XLT86™

8080 to 8086 Assembly Language Translator

USER'S GUIDE

Digital Research, Inc.
P.O. Box 579
801 Lighthouse Avenue
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

COPYRIGHT

DISCLAIMER

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research. CP/M-86, CP/NET, LINK-80, MP/M, MP/M-86, RMAC, XLT86 and PL/I-80 are trademarks of Digital Research. Z80 is a registered trademark of Zilog, Inc.

The XLT86 User's Guide was prepared using the Digital Research TEX-80 Text Formatter and was printed in the United States of America by Commercial Press/Monterey.

First Printing: September 1981

FOREWORD

XLT86™ is a Digital Research software product that aids in the translation of 8080 assembly language programs to equivalent 8086 programs. XLT86 takes the CP/M™ and MP/M™ environment into account, so that translated programs operate properly under both CP/M-86™ and MP/M-86™. XLT86 can also be used as a teaching tool by examining the output when XLT86 is applied to existing 8080 programs. Unlike other 8086 translators, XLT86 uses global data flow analysis techniques to determine 8080 register usage and reduce the number of generated 8086 instructions.

The XLT86 translator is available for operation under CP/M and MP/M for the 8080, 8085, and Z800 microprocessors with a minimum 40K Transient Program Area (TPA). XLT86 requires a 64K CP/M system to effectively translate any significant 8080 programs. Using a 4Mhz Z80 microprocessor, XLT86 translates programs at approximately 120 to 150 lines per minute, depending upon backup storage access speed. XLT86 is written in PL/I-80™ and thus can be adapted for use on computer systems that support Subset G. Specifically, XLT86 is available for cross-development on the Digital Equipment Corporation VAX 11/750 or 11/780 minicomputer, operating with the standard DEC VMS software. However, programs are supplied in machine code form, so it is not necessary to own PL/I-80 or any of its subsystems to operate XLT86.

The XLT86 system components, including the files XLT86.COM, XLTOO.OVL, and XLT01.OVL, are distributed in IBM-compatible single density disk form. Before operating XLT86, copy these system components to a working disk and save the distribution disk for archive purposes. If the working disk medium can be dismounted, it must be marked with the notice shown below to properly comply with the Software License Agreement:

This User's Guide presents the overall translation process, along with operator interface and command syntax. This manual also describes the format of the translated program, including the details of the 8080 to 8086 operation code translation.

TABLE OF CONTENTS

Section 1
THE TRANSLATION PROCESS

1.1 Input and Output Files

XLT86 reads an 8080 program from a file with type ASM and produces a file of type A86 containing the equivalent translated 8086 assembly language program. The filename for the 8080 source program, as well as filenames for all output files from XLT86, is taken from the command line typed by the operator. For example, the console command:

```
XLT86 DUMP
```

executes the XLT86 program using the file "DUMP.ASM" as input. The translation produces the output file "DUMP.A86".

The 8080 source program must be in a form acceptable to the standard Digital Research assembly language translators ASM, MAC, or RMAC. XLT86 processes conditional assembly statements, and produces an output program that results from evaluation of the particular conditions included in the 8080 program. However, macro definitions, macro invocations, and repeat loops are not altered in the translation. To properly translate programs that include macros or repeat loops, first assemble the programs under MAC or RMAC to produce a printer listing file of type PRN. Rename this PRN file to type ASM and edit the file to remove the beginning column positions, resulting in a file acceptable as input to XLT86. The A86 output file is now in a form acceptable to the Digital Research ASM86 assembler, requiring little or no modification for execution under CP/M-86 and MP/M-86.

XLT86 produces two additional files: a PRN file and a $$$ file. A file of type PRN contains error lines and messages along with optional listing and trace information. The PRN file is in a form suitable for listing on the system printer and contains embedded form-feed and tab characters. A temporary file of type $$$ is also created during translation. This temporary file is automatically deleted upon normal completion of XLT86.

The XLT86 program consists of a "root module" called XLT86.COM, which is loaded and executed when you enter the XLT86 command line shown above. There are two additional "overlays" called XLTOO.OVL and XLT01.OVL that must be present on your default disk drive. These two overlays are automatically loaded and executed at the appropriate time during the translation.

1.2 Translation Phases

The translation itself takes place in five phases. Each phase has a specific name that appears at the console during translation so that the operator can monitor the progress of XLT86. Table 1-1 lists the phase names.

Phase	**Meaning**
Symbol Setup	Determines the location of each symbol in the 8080 source program.
Setup Blocks	Determine the "Basic Blocks" necessary for the data flow analysis.
Join Blocks	Construct a "Directed Graph" connecting each basic block, corresponding to program flow of control.
List Blocks	Produce an optional list of Basic Blocks following flow analysis showing register and flag usage for each 8080 instruction.

Phase	Meaning
Translate-86	Translates the 8080 instructions to 8086 form, using the information gathered by the flow analysis.

Table 1-1. XLT86 Translation Phases

The command line:

```
XLT86 DUMP
```

activates the XLT86 translator using the DUMP.ASM program as input. The default action of XLT86 prints the name of each phase at the console as the translation proceeds, as shown below.

Symbol Setup
Setup Blocks
Join Blocks
List Blocks
Translate-86

The files processed by the "XLT86 DUMP" command are shown in Figure 1-1, below.

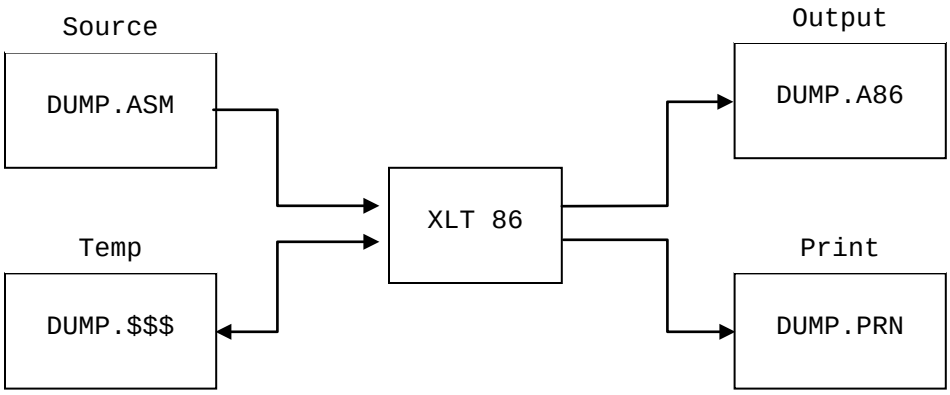

Figure 1-1. Processed Files

All files are placed on the drive specified by the operator as the prefix on the source filename. In the above example, all files are placed on the current default drive, which must also contain the XLT86 program along with its overlays. An alternative form:

```
XLT86 B:DUMP
```

overrides the default drive and obtains the source file from drive B. XLT86 creates the output, temporary, and print files on drive B as well. When several drives are available, it may be advantageous to place the various files on separate disks. In this case, you must use XLT86 "parameters," described in the following section, to override the default values.

Section 2
TRANSLATION PARAMETERS

2.1 Parameter Syntax

Several XLT86 parameters can be included in the command line by the operator or embedded within the 8080 source program to control the translation process. Parameters are grouped together into a parameter list enclosed within square brackets:

```
[p1 p2 … pn]
```

where p1 through pn denote one or more parameters optionally separated by blanks. When included on the command line, the XLT86 invocation appears as follows:

```
XLT86 filename [p1 p2 … pn]
```

When included within the source program, the opening bracket of the parameter list must begin in the first column position. The parameters denoted by p1 through pn are one or two character sequences in upper- or lower-case, with optional intervening blanks, as listed in Table 2-1, below.

Parameter	Meaning
Ax	Place the A86 file on drive x where x = A, B, P.
B	Produce a list of Basic Blocks in the PRN file.
C	Assume the 8086 "compact model" for execution.
J	Translate conditional jumps to short conditionals.
L	Send the PRN file directly to the system printer.
N	Show the line and statement number being processed.
P	Place the PRN file on drive x where x = A, B, P.
R	Assume all flags active at subroutine returns.
S	Assume non-overlapping 8086 code and data segments.
T	Place the $$$ file on drive x where x = A, B, … P.
80	Create an 8080 assembly listing in the PRN file.
86	Create an 8086 line and statement listing.

Table 2-1. Translation Parameters

2.2 The B (Block Trace) Parameter

The A (A86), P (PRN), and T (TMP) parameters allow you to select alternate disk drives for use during the translation process when only limited disk space is available on each drive. Otherwise, disk drives are selected as described above.

The B (Block Trace) parameter provides a trace in the PRN file showing register usage information collected by the data flow analyzer. This parameter is not normally selected since the trace information is of no particular value unless you are interested in detailed register usage. The B parameter trace consists of a sequence of register usage tables for each Basic Block in the form shown below.

```
Block At 011E (subr), A86 = 083F
        Entry Active: B-D-HL-AOZSPI Exit Active: BCDEHL-------
```

stmt#	opcode uses	op	v1	v2	opcode kills	live regs
23	-------AOZSPI	PUSH	PSW		----------	B-D-HL-A-----
24	-------A-----	MOV	E	A	---E-------	B-DEHL-------
25	----------	MVI	C	05	-C---------	BCDEHL-------
26	----------	CALL	0005		----------	BCDEHL-------

The Basic Block address in the original 8080 program is listed and the type of block is identified. The block type is "subr" for subroutines, "code" for main-line code, and "data" for data blocks. The A86 address is an approximation of the corresponding 8086 address used to determine short and long branch jump ranges. The remaining information shows register and flag use at block entry and at each instruction within the block. The registers and flags are displayed as a vector of letters and hyphens, where each letter represents the presence of a register or flag in the display, and each hyphen signifies that the corresponding register or flag is absent in the vector. Given that all registers and flags are present, the display appears as follows:

```
BCDEHLMAZOSPI
```

Table 2-2 lists the letter denotations of the above display.

Letter	**Meaning**
B	Register B, or high(BC)
C	Register C, or low (BC)
D	Register D, or high(DE)
E	Register E, or low (DE)
H	Register H, or high(HL)
L	Register L, or low (HL)
M	Register M, memory operand
A	Register A, 8-bit Accumulator
0	overflow Flag, carry or borrow
Z	Zero Flag
S	Sign Flag
P	Even Parity Flag
I	Interdigit Carry Flag

Table 2-2. Letter Denotations for Registers and Flags

The registers active upon entry are listed first. In the example shown above, the data flow analysis has determined that the B, D, and HL registers, along with all flag registers, are in use upon entry to the block. The active registers following this block are then listed, consisting of the BC, DE, and HL register pairs. Then each instruction in the Basic Block is given, with a preceding statement number that can be cross-referenced with the 8080 source program. The instruction itself is listed with the hexadecimal values of its two optional parameters.

The "opcode uses" field shows the register set used by the operation code, while the "opcode kills" field lists the registers destroyed by the operation. The "live registers" field provides the information used by the Translate-86 phase to minimize the generated code. This field lists the registers and flags that are referenced following the instruction and is derived by examining the Directed Graph corresponding to the 8080 source program. Again, the information collected by the flow analyzer is optionally displayed using the B parameter. This display is not required for normal operation of the translator.

2.3 The C (Compact) Parameter

The C (Compact) parameter causes XLT86 to generate 8086 machine code using the "Compact Memory Model" described in the CP/M-86 System Guide. Under normal circumstances, XLT86 assumes the "8080

Memory Model" where code and data segments overlap. To accomplish this overlap of segments, the program is analyzed to determine Basic Blocks that contain code and data. The program is assumed to begin with a code segment and, if a data segment is encountered as defined by a sequence of DS, DB, or DW statements, XLT86 produces the following statements that provide the proper transition:

```
L@n    EQU    $
       DSEG
       ORG    Offset L@n
```

Similarly, the transition from a data segment back to a code sequence is marked by the generated statements:

```
L@n    EQU    $
       CSEG
       ORG    Offset L@n
```

where L@n is a sequentially generated label. The labels are generated as required by XLT86, taking the form:

```
L@1    L@2    L@3    L@4    …      L@32767
```

Enabling the "C" parameter prevents the code and data segments from overlapping. In this case, the transition from code to data and data to code is marked by either DSEG or CSEG respectively. See also the description of the S (Segments) parameter. When enabled, the S parameter completely overrides the C parameter.

2.4 The J (Jump) Parameter

The J (Jump) parameter enables the short jump analysis option of XLT86. When enabled, XLT86 translates 8080 conditional jumps to either short conditional jumps or negated short conditional jumps followed by short unconditional jumps, depending upon the byte count to the target of the jump. That is, a "JZ x" instruction becomes either

```
JZ x
```

```
X.
```

or

```
JNZ    L@n
JMPS   x
L@n:
```

The first case results if the label "x" is within the range of a short jump, while the second form results from a target label beyond the range of a short jump. The J parameter is enabled by default, and should be disabled using the NOJ form described below only if you want to manually edit your conditional jumps following program translation.

2.5 The L (List) Parameter

The L (List) parameter sends the listing file directly to the system printer, thus avoiding the intermediate PRN file. The system printer, or printer driver, must handle form-feeds (ctl-L) and tabs (ctl-I) to every eighth column position. If your printer does not properly support these characters, you can leave the L parameter disabled and use the CP/M PIP utility command form:

```
PIP LST:=filename.PRN[T8F]
```

where the PIP parameter "TV expands tabs to blanks at every eighth column, and the "F" parameter deletes the form-feed character on transmission.

2.6 The N (Number) Parameter

The N (Number) parameter displays the current line and statement number on CRT-type console devices as the translation proceeds. Each line and statement number is displayed with an intervening carriage return, without a line feed, so that each successive display overwrites the previous value. In this way, you can easily monitor the progress of XLT86 as it proceeds through the source program during the translation.

2.7 The R (Return) Parameter

The R (Return) parameter overrides the default assumptions about register usage at the end of a subroutine. XLT86, by default, assumes that all registers are in use at the end of a subroutine in the absence of additional information. This is a safe, but possibly restrictive, assumption that might cause more 8086 code to be generated near the return statements of each subroutine. If you know that the entire 8080 program being translated contains subroutines that do not return flag registers, then you should include the R parameter in the command line to reduce the amount of generated code.

Alternatively, you can precede the return statements of various subroutines with "[R]" parameters when they do not return flag registers, as long as balancing "[NOR]" parameters, described below, are included to return to the default assumptions, where necessary.

2.8 The S (Segment) Parameter

The S (Segment) parameter informs XLT86 that the original source program contains embedded CSEG and DSEG directives that delimit the code and data segments. In this case, XLT86 makes no attempt to de-rive the code and data segment information and, instead, assumes that the CSEG and DSEG directives passed through to the 8086 program correctly define the appropriate segments. The S parameter is automatically set when the source program contains ASEG, CSEG, or DSEG directives, and completely overrides the effect of the C (Compact) parameter.

2.9 The 80 Parameter

The 80 parameter causes XLT86 to produce a pseudo-assembly listing of the original 8080 source program, giving the source line and statement number along with the assembled machine code location. If the B parameter is simultaneously enabled, additional Basic Block information precedes each straight-line code segment. When both 80 and B are enabled, the trace appears as shown below:

```
--------- Basic Block (2) 011E
              Predecessors:  0119 0111 0105 0100
              Successors  :  0125
              Reg's Killed:  -C-E --------
              Reg's Used  :  ------- AOZSPI
         22          22  011E        pr:
         23          23  011E                    push        psw
         24          24  011F                    mov         e,a
         25          25  0120                    mvi         c,lst
         26          26  0122                    call        bdos
```

Each Basic Block of the listing is preceded by the Basic Block Header consisting of the location (011E in the example above) , a set of predecessor blocks where the program flow of control comes from (0119, 0111, 0105, and 0100), and a set of successor blocks where program flow could continue (0125, above). The set of registers killed are listed, along with the set of registers used by the operation codes within the block. No global data flow information is displayed in this trace (see the B parameter described earlier).

2.10 The NO Parameter

The two character sequence "NO" preceding the B, C, J, L, N, R, S, 80, and 86 parameters negates the effect of the parameter once it has been set. Further, the A, P, and T parameters are ignored when they

occur within the source file and are effective only on the command line. The parameters B, C, J, L, N, R, S, 80, and 86 parameters, along with their negated forms, can occur in the command line or within the source program. When they occur within the program, they apply to the segment of code following their occurrence. Assuming that the default drive is d, where d is a valid drive code A, B, … or P, the default values assumed for each parameter are identical to the complete, but redundant, command line shown below:

```
XLT86 d:filename [Ad NOB NOC J NOL NON Pd NOS NOR Td N080 N086]
```

Section 3
TRANSLATED PROGRAM FORMAT AND CONTENT

3.1 Translated Program Format

XLT86 constructs the 8086 program from the original 8080 program by first analyzing the program register usage. Then, using the collected information, XLT86 translates each label, operation code, and operand expression into an equivalent 8086 program segment. In performing the translation, XLT86 uses as many program fragments from the original 8080 source program as possible. These program fragments include labels, expressions, and comment fields. Due to differences in assembly language formats, however, labels and expressions might be altered somewhat to maintain their original meaning.

The translation occurs line-by-line, where each 8080 source line may contain several statements delimited by exclamation symbols. XLT86, however, always generates a single statement per output line. The output line includes an optional label in column one, followed by a single tab character. The translated operation code field is placed immediately following the tab character. If the operation code has one or two operand fields, another tab character is included and the operand fields are inserted. The operand fields themselves are constructed by either translating 8080 registers to their 8086 equivalents, or through the construction of an expression that is the translation of the original form. If a comment field is present in the source program, it is copied to the 8086 program intact with sufficient leading tabs to position the comment to column forty, if that position has not already been reached. Comments beginning in column one are reproduced without leading tab characters. Further, comments that begin in column one with the character "*" are started, instead, by the two character sequence ";*" to maintain compatibility with ASM-86.

For pseudo-assembly purposes, the assumed origin of the 8080 program is 0100H, corresponding to the base of the TPA under CP/M. This assumed origin resolves label addresses during pseudo-assembly and does not normally affect the translation process. However, if an ORG statement is encountered at the beginning of the program before any code or data is encountered, the program origin is set to the value given in the operand field of the ORG statement.

Program-relative operand references, along with absolute addresses, are allowed in the source program. In this case, XLT86 generates a label of the form "L@n" at the target location. For example below, the 8080 instruction sequence shown to the left results in the 8086 program shown to the right:

```
NOP   L@l:  NOP
NOP         NOP
JMP $-2     JMPS L@l
```

Similarly, the absolute 8080 assembly language shown to the left below results in the program shown to the right:

```
        ORG     300H
        NOP
        NOP
        JMP     300H
```

```
        ORG     300H
L@l:    NOP
        NOP
        JMP     L@l
```

In this case, the ORG statement is necessary to override the default assumption.

From this last example, it appears that XLT86 is capable of translating 8080 programs produced through disassembly. Unfortunately, disassemblers cannot generally distinguish between code and data areas. If the code and data sections can be separated into distinct areas, where the code is disassembled with absolute address operands and the data areas consist of DS, DB, and DW operations, then XLT86 performs the translation.

Operand fields are translated according to their context and, for notational purposes, we make the following definitions.

Abbreviation	Definition	Example
ib	immediate byte operand	(MVI A,ib)
iw	immediate word operand	(LXI H,iw)
mb	byte in memory	(STA x)
mw	word in memory	(LHLD x)
mn	near memory	(CALL x)
rb	byte in register	(ADD B)
rw	word in register	(DAD B)

Table 3-1. Operand Field Abbreviations

The translation of an expression is denoted by a prime following the expression type. Thus, ib is translated to ib' , iw to iw', and so forth. Register translation takes place according to the following table.

8080 Register (rb)	8086 Register (rb')
A	AL
B	CH
C	CL
D	DH
E	DL
H	BH
L	BL

Table 3-2. Register Translation

The M (Memory) register has no direct equivalent in the 8086 environment, so XLT86 produces an "equate" statement in the following form at the beginning of each program.

```
M EQU Byte Ptr 0[BX]
```

Thus, the M register remains unchanged in the translation with the assumption that the BX register contains the offset to the proper memory location.

The 16-bit register pair translation occurs as shown in Table 3-3, below.

8080 Register (rw)	1 8086 Register (rw')
PSW	AX
B	CX
D	DX
H	BX
SP	SP

Table 3-3. 16-Bit Register Translation

The 8080 PSW and 8086 AX register have a loose correspondence depending upon register usage at the time of translation. The exact correspondence is defined below under the PUSH and POP operators.

3.2 Translated Program Content

Expressions are normally composed of literal constants, data variable references, program label references, and register references. XLT86 computes the type of each expression as the translation proceeds, resulting in one of the following expressions.

Expression	Meaning
constant	consists only of literal constants
variable	consists of zero or more constants and one or more variable references

Expression	Meaning
label	consists of zero or more constants or variables, and one or more labels
register	consists of zero or more constants, variables, or labels and one or more register references

Table 3-4. Expressions

The translation of ib, iw, mb, mw, and mn is described in Table 3-5, below. This translation takes place after XLT86 scans the expression to determine its type, as described above.

Operand Field	Translation
ib and iw	ib' and iw' are constructed from the original ib and iw by first determining the expression type. If the type is "constant," the expression ib or iw remains unchanged in the translation. Otherwise, for each variable, label, dollar sign ($) , or register reference in the expression, XLT86 changes the reference, denoted by x, to " (offset x) " so that the resulting expression ib' or iw' represents a CS or DS relative offset computation.
mb	The resulting expression mb' is constructed from the original expression mb according to the type of mb. If mb is "constant" then mb' becomes "Byte Ptr mb" denoting a single byte operand located at a literal constant address relative to DS or CS. Otherwise, the expression mb' becomes "Byte Ptr mb" denoting a byte variable or label address.
mw	Similar to mb, mw1 becomes "Byte Ptr .mw" if mw is "constant" and "Byte Ptr mw" otherwise.
mn	The expression mn' is the same as the original mn unless there is no literal label at the target address. In this latter case, a label of the form "L@n" is created at the target address, which becomes the value of mn

Table 3-5. Operand Field Translation

Due to differences in 8080 and 8086 program formulation requirements, not all valid 8080 expressions can be successfully converted to valid 8086 expressions. Thus, you must be aware that additional editing is required if your translated program produces errors during assembly with ASM-86. In particular, expressions that use arbitrary operations upon constants, variables, labels, and registers are unlikely to assemble correctly under ASM-86, or any other assembler that uses the Intel conventions.

In the translation table given below, the 8080 operation code is shown to the left, with the translated 8086 code sequence shown to the right. In many cases, the registers that are live at the point where the 8080 operation code occurs determine the exact sequence of code that is generated. In these cases, the alternative forms are given separately. Conditional assembly notation specifies the alternative forms, with the introduction of the following two pseudo-functions:

```
live(rl,r2, …, rn)
```

and

```
short(mn')
```

The "live" function takes a variable number of register arguments and results in a TRUE value if one or more of these registers is live at the point of translation. Otherwise, the "live" function results in a FALSE value. In the Section 2 example for the B parameter, statement 24 (MOV E,A) has the live register set given by the vector:

```
B-DEHL -------
```

so that

```
live(B,C,D) = TRUE      and   live(A,O) = FALSE
```

The "short" function is used in the translation of conditional jump instructions where the value of short(mn') is TRUE if the target of the translated jump address mn' is within the range of a conditional jump, or if the "J" parameter is enabled. otherwise, short(mn') results in a FALSE value. XLT86 also uses the notation in Section 2 for label generation. The form "L@n" represents labels produced sequentially, starting at n = 1, used in the translation of conditional calls, returns, and conditional jumps outside the range of an 8086 conditional transfer. The CC (Call if Carry) operator, for example, translates to a jump conditional to a generated label followed by a direct call. The generated label is then inserted, as shown in the expansion of the 8080 instruction CC SUBR:

```
JNB L@1
CALL SUBR
L@l:
```

Table 3-6 gives the translation of each operation code. Note in particular that the following BDOS entry operations:

```
CALL 0      CALL 5      imp 0 JMP 5
```

are treated as special cases that are translated to Interrupt 224, reserved by Intel Corporation for entry to CP/M-86 and MP/M-86.

Operation Code	**Translation**
ACI ib	ADC AL,ibl
ADC rb	ADC AL,rb'
ADD rb	ADD AL,rb'
ADI ib	ADD AL,ib
ANA rb	AND AL,rb'
ANI rb	AND AL,rb'

CALL 0	MOV CL,O
	MOV DL,O
	INT 224

CALL 5	INT 224
CALL mn	CALL mn'
CC mn	JNB L@n CALL mn' L@n:
CM mn	JNS L@n CALL mn' L@n:
CMA	NOT AL
CMC	CMC
CMP rb	CMP AL,rb'
CNC mn	JNAE L@n CALL mn' L@n:
CNZ mn	JZ L@n CALL mn' L@n:

Operation Code	Translation
CP mn	JS L@n CALL mn' L@n:
CPE mn	JNP L@n CALL mn' L@n:
CPI ib	CMP AL,ib
CPO mn	JP L@n CALL mn' L@n:
CZ mn	JNZ L@n CALL mn' L@n:
DAA	DAA

DAD rw	IF rw = H
	SHL BX,1
	ELSE
	IF live(O) AND NOT live(Z,S,P,I)
	ADD BX,rw1
	ELSE
	IF NOT live(O) AND live(Z,S,P,I)
	LAHF
	ADD BX,rw'
	SAHF
	ELSE
	LAHF
	ADD BX,rw'
	RCR SI,l
	SAHF
	RCL SI,l
	ENDIF
	ENDIF
	ENDIF

DEC rb	DEC rb1
DCX rw	DEC rwl
DI	CLI
EI	STI
HLT	HLT
IN ib	IN AL,ibl
INR rb	INC rb'

INX rw	IF NOT live(Z,S,P,I)
	INC rw'
	ELSE
	LAHF
	INC rw'
	SAHF
	ENDIF

JC mn	IF short(mn')
	JB mn'
	ELSE
	JNB L@n
	JMPS mn'
	L@n:
	ENDIF

Operation Code	Translation

JM mn	IF short(mn')
	JS mn'
	ELSE
	JNS L@n
	JMPS mn'
	L@n:
	ENDIF

imp 0	MOV CL,O
	MOV DL,O
	INT 224
	RET

imp 5	INT 224 RET
JMP mn	JMPS mn'

JNC mn	IF short(mn')
	JNB mn'
	ELSE
	JNAE L@n
	JMPS mn'
	L@n:
	ENDIF

JNZ mn	IF short(mn')
	JNZ mn'
	ELSE
	JZ L@n
	JMPS mn'
	L@n:
	ENDIF

JP mn	IF short(mn')
	JNS mn'
	ELSE
	JS L@n
	JMPS mn'
	L@n:
	ENDIF

JPE mn	IF short(mn')
	JPE mn'
	ELSE
	JNP L@n
	JMPS mn'
	L@n:
	ENDIF

JPO mn	IF short(mn')
	JPO mn'
	ELSE
	JP L@n
	JMPS mn'
	L@n:
	ENDIF

Operation Code	Translation

JZ mn	IF short(mn')
	  JZ mn'
	ELSE
	  JNZ L@n
	  JMPS mn'
	  L@n:
	ENDIF

LDA mb	MOV AL,mb'
LDAX rw	MOV SI,rw1 MOV AL,[SI]
LHLD mw	MOV BX,mw'
LXI rw,iw	MOV rwl,iwl
MOV rbl,rb2	MOV rbl',rb2l
MVI rb,ib	MOV rb',ib'
NOP	NOP
ORA rb	OR AL,rb'
ORI ib	OR AL,rb'
OUT ib	OUT ib',AL
PCHL	JMP BX

POP rw	POP rwl
	IF rw = PSW AND live(OIZIS'Pli)
	  XCHG AL,AH
	  SAHF
	ELSE
	  IF rw = PSW AND live(A)
	  XCHG AL,AH
	  ENDIF
	ENDIF

PUSH rw	IF rw = PSW AND live(A)
	  LAHF
	  XCHG AL,AH
	  PUSH AX
	  XCHG AL,AH
	ELSE
	  if rw = PSW
	  LAHF
	  XCHG AL,AH
	  PUSH AX
	  ELSE
	  PUSH rwl
	  ENDIF
	ENDIF

RAL	RCL AL,l
RAR	RCR AL,l
RC	JNB L@n RET L@n:
RET	RET
RLC	ROL AL,l
RM	JNS L@n RET L@n:

Operation Code	Translation
RNC	JNAE L@n RET L@n:
RNZ	JZ L@n RET L@n:
RP	JS L@n RET L@n:
RPE	JNP L@n RET L@n:
RPO	JP L@n RET L@n:
RRC	ROR AL,l
RST ib	INT ibl
RZ	JNZ L@n RET L@n:
SBB rb	SBB AL,rb'
SBI ib	SBB AL,ibl
SHLD mw	MOV mw',BX
SPHL	MOV SP,BX
STA mb	MOV mb',AL
STAX rw	MOV DI,rw1 MOV [DI1,AL
STC	STC
SUB rb	SUB AL,rb'
SUI ib	SUB AL,ibl
XCHG	XCHG BX,DX
XRA rb	XOR AL,rb'
XRI ib	XOR AL,ib
XTHL	MOV BP,SP XCHG BX,[BPI

Table 3-6. Translation Table

Section 4
XLT86 ERROR MESSAGES

4.1 Pseudo-assembly Process Error Messages

XLT86 issues error messages that fall into two categories: those produced by the pseudo-assembly process, and those produced during translation. Errors in the first category are not considered fatal, but are simply annotated in the source listing file following the line in which the error occurs. If errors are present, the message:

```
Number of Errors: n
```

is displayed at the console following the pseudo-assembly. Examine the PRN file to determine if the errors are significant. Error messages take the form:

```
** Error: e **, Near t
```

where e is one of the error codes, and t is a program element near the position where the error occurred. Table 4-1 lists the error codes.

Error Code	Meaning
Bad Flag	invalid parameter list [pl pn]
Balance	Unmatched right parenthesis or missing trailing string quote.
Boundary	Invalid program boundary, usually results from a branch to the middle of an instruction.
Convert	Cannot convert an operand to internal form.
End-Line	The end of a program line contains extraneous characters.
Exp Ovfl	Expression stack overflow; the expression is nested too deeply.
Gtr 7	An expression produced a value greater than 7, where a value from 0-7 is required.
Gtr 255	An expression produced a value greater than 255, where a value from 0-255 is required.
Mov M,M?	The source line contains the invalid instruction MOV M,M.
No Comma	Missing comma where comma is required.
No Value	A label or variable was encountered that does not have an assigned value.
Not Impl	The instruction or directive is not implemented in XLT86.
Phase	A label or variable has a different value on two passes through the source program.
Str Len	A string was encountered that exceeds the capacity of XLT86, check for missing right quote mark.
Value	The value produced by an expression is not compatible with the context in which it occurs.

Table 4-1. XLT86 Error Codes

4.2 Translate-86 Error Messages

The Translate-86 phase also produces a limited number of error messages. All errors produced by this phase are fatal, and cause immediate termination of XLT86. Table 4-2 lists these error messages.

Error Message	Meaning
Bad Oper	Invalid 8080 operation code was encountered during translation; probably due to bad disk 1/0 operation. Check for hardware controller faults.
Not BDOS	A CALL or JMP occurred below the base origin of the program where the target is not OOOOH (warm boot) or0005H (BDOS entry).
Phase (B)	The Directed Graph does not correspond to the source program at the Basic Block level; usually due to a hardware malfunction.
Phase (S)	The Directed Graph does not correspond to the source program at the statement level; usually due to a hardware malfunction

Table 4-2. Translate-86 Error Messages

An error produced by Translate-86 is accompanied by the console error message:

```
Fatal Error (See PRN file)
```

to indicate that such an error occurred.

4.3 Memory overflow

The XLT86 program occupies approximately 30K bytes of main memory. The remainder of memory, up to the base of CP/M, stores the program graph that represents the 8086 program being translated. The error message:

```
ERROR (7) "Free Space Exhausted"
```

is issued if the program graph exceeds available memory. A 64K CP/M system allows translation of 8080 programs of up to approximately 6K.

The above error causes XLT86 to terminate. To continue, you must divide your source program into smaller modules and retry the translation.

Appendix A
SAMPLE PROGRAM TRANSLATIONS

The DUMP.ASM program presented here and normally included as a sample assembly language program with CP/M illustrates the translation process. The XLT86 command line:

```
XLT86 DUMP [80 86]
```

produces the first example shown below. The "80" parameter selects the 8080 program listing option, while the "86" parameter selects the 8086 listing option. XLT86 places full lines of dashes (" ---- 11) between the Basic Blocks in the 8080 listing. This translation of the DUMP program, however, requires modification to run under CP/M 86. In particular, the DUMP.ASM program contains initialization code that saves the entry SP (statements 34 to 37) and resets the SP to a local stack (statement 39). The return statement following the FINIS label (statement 95) returns control to the CCP.

To perform an exactly equivalent sequence of operations, you must also save the stack segment register (SS) upon entry to the DUMP program, and restore this value before executing the return. Further, the simple RET operation must be replaced by a Far Return (RETF) to balance the original Far Call from the CCP. A simpler solution is to eliminate the initialization code (statements 33 through 39) and use the CCP's built-in 96 byte stack. Control returns to the CCP by executing a RETF at statement 95. If you want to use a local stack, set the SS register to the value of DS upon entry, and set SP to the Offset of STKTOP. Control returns to the CCP through execution of function call #0 in place of the RET in statement 95, as follows:

```
MOV CL,O
MOV DL,O
INT 224
```

The second listing shows the Basic Block information collected by the flow analyzer, and produced by the command line:

```
XLT86 DUMP [B]
```

where the "B" parameter selects the Basic Block trace. Under normal circumstances, either of the commands shown below are sufficient and reduce the amount of trace information:

```
XLT86 DUMP [N]
```

or

```
XLT86 DUMP
```

The first command is used only with a CRT-type device where the carriage-return character does not cause an automatic line-feed (see the description of the "N" parameter).

```
------------------------------------------------------------------
       1       1 0100   ;     FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
       2       2 0100   ;
       3       3 0100   ;     COPYRIGHT (C) 1975, 1976, 1977, 1978
       4       4 0100   ;     DIGITAL RESEARCH
       5       5 0100   ;     BOX 579, PACIFIC GROVE
       6       6 0100   ;     CALIFORNIA, 93950
       7       7 0100   ;
       8       8 0100         ORG     100H
       9       9 0100   BDOS  EQU     0005H   ;DOS ENTRY POINT
      10      10 0100   CONS  EQU     1       ;READ CONSOLE
      11      11 0100   TYPEF EQU     2       ;TYPE FUNCTION
      12      12 0100   PRINTF        EQU     9       ;BUFFER PRINT ENTRY
      13      13 0100   BRKF  EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
      14      14 0100   OPENF EQU     15      ;FILE OPEN
      15      15 0100   READF EQU     20      ;READ FUNCTION
      16      16 0100   ;
      17      17 0100   FCB   EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
      18      18 0100   BUFF  EQU     80H     ;INPUT DISK BUFFER ADDRESS
      19      19 0100   ;
      20      20 0100   ;     NON GRAPHIC CHARACTERS
      21      21 0100   CR    EQU     0DH     ;CARRIAGE RETURN
      22      22 0100   LF    EQU     0AH     ;LINE FEED
      23      23 0100   ;
      24      24 0100   ;     FILE CONTROL BLOCK DEFINITIONS
      25      25 0100   FCBDN EQU     FCB+0   ;DISK NAME
      26      26 0100   FCBFN EQU     FCB+1   ;FILE NAME
      27      27 0100   FCBFT EQU     FCB+9   ;DISK FILE TYPE (3 CHARACTERS)
      28      28 0100   FCBRL EQU     FCB+12  ;FILE'S CURRENT REEL NUMBER
      29      29 0100   FCBRC EQU     FCB+15  ;FILE'S RECORD COUNT (0 TO 128)
      30      30 0100   FCBCR EQU     FCB+32  ;CURRENT (NEXT) RECORD NUMBER (0 TO 127)
      31      31 0100   FCBLN EQU     FCB+33  ;FCB LENGTH
      32      32 0100   ;
      33      33 0100   ;     SET UP STACK
      34      34 0100         LXI     H,0
      35      35 0103         DAD     SP
      36      36 0104   ;     ENTRY STACK POINTER IN HL FROM THE CCP
      37      37 0104         SHLD    OLDSP
      38      38 0107   ;     SET SP TO LOCAL STACK AREA (RESTORED AT FINIS)
      39      39 0107         LXI     SP,STKTOP
      40      40 010A   ;     READ AND PRINT SUCCESSIVE BUFFERS
      41      41 010A         CALL    SETUP   ;SET UP INPUT FILE
------------------------------------------------------------------
      42      42 010D         CPI     255     ;255 IF FILE NOT PRESENT
      43      43 010F         JNZ     OPENOK  ;SKIP IF OPEN IS OK
------------------------------------------------------------------
      44      44 0112   ;
      45      45 0112   ;     FILE NOT THERE, GIVE ERROR MESSAGE AND RETURN
      46      46 0112         LXI     D,OPNMSG
      47      47 0115         CALL    ERR
------------------------------------------------------------------
      48      48 0118         JMP     FINIS   ;TO RETURN
------------------------------------------------------------------
      49      49 011B   ;
      50      50 011B   OPENOK:       ;OPEN OPERATION OK, SET BUFFER INDEX TO END
      51      51 011B         MVI     A,80H
      52      52 011D         STA     IBP     ;SET BUFFER POINTER TO 80H
      53      53 0120   ;     HL CONTAINS NEXT ADDRESS TO PRINT
      54      54 0120         LXI     H,0     ;START WITH 0000
------------------------------------------------------------------
      55      55 0123   ;
      56      56 0123   GLOOP:
      57      57 0123         PUSH    H       ;SAVE LINE POSITION
      58      58 0124         CALL    GNB
------------------------------------------------------------------
      59      59 0127         POP     H       ;RECALL LINE POSITION
      60      60 0128         JC      FINIS   ;CARRY SET BY GNB IF END FILE
------------------------------------------------------------------
      61      61 012B         MOV     B,A
      62      62 012C   ;     PRINT HEX VALUES
      63      63 012C   ;     CHECK FOR LINE FOLD
```

```
      64      64 012C         MOV     A,L
      65      65 012D         ANI     0FH     ;CHECK LOW 4 BITS
      66      66 012F         JNZ     NONUM
-----------------------------------------------------------------
      67      67 0132   ;     PRINT LINE NUMBER
      68      68 0132         CALL    CRLF
-----------------------------------------------------------------
      69      69 0135   ;
      70      70 0135   ;     CHECK FOR BREAK KEY
      71      71 0135         CALL    BREAK
-----------------------------------------------------------------
      72      72 0138   ;     ACCUM LSB = 1 IF CHARACTER READY
      73      73 0138         RRC             ;INTO CARRY
      74      74 0139         JC      FINIS   ;DON'T PRINT ANY MORE
-----------------------------------------------------------------
      75      75 013C   ;
      76      76 013C         MOV     A,H
      77      77 013D         CALL    PHEX
-----------------------------------------------------------------
      78      78 0140         MOV     A,L
      79      79 0141         CALL    PHEX
-----------------------------------------------------------------
      80      80 0144   NONUM:
      81      81 0144         INX     H       ;TO NEXT LINE NUMBER
      82      82 0145         MVI     A,' '
      83      83 0147         CALL    PCHAR
-----------------------------------------------------------------
      84      84 014A         MOV     A,B
      85      85 014B         CALL    PHEX
-----------------------------------------------------------------
      86      86 014E         JMP     GLOOP
-----------------------------------------------------------------
      87      87 0151   ;
      88      88 0151   FINIS:
      89      89 0151   ;     END OF DUMP, RETURN TO CCP
      90      90 0151   ;     (NOTE THAT A JMP TO 0000H REBOOTS)
      91      91 0151         CALL    CRLF
-----------------------------------------------------------------
      92      92 0154         LHLD    OLDSP
      93      93 0157         SPHL
      94      94 0158   ;     STACK POINTER CONTAINS CCP'S STACK LOCATION
      95      95 0158         RET             ;TO THE CCP
-----------------------------------------------------------------
      96      96 0159   ;
      97      97 0159   ;
      98      98 0159   ;     SUBROUTINES
      99      99 0159   ;
     100     100 0159   BREAK:        ;CHECK BREAK KEY (ACTUALLY ANY KEY WILL DO)
     101     101 0159         PUSH H! PUSH D! PUSH B; ENVIRONMENT SAVED
     102     104 015C         MVI     C,BRKF
     103     105 015E         CALL    BDOS
-----------------------------------------------------------------
     104     106 0161         POP B! POP D! POP H; ENVIRONMENT RESTORED
     105     109 0164         RET
-----------------------------------------------------------------
     106     110 0165   ;
     107     111 0165   PCHAR:        ;PRINT A CHARACTER
     108     112 0165         PUSH H! PUSH D! PUSH B; SAVED
     109     115 0168         MVI     C,TYPEF
     110     116 016A         MOV     E,A
     111     117 016B         CALL    BDOS
-----------------------------------------------------------------
     112     118 016E         POP B! POP D! POP H; RESTORED
     113     121 0171         RET
-----------------------------------------------------------------
     114     122 0172   ;
     115     123 0172   CRLF:
     116     124 0172         MVI     A,CR
     117     125 0174         CALL    PCHAR
-----------------------------------------------------------------
     118     126 0177         MVI     A,LF
```

```
     119     127 0179         CALL    PCHAR
-------------------------------------------------------------------
     120     128 017C         RET
-------------------------------------------------------------------
     121     129 017D   ;
     122     130 017D   ;
     123     131 017D   PNIB: ;PRINT NIBBLE IN REG A
     124     132 017D         ANI     0FH     ;LOW 4 BITS
     125     133 017F         CPI     10
     126     134 0181         JNC     P10
-------------------------------------------------------------------
     127     135 0184   ;     LESS THAN OR EQUAL TO 9
     128     136 0184         ADI     '0'
     129     137 0186         JMP     PRN
-------------------------------------------------------------------
     130     138 0189   ;
     131     139 0189   ;     GREATER OR EQUAL TO 10
     132     140 0189   P10:  ADI     'A' - 10
-------------------------------------------------------------------
     133     141 018B   PRN:  CALL    PCHAR
-------------------------------------------------------------------
     134     142 018E         RET
-------------------------------------------------------------------
     135     143 018F   ;
     136     144 018F   PHEX: ;PRINT HEX CHAR IN REG A
     137     145 018F         PUSH    PSW
     138     146 0190         RRC
     139     147 0191         RRC
     140     148 0192         RRC
     141     149 0193         RRC
     142     150 0194         CALL    PNIB    ;PRINT NIBBLE
-------------------------------------------------------------------
     143     151 0197         POP     PSW
     144     152 0198         CALL    PNIB
-------------------------------------------------------------------
     145     153 019B         RET
-------------------------------------------------------------------
     146     154 019C   ;
     147     155 019C   ERR:  ;PRINT ERROR MESSAGE
     148     156 019C   ;     D,E ADDRESSES MESSAGE ENDING WITH "$"
     149     157 019C         MVI     C,PRINTF        ;PRINT BUFFER FUNCTION
     150     158 019E         CALL    BDOS
-------------------------------------------------------------------
     151     159 01A1         RET
-------------------------------------------------------------------
     152     160 01A2   ;
     153     161 01A2   ;
     154     162 01A2   GNB:  ;GET NEXT BYTE
     155     163 01A2         LDA     IBP
     156     164 01A5         CPI     80H
     157     165 01A7         JNZ     G0
-------------------------------------------------------------------
     158     166 01AA   ;     READ ANOTHER BUFFER
     159     167 01AA   ;
     160     168 01AA   ;
     161     169 01AA         CALL    DISKR
-------------------------------------------------------------------
     162     170 01AD         ORA     A       ;ZERO VALUE IF READ OK
     163     171 01AE         JZ      G0      ;FOR ANOTHER BYTE
-------------------------------------------------------------------
     164     172 01B1   ;     END OF DATA, RETURN WITH CARRY SET FOR EOF
     165     173 01B1         STC
     166     174 01B2         RET
-------------------------------------------------------------------
     167     175 01B3   ;
     168     176 01B3   G0:   ;READ THE BYTE AT BUFF+REG A
     169     177 01B3         MOV     E,A     ;LS BYTE OF BUFFER INDEX
     170     178 01B4         MVI     D,0     ;DOUBLE PRECISION INDEX TO DE
     171     179 01B6         INR     A       ;INDEX=INDEX+1
     172     180 01B7         STA     IBP     ;BACK TO MEMORY
     173     181 01BA   ;     POINTER IS INCREMENTED
```

```
     174     182 01BA   ;     SAVE THE CURRENT FILE ADDRESS
     175     183 01BA         LXI     H,BUFF
     176     184 01BD         DAD     D
     177     185 01BE   ;     ABSOLUTE CHARACTER ADDRESS IS IN HL
     178     186 01BE         MOV     A,M
     179     187 01BF   ;     BYTE IS IN THE ACCUMULATOR
     180     188 01BF         ORA     A       ;RESET CARRY BIT
     181     189 01C0         RET
-----------------------------------------------------------------
     182     190 01C1   ;
     183     191 01C1   SETUP:        ;SET UP FILE
     184     192 01C1   ;     OPEN THE FILE FOR INPUT
     185     193 01C1         XRA     A       ;ZERO TO ACCUM
     186     194 01C2         STA     FCBCR   ;CLEAR CURRENT RECORD
     187     195 01C5   ;
     188     196 01C5         LXI     D,FCB
     189     197 01C8         MVI     C,OPENF
     190     198 01CA         CALL    BDOS
-----------------------------------------------------------------
     191     199 01CD   ;     255 IN ACCUM IF OPEN ERROR
     192     200 01CD         RET
-----------------------------------------------------------------
     193     201 01CE   ;
     194     202 01CE   DISKR:        ;READ DISK FILE RECORD
     195     203 01CE         PUSH H! PUSH D! PUSH B
     196     206 01D1         LXI     D,FCB
     197     207 01D4         MVI     C,READF
     198     208 01D6         CALL    BDOS
-----------------------------------------------------------------
     199     209 01D9         POP B! POP D! POP H
     200     212 01DC         RET
-----------------------------------------------------------------
     201     213 01DD   ;
     202     214 01DD   ;     FIXED MESSAGE AREA
     203     215 01DD   SIGNON:       DB      'FILE DUMP VERSION 1.4$'
     204     216 01F3   OPNMSG:       DB      CR,LF
     205     217 01F5   ;
     206     218 01F5   ;     VARIABLE AREA
     207     219 01F5   IBP:  DS      2       ;INPUT BUFFER POINTER
     208     220 01F7   OLDSP DS      2       ;ENTRY STACK POINTER VALUE
     209     221 01F9   ;
     210     222 01F9   ;     STACK AREA
     211     223 01F9         DS      64      ;RESERVE 32 LEVEL STACK
-----------------------------------------------------------------
     212     224 0239   STKTOP:
     213     225 0239   ;
     214     226 0239         END
     215     227 0239
```

```
 1      0  M      EQU     Byte Ptr 0[BX]
 1      1  ;      FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
 2      2  ;
 3      3  ;      COPYRIGHT (C) 1975, 1976, 1977, 1978
 4      4  ;      DIGITAL RESEARCH
 5      5  ;      BOX 579, PACIFIC GROVE
 6      6  ;      CALIFORNIA, 93950
 7      7  ;
 8      8         ORG     100H
 9      9  BDOS   EQU     0005H           ;DOS ENTRY POINT
10     10  CONS   EQU     1               ;READ CONSOLE
11     11  TYPEF  EQU     2               ;TYPE FUNCTION
12     12  PRINTF EQU     9               ;BUFFER PRINT ENTRY
13     13  BRKF   EQU     11              ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
14     14  OPENF  EQU     15              ;FILE OPEN
15     15  READF  EQU     20              ;READ FUNCTION
16     16  ;
17     17  FCB    EQU     5CH             ;FILE CONTROL BLOCK ADDRESS
18     18  BUFF   EQU     80H             ;INPUT DISK BUFFER ADDRESS
19     19  ;
20     20  ;      NON GRAPHIC CHARACTERS
21     21  CR     EQU     0DH             ;CARRIAGE RETURN
22     22  LF     EQU     0AH             ;LINE FEED
23     23  ;
24     24  ;      FILE CONTROL BLOCK DEFINITIONS
25     25  FCBDN  EQU     FCB+0           ;DISK NAME
26     26  FCBFN  EQU     FCB+1           ;FILE NAME
27     27  FCBFT  EQU     FCB+9           ;DISK FILE TYPE (3 CHARACTERS)
28     28  FCBRL  EQU     FCB+12          ;FILE'S CURRENT REEL NUMBER
29     29  FCBRC  EQU     FCB+15          ;FILE'S RECORD COUNT (0 TO 128)
30     30  FCBCR  EQU     FCB+32          ;CURRENT (NEXT) RECORD NUMBER (0 TO 127)
31     31  FCBLN  EQU     FCB+33          ;FCB LENGTH
32     32  ;
33     33  ;      SET UP STACK
34     34         MOV     BX,0
35     35         ADD     BX,SP
36     36  ;      ENTRY STACK POINTER IN HL FROM THE CCP
37     37         MOV     Word Ptr OLDSP,BX
38     38  ;      SET SP TO LOCAL STACK AREA (RESTORED AT FINIS)
39     39         MOV     SP,(Offset STKTOP)
40     40  ;      READ AND PRINT SUCCESSIVE BUFFERS
41     41         CALL    SETUP                   ;SET UP INPUT FILE
42     42         CMP     AL,255                  ;255 IF FILE NOT PRESENT
43     43         JNZ     OPENOK                  ;SKIP IF OPEN IS OK
44     44  ;
45     45  ;      FILE NOT THERE, GIVE ERROR MESSAGE AND RETURN
46     46         MOV     DX,(Offset OPNMSG)
47     47         CALL    ERR
48     48         JMPS    FINIS                   ;TO RETURN
49     49  ;
50     50  OPENOK:                         ;OPEN OPERATION OK, SET BUFFER INDEX TO END
51     51         MOV     AL,80H
52     52         MOV     Byte Ptr IBP,AL              ;SET BUFFER POINTER TO 80H
53     53  ;      HL CONTAINS NEXT ADDRESS TO PRINT
54     54         MOV     BX,0                    ;START WITH 0000
55     55  ;
56     56  GLOOP:
57     57         PUSH    BX                      ;SAVE LINE POSITION
58     58         CALL    GNB
59     59         POP     BX                      ;RECALL LINE POSITION
60     60         JB      FINIS                   ;CARRY SET BY GNB IF END FILE
61     61         MOV     CH,AL
62     62  ;      PRINT HEX VALUES
63     63  ;      CHECK FOR LINE FOLD
64     64         MOV     AL,BL
65     65         AND     AL,0FH                  ;CHECK LOW 4 BITS
66     66         JNZ     NONUM
67     67  ;      PRINT LINE NUMBER
68     68         CALL    CRLF
69     69  ;
```

```
 70     70  ;      CHECK FOR BREAK KEY
 71     71         CALL    BREAK
 72     72  ;      ACCUM LSB = 1 IF CHARACTER READY
 73     73         ROR     AL,1                    ;INTO CARRY
 74     74         JB      FINIS                   ;DON'T PRINT ANY MORE
 75     75  ;
 76     76         MOV     AL,BH
 77     77         CALL    PHEX
 78     78         MOV     AL,BL
 79     79         CALL    PHEX
 80     80  NONUM:
 81     81         LAHF                            ;TO NEXT LINE NUMBER
 81     81         INC     BX
 81     81         SAHF
 82     82         MOV     AL,' '
 83     83         CALL    PCHAR
 84     84         MOV     AL,CH
 85     85         CALL    PHEX
 86     86         JMPS    GLOOP
 87     87  ;
 88     88  FINIS:
 89     89  ;      END OF DUMP, RETURN TO CCP
 90     90  ;      (NOTE THAT A JMP TO 0000H REBOOTS)
 91     91         CALL    CRLF
 92     92         MOV     BX,Word Ptr OLDSP
 93     93         MOV     SP,BX
 94     94  ;      STACK POINTER CONTAINS CCP'S STACK LOCATION
 95     95         RET                             ;TO THE CCP
 96     96  ;
 97     97  ;
 98     98  ;      SUBROUTINES
 99     99  ;
100    100  BREAK:                          ;CHECK BREAK KEY (ACTUALLY ANY KEY WILL DO)
101    101         PUSH    BX
101    102         PUSH    DX
101    103         PUSH    CX                      ; ENVIRONMENT SAVED
102    104         MOV     CL,BRKF
103    105         INT     224
104    106         POP     CX
104    107         POP     DX
104    108         POP     BX                      ; ENVIRONMENT RESTORED
105    109         RET
106    110  ;
107    111  PCHAR:                             ;PRINT A CHARACTER
108    112         PUSH    BX
108    113         PUSH    DX
108    114         PUSH    CX                      ; SAVED
109    115         MOV     CL,TYPEF
110    116         MOV     DL,AL
111    117         INT     224
112    118         POP     CX
112    119         POP     DX
112    120         POP     BX                      ; RESTORED
113    121         RET
114    122  ;
115    123  CRLF:
116    124         MOV     AL,CR
117    125         CALL    PCHAR
118    126         MOV     AL,LF
119    127         CALL    PCHAR
120    128         RET
121    129  ;
122    130  ;
123    131  PNIB:                              ;PRINT NIBBLE IN REG A
124    132         AND     AL,0FH                  ;LOW 4 BITS
125    133         CMP     AL,10
126    134         JNB     P10
127    135  ;      LESS THAN OR EQUAL TO 9
128    136         ADD     AL,'0'
129    137         JMPS    PRN
130    138  ;
```

```
131    139  ;      GREATER OR EQUAL TO 10
132    140  P10:   ADD     AL,'A' - 10
133    141  PRN:   CALL    PCHAR
134    142         RET
135    143  ;
136    144  PHEX:                                ;PRINT HEX CHAR IN REG A
137    145         LAHF
137    145         XCHG    AL,AH
137    145         PUSH    AX
137    145         XCHG    AL,AH
138    146         ROR     AL,1
139    147         ROR     AL,1
140    148         ROR     AL,1
141    149         ROR     AL,1
142    150         CALL    PNIB                 ;PRINT NIBBLE
143    151         POP     AX
143    151         XCHG    AL,AH
143    151         SAHF
144    152         CALL    PNIB
145    153         RET
146    154  ;
147    155  ERR:                                 ;PRINT ERROR MESSAGE
148    156  ;      D,E ADDRESSES MESSAGE ENDING WITH "$"
149    157         MOV     CL,PRINTF            ;PRINT BUFFER FUNCTION
150    158         INT     224
151    159         RET
152    160  ;
153    161  ;
154    162  GNB:                                 ;GET NEXT BYTE
155    163         MOV     AL,Byte Ptr IBP
156    164         CMP     AL,80H
157    165         JNZ     G0
158    166  ;      READ ANOTHER BUFFER
159    167  ;
160    168  ;
161    169         CALL    DISKR
162    170         OR      AL,AL                ;ZERO VALUE IF READ OK
163    171         JZ      G0                   ;FOR ANOTHER BYTE
164    172  ;      END OF DATA, RETURN WITH CARRY SET FOR EOF
165    173         STC
166    174         RET
167    175  ;
168    176  G0:                                  ;READ THE BYTE AT BUFF+REG A
169    177         MOV     DL,AL                ;LS BYTE OF BUFFER INDEX
170    178         MOV     DH,0                 ;DOUBLE PRECISION INDEX TO DE
171    179         INC     AL                   ;INDEX=INDEX+1
172    180         MOV     Byte Ptr IBP,AL              ;BACK TO MEMORY
173    181  ;      POINTER IS INCREMENTED
174    182  ;      SAVE THE CURRENT FILE ADDRESS
175    183         MOV     BX,BUFF
176    184         ADD     BX,DX
177    185  ;      ABSOLUTE CHARACTER ADDRESS IS IN HL
178    186         MOV     AL,M
179    187  ;      BYTE IS IN THE ACCUMULATOR
180    188         OR      AL,AL                ;RESET CARRY BIT
181    189         RET
182    190  ;
183    191  SETUP:                               ;SET UP FILE
184    192  ;      OPEN THE FILE FOR INPUT
185    193         XOR     AL,AL                ;ZERO TO ACCUM
186    194         MOV     Byte Ptr .FCBCR,AL   ;CLEAR CURRENT RECORD
187    195  ;
188    196         MOV     DX,FCB
189    197         MOV     CL,OPENF
190    198         INT     224
191    199  ;      255 IN ACCUM IF OPEN ERROR
192    200         RET
193    201  ;
194    202  DISKR:                               ;READ DISK FILE RECORD
195    203         PUSH    BX
195    204         PUSH    DX
```

```
195    205         PUSH    CX
196    206         MOV     DX,FCB
197    207         MOV     CL,READF
198    208         INT     224
199    209         POP     CX
199    210         POP     DX
199    211         POP     BX
200    212         RET
200    212  L_1    EQU     $
200    212         DSEG
200    212         ORG     Offset L_1
201    213  ;
202    214  ;      FIXED MESSAGE AREA
203    215  SIGNON DB      'FILE DUMP VERSION 1.4$'
204    216  OPNMSG DB      CR,LF
205    217  ;
206    218  ;      VARIABLE AREA
207    219  IBP    RS      2                       ;INPUT BUFFER POINTER
208    220  OLDSP  RS      2                       ;ENTRY STACK POINTER VALUE
209    221  ;
210    222  ;      STACK AREA
211    223         RS      64                      ;RESERVE 32 LEVEL STACK
212    224  STKTOP:
213    225  ;
214    226         END
```

L I S T O F B A S I C B L O C K S

```
Block At 0005 (subr), A86 = 7A74
        Entry Active: ------------ Exit Active: ------------
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
-----------------------------------------------------------------------

Block At 0100 (code), A86 = 0100
        Entry Active: BCDE---A----- Exit Active: BCDEHL-A-----
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   34|------------ |LXI |   H|0000|----HL------- |BCDEHL-A----- |
|   35|----HL------- |DAD |  SP|    |----HL--O---- |BCDEHL-A----- |
|   37|----HL------- |SHLD|01F7|    |------------- |BCDEHL-A----- |
|   39|------------ |LXI |   M|0239|------------- |BCDEHL-A----- |
|   41|------------ |CALL|01C1|    |------------- |BCDEHL-A----- |
-----------------------------------------------------------------------

Block At 010D (code), A86 = 0115
        Entry Active: BCDEHL-A----- Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   42|-------A----- |CPI |  FF|    |--------OZSPI |BCDEHL-AOZSPI |
|   43|---------Z--- |JNZ |011B|    |------------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------------

Block At 0112 (code), A86 = 011C
        Entry Active: BC--HL-AOZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   46|------------ |LXI |   D|01F3|--DE--------- |BCDEHL-AOZSPI |
|   47|------------ |CALL|019C|    |------------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------------

Block At 0118 (code), A86 = 0122
        Entry Active: BCDEHL-AOZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   48|------------ |JMP |0151|    |------------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------------

Block At 011B (code), A86 = 0125
        Entry Active: BCDE----OZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   51|------------ |MVI |   A|  80|-------A----- |BCDE---AOZSPI |
|   52|-------A----- |STA |01F5|    |------------- |BCDE---AOZSPI |
|   54|------------ |LXI |   H|0000|----HL------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------------
```

```
Block At 0123 (code), A86 = 012D
        Entry Active: BCDEHL-AOZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------
```

```
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   57|----HL------- |PUSH|   H|    |------------- |BCDEHL-AOZSPI |
|   58|------------- |CALL|01A2|    |------------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------------

Block At 0127 (code), A86 = 0136
        Entry Active: BCDE---AOZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   59|------------- |POP |   H|    |----HL------- |BCDEHL-AOZSPI |
|   60|--------O---- |JC  |0151|    |------------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------------

Block At 012B (code), A86 = 0141
        Entry Active: -CDEHL-A----- Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   61|-------A----- |MOV |   B|   A|B------------ |BCDEHL------- |
|   64|-----L------- |MOV |   A|   L|-------A----- |BCDEHL-A----- |
|   65|-------A----- |ANI |  0F|    |-------AOZSPI |BCDEHL-AOZSPI |
|   66|---------Z--- |JNZ |0144|    |------------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------------

Block At 0132 (code), A86 = 014C
        Entry Active: BCDEHL-AOZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   68|------------- |CALL|0172|    |------------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------------

Block At 0135 (code), A86 = 014F
        Entry Active: BCDEHL-AOZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   71|------------- |CALL|0159|    |------------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------------

Block At 0138 (code), A86 = 0152
        Entry Active: BCDEHL-A-ZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   73|-------A----- |RRC |    |    |-------AO---- |BCDEHL-AOZSPI |
|   74|--------O---- |JC  |0151|    |------------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------------

Block At 013C (code), A86 = 015A
        Entry Active: BCDEHL--OZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------------
|   76|----H-------- |MOV |   A|   H|-------A----- |BCDEHL-AOZSPI |
```

```
|   77|------------- |CALL|018F|    |------------- |BCDEHL-AOZSPI |
-------------------------------------------------------------------
```

```
Block At 0140 (code), A86 = 015F
        Entry Active: BCDEHL--OZSPI Exit Active: BCDEHL-AOZSPI
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|   78|-----L------- |MOV |   A|   L|-------A----- |BCDEHL-AOZSPI |
|   79|------------- |CALL|018F|    |------------- |BCDEHL-AOZSPI |
-------------------------------------------------------------------

Block At 0144 (code), A86 = 0164
        Entry Active: BCDEHL--OZSPI Exit Active: BCDEHL-AOZSPI
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|   81|----HL------- |INX |   H|    |----HL------- |BCDEHL--OZSPI |
|   82|------------- |MVI |   A|  20|-------A----- |BCDEHL-AOZSPI |
|   83|------------- |CALL|0165|    |------------- |BCDEHL-AOZSPI |
-------------------------------------------------------------------

Block At 014A (code), A86 = 016C
        Entry Active: BCDEHL--OZSPI Exit Active: BCDEHL-AOZSPI
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|   84|B------------ |MOV |   A|   B|-------A----- |BCDEHL-AOZSPI |
|   85|------------- |CALL|018F|    |------------- |BCDEHL-AOZSPI |
-------------------------------------------------------------------

Block At 014E (code), A86 = 0171
        Entry Active: BCDEHL-AOZSPI Exit Active: BCDEHL-AOZSPI
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|   86|------------- |JMP |0123|    |------------- |BCDEHL-AOZSPI |
-------------------------------------------------------------------

Block At 0151 (code), A86 = 0174
        Entry Active: BCDEHL-AOZSPI Exit Active: BCDEHL-AOZSPI
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|   91|------------- |CALL|0172|    |------------- |BCDEHL-AOZSPI |
-------------------------------------------------------------------

Block At 0154 (subr), A86 = 0177
        Entry Active: BCDE---AOZSPI Exit Active: -------------
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|   92|------------- |LHLD|01F7|    |----HL------- |BCDEHL-AOZSPI |
|   93|----HL------- |SPHL|    |    |------------- |BCDEHL-AOZSPI |
|   95|BCDEHL-AOZSPI |RET |    |    |------------- |------------- |
-------------------------------------------------------------------

Block At 0159 (subr), A86 = 017E
        Entry Active: BCDEHL-AOZSPI Exit Active: -------AOZSPI
-------------------------------------------------------------------
```

```
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|  101|----HL------- |PUSH|   H|    |------------- |BCDE---AOZSPI |
```

```
|  102|--DE--------- |PUSH|   D|    |------------ |BC-----AOZSPI |
|  103|BC----------- |PUSH|   B|    |------------ |-------AOZSPI |
|  104|------------ |MVI |   C|  0B|-C---------- |-------AOZSPI |
|  105|------------ |CALL|0005|    |------------ |-------AOZSPI |
-----------------------------------------------------------------

Block At 0161 (subr), A86 = 0195
        Entry Active: -------AOZSPI Exit Active: -------------
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  106|------------ |POP |   B|    |BC---------- |BC-----AOZSPI |
|  107|------------ |POP |   D|    |--DE-------- |BCDE---AOZSPI |
|  108|------------ |POP |   H|    |----HL------ |BCDEHL-AOZSPI |
|  109|BCDEHL-AOZSPI |RET |    |    |------------ |------------- |
-----------------------------------------------------------------

Block At 0165 (subr), A86 = 01A8
        Entry Active: BCDEHL-AOZSPI Exit Active: -------AOZSPI
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  112|----HL------- |PUSH|   H|    |------------ |BCDE---AOZSPI |
|  113|--DE--------- |PUSH|   D|    |------------ |BC-----AOZSPI |
|  114|BC----------- |PUSH|   B|    |------------ |-------AOZSPI |
|  115|------------ |MVI |   C|  02|-C---------- |-------AOZSPI |
|  116|-------A----- |MOV |   E|   A|---E-------- |-------AOZSPI |
|  117|------------ |CALL|0005|    |------------ |-------AOZSPI |
-----------------------------------------------------------------

Block At 016E (subr), A86 = 01C1
        Entry Active: -------AOZSPI Exit Active: -------------
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  118|------------ |POP |   B|    |BC---------- |BC-----AOZSPI |
|  119|------------ |POP |   D|    |--DE-------- |BCDE---AOZSPI |
|  120|------------ |POP |   H|    |----HL------ |BCDEHL-AOZSPI |
|  121|BCDEHL-AOZSPI |RET |    |    |------------ |------------- |
-----------------------------------------------------------------

Block At 0172 (subr), A86 = 01D4
        Entry Active: BCDEHL--OZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  124|------------ |MVI |   A|  0D|-------A----- |BCDEHL-AOZSPI |
|  125|------------ |CALL|0165|    |------------ |BCDEHL-AOZSPI |
-----------------------------------------------------------------

Block At 0177 (code), A86 = 01D9
        Entry Active: BCDEHL--OZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  126|------------ |MVI |   A|  0A|-------A----- |BCDEHL-AOZSPI |
|  127|------------ |CALL|0165|    |------------ |BCDEHL-AOZSPI |
```

```
-----------------------------------------------------------------

Block At 017C (subr), A86 = 01DE
```

```
        Entry Active: BCDEHL-AOZSPI Exit Active: ------------
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|  128|BCDEHL-AOZSPI |RET |    |    |------------ |------------ |
-------------------------------------------------------------------

Block At 017D (code), A86 = 01DF
        Entry Active: BCDEHL-A----- Exit Active: BCDEHL-A-----
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|  132|-------A----- |ANI |  0F|    |-------AOZSPI |BCDEHL-A----- |
|  133|-------A----- |CPI |  0A|    |--------OZSPI |BCDEHL-AO---- |
|  134|--------O---- |JNC |0189|    |------------ |BCDEHL-A----- |
-------------------------------------------------------------------

Block At 0184 (code), A86 = 01E8
        Entry Active: BCDEHL-A----- Exit Active: BCDEHL-AOZSPI
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|  136|-------A----- |ADI |  30|    |-------AOZSPI |BCDEHL-AOZSPI |
|  137|------------ |JMP |018B|    |------------ |BCDEHL-AOZSPI |
-------------------------------------------------------------------

Block At 0189 (code), A86 = 01ED
        Entry Active: BCDEHL-A----- Exit Active: BCDEHL-AOZSPI
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|  140|-------A----- |ADI |  37|    |-------AOZSPI |BCDEHL-AOZSPI |
-------------------------------------------------------------------

Block At 018B (code), A86 = 01EF
        Entry Active: BCDEHL-AOZSPI Exit Active: BCDEHL-AOZSPI
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|  141|------------ |CALL|0165|    |------------ |BCDEHL-AOZSPI |
-------------------------------------------------------------------

Block At 018E (subr), A86 = 01F2
        Entry Active: BCDEHL-AOZSPI Exit Active: ------------
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|  142|BCDEHL-AOZSPI |RET |    |    |------------ |------------ |
-------------------------------------------------------------------

Block At 018F (subr), A86 = 01F3
        Entry Active: BCDEHL-AOZSPI Exit Active: BCDEHL-A-----
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
|  145|-------AOZSPI |PUSH| PSW|    |------------ |BCDEHL-A----- |
|  146|-------A----- |RRC |    |    |-------AO---- |BCDEHL-A----- |
```

```
|  147|-------A----- |RRC |    |    |-------AO---- |BCDEHL-A----- |
|  148|-------A----- |RRC |    |    |-------AO---- |BCDEHL-A----- |
|  149|-------A----- |RRC |    |    |-------AO---- |BCDEHL-A----- |
```

```
|  150|------------ |CALL|017D|    |------------ |BCDEHL-A----- |
-----------------------------------------------------------------

Block At 0197 (code), A86 = 0208
        Entry Active: BCDEHL------- Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  151|------------ |POP | PSW|    |-------AOZSPI |BCDEHL-AOZSPI |
|  152|------------ |CALL|017D|    |------------ |BCDEHL-AOZSPI |
-----------------------------------------------------------------

Block At 019B (subr), A86 = 0211
        Entry Active: BCDEHL-AOZSPI Exit Active: ------------
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  153|BCDEHL-AOZSPI |RET |    |    |------------ |------------ |
-----------------------------------------------------------------

Block At 019C (subr), A86 = 0212
        Entry Active: B-DEHL-AOZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  157|------------ |MVI |   C|  09|-C----------- |BCDEHL-AOZSPI |
|  158|------------ |CALL|0005|    |------------ |BCDEHL-AOZSPI |
-----------------------------------------------------------------

Block At 01A1 (subr), A86 = 0217
        Entry Active: BCDEHL-AOZSPI Exit Active: ------------
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  159|BCDEHL-AOZSPI |RET |    |    |------------ |------------ |
-----------------------------------------------------------------

Block At 01A2 (subr), A86 = 0218
        Entry Active: BCDEHL------- Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  163|------------ |LDA |01F5|    |-------A----- |BCDEHL-A----- |
|  164|-------A----- |CPI |  80|    |--------OZSPI |BCDEHL-AOZSPI |
|  165|---------Z--- |JNZ |01B3|    |------------ |BCDEHL-AOZSPI |
-----------------------------------------------------------------

Block At 01AA (code), A86 = 0222
        Entry Active: BCDEHL-AOZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  169|------------ |CALL|01CE|    |------------ |BCDEHL-AOZSPI |
-----------------------------------------------------------------

Block At 01AD (code), A86 = 0225
        Entry Active: BCDEHL-A----- Exit Active: BCDEHL-A-ZSPI
```

```
-------------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-------------------------------------------------------------------
```

```
|  170|-------A----- |ORA |   A|    |-------AOZSPI |BCDEHL-A-ZSPI |
|  171|---------Z--- |JZ  |01B3|    |------------- |BCDEHL-A-ZSPI |
-----------------------------------------------------------------

Block At 01B1 (subr), A86 = 022C
        Entry Active: BCDEHL-A-ZSPI Exit Active: -------------
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  173|------------- |STC |    |    |--------O---- |BCDEHL-AOZSPI |
|  174|BCDEHL-AOZSPI |RET |    |    |------------- |------------- |
-----------------------------------------------------------------

Block At 01B3 (subr), A86 = 022E
        Entry Active: BC-----A----- Exit Active: -------------
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  177|-------A----- |MOV |   E|   A|---E--------- |BC-E---A----- |
|  178|------------- |MVI |   D|  00|--D---------- |BCDE---A----- |
|  179|-------A----- |INR |   A|    |-------A-Z--- |BCDE---A----- |
|  180|-------A----- |STA |01F5|    |------------- |BCDE--------- |
|  183|------------- |LXI |   H|0080|----HL------- |BCDEHL------- |
|  184|--DEHL------- |DAD |   D|    |----HL--O---- |BCDEHL------- |
|  186|------------- |MOV |   A|   M|-------A----- |BCDEHL-A----- |
|  188|-------A----- |ORA |   A|    |-------AOZSPI |BCDEHL-AOZSPI |
|  189|BCDEHL-AOZSPI |RET |    |    |------------- |------------- |
-----------------------------------------------------------------

Block At 01C1 (subr), A86 = 0247
        Entry Active: B---HL-A----- Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  193|-------A----- |XRA |   A|    |-------AOZSPI |B---HL-AOZSPI |
|  194|-------A----- |STA |007C|    |------------- |B---HL-AOZSPI |
|  196|------------- |LXI |   D|005C|--DE--------- |B-DEHL-AOZSPI |
|  197|------------- |MVI |   C|  0F|-C----------- |BCDEHL-AOZSPI |
|  198|------------- |CALL|0005|    |------------- |BCDEHL-AOZSPI |
-----------------------------------------------------------------

Block At 01CD (subr), A86 = 0254
        Entry Active: BCDEHL-AOZSPI Exit Active: -------------
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  200|BCDEHL-AOZSPI |RET |    |    |------------- |------------- |
-----------------------------------------------------------------

Block At 01CE (subr), A86 = 0255
        Entry Active: BCDEHL-AOZSPI Exit Active: -------AOZSPI
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  203|----HL------- |PUSH|   H|    |------------- |BCDE---AOZSPI |
|  204|--DE--------- |PUSH|   D|    |------------- |BC-----AOZSPI |
|  205|BC----------- |PUSH|   B|    |------------- |-------AOZSPI |
```

```
|  206|------------ |LXI |   D|005C|--DE--------- |-------AOZSPI |
|  207|------------ |MVI |   C|  14|-C----------- |-------AOZSPI |
|  208|------------ |CALL|0005|    |------------- |-------AOZSPI |
```

```
-----------------------------------------------------------------

Block At 01D9 (subr), A86 = 026F
        Entry Active: -------AOZSPI Exit Active: ------------
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  209|------------- |POP |   B|    |BC----------- |BC-----AOZSPI |
|  210|------------- |POP |   D|    |--DE--------- |BCDE---AOZSPI |
|  211|------------- |POP |   H|    |----HL------- |BCDEHL-AOZSPI |
|  212|BCDEHL-AOZSPI |RET |    |    |------------- |------------- |
-----------------------------------------------------------------

Block At 01DD (data), A86 = 0282
        Entry Active: ------------ Exit Active: ------------
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
|  215|------------- |DB  |0016|    |------------- |------------- |
|  216|------------- |DB  |0002|    |------------- |------------- |
|  219|------------- |DS  |0002|    |------------- |------------- |
|  220|------------- |DS  |0002|    |------------- |------------- |
|  223|------------- |DS  |0040|    |------------- |------------- |
-----------------------------------------------------------------

Block At 0239 (code), A86 = 02DE
        Entry Active: BCDEHL-AOZSPI Exit Active: BCDEHL-AOZSPI
-----------------------------------------------------------------
|stmt#| opcode uses  | op | v1 | v2 | opcode kills |  live regs   |
-----------------------------------------------------------------
-----------------------------------------------------------------
```

INDEX