XLT86.WS4 (Courtesy of Emmanuel ROCHE.)
---

XLT86 T.M.

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 T.M. 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 T.M. environment into account, so that translated programs operate properly under both CP/M-86 and MP/M-86. XLT86 T.M. 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 T.M. and thus can be adapted for use on computer systems that support PL/I 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, XLT00.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 I

# 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 XLT00.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.

Table 1-1. XLT86 Translation Phases

| 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. |
| Translate-86 | translates the 8080 instructions to 8086 form, using the information gathered by the flow analysis. |

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.

```
Source          Output
DUMP.ASM        DUMP.A86
   | ----- | ----- |
Temp    XLT 86  Print
DUMP.$$$        DUMP.PRN
```

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:

[pl 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 pl through pn are one or two character sequences in upper- or lower-case, with optional intervening blanks, as listed in Table 2-1, below.

Table 2-1. Translation Parameters

| 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. |
| Px | 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. |
| Tx | 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. |

## 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 -------
-------------------------------------------------------------
Istmt#l opcode uses I op I vl I v2 I opcode kills I live regs I
```

```
-------------------------------------------------------------
I   23I ------ AOZSPIPUSHI PSWI    I ------------ I JB-D-HL-A I
I   24I ------ A -- IMOV I   El   Al --- E -------I JB-DEHL - I
I   25I ----------- IMVI I   Cl  05I -C ----------I JBCDEHL - I
I   26I ----------- ICALLI0005I    I -------------I JBCDEHL - I
-------------------------------------------------------------
```

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.

Table 2-2. Letter Denotations for Registers and Flags

| 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 |
| O | overflow Flag, carry or borrow |
| Z | Zero Flag |
| S | Sign Flag |
| P | Even Parity Flag |
| I | Interdigit Carry Flag |

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@l     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 (Ctrl-L) and tabs (Ctrl-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 "T" 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 derive 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, ..., 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@1
```

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

```
ORG     300H        ORG     300H
NOP         L@l:    NOP
NOP     NOP
JMP     300H    JMP     L@1
```

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.

Table 3-1. Operand Field Abbreviations

| Abbreviation | Definition |
|---|---|
| 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) |

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.

Table 3-2. Register Translation

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

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.

Table 3-3. 16-Bit Register Translation

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

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 followng expressions.

Table 3-4. Expressions

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

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.

Table 3-5. Operand Field Translation

| 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 ibl or iwl 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, mw' 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. |

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 JMP 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.

Table 3-6. Translation Table

```
Operation Code  Translation
--------------  -----------
ACI     ib      ADC     AL,ib'
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,0
                MOV     DL,0
                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:
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,rw'
                  ELSE
                    IF NOT live(O) AND
                      live(Z,S,P,I)
                      LAHF
                      ADD    BX,rw'
```

```
                SAHF
              ELSE
                LAHF
                ADD     BX,rw'
                RCR     SI,1
                SAHF
                RCL     SI,1
              ENDIF
            ENDIF
          ENDIF
DEC     rb      DEC     rb'
DCX     rw      DEC     rw'
DI              CLI
EI              STI
HLT             HLT
IN      ib      IN      AL,ib'
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
JM      mn      IF short(mn')
                JS      mn'
              ELSE
                JNS     L@n
                JMPS    mn'
                L@n:
              ENDIF
JMP     0       MOV     CL,0
              MOV     DL,0
              INT     224
              RET
JMP     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
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,rw'
              MOV    AL,[SI]
LHLD    mw    MOV    BX,mw'
LXI     rw,iw MOV    rw',iw'
MOV     rb1,rb2 MOV  rb1',rb2'
MVI     rb,ib MOV    rb',ib'
NOP           NOP
ORA     rb    OR     AL,rb'
ORI     ib    OR     AL,ib'
OUT     ib    OUT    ib',AL
PCHL          JMP    BX
POP     rw    POP    rw'
            IF rw = PSW AND live(O'Z'S'P'I)
              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    rw'
                  ENDIF
                ENDIF
RAL             RCL     AL,1
RAR             RCR     AL,1
RC              JNB     L@n
                RET
                L@n:
RET             RET
RLC             ROL     AL,1
RM              JNS     L@n
                RET
                L@n:
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,1
RST     ib      INT     ib'
RZ              JNZ     L@n
                RET
                L@n:
SBB     rb      SBB     AL,rb'
SBI     ib      SBB     AL,ib'
SHLD    mw      MOV     mw',BX
SPHL            MOV     SP,BX
STA     mb      MOV     mb',AL
STAX    rw      MOV     DI,rw'
                MOV     [DI],AL
STC             STC
SUB     rb      SUB     AL,rb'
SUI     ib      SUB     AL,ib'
```

```
XCHG          XCHG    BX,DX
XRA rb        XOR     AL,rb'
XRI ib        XOR     AL,ib'
XTHL          MOV     BP,SP
              XCHG    BX,[BP]
```

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.

Table 4-1. XLT86 Error Codes

| Error Code | Meaning |
|---|---|
| Bad Flag | invalid parameter list [p1 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.

## 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.

Table 4-2. Translate-86 Error Messages

| Error Message | Meaning |
|---|---|
| Bad Oper | Invalid 8080 operation code was encountered during translation; probably due to bad disk I/O 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 0000H (warm boot) or 0005H (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 |

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 [8086]

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 (" ---- ") 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,0
    MOV     DL,0
    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).

INDEX
---

8086 program segment, 13

EOF