TEST  75ØØ   75ØØ

0010 LOOP:INX H                                   > typed after line number, but not echoed
0011  DAD B                                       Auto line mode
0012  ORA A
0013  JNZ LOOP
0014  RET
0015                                              < typed after carriage return
A A000                                            Assemble file

                                                  Assembly listing
A000 23                   0010 LOOP    INX    H
A001 09                   0011         DAD    B
A002 B7                   0012         ORA    A
A003 C2 00 A0             0013         JNZ    LOOP
A006 C9                   0014         RET

SYMBOL TABLE

LOOP  A000

WD                                                Write source to disk
FILE
SAVE WRITTEN                                      Disk operation completed
```

Z8 INSTRUCTION SET

REGISTERS AND NOTATION

The notation for operands (condition codes and address modes) and the actual operands they represent are as follows:

Notation	Address Mode	Actual Operand/Range
cc	Condition code	See condition code list below
dst	Destination register	
src	Source register	
r	Working register only	Rn, where n = 0-15
R	Register or working register	reg = 0-127, 240-255; Rn as defined above
RR	Register pair or working register pair	reg, where reg is an even number in the range above or a variable whose address is even; RRp where p = 0, 2, 4, 6...14
Ir	Indirect working register only	@Rn, where n = 0-15
IR	Indirect register or working register	@reg, where reg is as defined above; @Rn, as defined above
Irr	Indirect working register pair only	@RRp, where p = 0, 2, 4, 6...14
IRR	Indirect register pair or working register pair	@reg, where reg is an even number in the range above, or a variable whose address is even; @RRp as defined above
X	Indexed	reg(Rn), where reg and Rn are as defined above
DA	Direct address	Program label or expression
RA	Relative address	Program label or $ + or - offset, where the location addressed must be in the range +127, -128 bytes from the start of the next instruction
IM	Immediate	#data, where data is an expression

CONDITION CODES AND STATUS FLAGS

Status flags are represented as follows:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

The condition codes and the flag settings they represent are:

Code	Meaning	Flag Settings	Value
	0 Always false	-	0000
(blank)	8 Always true	-	1000
Z	6 Zero	Z=1	0110
NZ	E Not zero	Z=0	1110
C	7 Carry	C=1	0111
NC	F No carry	C=0	1111
PL	D Plus	S=0	1101
MI	5 Minus	S=1	0101
NE	E Not equal	Z=0	1110
OV	4 Overflow	V=1	0100
NOV	C No overflow	V=0	1100
GE	9 Greater than or equal	(S XOR V)=0	1001
LT	1 Less than	(S XOR V)=1	0001
GT	A Greater than	(Z OR (S XOR V))=0	1010
LE	2 Less than or equal	(Z OR (S XOR V))=1	0010
UGE	F Unsigned greater than or equal	C=0	1111
ULT	7 Unsigned less than	C=1	0111
UGT	B Unsigned greater than	((C=0) & (Z=0))=1	1011
ULE	3 Unsigned less than or equal	(C OR Z)=1	0011

Note that some of the condition codes correspond to identical flag settings, i.e. Z-EQ, NZ-NE, C-ULT, NC-UGE.

Z8 INSTRUCTIONS

	Addr Mode		Opcode Byte	
Instruction	dst	src	(Hex)	Description
ADC dst, src	(Note 1)		1□	Add with carry
ADD dst, src	(Note 1)		0□	Add
AND dst, src	(Note 1)		5□	Logical AND
CALL dst	DA		D6	Call procedure
	IRR		D4	
CCF			EF	Complement carry flag
CLR dst	R		B0	Clear
	IR		B1	
COM dst	R		60	Complement
	IR		61	
CP dst, src	(Note 1)		A□	Compare
DA dst	R		40	Decimal adjust
	IR		41	
DEC dst	R		00	Decrement
	IR		01	
DECW dst	RR		80	Decrement word
	IR		81	
DI			8F	Disable interrupts
DJNZ r, dst	RA		rA	Decrement and jump if nonzero
EI			9F	Enable interrupts
INC dst	r		rE	Increment
	R		20	
	IR		21	
INCW dst	RR		A0	Increment word
	IR		A1	
IRET			BF	Interrupt return
JP cc, dst	DA		cD	Jump
	IRR		30	
JR cc, dst	RA		cB	Jump relative

	Addr Mode		Opcode Byte	
Instruction	dst	src	(Hex)	Description
LD dst, src	r	IM	rC	Load (except indexed)
	r	R	r8	
	R	r	r9	
	r	Ir	E3	
	Ir	r	F3	
	R	R	E4	
	R	IR	E5	
	R	IM	E6	
	IR	IM	E7	
	IR	R	F5	
LDRX dst, index, base	r	X	C7	Load (indexed). These instructions replace the Zilog indexed load.
LDXR base, index, src	X	r	D7	
LDC dst, src	r	Irr	C2	Load constant
	Irr	r	D2	
LDCI dst, src	Ir	Irr	C3	Load constant autoincrement
	Irr	Ir	D3	
LDE dst, src	r	Irr	82	Load external data
	Irr	r	92	
LDEI dst, src	Ir	Irr	83	Load external data autoincrement
	Irr	Ir	93	
NOP			FF	No operation
OR dst, src	(Note 1)		4□	Logical OR
POP dst	R		50	Pop
	IR		51	
PUSH src		R	70	Push
		IR	71	
RCF			CF	Reset carry flag
RET			AF	Return
RL dst	R		90	Rotate left
	IR		91	
RLC dst	R		10	Rotate left through carry
	IR		11	
RR dst	R		E0	Rotate right
	IR		E1	
RRC dst	R		C0	Rotate right through carry
	IR		C1	

	Addr Mode		Opcode Byte	
Instruction	dst	src	(Hex)	Description
SBC dst, src	(Note 1)		3□	Subtract with carry
SCF			DF	Set carry flag
SRA dst	R		D0	Shift right arithmetic
	IR		D1	
SRP src		IM	31	Set register pointer
SUB dst, src	(Note 1)		2□	Subtract
SWAP dst	R		F0	Swap nibbles
	IR		F1	
TCM dst, src	(Note 1)		6□	Test complement under mask
TM dst, src	(Note 1)		7□	Test under mask
XOR dst, src	(Note 1)		B□	Logical exclusive OR

NOTE 1: These instructions have an identical set of addressing modes, which are encoded for brevity in this table. The higher opcode nibble is found in the instruction set table above. The lower nibble is expressed symbolically by a □ in the table, and its value is found in the following table to the right of the applicable addressing mode pair. For example, the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

Addr Mode		Lower
dst	src	Opcode Nibble
r	r	$\boxed{2}$
r	Ir	$\boxed{3}$
R	R	$\boxed{4}$
R	IR	$\boxed{5}$
R	IM	$\boxed{6}$
IR	IM	$\boxed{7}$

Z8

ASSEMBLER TEST PROGRAM

```
0000                0010 ; WORKING REGISTERS RN 0 LE N LE 15
0000 12 43          0020            ADC     R4, R3
0002 13 43          0030            ADC     R4, @R3
0004 14 03 04       0040            ADC     4, 3
0007 15 03 04       0050            ADC     4, @3
000A 15 03 E4       0060            ADC     R4, @3
000D 16 04 01       0070            ADC     4, #1
0010 16 E4 01       0080            ADC     R4, #1
0013 17 04 01       0090            ADC     @4, #1
0016 17 E4 01       0100            ADC     @R4, #1
0019 02 43          0110            ADD     R4, R3
001B 52 43          0120            AND     R4, R3
001D A2 43          0130            CP      R4, R3
001F 42 43          0140            OR      R4, R3
0021 32 43          0150            SBC     R4, R3
0023 22 43          0160            SUB     R4, R3
0025 62 43          0170            TCM     R4, R3
0027 72 43          0180            TM      R4, R3
0029 B2 43          0190            XOR     R4, R3
002B D6 00 5E       0200            CALL    LOOP
002E D4 E4          0210            CALL    @RR4
0030 EF             0220            CCF
0031 8F             0230            DI
0032 9F             0240            EI
0033 BF             0250            IRET
0034 FF             0260            NOP
0035 CF             0270            RCF
0036 AF             0280            RET
0037 DF             0290            SCF
0038 B0 04          0300            CLR     4
003A B0 E4          0310            CLR     R4
003C B1 04          0320            CLR     @4
003E B1 E4          0330            CLR     @R4
0040 60 E4          0340            COM     R4
0042 40 E4          0350            DA      R4
0044 00 E4          0360            DEC     R4
0046 50 E4          0370            POP     R4
0048 70 E4          0380            PUSH    R4
004A 90 E4          0390            RL      R4
004C 10 E4          0400            RLC     R4
004E E0 E4          0410            RR      R4
0050 C0 E4          0420            RRC     R4
0052 D0 E4          0430            SRA     R4
0054 F0 E4          0440            SWAP    R4
0056 80 E4          0450            DECW    RR4
0058 81 E4          0460            DECW    @R4
005A 81 04          0470            DECW    @4
005C A0 E4          0480            INCW    RR4
005E 6A FE          0490 LOOP       DJNZ    R6, LOOP
0060 0A FC          0500            DJNZ    R0, LOOP
0062 FA FA          0510            DJNZ    R15, LOOP
0064 4E             0520            INC     R4
0065 21 E3          0530            INC     @R3
0067 21 03          0540            INC     @3
0069 5D 00 5E       0550            JP      MI, LOOP
006C 30 E4          0560            JP      @RR4
006E FD 00 5E       0570            JP      NC, LOOP
0071 ED 00 5E       0580            JP      NZ, LOOP
0074 DD 00 5E       0590            JP      PL, LOOP
```

```
0077 4D 00 5E           0600        JP      OV,LOOP
007A 6D 00 5E           0610        JP      EQ,LOOP
007D ED 00 5E           0620        JP      NE,LOOP
0080 9D 00 5E           0630        JP      GE,LOOP
0083 2D 00 5E           0640        JP      LE,LOOP
0086 1D 00 5E           0650        JP      LT,LOOP
0089 AD 00 5E           0660        JP      GT,LOOP
008C 5B D0              0670        JR      MI,LOOP
008E 4C 01              0680        LD      R4,#1
0090 48 03              0690        LD      R4,3
0092 39 04              0700        LD      4,R3
0094 E3 43              0710        LD      R4,@R3
0096 F3 43              0720        LD      @R4,R3
0098 E4 03 04           0730        LD      4,3
009B E5 E3 04           0740        LD      4,@R3
009E E6 04 01           0750        LD      4,#1
00A1 E7 04 01           0760        LD      @4,#1
00A4 F5 04 04           0770        LD      @4,4
00A7                    0780 ; BELOW ARE THE INDEXED LOADS
00A7 C7 A0 F0           0790        LDRX    R10,R0,240    ;IS LD R10, 240(R0)
00AA D7 A0 F0           0800        LDXR    240,R0,R10    ; IS LD 240(R0),R10
00AD C2 34              0810        LDC     R3,@RR4
00AF D2 43              0820        LDC     @RR4,R3
00B1 82 34              0830        LDE     R3,@RR4.
00B3 C3 34              0840        LDCI    @R3,@RR4
00B5 D3 43              0850        LDCI    @RR4,@R3
00B7 83 34              0860        LDEI    @R3,@RR4
00B9 50 E4              0870        POP     R4
00BB 50 04              0880        POP     4
00BD 51 04              0890        POP     @4
00BF 51 E4              0900        POP     @R4
00C1 70 04              0910        PUSH    4
00C3 31 00          V   0920        SRP     12
00C5 31 70              0930        SRP     70H
00C7                    0940;;RELA  LIMITS
00C7                    0950;;NOTE  $AT     REFERS  TO START OF DJNZ INSTRUCTION
00C7 6A 00          V   0960        DJNZ    R6,$-128
00C9 6A 00          V   0970        DJNZ    R6,$-127
00CB 6A 80              0980        DJNZ    R6,$-126
00CD 6A 7E              0990        DJNZ    R6,$+128
00CF 6A 7D              1000        DJNZ    R6,$+127
00D1 6A 7F              1010        DJNZ    R6,$+129
00D3 6A 00          V   1020        DJNZ    R6,$+130
00D5                    1030 ;END RELATIVE JUMPS
00D5                    1040 ;ERROR CHECKS
00D5 12 44          R   1050        ADC     R4,RR4   ;NO RR
00D7 12 44          R   1060        ADC     RR4,R4
00D9 12 04          R   1070        ADC     R16,R4
00DB D4 E4              1080        CALL    @R4
00DD D6 00 00       U   1090        CALL    RR4
00E0 B1 E4              1100        CLR     @RR4
00E2 80 E4              1110        DECW    R4
00E4 81 E4              1120        DECW    @RR4
00E6 20 1E              1130        INC     30
00E8 21 00          R   1140        INC     @RR4
00EA 0D 00 EA       U   1150        JP      NO,$
00ED E6 E4 64       R   1160        LD      RR4,#100
00F0 E6 64 64       S   1170        LD      #100,#100
00F3 E6 E4 E4       R   1180        LD      R4,@RR4
00F6 C2 E4          R   1190        LDC     30,@RR4
00F8 C2 4E          R   1200        LDC     @RR4,30
00FA 82 E4          R   1210        LDE     30,@RR4
```

```
00FC 82 4E       R    1220       LDE     @RR4,30   Z8-3
00FE 51 E4            1230       POP     @RR4
0100 50 E4            1240       POP     RR4
0102 31 00       V    1250       SRP     3
SYMBOL TABLE
LOOP  005E
```

ALLEN ASHLEY

PROFESSIONAL SOFTWARE FOR PERSONAL USE

The software products listed below are used in over 1500 installations throughout the world. The successful reception of this software is due to the recommendation of users who appreciate the outstanding performance, attractive price, and unparalleled user support. The development software is within the grasp of the beginner (PDS is the basis of dozens of high school computer science courses) and powerful enough to meet the needs of the most demanding programming professional (many commercially available software packages were developed with PDS).

PDS DEVELOPMENT SYSTEM (North Star, CP/M) - $99

PDS is an exceptionally powerful assembly language development system structured to be the most complete, well-rounded system available for microcomputer use. PDS includes:

ASMB	Assembler/Editor	DEBUG	Debug Monitor/Disassembler
MAKRO	Macro Assembler	LINKED	Linkage Editor
EDIT	Text Editor	KWIK	Relocating Loader

MAKRO and ASMB assemble the complete instruction set of the Z-80 and feature mnemonics which are a logical and syntactical extension of the widely familiar 8080 assembly language. The DEBUG module features breakpoint or single-step execution of programs, with trace display of all register contents, flag status, a memory window, and the mnemonics of the instruction just executed and the next instruction to be executed.

The power of PDS derives from the interactive environment afforded by the assembler/editor and the debug package. Program modules can be modified, assembled and checked in seconds under the tight control of trace execution.

While the many features of PDS will satisfy the demands of the most sophisticated programmer, PDS affords an exceptional educational environment for beginning assembly language programmers. The interactive combination of the ASMB editor/assembler and the DEBUG trace program allow the user to witness operation of his program first hand.

Each of the components of PDS is written in the 8080 instruction subset, and the entire system is thus operational on either Z-80 or 8080 machines.

Minimum operating system: 16K RAM and one disk drive. DEBUG, LINKED and KWIK are furnished in relocatable form to satisfy the requirements of individual systems. Full user support is provided by mail or phone.

STAR•TRAC BASIC DEBUG MONITOR (North Star) - $49

Get a handle on your BASIC programs with the STAR*TRAC extension to North Star BASIC 5.1. STAR*TRAC offers the first fully interactive debug monitor for <u>any</u> microcomputer BASIC. STAR*TRAC allows the user to insert a breakpoint in the BASIC program and assume full keyboard control over subsequent execution. Upon reaching the breakpoint, program control is turned over to the STAR*TRAC monitor, which allows execution of any direct mode command. Program variables can be examined or altered before resuming. The BASIC program can then be single stepped, with each program source line and the value of selected variables displayed before execution. The single-step feature of STAR*TRAC extends to multiple commands on a source line: each individual command is executed separately. The breakpoint can be relocated anywhere within the program or invoked after a program command has been executed a specified number of times.

The most powerful feature of STAR*TRAC is the ability to assert a conditional breakpoint: control is assumed whenever a specified logical expression becomes true. Often a faulty program can only be identified by its results -- the portion of the program responsible for the fault cannot be specified. The conditional breakpoint allows control over such a BASIC program to be assumed when a specified program symptom occurs, such as when the value of a variable is altered.

The STAR*TRAC monitor allows complete control over the BASIC program without any modification to the program itself. Neither special diagnostic PRINT statements nor tedious STOP/CONTINUE sequences are required to monitor program evolution -- these features and more are offered by STAR*TRAC.

COMSTAR

NORTH STAR BASIC COMPILER

$400

INCLUDING:
FULL COMPILER FOR NORTH STAR BASIC
RELOCATING MACRO ASSEMBLER
LINKING LOADER
TEXT EDITOR
CONSOLE COMMAND PROCESSOR

FEATURING:
TRANSLATION OF NORTH STAR BASIC PROGRAMS TO MACHINE CODE
OPERATIONAL ON 8Ø8Ø OR Z8Ø
PROGRAMS COMPATIBLE WITH NORTH STAR DOUBLE OR QUAD DENSITY

The COMSTAR compiler translates a North Star type 2 (program) file into an assembly language program and thence into a fully operational machine language program. The resulting programs run faster than their BASIC equivalents and as machine code fully protect the original source BASIC program.

The only major restrictions imposed on the program to be compiled are that only one NEXT is allowed for each FOR, and that variable dimensions and disk file numbers must be decimal constants. Thus DIM A(N) and READ #K are illegal constructs.

Compiled programs typically require substantially more memory than their BASIC equivalents. The increased memory requirement arises partly because of the compilation process and partly because the variable storage areas are included within the compiled program. The enhanced memory requirement is illustrated by the compilation of a 36-block BASIC program which generated a 108-block machine program, somewhat greater than the interpreter and BASIC program combined.

Compiled programs can use either software floating point functions or the North Star floating point board (for a very substantial increase in computational speed).

COMSTAR is available for double or quad density systems only. Neither the compiler nor the compiled programs will read or write single density disks. A dual drive disk system is desirable, and mandatory for large BASIC programs. Systems with one double density disk unit can compile and assemble a BASIC program of approximately 70 blocks maximum. The compiler consumes approximately 12K memory with additional space required for data storage. It is not the compiler but the BASIC program which will define the memory limit.

Programs generated by COMSTAR perform all their I/O through the North Star DOS. COMSTAR is available for DOS located at either 1ØØH or 2ØØØH. Either version of COMSTAR can generate programs for any DOS location.

Complete documentation is included, and full user support is provided by mail or phone.

REGENT

<u>Disk Disassembler</u> ($25): Generates a source file on disk from object program stored in memory. NOT for the casual or novice programmer. (NORTH STAR ONLY.)

EZ•80

<u>Assembly Language Tutorial</u> ($25): FOR the novice programmer. Teaches Z-80 instruction set and operations by executing assembly language commands individually. Registers and flags are displayed for each instruction executed. (NORTH STAR ONLY.)

395 Sierra Madre Villa • Pasadena, California • (213) 793-5748

HDS HYBRID DEVELOPMENT SYSTEM (North Star) - $40

If you use North Star BASIC then you need the HDS hybrid development system. Hybrid programs share the computation between BASIC and assembly language support routines. NOW:

1. Critical program segments may be coded in assembly language to achieve higher speed.
2. Proprietary program segments may be better protected when coded in assembly language.
3. Hybrid programs offer nearly the same execution speed as assembly code while retaining the ease of BASIC program development.
4. Certain operations are much more easily performed at the assembler level.
5. Hybrid programs can use internal BASIC routines for ease of program development.

HDS includes an easy-to-use assembler/editor as well as a roadmap to the internal routines of BASIC and their calling sequence. The HDS system includes modifications to BASIC which allow BASIC programs to utilize assembly support routines to greatly increase execution speed. Most of the operations of BASIC can be called directly from your routines, avoiding the interpretive overhead. With HDS you can extend the capability of BASIC to include such features as graphics output, text formatting, string manipulation, and array processing. Assembly routines can utilize BASIC variables and strings and return results back to BASIC.

The modifications to BASIC give access to the addresses of BASIC variables and extend the CALL function of BASIC to allow an unlimited parameter list. Access to the address of a BASIC variable is gained by enclosing the variable in square brackets. Thus A1 refers to the value of variable A1 while [A1] refers to the location of A1. Examples are provided to:

1. Load an assembly language routine from BASIC using the sequence: P$ = "FILE": Z9 = CALL (ADDR, LOCN, [P$])
2. Find the total of a BASIC array A(N) as: Z9 = CALL (ADDR, [A(1)], [S], N)
3. Find the minimum in a BASIC array as: Z9 = CALL (ADDR, [B], [A(1)], N)

The HDS package includes the ASMB assembler/editor operational at 40H (to be co-resident with BASIC) and complete documentation. As always, full user support is provided by mail or phone.

CROSS ASSEMBLERS (CP/M) - $150

Development software comparable to that offered by the microprocessor manufacturer enables any CP/M system to serve as a development station for the INTEL 8048 series, ZILOG Z-8, RCA COSMAC 1802/1804, and the National COP400 series processors. These development systems feature a macro-assembler, an interactive editor/assembler, and a text editor. With the exception of the instruction set and relocatable code, these components are equivalent to those of PDS.

The development systems share a common operational structure, with uniform procedures for program entry, modification, assembly, and disk file handling. With minor exceptions, the assemblers feature instruction mnemonics and syntax as defined by the processor manufacturers. The macro assembler includes full macro and conditional assembly features as well as the ability to chain a series of source files together during a single assembly.

Programs developed under these systems must be off-loaded to the target processor for test. Facilities are provided to implement the off-loading mechanism as a direct transfer from memory, via a byte stream over a CPU port, or via .COM or .HEX disk files. The development systems currently available are:

SYSTEM-48:	For the INTEL 8048 series	SYSTEM-20 :	For the AMI S2000 series
SYSTEM-18:	For the RCA 1802/1804 processors	SYSTEM-3870:	For the Fairchild F8/3870
SYSTEM-CP4:	For the National COP400 series		
SYSTEM-Z8:	For the ZILOG Z-8 processor		

Each development system is available for $150 on CP/M 8" soft sector (3741), 5" North Star, or 5" Micropolis Mod II (Lifeboat adaptation) diskette with complete documentation.

SOURCE MODULES DEVELOPMENT UTILITIES (NS, CP/M) - $100

To facilitate the development of assembly language application programs, and to encourage the use and sale of PDS, a number of assembly language program modules are available. These source modules are provided to facilitate your development efforts, and no restriction is imposed on their use. Interface requirements are clearly documented.

MODULE	FUNCTION	REQUIREMENTS	PRICE
ALPHSORT	High speed alphabetic sort	None	$ 20
NUMRSORT	High speed numeric sort	None	20
FPPACK	BCD floating point arithmetic	None	20
FOURIER	Fast Fourier transform	FPPACK	20
MINV	Matrix inversion	FPPACK	20
MATPRD	Matrix product	FPPACK	10
RATPOL	Rational function and utilities	FPPACK	15
SQRT	Square root	FPPACK	5
TRIGS	Sine, Cosine, TAN, ATAN	FPPACK, RATPOL	20
LOGEXP	Exponential, logarithm, Y^X	FPPACK, RATPOL	20
FPIOP	Floating point I/O	None	15
FORMAT	Formatted floating point output	None	10
NFILES	North Star buffered disk I/O	None	15

ENTIRE PACKAGE: $100 A LA CARTE: ADD $5 PER ORDER FOR DISK

Your order will be shipped within 24 hours on receipt of your check/money order. If individual source modules are also desired, please list on separate sheet. Dealer discounts are available on all programs.

NAME ____________________

ADDRESS ____________________

____

PDS Program Development System: $ 99 each

North Star Single Density □
North Star Double Density □
CP/M: 8" disk □
5" disk: North Star 10-sector □ Micropolis Mod II □

COMSTAR North Star BASIC Compiler (double or quad density only): $400 each

DOS at 100H □
DOS at 2000H □

CROSS ASSEMBLERS (CP/M only): $150 each

8" disk □
5" disk: North Star 10-sector □ Micropolis Mod II □

SYSTEM-48 (8048) □ SYSTEM-20 (AMI S2000) □
SYSTEM-18 (1802/1804) □ SYSTEM-3870 (F8/3870) □
SYSTEM-CP4 (COP400) □
SYSTEM-Z8 (Z8) □

STAR*TRAC North Star BASIC Debug Monitor:

North Star BASIC 5.1, 5.2 $ 49 □

HDS Hybrid Development System (North Star only) $ 40 □
REGENT Disk Disassembler (North Star only) $ 25 □
EZ-80 Assembly Language Tutorial (North Star only) $ 25 □
SOURCE MODULES (entire package) $ 100 □
SOURCE MODULES (individual, list attached) ___ □

Mail to: Allen Ashley
395 Sierra Madre Villa
Pasadena, CA 91107

Amount Enclosed: ________