MCS-86™
ASSEMBLY LANGUAGE CONVERTER
OPERATING INSTRUCTIONS
FOR ISIS-II USERS

Manual Order No. 9800642-02

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9).

The following are trademarks of Intel Corporation and may be used only to identify Intel products:

i
ICE
iCS
Insite
Intel
Intelevision
Intellec
iSBC
Library Manager
MCS
Megachassis
Micromap
Multibus
Multimodule
PROMPT
Promware
RMX
UPI
µScope

and the combination of ICE, iCS, iSBC, MCS, or RMX and a numerical suffix.

A175/280/7.5 FL

PREFACE

This manual describes how the ISIS-II user who is familiar with 8080/8085 assembly language can convert 8080/8085 source files to 8086 assembly language source files, which can then be assembled, linked, located, and run to perform their equivalent 8080/8085 functions on the upwardly compatible, 16-bit 8086.

Chapter 1 describes the scope and environment of conversion.

Chapter 2 describes how to operate the converter program CONV86.

Chapter 3 describes how to edit converter output to obtain MCS-86 source files.

Appendices describe the instruction, operand (expression), and directive mappings; reserved names; and sample conversions with 8080/8085 and MCS-86 Macro Assembler listings of source and output files.

The following publications contain detailed information on 8080/8085 and MCS-86 software related to this manual:

- *8080/8085 Assembly Language Programming Manual,* Order No. 9800301
- *ISIS-II 8080/8085 Macro Assembler Operator's Manual,* Order No. 9800292
- *ISIS-II User's Guide,* Order No. 9800306
- *8086 Family User's Manual* Order No. 9800722
- *MCS-86™ Macro Assembly Language Reference Manual,* Order No. 9800640
- *MCS-86™ Macro Assembler Operating Instructions for ISIS-II Users,* Order No. 9800641
- *MCS-86™ Software Development Utilities Operating Instructions for ISIS-II Users,* Order No. 9800639
- *ISIS-II PL/M-86 Compiler Operator's Manual,* Order No. 9800478

CONTENTS

CHAPTER 1 AN OVERVIEW OF CONVERSION

CHAPTER 2 OPERATING THE CONVERTER

CHAPTER 3 EDITING CONVERTER OUTPUT

APPENDIX A INSTRUCTION MAPPING

APPENDIX B CONVERSION OF EXPRESSIONS IN CONTEXT

APPENDIX C ASSEMBLER DIRECTIVES MAPPING

APPENDIX D RESERVED NAMES

APPENDIX E MACRO CONSTRUCT CONVERSION

APPENDIX F SAMPLE CONVERSION AND LISTINGS

APPENDIX G RELOCATION AND LINKAGE ERRORS AND WARNINGS

INDEX

TABLES

FIGURES

CHAPTER 1
AN OVERVIEW OF CONVERSION

Conversion and You

What Is Conversion?

Conversion is a way for you to obtain MCS-86 source files from your error-free 8080/8085 assembly-language source files. (Recall that an assembly-language source file consists of assembler control statements, assembler directives, and assembly-language instructions.)

Figure 1-1 shows the role of conversion in 8080/8085-to-8086 software development. Conversion consists of two phases:

1. Operating the program CONV86 under ISIS-II. As shown in Figure 1-2, CONV86 accepts as input an error-free 8080/8085 assembly-language source file and optional controls, and produces as output optional PRINT and OUTPUT files. The OUTPUT file contains machine-readable 8086 assembly-language source code generated by CONV86. The PRINT file is human-readable and contains:

    - Input 8080/8085 assembly-language source code—optionally controlled by SOURCELIST/NOSOURCELIST converter control
    - Output 8086 assembly-language source code with embedded diagnostic ("caution") messages

    Chapter 2 describes how to operate CONV86 under ISIS-II.

2. Manually editing (using the ISIS-II text editor) the OUTPUT file as indicated by the caution messages in the PRINT file. Chapter 3 describes how to edit CONV86 output according to the caution messages generated. Some machine-dependent sequences (such as software timing delays) are not detected by CONV86, but still require manual editing. Recall that in going from the 8080 to the 8086, both the instruction size (length) and time (clocks) change.

Figure 1-1 shows both phases of conversion, as well as subsequent assembling, linking, and (absolute) loading required for execution of your program.

Figure 1-3 shows the format of the PRINT file, and highlights features of conversion discussed here and elsewhere in this manual.

Why Convert?

If you want to capitalize on your software investment in the 8080/8085, and if your 8080/8085 source files are tried-and-true, then conversion may offer you a considerable head-start in your software development effort for the upwardly-compatible 8086.

What Preparation Does CONV86 Require of Source Code?

You must ensure that all 8080/8085 source files to be converted can be assembled without error by the ISIS-II 8080/8085 assembler. No source line can be longer than 129 characters, excluding carriage-return and line-feed. If your program contains more than 600 symbols, you must break your program down into smaller programs (even if you have 64K RAM).

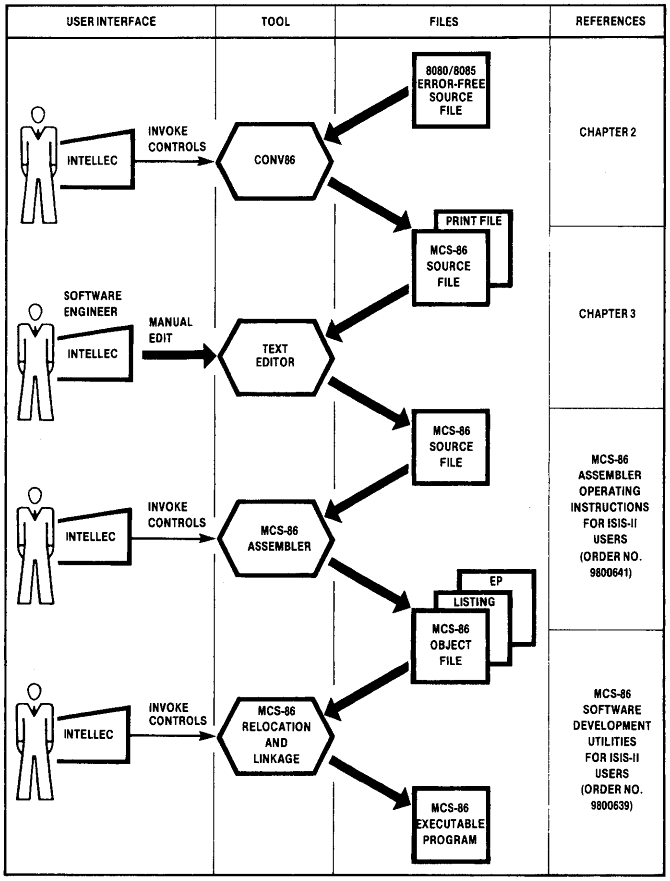


Figure 1-1. From 8080/8085 Assembly Language Source File to 8086 Execution

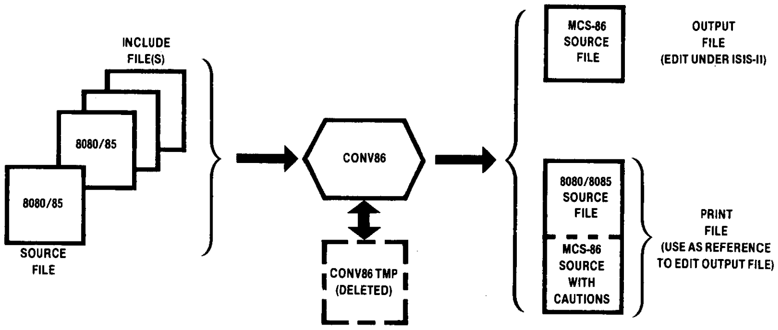

INCLUDE FILE(S)
8080/85
8080/85
SOURCE FILE
CONV86
CONV86 TMP (DELETED)
MCS-86 SOURCE FILE
OUTPUT FILE (EDIT UNDER ISIS-II)
8080/8085 SOURCE FILE
MCS-86 SOURCE WITH CAUTIONS
PRINT FILE (USE AS REFERENCE TO EDIT OUTPUT FILE)

Figure 1-2. CONV86 Input and Output Files

What About Macros?

All macro definitions and calls will be converted to their 8086 equivalents. However, macro-related constructs require special conversion. Appendix E lists all of these constructs and shows how they are mapped.

NOTE

ASM86 may misinterpret metacharacters (%) or unmatched parentheses appearing in comments as macro invocations.

What Hardware/Software Is Needed for Conversion?

You need an Intellec microcomputer development system with 64K bytes of RAM and at least one diskette unit. The CONV86 program occupies a single diskette and runs under ISIS-II. During execution, CONV86 creates a work file (CONV86.TMP) which requires seven bytes for each line of 8080/8085 code processed. Upon normal termination, CONV86 deletes this temporary file.

How Much Manual Editing of CONV86 Output Is Necessary?

Anywhere from none to a considerable amount, depending on the nature of the 8080/8085 source file. In general, the following kinds of source code are better implemented on the 8086 by recoding from scratch in 8086 assembly language, rather than by converting from 8080:

- ''Tricky'' code that modifies itself
- Code that uses operation mnemonics as operands (for example, the instruction MVI C,(MOV A,B); the intent of this instruction is to load C with the opcode for MOV A,B).
- Programs relying heavily on the 8085 instructions RIM and SIM (Read/Set Interrupt Mask) should be recoded from scratch in 8086 rather than converted. The 8086 has no functional counterparts for these instructions.

It is therefore recommended that source files not be blindly submitted for conversion. Each source file under consideration for conversion should be carefully examined for these problem areas.

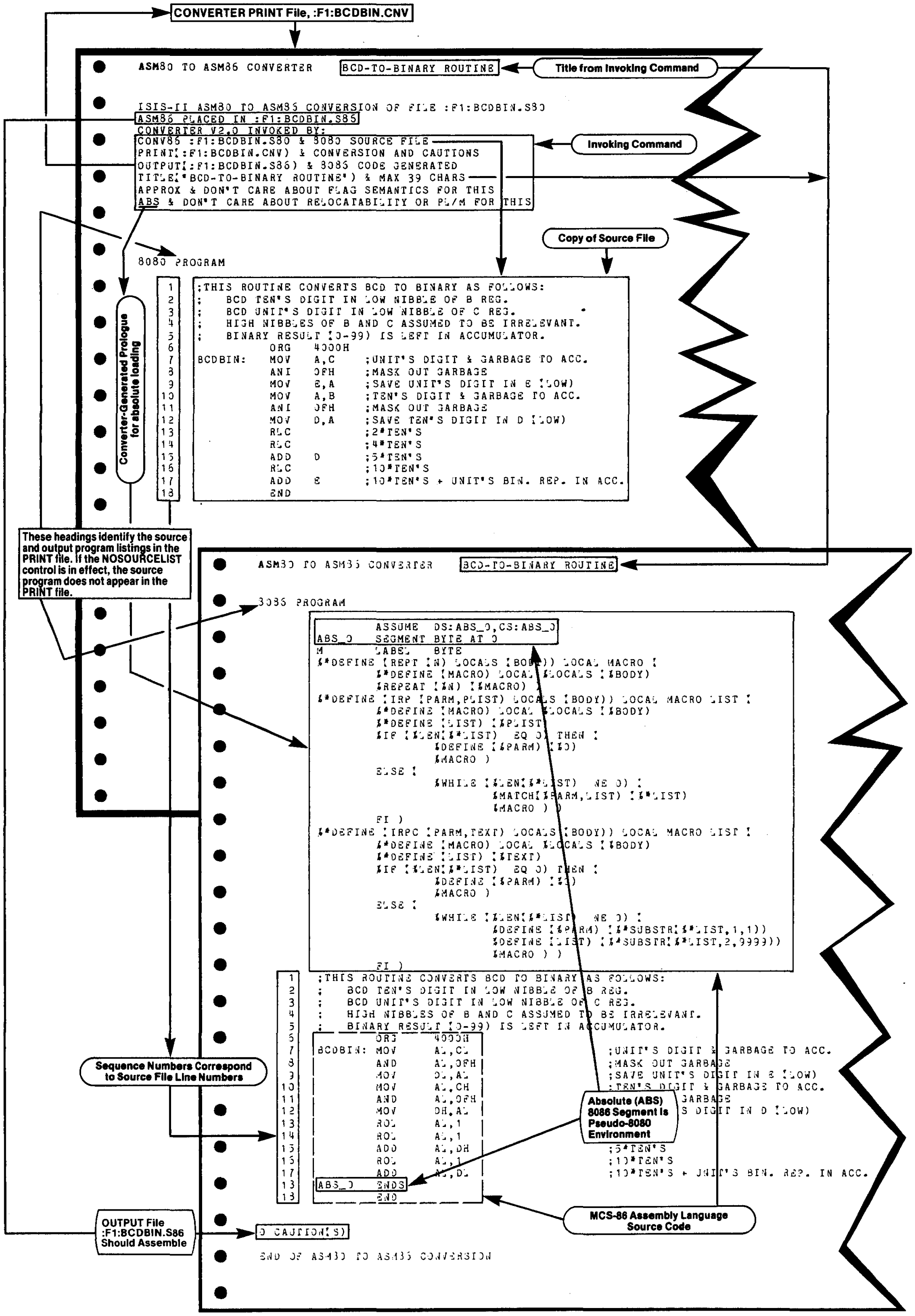
What Advantage Is There in Rewriting Programs in 8086 Assembly Language Rather Than Converting?

CONV86 converts most 8080/8085 assembly-language source programs adequately. You can take advantage of the more powerful 8086 by coding some routines directly in 8086 assembly language.

For example, Figure 1-4 shows assembled program listings for:

- 8080 Assembly of BCDBIN (13 bytes 8080 object code)
- MCS-86 Assembly of Conversion of BCDBIN (22 bytes 8086 object code)
- MCS-86 Assembly of BCDMCS Original 8086 Source (7 bytes 8086 object code)

(Recall that the PRINT file for the conversion of BCDBIN is shown in Figure 1-3.)



CONVERTER PRINT File, :F1:BCDBIN.CNV

Title from Invoking Command

Invoking Command

Copy of Source File

Converter-Generated Prologue for absolute loading

These headings identify the source and output program listings in the PRINT file. If the NOSOURCELIST control is in effect, the source program does not appear in the PRINT file.

Sequence Numbers Correspond to Source File Line Numbers

Absolute (ABS) 8086 Segment Is Pseudo-8080 Environment

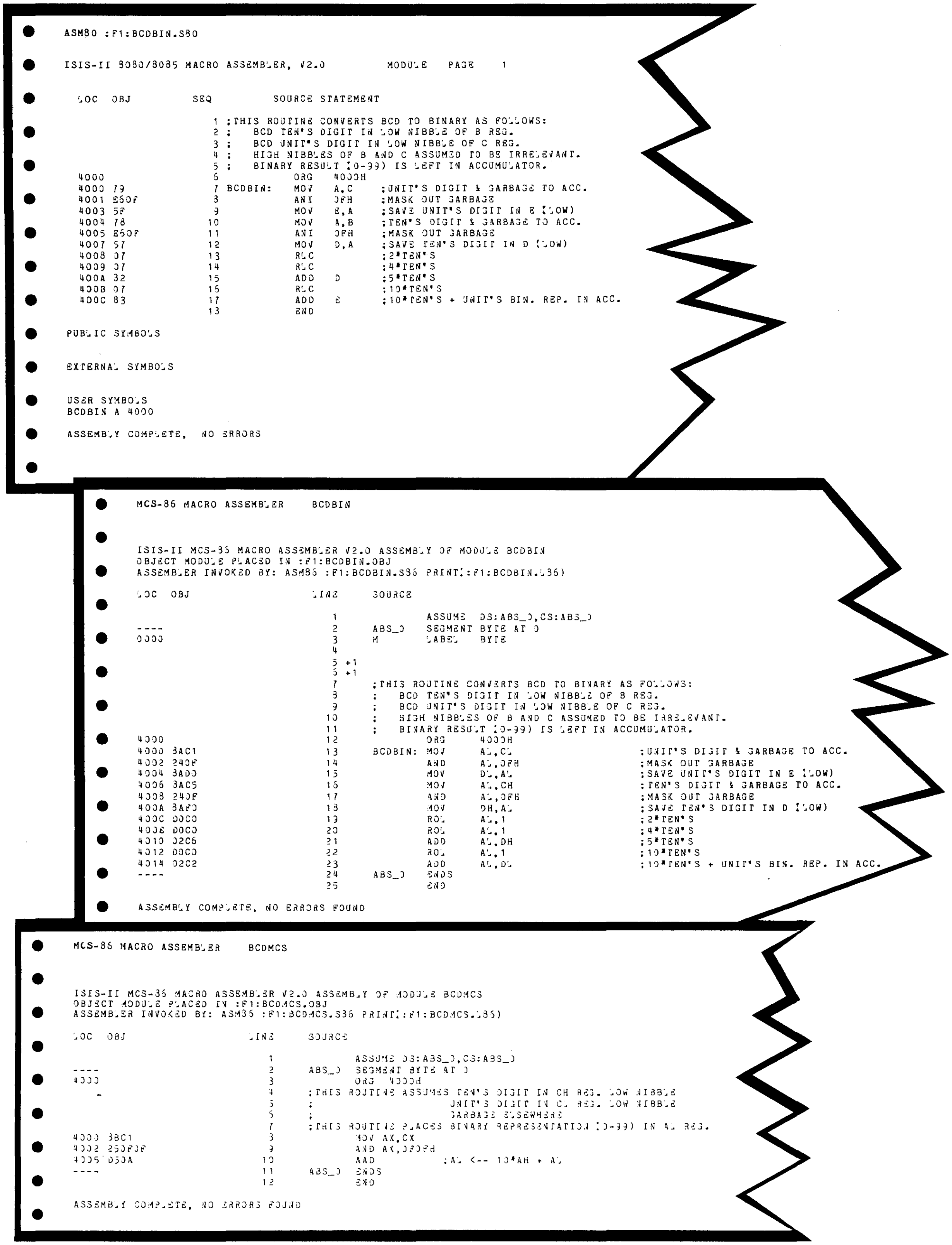
MCS-86 Assembly Language Source Code

OUTPUT File :F1:BCDBIN.S86 Should Assemble

```
ASM80 TO ASM86 CONVERTER      BCD-TO-BINARY ROUTINE

ISIS-II ASM80 TO ASM86 CONVERSION OF FILE :F1:BCDBIN.S80
ASM86 PLACED IN :F1:BCDBIN.S86
CONVERTER V2.0 INVOKED BY:
CONV86 :F1:BCDBIN.S80 & 8080 SOURCE FILE
PRINT(:F1:BCDBIN.CNV) & CONVERSION AND CAUTIONS
OUTPUT(:F1:BCDBIN.S86) & 8086 CODE GENERATED
TITLE('BCD-TO-BINARY ROUTINE') & MAX 39 CHARS
APPROX & DON'T CARE ABOUT FLAG SEMANTICS FOR THIS
ABS & DON'T CARE ABOUT RELOCATABILITY OR PL/M FOR THIS

8080 PROGRAM

 1  ;THIS ROUTINE CONVERTS BCD TO BINARY AS FOLLOWS:
 2  ;    BCD TEN'S DIGIT IN LOW NIBBLE OF B REG.
 3  ;    BCD UNIT'S DIGIT IN LOW NIBBLE OF C REG.
 4  ;    HIGH NIBBLES OF B AND C ASSUMED TO BE IRRELEVANT.
 5  ;    BINARY RESULT (0-99) IS LEFT IN ACCUMULATOR.
 6                  ORG     4000H
 7  BCDBIN:         MOV     A,C       ;UNIT'S DIGIT & GARBAGE TO ACC.
 8                  ANI     0FH       ;MASK OUT GARBAGE
 9                  MOV     E,A       ;SAVE UNIT'S DIGIT IN E (LOW)
10                  MOV     A,B       ;TEN'S DIGIT & GARBAGE TO ACC.
11                  ANI     0FH       ;MASK OUT GARBAGE
12                  MOV     D,A       ;SAVE TEN'S DIGIT IN D (LOW)
13                  RLC               ;2*TEN'S
14                  RLC               ;4*TEN'S
15                  ADD     D         ;5*TEN'S
16                  RLC               ;10*TEN'S
17                  ADD     E         ;10*TEN'S + UNIT'S BIN. REP. IN ACC.
18                  END
```

```
ASM80 TO ASM86 CONVERTER      BCD-TO-BINARY ROUTINE

8086 PROGRAM

                ASSUME  DS:ABS_0,CS:ABS_0
        ABS_0   SEGMENT BYTE AT 0
        M       LABEL   BYTE
        %*DEFINE (REPT (N) LOCALS (BODY)) LOCAL MACRO (
                %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
                %REPEAT (%N) (%MACRO) )
        %*DEFINE (IRP (PARM,PLIST) LOCALS (BODY)) LOCAL MACRO LIST (
                %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
                %*DEFINE (LIST) (%PLIST)
                %IF (%LEN(%*LIST)  EQ 0) THEN (
                        %DEFINE (%PARM) (%0)
                        %MACRO )
                ELSE (
                        %WHILE (%LEN(%*LIST)  NE 0) (
                                %MATCH(%PARM,LIST) (%*LIST)
                                %MACRO ) )
                FI )
        %*DEFINE (IRPC (PARM,TEXT) LOCALS (BODY)) LOCAL MACRO LIST (
                %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
                %*DEFINE (LIST) (%TEXT)
                %IF (%LEN(%*LIST)  EQ 0) THEN (
                        %DEFINE (%PARM) (%0)
                        %MACRO )
                ELSE (
                        %WHILE (%LEN(%*LIST)  NE 0) (
                                %DEFINE (%PARM) (%*SUBSTR(%*LIST,1,1))
                                %DEFINE (LIST) (%*SUBSTR(%*LIST,2,9999))
                                %MACRO ) )
                FI )
 1      ;THIS ROUTINE CONVERTS BCD TO BINARY AS FOLLOWS:
 2      ;    BCD TEN'S DIGIT IN LOW NIBBLE OF B REG.
 3      ;    BCD UNIT'S DIGIT IN LOW NIBBLE OF C REG.
 4      ;    HIGH NIBBLES OF B AND C ASSUMED TO BE IRRELEVANT.
 5      ;    BINARY RESULT (0-99) IS LEFT IN ACCUMULATOR.
 6              ORG     4000H
 7      BCDBIN: MOV     AL,CL                   ;UNIT'S DIGIT & GARBAGE TO ACC.
 8              AND     AL,0FH                  ;MASK OUT GARBAGE
 9              MOV     DL,AL                   ;SAVE UNIT'S DIGIT IN E (LOW)
10              MOV     AL,CH                   ;TEN'S DIGIT & GARBAGE TO ACC.
11              AND     AL,0FH                   GARBAGE
12              MOV     DH,AL                  S DIGIT IN D (LOW)
13              ROL     AL,1
14              ROL     AL,1
15              ADD     AL,DH                   ;5*TEN'S
16              ROL     AL,1                    ;10*TEN'S
17              ADD     AL,DL                   ;10*TEN'S + UNIT'S BIN. REP. IN ACC.
18      ABS_0   ENDS
19              END

0 CAUTION(S)

END OF ASM80 TO ASM86 CONVERSION
```

Figure 1-3. Sample PRINT File



```
ASM80 :F1:BCDBIN.S80

ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0         MODULE    PAGE     1

  LOC  OBJ         SEQ         SOURCE STATEMENT

                     1 ;THIS ROUTINE CONVERTS BCD TO BINARY AS FOLLOWS:
                     2 ;    BCD TEN'S DIGIT IN LOW NIBBLE OF B REG.
                     3 ;    BCD UNIT'S DIGIT IN LOW NIBBLE OF C REG.
                     4 ;    HIGH NIBBLES OF B AND C ASSUMED TO BE IRRELEVANT.
                     5 ;    BINARY RESULT (0-99) IS LEFT IN ACCUMULATOR.
  4000               6            ORG   4000H
  4000 79            7 BCDBIN:    MOV   A,C     ;UNIT'S DIGIT & GARBAGE TO ACC.
  4001 E60F          8            ANI   0FH     ;MASK OUT GARBAGE
  4003 5F            9            MOV   E,A     ;SAVE UNIT'S DIGIT IN E (LOW)
  4004 78           10            MOV   A,B     ;TEN'S DIGIT & GARBAGE TO ACC.
  4005 E60F         11            ANI   0FH     ;MASK OUT GARBAGE
  4007 57           12            MOV   D,A     ;SAVE TEN'S DIGIT IN D (LOW)
  4008 07           13            RLC           ;2*TEN'S
  4009 07           14            RLC           ;4*TEN'S
  400A 32           15            ADD   D       ;5*TEN'S
  400B 07           16            RLC           ;10*TEN'S
  400C 83           17            ADD   E       ;10*TEN'S + UNIT'S BIN. REP. IN ACC.
                    18            END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS
BCDBIN A 4000

ASSEMBLY COMPLETE,  NO ERRORS
```

```
MCS-86 MACRO ASSEMBLER    BCDBIN


ISIS-II MCS-86 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE BCDBIN
OBJECT MODULE PLACED IN :F1:BCDBIN.OBJ
ASSEMBLER INVOKED BY: ASM86 :F1:BCDBIN.S86 PRINT(:F1:BCDBIN.L86)

LOC  OBJ                  LINE     SOURCE

                            1               ASSUME  DS:ABS_0,CS:ABS_0
----                        2      ABS_0    SEGMENT BYTE AT 0
0000                        3      M        LABEL   BYTE
                            4
                            5 +1
                            6 +1
                            7      ;THIS ROUTINE CONVERTS BCD TO BINARY AS FOLLOWS:
                            8      ;    BCD TEN'S DIGIT IN LOW NIBBLE OF B REG.
                            9      ;    BCD UNIT'S DIGIT IN LOW NIBBLE OF C REG.
                           10      ;    HIGH NIBBLES OF B AND C ASSUMED TO BE IRRELEVANT.
                           11      ;    BINARY RESULT (0-99) IS LEFT IN ACCUMULATOR.
4000                       12               ORG     4000H
4000 8AC1                  13      BCDBIN:  MOV     AL,CL                ;UNIT'S DIGIT & GARBAGE TO ACC.
4002 240F                  14               AND     AL,0FH               ;MASK OUT GARBAGE
4004 8AD0                  15               MOV     DL,AL                ;SAVE UNIT'S DIGIT IN E (LOW)
4006 8AC5                  16               MOV     AL,CH                ;TEN'S DIGIT & GARBAGE TO ACC.
4008 240F                  17               AND     AL,0FH               ;MASK OUT GARBAGE
400A 8AF0                  18               MOV     DH,AL                ;SAVE TEN'S DIGIT IN D (LOW)
400C D0C0                  19               ROL     AL,1                 ;2*TEN'S
400E D0C0                  20               ROL     AL,1                 ;4*TEN'S
4010 02C6                  21               ADD     AL,DH                ;5*TEN'S
4012 D0C0                  22               ROL     AL,1                 ;10*TEN'S
4014 02C2                  23               ADD     AL,DL                ;10*TEN'S + UNIT'S BIN. REP. IN ACC.
----                       24      ABS_0    ENDS
                           25               END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
MCS-86 MACRO ASSEMBLER    BCDMCS


ISIS-II MCS-86 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE BCDMCS
OBJECT MODULE PLACED IN :F1:BCDMCS.OBJ
ASSEMBLER INVOKED BY: ASM86 :F1:BCDMCS.S86 PRINT(:F1:BCDMCS.L86)

LOC  OBJ                LINE     SOURCE

                          1             ASSUME DS:ABS_0,CS:ABS_0
----                      2      ABS_0  SEGMENT BYTE AT 0
4000                      3             ORG  4000H
                          4      ;THIS ROUTINE ASSUMES TEN'S DIGIT IN CH REG. LOW NIBBLE
                          5      ;                          UNIT'S DIGIT IN CL REG. LOW NIBBLE
                          6      ;                          GARBAGE ELSEWHERE
                          7      ;THIS ROUTINE PLACES BINARY REPRESENTATION (0-99) IN AL REG.
4000 8BC1                 8             MOV AX,CX
4002 250F0F               9             AND AX,0F0FH
4005 D50A                10             AAD            ;AL <-- 10*AH + AL
----                     11      ABS_0  ENDS
                         12             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Figure 1-4. Program Listings: Original 8080 (top); Converted 8080 (middle) Original 8086 (bottom)

Functional Mapping

What Are the 8086 Assembly Language Prologues Generated by CONV86?

The main source file of your 8080/8085 program should be converted using the (defaulted) control NOTINCLUDED. If NOTINCLUDED is in effect, the converted file begins with a converter-generated prologue. The prologue generated by the converter depends on whether the ABS or REL control is specified when CONV86 is run (REL is the default).

If the ABS control is specified (for subsequent absolute loading by 8086 relocation and linkage), CONV86 generates as a prologue:

```
          ASSUME DS:ABS_0,CS:ABS_0
ABS_0     SEGMENT BYTE AT 0
M         LABEL    BYTE
%*DEFINE (REPT (N) LOCALS (BODY)) LOCAL MACRO (
           %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
           %REPEAT (%N) (%MACRO) )
%*DEFINE (IRP (PARM,PLIST) LOCALS (BODY)) LOCAL MACRO LIST (
           %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
           %*DEFINE (LIST) (%PLIST)
           %IF (%LEN(%*LIST) EQ 0) THEN (
                    %DEFINE (%PARM) (%0)
                    %MACRO )
           ELSE (
                   %WHILE (%LEN(%*LIST) NE 0) (
                            %MATCH(%PARM,LIST) (%*LIST)
                            %MACRO ) )
           FI )
%*DEFINE (IRPC (PARM,TEXT) LOCALS (BODY)) LOCAL MACRO LIST (
           %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
           %*DEFINE (LIST) (%TEXT)
           %IF (%LEN(%*LIST) EQ 0) THEN (
                    %DEFINE (%PARM) (%0)
                    %MACRO )
           ELSE (
                   %WHILE (%LEN(%*LIST) NE 0) (
                            %DEFINE (%PARM) (%*SUBSTR(%*LIST,1,1))
                            %DEFINE (LIST) (%*SUBSTR(%*LIST,2,9999))
                            %MACRO ) )
           FI )
```

If the REL control is specified (for converting 8080/8085 source files with relocatability features, and/or for subsequent linking to PL/M-86 modules) CONV86 generates as a prologue:

```
CGROUP    GROUP      ABS_0,CODE,CONST,DATA,STACK,MEMORY
DGROUP    GROUP      ABS_0,CODE,CONST,DATA,STACK,MEMORY
          ASSUME     DS:DGROUP,CS:CGROUP,SS:DGROUP
CODE      SEGMENT    WORD PUBLIC 'CODE'
CODE      ENDS
CONST     SEGMENT    WORD PUBLIC 'CONST'
CONST     ENDS
DATA      SEGMENT    WORD PUBLIC 'DATA'
DATA      ENDS
STACK     SEGMENT    WORD STACK 'STACK'
          DB n DUP(?)
```

```
STACK__BASE  LABEL    BYTE
STACK        ENDS
MEMORY       SEGMENT  WORD MEMORY 'MEMORY'
MEMORY__     LABEL    BYTE
MEMORY       ENDS
ABS__0       SEGMENT  BYTE AT 0
M            LABEL    BYTE
%*DEFINE (REPT (N) LOCALS (BODY)) LOCAL MACRO (
          %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
          %REPEAT (%N) (%MACRO) )
%*DEFINE (IRP (PARM,PLIST) LOCALS (BODY)) LOCAL MACRO LIST (
          %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
          %*DEFINE (LIST) (%PLIST)
          %IF (%LEN(%*LIST) EQ 0) THEN (
                   %DEFINE (%PARM) (%0)
                   %MACRO )
          ELSE (
                   %WHILE (%LEN(%*LIST) NE 0) (
                             %MATCH(%PARM,LIST) (%*LIST)
                             %MACRO ) )
          FI )
%*DEFINE (IRPC (PARM,TEXT) LOCALS (BODY)) LOCAL MACRO LIST (
          %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
          %*DEFINE (LIST) (%TEXT)
          %IF (%LEN(%*LIST) EQ 0) THEN (
                   %DEFINE (%PARM) (%0)
                   %MACRO )
          ELSE (
                   %WHILE (%LEN(%*LIST) NE 0) (
                             %DEFINE (%PARM) (%*SUBSTR(%*LIST,1,1))
                             %DEFINE (LIST) (%*SUBSTR(%*LIST,2,9999))
                             %MACRO ) )
          FI )
```

The statement DB *n* DUP(?) in the STACK segment only appears when the 8080 source file contains a STKLN directive. In that case, *n* corresponds to the operand of the 8080 STKLN directive.

These statements help to set up a pseudo-8080 environment, since an 8086 segment cannot exceed 64K bytes. The register mappings help to complete the pseudo-8080 environment.

NOTE

If more than one module is linked, multiple ABS__0 segments will cause LINK86 to issue error messages concerning SEGMENT OVERLAP. These errors are nonfatal and can be ignored, but you should check your 8080 ASEG (now the 8086 ABS__0 segment) to make sure that you intend the overlap to occur. See Appendix G for further details.

What If a Converted Program Exceeds 64K?

If your 8080 object file exceeds 50K bytes, then there is a chance that your converted source file, when assembled, will exceed 64K bytes and therefore will be too large to fit into a single 8086 segment. (To determine this, you must first convert your 8080 source file, including required manual editing of 8086 source code, and then assemble under the MCS-86 Assembler. An error message will inform you if the resulting MCS-86 object file exceeds 64K bytes.)

If your converted program exceeds 64K bytes, you must reorganize your MCS-86 source code into two or more segments, or else optimize your converted program (by recoding portions directly in more efficient MCS-86 source code).

To reorganize your converted program into two or more segments, you will need to change the GROUP, SEGMENT, and ASSUME assembler directives as described in the manual, *MCS-86 Macro Assembly Language Reference Manual,* Order No. 9800640.

If you need to reorganize your converted program, you can place your data in one segment or group based at absolute location 0, and place your code in another segment or group located above the data segment (or group). You should pay particular attention to absolute addresses and pointers (address values stored as data) in this case, to ensure that your program accesses its data as originally intended.

How Does CONV86 Handle the Stack?

If present, "STKLN" is converted to "DB *n* DUP(?)" in the STACK segment, where *n* is taken from the operand of STKLN. The reserved name STACK is converted to STACK_BASE. (See also "Initializing Registers" under "8086 Checklist" in Chapter 3.)

How Are the 8080/8085 Registers Mapped into 8086 Registers?

Byte registers are mapped as follows:

8080/8085	8086
A	AL
B	CH
C	CL
D	DH
E	DL
H	BH
L	BL

Word registers are mapped as follows:

8080/8085	8086
PSW	AX
B	CX
D	DX
H	BX
SP	SP

How Are the 8080 Flags Mapped into the 8086 Flags?

The 8080 flags correspond to a subset[1] of the 8086 flags as shown in Table 1-1:

Table 1-1. 8080-8086 Flag Correspondence

Flag Name	8080 Designation	8086 Designation
Auxiliary-carry	AC	AF
Carry	C	CF
Zero	Z	ZF
Sign	S	SF
Parity	P	PF

[1] Four 8086 flags do not concern us here: DF (direction), IF (interrupt-enable), OF (overflow), and TF (trap).

How Are 8080/8085 Instructions Mapped into 8086 Instructions?

Appendix A shows how all instructions are mapped. But first, consider that it is not enough simply to map an 8080 instruction mnemonic directly into an 8086 instruction mnemonic, because the instruction operands must be examined as well.

How Are 8080 Operands (Expressions) Converted to 8086 Operands (Expressions)?

8086 Assembly Language is a typed language, whereas 8080/8085 is not. Thus, CONV86 must assign a type—BYTE, WORD, or NEAR—to each symbol encountered in your 8080/8085 source file. Each symbol is typed according to its most frequent usage. After each symbol has been assigned a type (at the end of the first pass of CONV86), CONV86 can explicitly override the type in 8086 source code when necessary.

Appendix B describes the conversion of 8080 expressions into 8086 expressions as a function of the context and the operand or expression type. For example, during its first pass in converting your 8080 source file, CONV86 may find the symbol LASZLO used in three different contexts:

8080

```
LDA     LASZLO    ;Load accumulator with byte at LASZLO.
 .
 .
 .
LHLD    LASZLO    ;Load (H,L) with word at LASZLO.
 .
 .
 .
JMP     LASZLO    ;Jump to symbolic location LASZLO.
```

Since all three usages of the same symbol are permitted in 8080/8085 assembly language, but since 8086 assembly language permits a symbol to be of only one type—BYTE, WORD, or NEAR—then CONV86 must assign a single type to LASZLO. In this case, LASZLO is assigned type BYTE, and the remaining two occurrences of LASZLO are overridden as follows:

```
   8086

MOV     AL, LASZLO                  ;Load AL with byte at LASZLO.
 .
 .
 .
MOV     BX,WORD PTR(LASZLO)         ;Load BX with word at LASZLO.
 .
 .
 .
JMP     NEAR PTR(LASZLO)            ;Jump to symbolic location LASZLO.
```

How Are Comments Mapped?

Comments are mapped unchanged. However, metacharacters (%) or unmatched parentheses in 8080 source comments may be misinterpreted by ASM86.

How Are 8080/8085 Assembler Directives Mapped Into 8086 Assembler Directives?

Appendix C shows the assembler directive mapping.

Operands (expressions) of all directives are mapped according to Appendix B.

How Are 8080/8085 Assembler Controls Mapped?

CONV86 deletes the MOD85, NOMACROFILE, COND, NOCOND, MACRODEBUG and NOMACRODEBUG controls, and issues corresponding caution messages.

The MACROFILE (:Fn:) control, specified with its argument, will be converted to WORKFILES (:Fn:,:Fn:). The MACROFILE control will not be converted correctly if you have not specified it with its optional argument. Such a control can be deleted from the 8080/8085 source file or from the converter output file. All other 8080/8085 assembler controls are copied unchanged to the 8086 output file.

The only 8080/8085 assembler control interpreted by the converter is the INCLUDE control, which causes included files to be processed in the first pass. Included files are neither listed nor converted when the main source file is converted; they are processed in order to evaluate symbol definitions and attributes. The maximum nesting level for included files is four.

How Does CONV86 Handle 8086 Reserved Names?

Whenever CONV86 encounters an 8086 reserved name (such as AL, TEST, or LOOP) in an 8080/8085 source file, CONV86 appends an underscore to the name (thus obtaining AL_, TEST_, or LOOP_). The only exception to this rule is STACK, which is converted to STACK_BASE. As a result, you don't need to be concerned about any 8086 reserved names that might be hiding in your 8080/8085 source files. Appendix D gives a complete list of 8086 reserved names.

Functional Equivalence

What Is Functional Equivalence?

The ideal conversion results in total functional equivalence, which means that the converted 8086 source file, when assembled, linked, located, and run, performs the equivalent function of the input 8080/8085 source file.

CONV86 cannot infer the *intent* of your source program.

While CONV86 cannot usually achieve total[1] functional equivalence on a per- program basis, CONV86 can, in almost every instance, achieve functional equivalence on a line-by-line basis. This means that CONV86 attempts to "map" each 8080/8085 instruction, directive, or control into its 8086 counterpart, if it exists.

Using the instruction mapping of Appendix A, the operand (expression) mapping of Appendix B, and the directive mapping of Appendix C, CONV86 achieves line-by-line functional equivalence. Problems encountered in achieving program functional equivalence arise from:

- Symbol-typing ambiguities — overridden symbol types might not yield the desired 8086 source code. CONV86 flags potential problems of this sort with caution messages.
- Machine-dependent sequences, such as software timing delays or other sequences which depend on instruction length or clock periods.

What About Program Execution Time?

The 8086 assembly-language instructions produced by CONV86 require, in general, more clock periods than did the original 8080/8085 instructions. Thus, the 8086 code produced is less efficient in terms of instruction cycles. However, since the 8086 can be driven by a faster clock, this loss of instruction-cycle efficiency is offset.

What Happens to Software Timing Delays in Conversion?

You should examine the 8086 code derived from timing delay loops. Then, taking into consideration the number of cycles for each 8086 instruction involved, as well as the bandwidth (frequency) of your 8086 clock, you can manually edit the 8086 source code to preserve your timing delays. You should also take into account the 8086 instruction queue (pipeline), which contains six prefetched bytes of in-line code.

Does the 8086 Code Produced Set Flags Exactly as on the 8080?

Yes, unless you specify the APPROX control when you run CONV86. Table 1-2 shows the five 8080 instructions whose 8086 counterparts set flags differently if APPROX is specified. The EXACT control (a default) forces all flag settings to be preserved.

[1] Total functional equivalence on a per-program basis would constrain instruction sequence sizes and clocks to be preserved.

Table 1-2. Flag Settings That Change If APPROX Is Specified

Source 8080 Instruction	8080 Flags Affected	Equivalent 8086 Instruction	8086 Flags Affected
DAD	CY	ADD BX,__	AF,CF,PF,SF,ZF
INX	none	INC	AF,PF,SF,ZF
DCX	none	DEC	AF,PF,SF,ZF
PUSH PSW	none; saved in stack	PUSH AX	none
POP PSW	Z,S,P,CY,AC	POP AX	[SEE NOTE 1]

[NOTE 1: No flags are set if APPROX is specified. EXACT sets AF, CF, PF, SF, and ZF (but not OF).]

How Does the EXACT Control Preserve Flag Semantics?

By inserting the LAHF (load AH with flags) and SAHF (store flags from AH) instructions before and after the 8086 counterpart of the 8080 instruction being converted. For example, the 8080 instruction INX B increments the 16-bit register-pair (B,C) without affecting any 8080/8085 flags, whereas the 8086 instruction INC CX not only increments the 16-bit register CX on the 8086, but also can affect four relevant flags:

- Auxiliary-carry flag (AF)
- Parity flag (PF)
- Sign flag (SF)
- Zero flag (ZF)

If your program is not concerned with these flag settings, then the APPROX mapping will suffice:

```
8080              8086
INX B——(APPROX)——►INC CX
```

However, if your program flow depends on the settings of any of the four flags mentioned, you will want to ensure that in your 8086 program, these flags are saved before INC CX is executed, and restored after INC CX is executed. The EXACT control does this for you as follows:

```
8080              8086      COMMENTS
INX B——(EXACT)——► LAHF     ;Load flags into AH.
                  INC CX
                  SAHF     ;Store flags from AH.
```

Similar flag-preserving code results from EXACT conversion of the 8080/8085 instructions DCX, DAD, PUSH PSW and POP PSW.

When in doubt, let CONV86 default to the EXACT control. More 8086 source code is generated than for APPROX, but the code can be counted on to preserve the flag-setting semantics of your 8080/8085 program.

Editing CONV86 Output for 8086 Assembly

What Output Files Does CONV86 Create?

Table 1-3 shows CONV86 output files, their default extensions, and uses.

Table 1-3. CONV86 Output Files

File Designation in Invoking Command	Default File-Name	Contents and Use
OUTPUT	:Fs:source.A86	Machine-readable 8086 source file; to be manually edited according to caution messages in PRINT file.
PRINT	:Fs:source.LST	1) Optional copy of 8080/8085 source. 2) Human-readable 8086 source file with embedded caution messages for manually editing OUTPUT file.

What Are Caution Messages?

In general, CONV86 issues a caution message when it detects a potential problem in the converted 8086 source code. Caution messages can alert you to possible symbol type ambiguities, such as a symbol used both as a byte and a word, or to possible displaced references, such as JMP $ + (*exp*). In the latter case, the displacement (*exp*) usually increases in going from the 8080 to the 8086. Chapter 3 describes caution messages and identifies what, if anything, you need to do to your 8086 source file.

Does a Caution Message Necessarily Mean a Manual Edit?

No. In some instances, such as displaced references, CONV86 cannot be sure if an error exists. In other instances, such as MOD85 CONTROL DELETED, the converter is simply informing you of a deliberately omitted source file control. Nevertheless, all caution messages and the lines to which they apply demand scrutiny.

Do Caution Messages Identify All Manual Editing?

No. Since CONV86 cannot infer the *intent* of a source program, you must be the final judge as to whether the 8086 source code produced will do a satisfactory job. In particular, you should be alert to machine-dependent sequences of instructions, bearing in mind that instruction sizes (lengths) and execution time (clocks) will change in going from the 8080/8085 to the 8086.

Also, certain 8080/8085 Assembly Language constructs, not valid in the MCS-86 Macro Assembly Language, are not detected by CONV86. These constructs are flagged as errors by ASM86. For example, a nested macro definition that uses the same macro name (a valid construct in the 8080/8085 Assembly Language) is invalid in the MCS-86 Macro Assembly Language. This construct is not detected by CONV86 but it is flagged as an error by ASM86, alerting you about the problem.

The 8080/8085 assembler control MACROFILE is not converted correctly if its optional argument is not present. CONV86 does not issue a caution for this condition and ASM86 processing of the converter output file is terminated by a fatal error, "BAD WORKFILE COMMAND." This problem can be corrected by editing the converter output file or removing the MACROFILE control from the 8080/8085 source file before it is converted.

CHAPTER 2 OPERATING THE CONVERTER

Source File Requirements

Before operating the converter program CONV86, you should ensure that the main source file and all included source files meet the following requirements:

1. The source file must be capable of being assembled without errors by the ISIS-II 8080/8085 Assembler.

2. Diskettes containing files INCLUDEd by the main source file must be mounted on their indicated diskette drives.

3. The maximum source line length is 129 characters, not including carriage-return and line-feed characters. Longer lines are converted to comments and flagged with a caution message.

4. The maximum number of symbols allowed per conversion is approximately 600. Programs having more than 600 symbols must be divided into smaller programs.

CONV86 Controls and Defaults

If the above requirements are met, you can invoke the converter under ISIS-II by entering the command:

```
:Fn:CONV86 source controls
```

where ***source*** is the name of the file to be converted, and ***controls*** are as described in Table 2-1.

Table 2-1. CONV86 Controls and Defaults

CONTROLS	DEFAULTS
PRINT(path-name) / NOPRINT	PRINT(:Fs:source.LST)
OUTPUT(path-name) / NOOUTPUT	OUTPUT(:Fs:source.A86)
DATE('date')	DATE(' ')
TITLE('title')	TITLE(' ')
PAGELENGTH(n) / NOPAGING	PAGELENGTH(60)
PAGEWIDTH(n)	PAGEWIDTH(120)
EXACT / APPROX	EXACT
INCLUDED / NOTINCLUDED	NOTINCLUDED
ABS/REL	REL
WORKFILES(:Fn:)	WORKFILES(:Fs:)
SOURCELIST/NOSOURCELIST	SOURCELIST

where:

Fs

specifies the diskette unit on which the source file resides.

PRINT

specifies an ISIS-II path-name (file or device designation) for a copy of your 8080/8085 source code together with generated 8086 source code and embedded caution messages.

NOPRINT

specifies that the PRINT file is not to be created.

OUTPUT

specifies an ISIS-II path-name for the output 8086 source code. Refer to Table 1-3, "CONV86 Output Files."

NOOUTPUT

specifies that the OUTPUT file is not to be created.

DATE

specifies a date (or other information) of up to nine characters to be printed in the page header of the PRINT file.

TITLE

specifies a title (or other information) of up to 40 characters to be printed in the page header of the PRINT file.

PAGELENGTH(n)

specifies the number of lines per output page in the PRINT file. The minimum is four lines per page; there is no effective maximum.

NOPAGING

specifies no forms control and is equivalent to PAGELENGTH (65535).

PAGEWIDTH(n)

specifies the number of characters per output line in the PRINT file. The minimum is 60 characters per line; there is no effective maximum.

EXACT

specifies that full flag-setting semantics are to be preserved in conversion. This control affects conversion of the DAD, DCX, INX, POP PSW, and PUSH PSW.

APPROX

specifies that full flag-setting semantics are not to be preserved for the instructions DAD, DCX, INX, POP PSW, and PUSH PSW. Refer to

Chapter 1, "Functional Equivalence," for a description of flag preservation.

INCLUDED

specifies that this module is included in another module for assembly. This control suppresses generation of a standard prologue.

NOTINCLUDED

specifies that this module is not included in another module for assembly. The converter therefore generates a standard prologue. Refer to Chapter 1, "Functional Mapping," for a description of prologues.

REL

specifies that this module will subsequently be assembled in relocatable format and/or linked to a PL/M-86 module. If REL and NOTINCLUDED are both specified or defaulted to (both are defaults), the standard prologue generated is compatible with PL/M-86, and informs the converter that 8080 relocation capabilities are present in the source file and must be mapped into 8086 relocation features. See "Functional Mapping" in Chapter 1.

ABS

specifies that this module is absolute and not relocatable (and hence not to be linked to a PL/M-86 module). If ABS and NOTINCLUDED are both in effect (NOTINCLUDED is a default), then the standard prologue generated is not compatible with PL/M-86, but is compatible with other 8086 assemblies. See "Functional Mapping" in Chapter 1 for a description of standard prologues.

WORKFILES(:Fn:)

specifies that the single, temporary workfile CONV86.TMP is to be created on (and subsequently deleted from) diskette unit :Fn:, where n defaults to the source file diskette unit number if the WORKFILES control is omitted. The single workfile created (the plural WORKFILES is used for consistency with other programs) requires seven (7) bytes for each source line.

SOURCELIST

specifies that the 8080/8085 source program is to be listed in the PRINT file (overridden by NOPRINT).

NOSOURCELIST

specifies that the 8080/8085 source program is not to be listed in the PRINT file.

Examples

Example 1. Full Default Saves Flags and Relocatability

Suppose CONV86 resides on diskette unit 0, and that the program to be converted is named MYASM.A80 and resides on diskette unit 1. Then the command:

```
CONV86 :F1:MYASM.A80
```

invokes the converter and results in the following controls:

- The 8080 source file and 8086 source file with embedded cautions are written to the file :F1:MYASM.LST
- The converted file (without embedded caution messages) is placed in the file :F1:MYASM.A86
- Blanks appear in the title and date fields of page headers.
- Page lengths default to 60 lines per page.
- Page widths (line lengths) default to 120 characters, not including carriage-return or line-feed.
- Flag-setting semantics are preserved for all instructions.
- The prologue generated in the OUTPUT file :F1:MYASM.A86 will cause the MCS-86 Assembler to generate relocatable object modules suitable for linking with other assemblies or PL/M-86 object modules.
- The temporary workfile CONV86.TMP is created on, and deleted from, diskette unit 1, the default.

Example 2: Absolute Code with No Flags Saved

If, in Example 1, you had entered the command:

```
CONV86 :F1:MYASM.A80 ABS APPROX
```

then the results would differ as follows:

- Full flag-setting semantics are *not* preserved for DAD, DCX, INX, PUSH PSW, or POP PSW.
- A standard 8086 assembly language absolute prologue is generated in the converted code. This prologue is not compatible with PL/M-86, but is compatible with other 8086 assemblies. Your MCS-86 Assembler object file will not be relocatable.

Example 3: Absolute Code with Flags Saved

The invoking command:

```
CONV86 :F1:MYASM.A80 ABS
```

generates an absolute prologue, and defaults to EXACT.

Example 4: Relocatable Code with No Flags Saved

The invoking command:

```
CONV86 :F1:MYASM.A80 APPROX
```

does not preserve flag semantics for the five instructions just mentioned, and defaults to REL.

NOTE

In the following examples, the double asterisks (**) indicating prompting are generated internally, and not by the user.

Example 5: Prompting and Continuation Lines

You need not enter the entire invoking command on a single line. If you wish to continue the command on one or more subsequent lines, you must enter an ampersand (&) as the last character of the current line. Characters entered following the ampersand and preceding the carriage-return are comments; they are echoed by CONV86 in the PRINT file header but are not processed. The converter then prompts for more command input with a double asterisk:

```
CONV86 :F1:MYASM.A80 & source file is MYASM.A80 on disk drive 1

** DATE('10/5/78') & date cannot exceed 9 chars. excluding quotes

** TITLE('CONVERSION TEST 39, PROJECT AXOLOTL') & 40 chars.
```

The date and title are included in the PRINT file headers as shown in Figure 1-3, Chapter 1. The remaining controls default as in Example 1.

Example 6: Overriding Controls

It may happen that you have entered a control incorrectly, or for some other reason wish to override a previously entered control. You can override any previously entered controls so long as prompting is in effect. Suppose you have entered the following:

```
CONV86 :F1:MYASM.80 &

** DATE('10/5/39') &

** TITLE('CONVERSION TEST 78, PROJECT AXOLOTL') &
```

If you happen to notice at this point that the wrong information has been entered — that is, the 39 and 78 have been interchanged, there is no problem, since prompting is still in effect. On subsequent continuation lines, you can enter:

```
** DATE('10/5/78') &

** TITLE('CONVERSION TEST 39, PROJECT AXOLOTL') &

**
```

Controls can be entered in any order and overridden in any order as many times as necessary. For this reason, it is good practice to end every line with an unquoted ampersand. When you are satisfied that the controls are correct, you can end the command with the last line consisting of a lone carriage return.

Console Output

When you have entered the command invoking CONV86, the converter responds with the message:

```
ISIS-II ASM80 TO ASM86 CONVERTER Vx.y
```

where x.y is the version designation.

Normal termination of the converter causes it to issue the message:

```
ASM80 TO ASM86 CONVERSION COMPLETE

nnnnn CAUTIONS ISSUED
```

where nnnnn is the number of messages generated for the run. Caution messages are described in Chapter 3.

CONV86 terminates abnormaly (aborts) if I/O or other fatal errors occur during execution, or if CONV86 has not been properly invoked.

Fatal I/O console messages are of the form:

```
ASM80-TO-86 I/O ERROR—

FILE: file-type

NAME: file-name

ERROR: error-message

CONVERSION TERMINATED
```

Table 2-2 shows the relationship between file-type and file-name.

Table 2-2. File-types and File-names in CONV86 Fatal I/O Errors

FILE-TYPE	FILE-NAME
LIST	Specified by PRINT control
OUTPUT	Specified by OUTPUT control (or default)
SOURCE	Specified by source field of command
INCLUD	Specified by ASM80 INCLUDE control
TEMP	CONV86.TMP—temporary work file
:CI:	Refers to console input device

Error-message is one of the following:

```
04 — ILLEGAL FILENAME SPECIFICATION
05 — ILLEGAL OR UNRECOGNIZED DEVICE SPECIFICATION IN FILENAME
12 — ATTEMPT TO OPEN AN ALREADY OPEN FILE
13 — NO SUCH FILE
14 — FILE IS WRITE PROTECTED
19 — FILE IS NOT ON A DIRECT ACCESS DEVICE
22 — DEVICE NAME NOT COMPATIBLE WITH INTENDED FILE USE
23 — FILENAME REQUIRED ON DIRECT ACCESS FILE
28 — NULL FILE EXTENSION
254 — ATTEMPT TO READ PAST EOF
```

Fatal errors (other than I/O) result in the following console display:

```
ASM80-TO-86 FATAL ERROR—

message

CONVERSION TERMINATED
```

Messages corresponding to (non-I/O) fatal errors are as follows:

MESSAGE	ACTION
CONDITIONALLY ASSEMBLED MACRO	Remove conditional directives
CONDITIONALLY ASSEMBLED ENDM	Remove conditional directives
INVALID FILENAME	Examine, correct file name
INVALID CONTROL FORMAT	Refer to beginning of Chapter 2
CONTROL STRING TOO LONG	Reduce length(s) of DATA/TITLE strings
INVALID CONTROL VALUE	Refer to Controls description
INVOCATION COMMAND DOES NOT END WITH <CR> <LF>	Reenter with carriage return
UNKNOWN CONTROL	Refer to Controls description
INSUFFICIENT MEMORY FOR DICTIONARY	Reduce the number of symbols used in your program
MAXIMUM MACRO NESTING LEVEL EXCEEDED	Check for recursive macro calls; reduce the number of nested macro calls

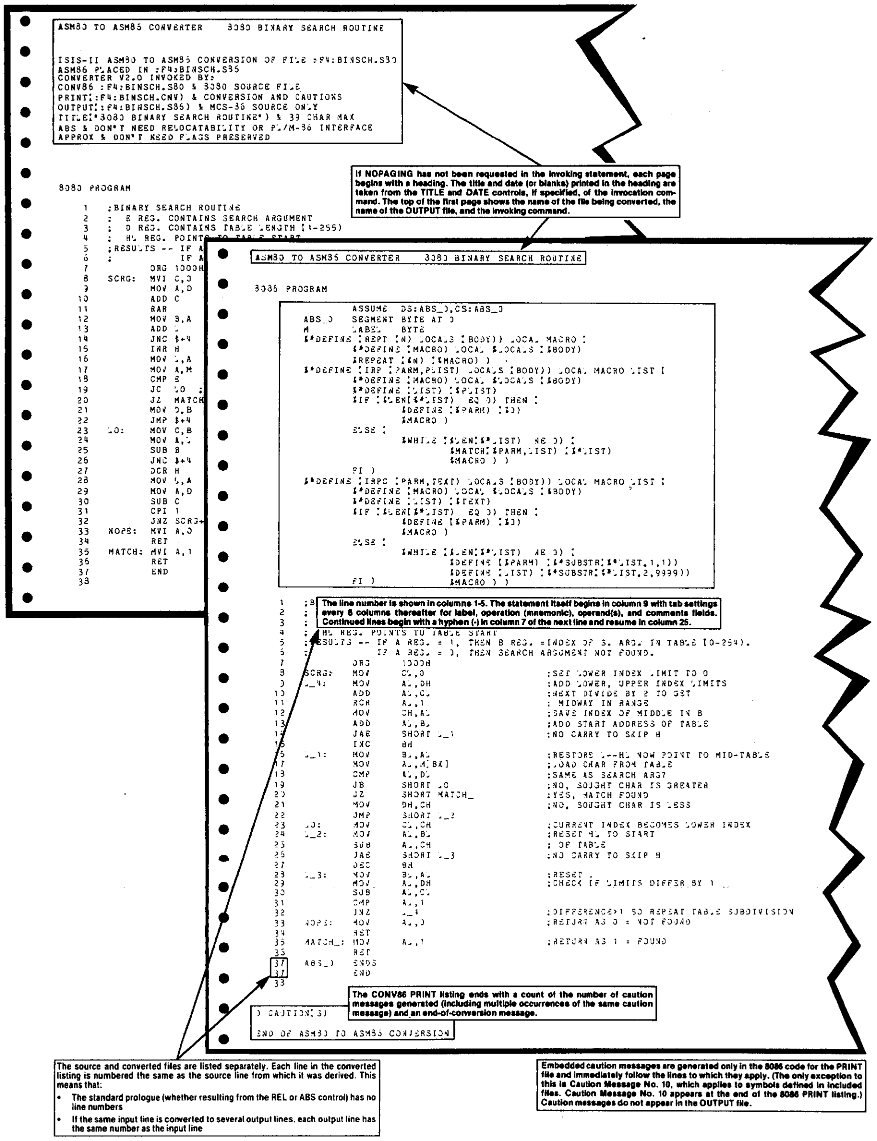
CHAPTER 3
EDITING CONVERTER OUTPUT

Interpreting the PRINT File

After you have run CONV86 and it has terminated normally, you should examine the PRINT file. As shown in Figure 3-1, the PRINT file consists of:

- A copy of the 8080/8085 assembly-language source file, unless the NOSOURCELIST control was specified
- MCS-86 assembly-language source code with embedded caution messages

Using the PRINT file as a reference, you can manually edit the OUTPUT file to obtain 8086 source code that can be assembled by the MCS-86 Macro Assembler.



```
ASM80 TO ASM86 CONVERTER    8080 BINARY SEARCH ROUTINE


ISIS-II ASM80 TO ASM86 CONVERSION OF FILE :F4:BINSCH.S80
ASM86 PLACED IN :F4:BINSCH.S86
CONVERTER V2.0 INVOKED BY:
CONV86 :F4:BINSCH.S80 & 8080 SOURCE FILE
PRINT(:F4:BINSCH.CNV) & CONVERSION AND CAUTIONS
OUTPUT(:F4:BINSCH.S86) & MCS-86 SOURCE ONLY
TITLE('8080 BINARY SEARCH ROUTINE') & 39 CHAR MAX
ABS & DON'T NEED RELOCATABILITY OR PL/M-86 INTERFACE
APPROX & DON'T NEED FLAGS PRESERVED


8080 PROGRAM

     1  ;BINARY SEARCH ROUTINE
     2  ;  E REG. CONTAINS SEARCH ARGUMENT
     3  ;  D REG. CONTAINS TABLE LENGTH (1-255)
     4  ;  HL REG. POINTS
     5  ;RESULTS -- IF A
     6  ;           IF A
     7          ORG 1000H
     8  SCRG:   MVI C,0
     9          MOV A,D
    10          ADD C
    11          RAR
    12          MOV B,A
    13          ADD L
    14          JNC $+4
    15          INR H
    16          MOV L,A
    17          MOV A,M
    18          CMP E
    19          JC  LO  ;
    20          JZ  MATCH
    21          MOV D,B
    22          JMP $+4
    23  LO:     MOV C,B
    24          MOV A,L
    25          SUB B
    26          JNC $+4
    27          DCR H
    28          MOV L,A
    29          MOV A,D
    30          SUB C
    31          CPI 1
    32          JNZ SCRG+
    33  NOPE:   MVI A,0
    34          RET
    35  MATCH:  MVI A,1
    36          RET
    37          END
    38
```

If NOPAGING has not been requested in the invoking statement, each page begins with a heading. The title and date (or blanks) printed in the heading are taken from the TITLE and DATE controls, if specified, of the invocation command. The top of the first page shows the name of the file being converted, the name of the OUTPUT file, and the invoking command.

```
ASM80 TO ASM86 CONVERTER    8080 BINARY SEARCH ROUTINE


8086 PROGRAM

                ASSUME  DS:ABS_0,CS:ABS_0
        ABS_0   SEGMENT BYTE AT 0
        M       LABEL   BYTE
        %*DEFINE (REPT (N) LOCALS (BODY)) LOCAL MACRO (
                %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
                %REPEAT (%N) (%MACRO) )
        %*DEFINE (IRP (PARM,PLIST) LOCALS (BODY)) LOCAL MACRO LIST (
                %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
                %*DEFINE (LIST) (%PLIST)
                %IF (%LEN(%*LIST)  EQ 0) THEN (
                        %DEFINE (%PARM) (%0)
                        %MACRO )
                ELSE (
                        %WHILE (%LEN(%*LIST)  NE 0) (
                                %MATCH(%PARM,LIST) (%*LIST)
                                %MACRO ) )
                FI )
        %*DEFINE (IRPC (PARM,TEXT) LOCALS (BODY)) LOCAL MACRO LIST (
                %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
                %*DEFINE (LIST) (%TEXT)
                %IF (%LEN(%*LIST)  EQ 0) THEN (
                        %DEFINE (%PARM) (%0)
                        %MACRO )
                ELSE (
                        %WHILE (%LEN(%*LIST)  NE 0) (
                                %DEFINE (%PARM) (%*SUBSTR(%*LIST,1,1))
                                %DEFINE (LIST) (%*SUBSTR(%*LIST,2,9999))
                FI )                    %MACRO ) )
```

The line number is shown in columns 1-5. The statement itself begins in column 9 with tab settings every 8 columns thereafter for label, operation (mnemonic), operand(s), and comments fields. Continued lines begin with a hyphen (-) in column 7 of the next line and resume in column 25.

```
     1  ;B
     2  ;
     3  ;
     4  ;  HL REG. POINTS TO TABLE START
     5  ;RESULTS -- IF A REG. = 1, THEN B REG. =INDEX OF S. ARG. IN TABLE (0-254).
     6  ;           IF A REG. = 0, THEN SEARCH ARGUMENT NOT FOUND.
     7          ORG     1000H
     8  SCRG:   MOV     CL,0                    ;SET LOWER INDEX LIMIT TO 0
     9  L_4:    MOV     AL,DH                   ;ADD LOWER, UPPER INDEX LIMITS
    10          ADD     AL,CL                   ;NEXT DIVIDE BY 2 TO GET
    11          RCR     AL,1                    ; MIDWAY IN RANGE
    12          MOV     CH,AL                   ;SAVE INDEX OF MIDDLE IN B
    13          ADD     AL,BL                   ;ADD START ADDRESS OF TABLE
    14          JAE     SHORT L_1               ;NO CARRY TO SKIP H
    15          INC     BH
    16  L_1:    MOV     BL,AL                   ;RESTORE L--HL NOW POINT TO MID-TABLE
    17          MOV     AL,M[BX]                ;LOAD CHAR FROM TABLE
    18          CMP     AL,DL                   ;SAME AS SEARCH ARG?
    19          JB      SHORT LO                ;NO, SOUGHT CHAR IS GREATER
    20          JZ      SHORT MATCH_            ;YES, MATCH FOUND
    21          MOV     DH,CH                   ;NO, SOUGHT CHAR IS LESS
    22          JMP     SHORT L_2
    23  LO:     MOV     CL,CH                   ;CURRENT INDEX BECOMES LOWER INDEX
    24  L_2:    MOV     AL,BL                   ;RESET HL TO START
    25          SUB     AL,CH                   ; OF TABLE
    26          JAE     SHORT L_3               ;NO CARRY TO SKIP H
    27          DEC     BH
    28  L_3:    MOV     BL,AL                   ;RESET L
    29          MOV     AL,DH                   ;CHECK IF LIMITS DIFFER BY 1
    30          SUB     AL,CL
    31          CMP     AL,1
    32          JNZ     L_4                     ;DIFFERENCE>1 SO REPEAT TABLE SUBDIVISION
    33  NOPE:   MOV     AL,0                    ;RETURN AS 0 = NOT FOUND
    34          RET
    35  MATCH_: MOV     AL,1                    ;RETURN AS 1 = FOUND
    36          RET
    37  ABS_0   ENDS
    37          END
    38

0 CAUTION(S)

END OF ASM80 TO ASM86 CONVERSION
```

The CONV86 PRINT listing ends with a count of the number of caution messages generated (including multiple occurrences of the same caution message) and an end-of-conversion message.

The source and converted files are listed separately. Each line in the converted listing is numbered the same as the source line from which it was derived. This means that:

- The standard prologue (whether resulting from the REL or ABS control) has no line numbers
- If the same input line is converted to several output lines, each output line has the same number as the input line

Embedded caution messages are generated only in the 8086 code for the PRINT file and immediately follow the lines to which they apply. (The only exception to this is Caution Message No. 10, which applies to symbols defined in included files. Caution Message No. 10 appears at the end of the 8086 PRINT listing.) Caution messages do not appear in the OUTPUT file.

Figure 3-1. Annotated PRINT File

8086 Checklist

Caution messages and the modifications they may require are described later in this chapter. This section provides a list of items that you should check yourself.

1. **Initializing Registers.** Before your converted program can be assembled for subsequent linking, locating, and execution, you must insert register initialization code at the entry point to your main program. The register initialization code that you insert must be the first sequence of instructions executed by your program. If you omit this code from your main program, neither the segment registers nor the stack pointer (SP) can be depended on to contain meaningful data, and the results are unpredictable.

   The code that you insert follows. Note that *expr* should not be coded verbatim; what you substitute for *expr* depends on whether you converted using the ABS or REL control (REL is the default), and how your 8080/8085 program initialized SP.

```
mainentrypoint:   CLI            ;First instruction to be executed in your main
                                 ;program
                  MOV AX,CS      ;Use CS to initialize:
                  MOV DS,AX      ; —data segment register
                  MOV ES,AX      ; —extra segment register
                  MOV SS,AX      ; —stack segment register
                  LEA SP, expr   ;See below for what to code for expr
                  STI            ;Enable interrupts
```

   where:

   mainentrypoint is the symbolic location of the first instruction to be executed in your main program. If, in your original 8080 program development, you used the 8080 LOCATE control RESTART0 (to have the locater insert code to jump to the entry point of your main module when the 8080 was reset), the corresponding LOC86 control is BOOTSTRAP.

   expr is STACK_BASE if you converted using the REL control *and* your original 8080 program used the STKLN directive to set the stack size.

   Otherwise *expr* is a constant, expression, or program label that your original 8080 program used to set SP. For constants or expressions, you should check that these values are really what you want.

   You should check every instance in your program where SP is loaded to ensure that the stack reinitialization has the intended effect in your converted program.

2. **Absolute Addressing.** Absolute addresses should be checked for correctness. This includes ORGs in the absolute segment, LHLD and LDA from a constant location, and immediate operations such as LXI whose constant operands represent addresses. Remember that 8086 instruction lengths are generally different from those of their 8080/8085 counterparts.

3. **Relative Addressing.** Relative addressing should be checked, since the number of bytes between instructions will in general increase in going from 8080/8085 to 8086. In some instances, CONV86 generates and inserts a label of the form L_*n* for a displaced reference, as in the following:

8080 Source **MCS-86 (CONV86-Generated) PRINT File**

```
 2          MOV D,B    |  2              MOV      DH,CH
 3          JMP $+4    |  3              JMP      SHORT L_1
 4    LO:   MOV C,B    |  4    LO:       MOV      CL,CH
 5          MOV A,L    |  5    L_1:      MOV      AL,BL
```

In some instances, however, CONV86 does *not* generate such a label, as in the following:

8080 Source **MCS-86 (CONV86-Generated) PRINT File**

```
 7        MOV A,C                 |  7               MOV      AL,CL
 8        JMP $+3*((3+2)*2-7)     |  8               JMP      $+3*((3+2)*2-7)
 9        DB  78H                 |  CAUTION 017     ***    ADDRESS EXPRESSION
10        DB  10111101B           |  9               DB       78H
11        DW  0BABAH              | 10               DB       10111101B
12        DW  0BEACH              | 11               DW       0BABAH
13        CMA                     | 12               DW       0BEACH
                                  | 13               NOT      AL
```

CONV86 does not attempt to evaluate the expression or insert a label, although Caution Message 17 is issued for a possible displaced reference. Thus, it is up to you to insert a label. At the same time, since the jump (forward) is less than 127 bytes, the SHORT label attribute can be used, as follows:

CONV86 OUTPUT File

```
     MOV     AL,CL                |                MOV      AL,CL
     JMP     $+3*((3+2)*2-7)      |                JMP      SHORT LASZLO
     DB      78H                  |                DB       78H
     DB      10111101B            |                DB       10111101B
     DW      0BABAH               |                DW       0BABAH
     DW      0BEACH               |                DW       0BEACH
     NOT     AL                   |    LASZLO:     NOT      AL
```

Before Your Edit **After Your Edit**

In general, you should check all relative addressing.

4. **Interrupts.** Figure 3-2 shows how interrupt service routines on the 8080/8085 can be converted to interrupt service routines on the 8086.

   The principal difference between the two schemes is that on the 8080/8085, control traps to location 8*N, where executable code resides; whereas on the 8086, control traps to the location *pointed to* by the 16-bit offset and 16-bit base values stored at location 4*N.

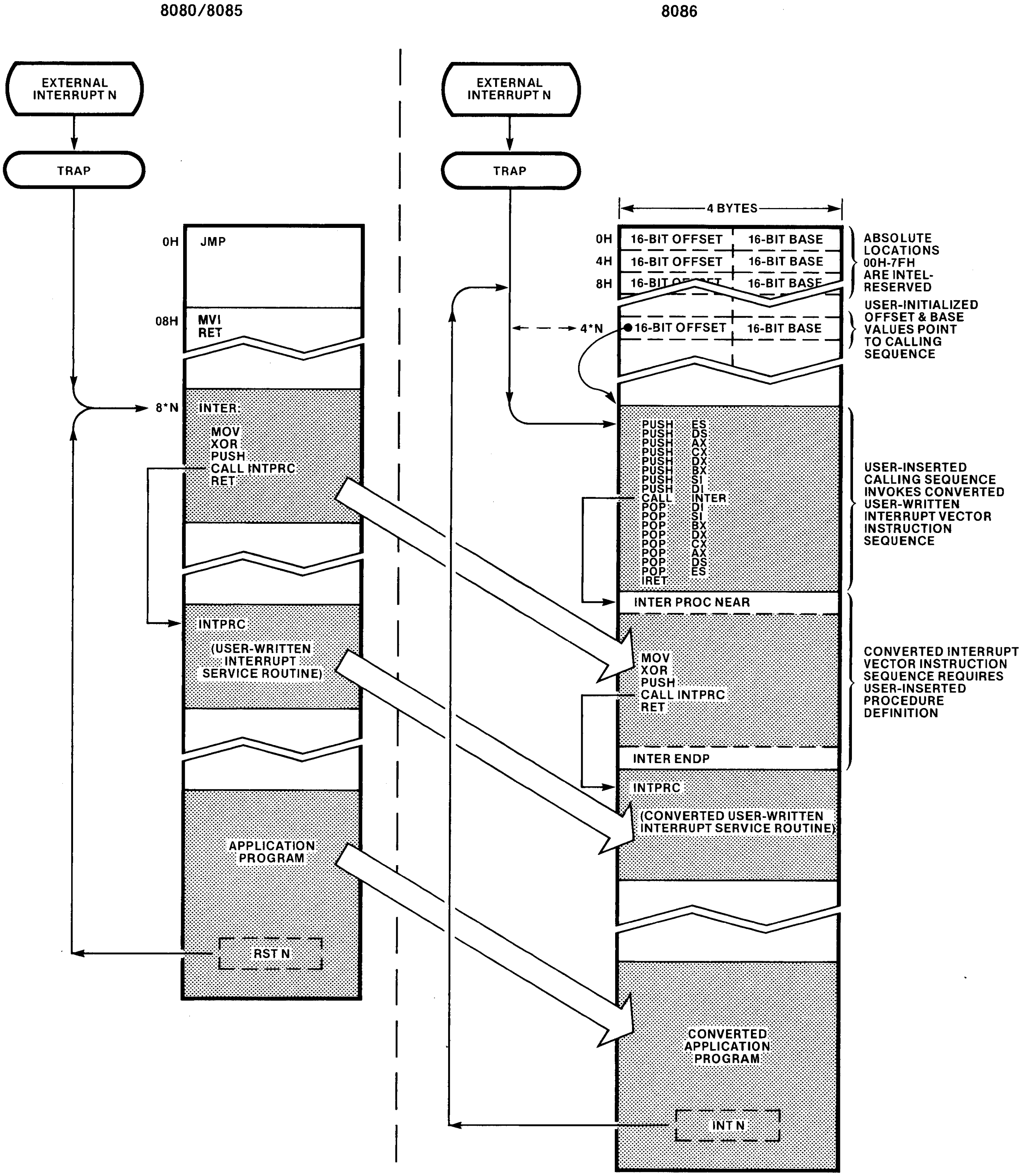


Figure 3-2. Converting Your Interrupt Procedures

You can convert your 8080 interrupt service routines as follows:

1. Insert, at a convenient place in your 8086 source code, the following calling sequence, using your own label (be sure not to use a reserved name given in Appendix D):

```
INTSEQ:   PUSH   ES
          PUSH   DS
          PUSH   AX
          PUSH   CX
          PUSH   DX
          PUSH   BX
          PUSH   SI
          PUSH   DI
          CALL   INTER  ;INTER used here for example in Figure 3-2.
          POP    DI
          POP    SI
          POP    BX
          POP    DX
          POP    CX
          POP    AX
          POP    DS
          POP    ES
          IRET          ;Note that this is IRET, and not RET.
```

2. Insert the following initialization sequence for absolute location 4*N in the ABS_0 segment:

```
 ORG 4*N                       ;N is the interrupt number on the 8086.
                               ;INTSEQ used here for example above.

{ DD  CGROUP:INTSEQ             ;If REL control was used. }
{ DD  INTSEQ                    ;If ABS control was used. }
```

3. Sandwich the converted code from INTER (used here for example in Figure 3-2) between PROC and ENDP statements as follows:

```
INTER PROC NEAR    ;Nothing special about the word INTER.

[converted code]

INTER ENDP         ;Nothing special about the word INTER.
```

While these steps are general enough to cover virtually any application, you may find that as you become familiar with the 8086, you can recode your interrupt service routines in MCS-86 Macro Assembly Language to obtain optimal code more suited to your application.

PL/M-86 Linkage Conventions

The only PL/M-86 model of computation relevant to conversion is the SMALL model.

Case 1: When PL/M Calls

Converted assembly-language programs called from PL/M programs must be changed if *any* parameters are passed, since PL/M-80 passes parameters in registers and on the stack, and PL/M-86 passes *all* parameters on the stack. PL/M-86 parameter passing is as follows:

- Arguments are pushed on the stack in left-to-right order and therefore occupy successively lower memory locations. The return address is pushed on the stack last.
- Each argument occupies two bytes. One-byte arguments are passed in the lower half (least significant byte) of a word.

Therefore, converted 8086 assembly language programs called from PL/M-86 programs need to access arguments from the stack, and not from registers. However, since the calling PL/M-86 program has pushed the return address on the stack last, the called 8086 assembly language program needs to:

1. POP the return address to any convenient word register, such as BX.
2. POP arguments as needed into their 8086 register counterparts, as follows:
   - If no arguments are expected, POP no further. Go to Step 3 below.
   - If one argument is expected, then it was originally expected in (B,C). Therefore the converted assembly language program is accessing the single argument from the 8086 CX register. This means that you need to insert the instruction:

     ```
     POP CX      ;Retrieve only PL/M-86 argument.
     ```

     immediately after POP BX (the return address) in order for the converted 8086 assembly language program to access the single argument as intended.
   - If two arguments are expected, then they were originally expected in (B,C) and (D,E). Therefore the converted assembly language program accesses its arguments from the 8086 CX and DX registers. Since PL/M-86 passes these arguments on the stack *in order*, this means that you need to insert the instructions:

     ```
     POP DX     ;Retrieve second PL/M-86 argument.
     POP CX     ;Retrieve first PL/M-86 argument.
     ```

     immediately after POP BX (the return address) in order for the converted 8086 assembly language program to access the two arguments as intended.
   - If more than two arguments are expected, the remainder are in the stack (where the converted assembly language program expects them), and there is no problem. The last two arguments are accessed as described in the preceding paragraph.
3. PUSH the return address back on the stack *immediately* after accessing the arguments as just described. If BX was used in Step 1 above to retain the return address, then you need to insert the instruction:

   ```
   PUSH BX     ;Replace return address on stack.
   ```

   immediately following your argument-accessing sequence of POPs.
4. PL/M-86 expects the return value (a one-word pointer or data item) of the assembly language program to be in the AX register. If the return value is a byte, it is expected in AL.

Case 2: When Your Converted Program Calls

If your 8080/8085 source program calls another routine (written either in MCS-86 Macro Assembly Language or PL/M-86) which expects arguments to be passed on the stack, you need to insert 8086 source code in your converted program.

If your original 8080 source program passed only one argument to the CALLed routine, that argument was passed in the (B,C) register-pair. Hence you need to insert:

```
PUSH CX     ;Push (B,C) argument on stack.
```

immediately before the CALL.

If your original 8080 source program passed two or more arguments to the CALLed routine, those arguments were passed in the (B,C) register-pair, in the (D,E) register-pair, and remaining arguments on the stack. Hence you need to insert:

```
PUSH CX     ;Push (B,C) argument on stack.
PUSH DX     ;Push (D,E) argument on stack.
```

immediately before the CALL. The remaining arguments (if any) are already on the stack in the correct order. PL/M-86 return values are placed in AX or AL as described in Case 1.

Caution Messages

Caution messages do not necessarily imply manual editing, but they do demand scrutiny. In many cases, CONV86 cannot be sure if an error actually exists (as for instance, in expression evaluation). This section lists all possible caution messages. The next section lists caution message descriptions and indicates what manual editing of the output file may be necessary.

The entire list of caution messages is as follows (note that caution messages 9, 15, 26, and 29 do not exist):

1 BYTE REGISTER USED IN WORD CONTEXT OR VICE VERSA

2 8080 REGISTER MNEMONIC APPEARING IN IRPC STRING

3 MACRO PARAMETER BOTH CONCATENATED AND USED AS PARAMETER

4 EXPANDED NAME MAY BE RESERVED WHEN CONCATENATED

5 MACRO PARAMETER USED IN BOTH BYTE AND WORD CONTEXTS

6 EQU'D OR SET REGISTER SYMBOL USED IN BOTH BYTE AND WORD CONTEXTS

7 MULTIPLY DEFINED EQU MAY NOT BE ASSIGNED PROPER TYPE

8 UNKNOWN STATEMENT

10 TYPE ASSIGNED TO INCLUDED SYMBOL MAY NOT AGREE WITH DEFINITION

11 TRANSLATION OF NOP MAY NOT YIELD DESIRED RESULTS

12 TRANSLATION OF RST MAY NOT YIELD DESIRED RESULTS

13 8085-SPECIFIC INSTRUCTION CANNOT BE TRANSLATED

14 FORWARD REFERENCE TO A SYMBOL WHICH IS A REGISTER OR [BX] CANNOT BE CORRECTLY ASSEMBLED

16 EXPRESSION ASSUMED TO BE A VARIABLE

17 ADDRESS EXPRESSION MAY BE INVALID FOR 8086

18 INSTRUCTION AS OPERAND CANNOT BE TRANSLATED

19 REGISTER USED IN UNKNOWN CONTEXT

20 OUTPUT LINE TOO LONG; TRUNCATED

21 LABEL ASSUMED TO BE NEAR

22 NOMACROFILE CONTROL DELETED

23 MOD85 CONTROL DELETED

24 SOURCE LINE TOO LONG; IGNORED

25 CURRENT SEGMENT UNKNOWN; CANNOT GENERATE ENDS

27 SYMBOL NAME TOO LONG

28 CONDITIONAL ASSEMBLY GENERATED

30 UNKNOWN INSTRUCTION ASSUMED TO BE A MACRO

31 GENERATED LABEL MIGHT NEED TO BE DECLARED LOCAL

32 (NO) COND CONTROL DELETED

33 (NO) MACRODEBUG CONTROL DELETED

34 METACHARACTER OR PARENTHESIS FOUND IN IRPC STRING

35 EXPRESSION ASSUMED TO BE A CONSTANT

36 SYMBOLIC EXPRESSION MAY BE CONTEXTUALLY INVALID FOR ASM86

Caution Message Descriptions

1 BYTE REGISTER USED IN WORD CONTEXT OR VICE VERSA

A register variable defined in an EQU directive or as a macro parameter has been classed as BYTE or WORD according to its predominant usage. In this statement, the register variable appears in the opposite context. This is unacceptable for the 8086, since byte and word register mnemonics are different. You should insert the appropriate register mnemonic.

2 8080 REGISTER MNEMONIC APPEARING IN IRPC STRING

The parameter of this IRPC directive is used in a register context. Since 8086 register mnemonics are two characters long, you should change the IRPC directive (possibly to an equivalent IRP).

3 MACRO PARAMETER BOTH CONCATENATED AND USED AS PARAMETER

One of the arguments of this macro is both concatenated and used as a register. You may need to manually convert the mnemonics yourself.

4 EXPANDED NAME MAY BE RESERVED WHEN CONCATENATED

One of the arguments of this macro is concatenated. You should examine the resulting symbol and see if it corresponds to the intent of the 8080/8085 source code. You should also check to see if the resulting concatenated name is reserved. A list of reserved symbols appears in Appendix D.

5 MACRO PARAMETER USED IN BOTH BYTE AND WORD CONTEXTS

A macro argument is used in both byte and word register contexts. Since the argument can be of only one type, you should manually alter the macro or override the argument type.

6 EQU'D OR SET REGISTER SYMBOL USED IN BOTH BYTE AND WORD CONTEXTS

An EQU or SET symbol is used in both byte register and word register contexts. You should manually insert the appropriate register mnemonic(s). You may need to use two EQUs: one for byte usage, and one for word usage.

7 MULTIPLY DEFINED EQU MAY NOT BE ASSIGNED PROPER TYPE

An EQU symbol has been multiply defined, perhaps due to conditional compilation. You should eliminate the excess definition(s), and redefine as necessary. CONV86 may have assigned the wrong type.

8 UNKNOWN STATEMENT

The converter is unable to recognize this statement, possibly because its mnemonic is a macro parameter. You should either recode the 8080 source to produce recognizable statements (legal instructions) and submit the recoded 8080 file to CONV86, or else simply insert the appropriate 8086 source code in the OUTPUT file.

10 TYPE ASSIGNED TO INCLUDED SYMBOL MAY NOT AGREE WITH DEFINITION

The specified symbol is defined in an INCLUDE file. When the INCLUDE file is converted, the usage of the symbol may not be the same as inferred by CONV86 here. You should convert the INCLUDE file and examine the type CONV86 has assigned to it there, and then ensure that both usages are the same. If they are not, you should override the assigned usage in either file so as to make their types identical.

11 TRANSLATION OF NOP MAY NOT YIELD DESIRED RESULTS

An NOP instruction has been converted to XCHG AX,AX. This may not be the desired mapping, as it assembles into a one-byte instruction (3 clocks).

12 TRANSLATION OF RST MAY NOT YIELD DESIRED RESULTS

An RST instruction has been converted to an INT instruction for the 8086. You should verify that the original intent of the RST instruction was to cause an interrupt. You should examine the operand carefully to ensure that the instruction traps to the desired absolute address, and that the intended routine to be trapped to will be bound to (loaded at) that address.

13 8085-SPECIFIC INSTRUCTION CANNOT BE TRANSLATED

The 8086 has no counterpart for RIM or SIM. You should recode according to the 8086 interrupt scheme as described in the *8086 Family User's Manual* under "Interrupts."

14 FORWARD REFERENCE TO A SYMBOL WHICH IS A REGISTER OR [BX] CANNOT BE CORRECTLY ASSEMBLED

The 8086 assembler does not accept forward references to registers. You should move your register EQUs to the beginning of your file.

16 EXPRESSION ASSUMED TO BE A VARIABLE

CONV86 has not been able to determine what type of expression is in this instruction. CONV86 has assumed that the expression is a variable. If this assumption is incorrect, you should examine the resulting 8086 statement and recode the mapped expression to suit your intent. You may find it helpful to insert additional labels.

17 ADDRESS EXPRESSION MAY BE INVALID FOR 8086

Case 1: Displaced Reference

CONV86 may not have mapped a displaced symbol reference (for instance, $+BAZ*(FOO-N)) correctly. You can manually check the mapped displacement. You may find it simpler (and safer) to insert additional labels or variables rather than manually calculating displacements.

Case 2: HIGH/LOW Applied to Symbolic Address Expressions

You should check the symbols operated on by the HIGH/LOW functions to ensure that their alignments in 8086 memory correspond to their 8080 page alignments.

In addition, if you converted using the REL control (a default), you should insert a group override prefix as follows:

Before Your Editing	**After Your Editing**
LOW(expr)	LOW DGROUP:(expr')
HIGH(expr)	HIGH DGROUP:(expr')

Case 3: Overly Complex Expressions

It is possible that an overly complex 8080 expression has resulted in unacceptable MCS-86 source code in your OUTPUT file. You should examine the original 8080 expression carefully to determine its intent, and then hand-translate the expression to a valid MCS-86 expression that corresponds to the original intent.

18 INSTRUCTION AS OPERAND CANNOT BE TRANSLATED

8080/8085 instructions are not permitted as operands in your source file.

19 REGISTER USED IN UNKNOWN CONTEXT

A register was used in an unknown context, such as:

```
REG EQU B
```

If this directive appears in an INCLUDE file which does not reference REG, conversion of the INCLUDE file will result in a type ambiguity for B. That is, CONV86 will not know at the time of the INCLUDE file's conversion whether B maps into CH or CX. You should check to see whether you want B to map into a byte register or a word register, and change the converter's mapping accordingly.

20 OUTPUT LINE TOO LONG; TRUNCATED

An output line has exceeded 129 characters and has been truncated. You should recode the line in 8086 accordingly.

21 LABEL ASSUMED TO BE NEAR

CONV86 has been unable to determine how this label is used; it is assumed to be of type NEAR. Since CONV86 has no information on how to type this symbol, you should check its usage and change its type accordingly.

22 NOMACROFILE CONTROL DELETED

No corresponding control exists for the 8086 assembler. No manual editing is required for this caution.

23 MOD85 CONTROL DELETED

No corresponding control exists for the 8086 assembler. No manual editing is required for this caution.

24 SOURCE LINE TOO LONG; IGNORED

The current source line exceeds 129 characters and has been mapped into a comment in both 8080/8085 and 8086 output files. You can either recode the source line and reconvert the source file using CONV86, or you can insert 8086 code in the OUTPUT file to accomplish the intent of the source line.

25 CURRENT SEGMENT UNKNOWN; CANNOT GENERATE ENDS

An END or SEG directive in 8086 implies a preceding ENDS directive to close the currently open segment. This segment is unknown. You should insert an ENDS directive of the appropriate type.

27 SYMBOL NAME TOO LONG

Symbol names in 8086 cannot exceed 31 characters.

28 CONDITIONAL ASSEMBLY GENERATED

CONV86 has assumed that it is possible that the operand of this PUSH or POP instruction is the PSW. Conditional assembler directives have been generated to take this possibility into account. If you know the operand is the PSW, you can substitute the appropriate mapping from Appendix A for:

- POP PSW (Using EXACT Control)
- POP PSW (Using APPROX Control)
- PUSH PSW (Using EXACT Control)
- PUSH PSW (Using APPROX Control)

On the other hand, if you know the operand is *definitely not* the PSW, you can substitute the appropriate mapping from Appendix A for:

- POP rw (Using either EXACT or APPROX)
- PUSH rw (Using either EXACT or APPROX)

If you cannot determine whether the operand is the PSW, you should desk-check or single-step your source program until you are able to make that determination. Otherwise, the conditional assembly statements placed by CONV86 in your OUTPUT file will not assemble under version V2.0 of the MCS-86 Macro Assembler.

30 UNKNOWN INSTRUCTION ASSUMED TO BE A MACRO

The converter is unable to recognize this statement and has assumed that it is a macro call. You should verify this assumption and recode if necessary.

31 GENERATED LABEL MIGHT NEED TO BE DECLARED LOCAL

The converter has generated a label within a macro definition. This label must be made local if the macro is invoked more than once.

32 (NO)COND CONTROL DELETED

No corresponding control exists for the 8086 assembler. No manual editing is required for this caution.

33 (NO)MACRODEBUG CONTROL DELETED

No corresponding control exists for the 8086 assembler. No manual editing is required for this caution.

34 METACHARACTER OR PARENTHESIS FOUND IN IRPC STRING

A '%,' '(' or ')' character was left in an IRPC string but will not be correctly interpreted by the 8086 assembler. This requires your attention.

35 EXPRESSION ASSUMED TO BE A CONSTANT

CONV86 has not been able to determine what type of expression is in this instruction. CONV86 has assumed that the expression is a numeric constant. If this assumption is incorrect, you should examine the resulting 8086 statement and recode the mapped expression to suit your intent. You may find it helpful to insert additional labels.

36 SYMBOLIC EXPRESSION MAY BE CONTEXTUALLY INVALID FOR ASM86

A symbolic expression has been encountered in a context in which the 8086 assembler allows expressions containing only two type of operands:

a. Numeric constants, and

b. Macro symbols (preceded or followed by a '%') that evaluate to numeric constants.

If the expression contains symbols which do not conform to b, above, they must be replaced by their numeric values or redefined via the % SET macro.

APPENDIX A
INSTRUCTION MAPPING

Following are instruction mappings from 8080/8085 to 8086 assembly language. Operands are mapped according to Appendix B. Operand designations are as follows:

ib = byte immediate	mn = near memory
iw = word immediate	rb = byte register
mb = byte memory	rw = word register
mw = word memory	

Similarly, ib' refers to the mapping of ib, iw' refers to the mapping of iw, and so on. Thus, if B = rb, then rb' = CH. But if B = rw, then rw' = CX.

Constructs of the form L_n are generated internally by CONV86 for use as labels in mappings of conditional CALLs, conditional RETurns, conditional JMPs.

8080/8085	8086	Remarks
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,ib'	
CALL mn	CALL mn'	
CC mn	JNB SHORT L_n CALL mn'	(L_n inserted as label for instruction following CALL)
CM mn	JNS SHORT L_n CALL mn'	(L_n inserted as label for instruction following CALL)
CMA	NOT AL	
CMC	CMC	
CMP rb	CMP AL,rb'	
CNC mn	JNAE SHORT L_n CALL mn'	(L_n inserted as label for instruction following CALL)
CNZ mn	JZ SHORT L_n CALL mn'	(L_n inserted as label for instruction following CALL)
CP mn	JS SHORT L_n CALL mn'	(L_n inserted as label for instruction following CALL)
CPE mn	JNP SHORT L_n CALL mn'	(L_n inserted as label for instruction following CALL)
CPI ib	CMP AL,ib'	
CPO mn	JP SHORT L_n CALL mn'	(L_n inserted as label for instruction following CALL)
CZ mn	JNZ SHORT L_n CALL mn'	(L_n inserted as label for instruction following CALL)

8080/8085	8086	Remarks
DAA	DAA	
DAD rw	ADD BX,rw'	(Using APPROX Control)
DAD rw	LAHF ADD BX,rw' RCR SI,1 SAHF RCL SI,1	(Using EXACT Control)
DCR rb	DEC rb'	
DCX rw	DEC rw'	(Using APPROX Control)
DCX rw	LAHF DEC rw' SAHF	(Using EXACT Control)
DI	CLI	
EI	STI	
HLT	HLT	
IN ib	IN AL, ib'	
INR rb	INC rb'	
INX rw	INC rw'	(Using APPROX Control)
INX rw	LAHF INC rw' SAHF	(Using EXACT Control)

8080/8085	8086	Remarks
JC mn	JB SHORT mn'	(For forward short branch)
JC mn	JB mn'	(For backward short branch)
JC mn	JAE SHORT L_n JMP mn'	(Otherwise)
JM mn	JS SHORT mn'	(For forward short branch)
JM mn	JS mn'	(For backward short branch)
JM mn	JNS SHORT L_n JMP mn'	(Otherwise)
JMP mn	JMP SHORT mn'	(For forward short branch)
JMP mn	JMP mn'	(Otherwise)
JNC mn	JAE SHORT mn'	(For forward short branch)
JNC mn	JAE mn'	(For backward short branch)
JNC mn	JNAE SHORT L_n JMP mn'	(Otherwise)
JNZ mn	JNZ SHORT mn'	(For forward short branch)
JNZ mn	JNZ mn'	(For backward short branch)
JNZ mn	JZ SHORT L_n JMP mn'	(Otherwise)
JP mn	JNS SHORT mn'	(For forward short branch)
JP mn	JNS mn'	(For backward short branch)
JP mn	JS SHORT L_n JMP mn'	(Otherwise)
JPE mn	JP SHORT mn'	(For forward short branch)
JPE mn	JP mn'	(For backward short branch)
JPE mn	JNP SHORT L_n JMP mn'	(Otherwise)
JPO mn	JNP SHORT mn'	(For forward short branch)
JPO mn	JNP mn'	(For backward short branch)
JPO mn	JP SHORT L_n JMP mn'	(Otherwise)
JZ mn	JZ SHORT mn'	(For forward short branch)
JZ mn	JZ mn'	(For backward short branch)
JZ mn	JNZ SHORT L_n JMP mn'	(Otherwise)

8080/8085	8086	Remarks
LDA mb	MOV AL,mb'	
LDAX rw	MOV SI,rw' LODS DS:M[SI]	
LHLD mw	MOV BX,mw'	
LXI rw,iw	MOV rw',iw'	(When 2nd operand immed. or near)
LXI rw,iw	LEA rw',iw'	(When 2nd operand is byte or word)
MOV rb1,rb2	MOV rb1',rb2'	
MOV M, rb	MOV M[BX], rb'	
MVI rb,ib	MOV rb',ib'	
MVI M, ib	MOV M[BX], ib'	
NOP	NOP	XCHG AX,AX (1 byte, 3 clocks)
ORA rb	OR AL,rb'	
ORI ib	OR AL,ib'	
OUT ib	OUT ib', AL	
PCHL	JMP BX	
POP rw	POP rw'	(For EXACT or APPROX when rw is definitely not PSW)
POP PSW	POP AX XCHG AL, AH	(Using APPROX Control)
POP PSW	POP AX XCHG AL, AH SAHF	(Using EXACT Control)
POP rw	% IF (%EQS (rw',AX)) THEN(POP rw' XCHG AL, AH)ELSE(POP rw')FI	(Using APPROX when rw could be PSW)
POP rw	%IF (%EQS (rw',AX)) THEN(POP rw' XCHG AL, AH SAHF)ELSE(POP rw')FI	(Using EXACT Control when rw could be PSW)

8080/8085	8086	Remarks
PUSH rw	PUSH rw'	(For EXACT or APPROX when rw is definitely not PSW)
PUSH PSW	LAHF XCHG AL, AH PUSH AX XCHG AL, AH	(Using EXACT Control)
PUSH PSW	XCHG AL, AH PUSH AX XCHG AL, AH	(Using APPROX Control)
PUSH rw	%IF (%EQS (rw',AX)) THEN(XCHG AL, AH PUSH rw' XCHG AL, AH)ELSE(PUSH rw')FI	(Using APPROX Control when rw could be PSW)
PUSH rw	%IF (%EQS (rw',AX)) THEN(LAHF XCHG AL, AH PUSH rw' XCHG AL, AH)ELSE(PUSH rw')FI	(Using EXACT Control when rw could be PSW)
RAL	RCL AL,1	
RAR	RCR AL,1	
RC	JNB SHORT L_n RET	(L_n inserted as label for instruction following RET)
RET	RET	
RIM	***error***	
RLC	ROL AL,1	
RM	JNS SHORT L_n RET	(L_n inserted as label for instruction following RET)
RNC	JNAE SHORT L_n RET	(L_n inserted as label for instruction following RET)
RNZ	JZ SHORT L_n RET	(L_n inserted as label for instruction following RET)
RP	JS SHORT L_n RET	(L_n inserted as label for instruction following RET)
RPE	JNP SHORT L_n RET	(L_n inserted as label for instruction following RET)
RPO	JP SHORT L_n RET	(L_n inserted as label for instruction following RET)
RRC	ROR AL,1	
RST ib	INT ib'	
RZ	JNZ SHORT L_n RET	(L_n inserted as label for instruction following RET)

8080/8085	8086	Remarks
SBB rb	SBB AL,rb'	
SBI ib	SBB AL,ib'	
SHLD mw	MOV mw',BX	
SIM	***error***	
SPHL	MOV SP,BX	
STA mb	MOV mb',AL	
STAX rw	MOV DI,rw' MOV DS:[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	POP SI XCHG BX,SI PUSH SI	
unknown expr	unknown' expr'	

APPENDIX B
CONVERSION OF EXPRESSIONS IN CONTEXT

The following describes how 8080/8085 expressions are converted to 8086 expressions according to the context in which an operand or expression occurs. The context is simply what CONV86 infers from the use of the operand in the instruction:

ib = byte immediate
iw = word immediate
mb = byte memory
mw = word memory
mn = near memory
rb = byte register
rw = word register

M is defined to be a byte located at absolute location 0. In contexts 3 and 5 below, forward-referenced memory items are treated as "unknown."

1. Context = ib
   - Operand = ib: expr → expr'
   - Operand = iw: expr → LOW(expr')
   - Operand = mn, mw, mb, or unknown: [1] [2]
     If REL control, then
       expr → LOW DGROUP:(expr')
     If ABS control, then
       expr → LOW(expr')
2. Context = iw
   - Operand = ib or iw: expr → expr'
   - Operand = mb, mw, mn, or unknown[2]:
     If REL control, then
       expr → OFFSET DGROUP:(expr')
     If ABS control, then
       expr → OFFSET(expr')
3. Context = mb
   - Operand = mb: expr → expr'
   - Operand = mn or mw or unknown: expr → BYTE PTR(expr')
   - Operand = ib or iw: expr → M[expr']
4. Context = mn
   - Operand = mn: expr → expr'
   - Operand = mb or mw or unknown: expr → NEAR PTR(expr')
   - Operand = ib or iw: expr → NEAR PTR M[expr']
5. Context = mw
   - Operand = mw: expr → expr'
   - Operand = mb or mn or unknown: expr → WORD PTR(expr')
   - Operand = ib or iw: expr → WORD PTR M[expr']

[1] mn, mw, and mb are illegal in 8080 in this context, but give an implicit LOW.

[2] unknown generates Caution Message 17.

6. Context = rb
   - Operand = rb:
   - A → AL
   - B → CH
   - C → CL
   - D → DH
   - E → DL
   - H → BH
   - L → BL
   - Operand = mb:M → M[BX]
7. Context = rw
   - Operand = rw:
   - B → CX
   - D → DX
   - H → BX
   - SP → SP
   - PSW → AX

APPENDIX C
ASSEMBLER DIRECTIVES MAPPING

This appendix shows how 8080/8085 assembler directives are converted by CONV86 into 8086 assembler directives. Expression mapping is described in Appendix B. Context symbols (for instance, "expr", "mn", and so on) used as directive operands are mapped according to Appendix B.

In certain cases (EQU, IRP, macro call, and SET), it is possible to determine that an assignment is being made to a byte or word register. In such cases, the appropriate rb or rw expression conversion is performed. The STKLN expression is converted in the prologue (see Chapter 1, "Functional Mapping").

Table C-1. Assembler Directives Mapping

8080/8085	8086	NOTES
ASEG	prev-seg ENDS ABS_0 SEGMENT BYTE AT 0	
CSEG	prev-seg ENDS CODE SEGMENT WORD PUBLIC 'CODE'	
DB expr-list	DB expr-list'	
DS expr	DB expr' DUP (?)	
DSEG	prev-seg ENDS DATA SEGMENT WORD PUBLIC 'DATA'	
DW expr-list	DW expr-list'	
END [mn]	prev-seg ENDS END [mn']	
name EQU expr	name' EQU expr'	
EXTRN name-list	EXTRN name:usage-list'	
NAME name	NAME name'	
ORG mn	ORG mn'	
PUBLIC name-list	PUBLIC name-list'	
STKLN expr	***deleted***	If the REL control (a default) is used, STKLN converts to information in the prologue. Refer to Chapter 1, "Functional Mapping."
a SET b	% SET (a',b')	If the symbol being defined is never set to a non-constant.
	PURGE a' a' EQU b'	If the symbol being defined is ever set to a non-constant and the SET is not self-relative.
	T_a' EQU b' PURGE a' a' EQU T_a' PURGE T_a'	If the symbol being defined is ever set to a non-constant and the set is self-relative, e.g., X SET X+5.
IF a	%IF (a') THEN (	
ELSE	) ELSE (	
ENDIF	) FI	

Table C-1. Assembler Directives Mapping (Cont'd.)

8080/8085	8086	NOTES
a MACRO b,...	%*DEFINE (a'(b',...)) LOCAL c' ... (	All local labels for the macro (c'...) are moved to the local list in the macro definition, with blanks replacing commas. LOCAL statements disappear. The word LOCAL is not produced if there are no local labels. The parentheses around b',... are omitted when the parameter list is null.
LOCAL c, ...	none	
ENDM	)	If this directive closes a macro.
	))	If this directive closes a REPT, IRP or IRPC definition.
mcall b, ...	%mcall (b', ...)	The parentheses are omitted when the parameter list is null.
IRP a,b	%IRP(a',b')c'...(%(	All local labels for the macro (c'...) are moved to the local list in the macro definition, with blanks replacing commas. LOCAL statements disappear.
IRPC a,b	%IRPC(a',b')c'...(%(	All local labels for the macro (c'...) are moved to the local list in the macro definition, with blanks replacing commas. LOCAL statements disappear.
REPT a	%REPT(a')c'...(%(	All local labels for the macro (c'...) are moved to the local list in the macro definition, with blanks replacing commas. LOCAL statements disappear.
EXITM	%EXIT	

APPENDIX D
RESERVED NAMES

A name appearing in an 8080/8085 expression may have a special 8086 interpretation (for instance, AL or TEST), or it may be reserved for a segment or group name (for instance, CODE). Except for STACK, which is converted to STACK_BASE, each such name is automatically converted by CONV86 by appending an underscore to it (for instance, AL_ or TEST_). The 8080 reserved word MEMORY is treated specially.

The following ASM86 reserved names are modified by CONV86:

Table D-1. Reserved Names

AAA
AAD
AAM
AAS
ABS
AH
AL
ASSUME
AT
AX
BH
BL
BP
BX
BYTE
CBW
CH
CL
CLC
CLD
CLI
CMPS
CODEMARCO
COMMON
CS
CWD
CX
DAS
DD
DEC
DEFINE
DH
DIV
DL
DUP
DWORD
DX
ELSE
ELSEIF
ENDIF
ENDM
ENDP
ENDS
EQS
ES
ESC
EVAL
EXIT
FAR
GES
GROUP
GTS
IDIV
IMUL
INC
INCHAR
INT
INTO
IRET
JA
JAE
JB
JBE
JCXZ
JE
JG
JGE
JL
JLE
JNA
JNAE
JNR
JNBE
JNE
JNG
JNGE
JNL
JNLE
JNO
JNP
JNS
JO
JS
LABEL
LAHF
LDS
LEA
LEN
LENGTH
LES
LOCK
LODS
LOOP
LOOPE
LOOPNE
LOOPNZ
LOOPZ
LTS
MASK
MATCH
METACHAR
MODRM
MOVS
MUL
NEAR
NEG
NES
NIL
NOSEGFIX
NOTHING
OFFSET
PARA
POPF
PREFX
PROC
PROCLEN
PIR
PURGE
PUSHF
RCL
RCR
RECORD
RELB
RELW
REP
REPE
REPEAT
REPNE
REPNZ
REPZ
ROL
SAHF
SAL
SAR
SCAS
SEG
SEGFIX
SEGMENT
SHORT
SI
SIZE
SS
STD
STI
STOS
STRUC
SUBSTR
TEST
THIS
TYPE
WAIT
WHILE
WIDTH
WORD
XLAT

The names CGROUP, CODE, CONST, DATA, and DGROUP are reserved by CONV86 to set up a PL/M-86 environment.

The assembler-reserved symbols ? and ??SEG are not permitted as user mnemonics.

APPENDIX E
MACRO CONSTRUCT CONVERSION

All macro definitions and calls will be translated to their 8086 macro processing language equivalents. However, macro related constructs require special conversion.

The following 8080/8085 macro constructs are converted to their 8086 equivalent as shown:

Table E-1. Macro Construct Conversion

8080 CONSTRUCT	8086 EQUIVALENT	NOTES
;;	%'	Within a macro definition body.
!	%1	When quoted or within a list or IRPC string.
NUL *operand*	%EQS(*operand'*,%0)	Within any expression.
<list>	%(*list'*)	Within any macro argument field, but '< >' is stripped when surrounding an IRPC string.
(	%1(	Within < > or ' ' in macro call parameter, macro definition, IF expression or body, or SET body.
)	%1)	Within < > or ' ' in macro call parameter, macro definition, IF expression or body, or SET body.
%expression	*expression'*	Within macro argument field.
symbol	%(*symbol'*)	When symbol is a macro parameter and is being passed to another macro in an argument field that does not use %.
symbol	%*symbol'*	When symbol is a parameter or local symbol in a macro definition, a macro itself, or defined with a SET directive.
%	%1%	Within quotes when not causing concatenation.
&	%	Concatenation translation.

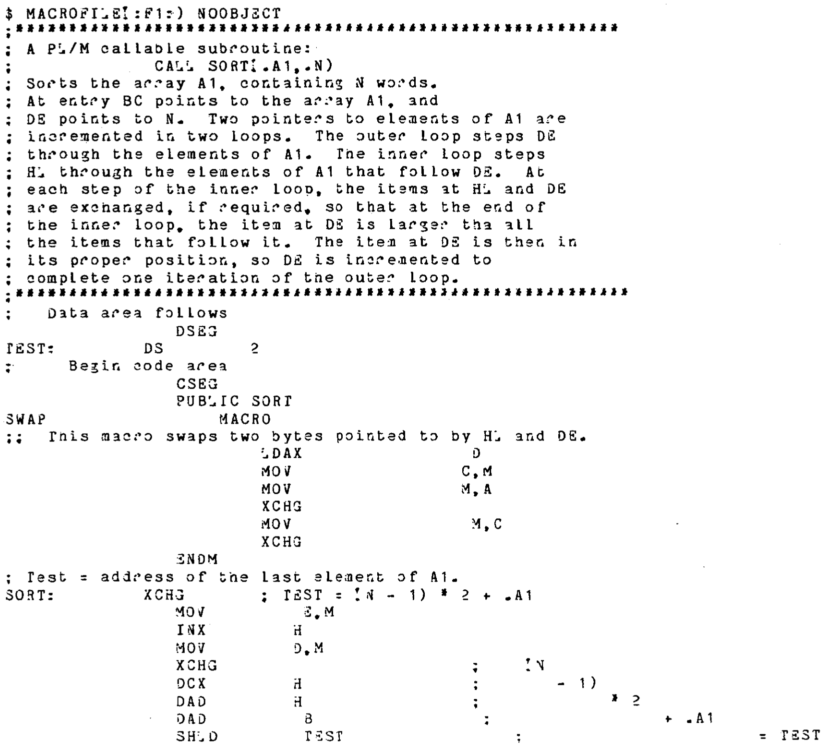
APPENDIX F
SAMPLE CONVERSION AND LISTINGS

This appendix consists of:

- Figure F-1. 8080 Sort Routine Source File
- Figure F-2. CONV86 PRINT File of Conversion of 8080 Sort Routine
- Figure F-3. MCS-86 Macro Assembler Listing of Conversion of 8080 Sort Routine
- Figure F-4. MCS-86 Macro Assembler Listing of Originally Coded 8086 Sort Routine

Please note that the CONV86 OUTPUT file was edited before submitting it to ASM86 for assembly. The OUTPUT file was edited as follows:

1. To retrieve PL/M-86 stack parameters, code (corresponding to lines 44-47 in Figure F-3) was inserted as described in Chapter 3.
2. To correct incomplete register mapping due to mnemonics appearing in an IRPC string, IRPC calls have been deleted at lines 69 and 85 in Figure F-2, and the code has been expanded by hand to that at lines 91-94 and 132-133 in Figure F-3. This edit is in response to the converter generated caution.
3. For space/time considerations, only the necessary LAHF/SAHF instructions were retained from the OUTPUT file. Since the file was converted using the (default) control EXACT, flag-preserving code for all occurrences of DAD, DCX, INX, and PUSH/POP PSW was generated. You can determine which flag-preserving code has been retained by comparing Figures F-2 and F-3

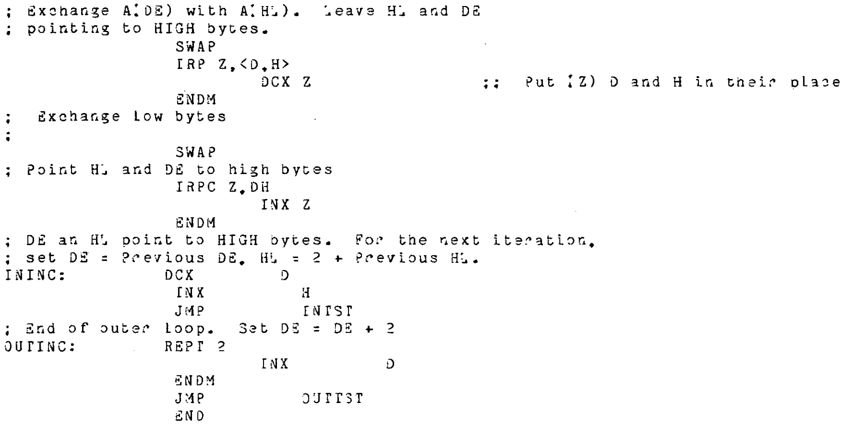


```
$ MACROFILE(:F1:) NOOBJECT
;*****************************************************************
; A PL/M callable subroutine:
;             CALL SORT(.A1,.N)
; Sorts the array A1, containing N words.
; At entry BC points to the array A1, and
; DE points to N.  Two pointers to elements of A1 are
; incremented in two loops.  The outer loop steps DE
; through the elements of A1.  The inner loop steps
; HL through the elements of A1 that follow DE.  At
; each step of the inner loop, the items at HL and DE
; are exchanged, if required, so that at the end of
; the inner loop, the item at DE is larger tha all
; the items that follow it.  The item at DE is then in
; its proper position, so DE is incremented to
; complete one iteration of the outer loop.
;*****************************************************************
;   Data area follows
                DSEG
TEST:        DS         2
;     Begin code area
                CSEG
                PUBLIC SORT
SWAP                MACRO
;;  This macro swaps two bytes pointed to by HL and DE.
                        LDAX                D
                        MOV                C,M
                        MOV                M,A
                        XCHG
                        MOV                 M,C
                        XCHG
                ENDM
; Test = address of the last element of A1.
SORT:        XCHG       ; TEST = (N - 1) * 2 + .A1
                MOV         E,M
                INX        H
                MOV        D,M
                XCHG                        ;    (N
                DCX        H                ;       - 1)
                DAD        H                ;            * 2
                DAD         B                ;                + .A1
                SHLD        TEST               ;                       = TEST
```

Figure F-1A. 8080 Sort Routine Source File

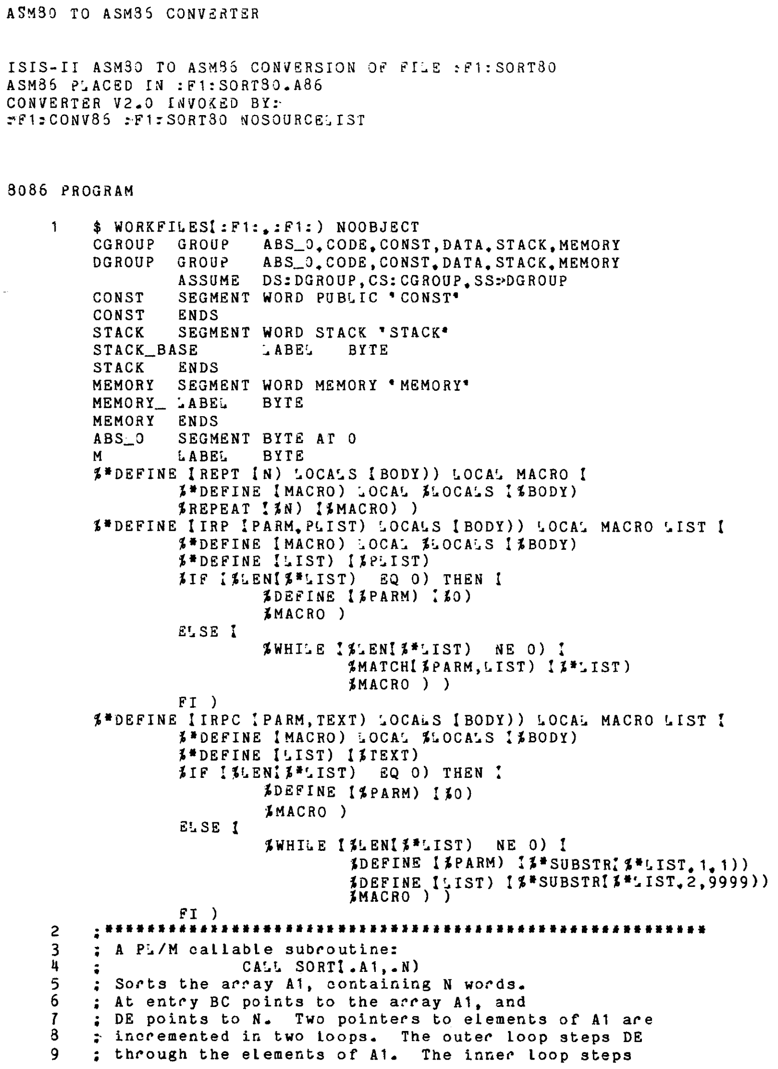
```
;    OUTER LOOP: DO DE = .A1 TO TEST BY 2;
                MOV        E,C                  ; BC CONTAINS .A1
                MOV         D,B
OUTTST:       LDA        TEST                   ; IF DE > TEST THEN RETURN
                SUB         E
                LDA         TEST + 1
                SBB         D
                RC
; INNER LOOP:  DO HL = DE+2 TO TEST BY 2
                MOV          L,E
                MOV         H,D
                REPT         2
                        INX         H
                ENDM
                                                ;HL = DE + 2
; IF HL > TEST THEN GOTO OUTINC
INTST:        LDA        TEST
                SUB          L
                LDA         TEST + 1
                SBB  H
                JC OUTINC
; IF A1(HL) < A1(DE) THEN GOTO ININC
; As a side effect, HL and DE are incremented by 1
; to point to the high bytes of their array elements.
                LDAX        D
                SUB         M
                IRPC Z,DH
                        INX         Z
                ENDM
                LDAX        D
                SBB         M
                JNC        ININC
```

Figure F-1B. 8080 Sort Routine Source File



```
; Exchange A(DE) with A(HL).  Leave HL and DE
; pointing to HIGH bytes.
                SWAP
                IRP Z,<D,H>
                        DCX Z                 ;;  Put (Z) D and H in their place
                ENDM
;  Exchange low bytes
;
                SWAP
; Point HL and DE to high bytes
                IRPC Z,DH
                        INX Z
                ENDM
; DE an HL point to HIGH bytes.  For the next iteration,
; set DE = Previous DE, HL = 2 + Previous HL.
ININC:         DCX        D
                INX        H
                JMP        INTST
; End of outer loop.  Set DE = DE + 2
OUTINC:        REPT 2
                        INX         D
                ENDM
                JMP        OUTTST
                END
```

Figure F-1C. 8080 Sort Routine Source File



```
ASM80 TO ASM86 CONVERTER


ISIS-II ASM80 TO ASM86 CONVERSION OF FILE :F1:SORT80
ASM86 PLACED IN :F1:SORT80.A86
CONVERTER V2.0 INVOKED BY:-
:F1:CONV86 :F1:SORT80 NOSOURCELIST


8086 PROGRAM

    1    $ WORKFILES(:F1:,:F1:) NOOBJECT
         CGROUP  GROUP    ABS_0,CODE,CONST,DATA,STACK,MEMORY
         DGROUP  GROUP    ABS_0,CODE,CONST,DATA,STACK,MEMORY
                 ASSUME   DS:DGROUP,CS:CGROUP,SS:DGROUP
         CONST   SEGMENT WORD PUBLIC 'CONST'
         CONST   ENDS
         STACK   SEGMENT WORD STACK 'STACK'
         STACK_BASE      LABEL    BYTE
         STACK   ENDS
         MEMORY  SEGMENT WORD MEMORY 'MEMORY'
         MEMORY_ LABEL    BYTE
         MEMORY  ENDS
         ABS_0   SEGMENT BYTE AT 0
         M       LABEL    BYTE
         %*DEFINE (REPT (N) LOCALS (BODY)) LOCAL MACRO (
                 %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
                 %REPEAT (%N) (%MACRO) )
         %*DEFINE (IRP (PARM,PLIST) LOCALS (BODY)) LOCAL MACRO LIST (
                 %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
                 %*DEFINE (LIST) (%PLIST)
                 %IF (%LEN(%*LIST)  EQ 0) THEN (
                         %DEFINE (%PARM) (%0)
                         %MACRO )
                 ELSE (
                         %WHILE (%LEN(%*LIST)  NE 0) (
                                 %MATCH(%PARM,LIST) (%*LIST)
                                 %MACRO ) )
                 FI )
         %*DEFINE (IRPC (PARM,TEXT) LOCALS (BODY)) LOCAL MACRO LIST (
                 %*DEFINE (MACRO) LOCAL %LOCALS (%BODY)
                 %*DEFINE (LIST) (%TEXT)
                 %IF (%LEN(%*LIST)  EQ 0) THEN (
                         %DEFINE (%PARM) (%0)
                         %MACRO )
                 ELSE (
                         %WHILE (%LEN(%*LIST)  NE 0) (
                                 %DEFINE (%PARM) (%*SUBSTR(%*LIST,1,1))
                                 %DEFINE (LIST) (%*SUBSTR(%*LIST,2,9999))
                                 %MACRO ) )
                 FI )
    2    ;*********************************************************
    3    ; A PL/M callable subroutine:
    4    ;             CALL SORT(.A1,.N)
    5    ; Sorts the array A1, containing N words.
    6    ; At entry BC points to the array A1, and
    7    ; DE points to N.  Two pointers to elements of A1 are
    8    ; incremented in two loops.  The outer loop steps DE
    9    ; through the elements of A1.  The inner loop steps
```

Figure F-2A. CONV86 PRINT File Conversion of 8080 Sort Routine

```
ASM80 TO ASM86 CONVERTER

10   ; HL through the elements of A1 that follow DE.  At
11   ; each step of the inner loop, the items at HL and DE
12   ; are exchanged, if required, so that at the end of
13   ; the inner loop, the item at DE is larger tha all
14   ; the items that follow it.  The item at DE is then in
15   ; its proper position, so DE is incremented to
16   ; complete one iteration of the outer loop.
17   ;*************************************************************
18   ;   Data area follows
19   ABS_0   ENDS
19   DATA    SEGMENT WORD PUBLIC 'DATA'
20   TEST_   DB      2 DUP [?)
21   ;       Begin code area
22   DATA    ENDS
22   CODE    SEGMENT WORD PUBLIC 'CODE'
23           PUBLIC  SORT
24   %*DEFINE [SWAP) [
25   %*  This macro swaps two bytes pointed to by HL and DE.
26           MOV     SI,DX
26           LODS    DS:M[SI]
27           MOV     CL,M[BX]
28           MOV     M[BX],AL
29           XCHG    BX,DX
30           MOV     M[BX],CL
31           XCHG    BX,DX
32           )
33   ; Test = address of the last element of A1.
34   SORT:   XCHG    BX,DX                   ; TEST = [N - 1) * 2 + .A1
35           MOV     DL,M[BX]
36           LAHF
36           INC     BX
36           SAHF
37           MOV     DH,M[BX]
38           XCHG    BX,DX                   ;    [N
39           LAHF
39           DEC     BX
39           SAHF                            ;       - 1)
40           LAHF
40           ADD     BX,BX
40           RCR     SI,1
40           SAHF
40           RCL     SI,1                    ;            * 2
41           LAHF
41           ADD     BX,CX
41           RCR     SI,1
41           SAHF
41           RCL     SI,1                    ;                + .A1
42           MOV     WORD PTR[TEST_),BX      ;                      = TEST
43   *   OUTER LOOP: DO DE = .A1 TO TEST BY 2;
44           MOV     DL,CL                   ; BC CONTAINS .A1
45           MOV     DH,CH
46   OUTTST: MOV     AL,TEST_                ; IF DE > TEST THEN RETURN
47           SUB     AL,DL
48           MOV     AL,TEST_+1
49           SBB     AL,DH
50           JNB     SHORT L_1
50           RET
```

Figure F-2B. CONV86 PRINT File Conversion of 8080 Sort Routine

```
ASM80 TO ASM86 CONVERTER


   50    L_1:
   51    ; INNER LOOP>  DO HL = DE+2 TO TEST BY 2
   52            MOV      BL,DL
   53            MOV      BH,DH
   54            %REPT    [2)  [%[
   55            LAHF
   55            INC      BX
   55            SAHF
   56            ))
   57                                             ;HL = DE + 2
   58    ; IF HL > TEST THEN GOTO OUTINC
   59    INTST:  MOV      AL,TEST_
   60            SUB      AL,BL
   61            MOV      AL,TEST_+1
   62            SBB      AL,BH
   63            JB       SHORT OUTINC
   64    ; IF A1[HL) < A1[DE) THEN GOTO ININC
   65    ; As a side effect, HL and DE are incremented by 1
   66    ; to point to the nigh bytes of their array elements.
   67            MOV      SI,DX
   67            LODS     DS:M[SI]
   68            SUB      AL,M[BX]
   69            %IRPC    [Z,DH)  [%[
*** CAUTION 002 ***  8080 REGISTER MNEMONIC APPEARING IN IRPC STRING
   70            LAHF
   70            INC      %Z
   70            SAHF
   71            ))
   72            MOV      SI,DX
   72            LODS     DS:M[SI]
   73            SBB      AL,M[BX]
   74            JAE      SHORT ININC
   75    ; Exchange A[DE) with A[HL).  Leave HL and DE
   76    ; pointing to HIGH bytes.
   77            %SWAP
   78            %IRP     [Z,%[DX,BX))  [%[
   79            LAHF
   79            DEC      %Z
   79            SAHF                              %*  Put [Z) D and H in their place
   80            ))
   81    ;  Exchange low bytes
   82    ;
   83            %SWAP
   84    ; Point HL and DE to high bytes
   85            %IRPC    [Z,DH)  [%[
*** CAUTION 002 ***  8080 REGISTER MNEMONIC APPEARING IN IRPC STRING
   86            LAHF
   86            INC      %Z
   86            SAHF
   87            ))
   88    ; DE an HL point to HIGH bytes.  For the next iteration,
   89    ; set DE = Previous DE, HL = 2 + Previous HL.
   90    ININC:  LAHF
   90            DEC      DX
   90            SAHF
   91            LAHF
   91            INC      BX
```

Figure F-2C. CONV86 PRINT File Conversion of 8080 Sort Routine

```
ASM80 TO ASM86 CONVERTER


  91          SAHF
  92          JMP     INTST
  93   ; End of outer loop.  Set DE = DE + 2
  94   OUTINC: %REPT   (2)  (%(
  95          LAHF
  95          INC     DX
  95          SAHF
  96          ))
  97          JMP     OUTTST
  98   CODE   ENDS
  98          END

2 CAUTION(S)

END OF ASM80 TO ASM86 CONVERSION
```

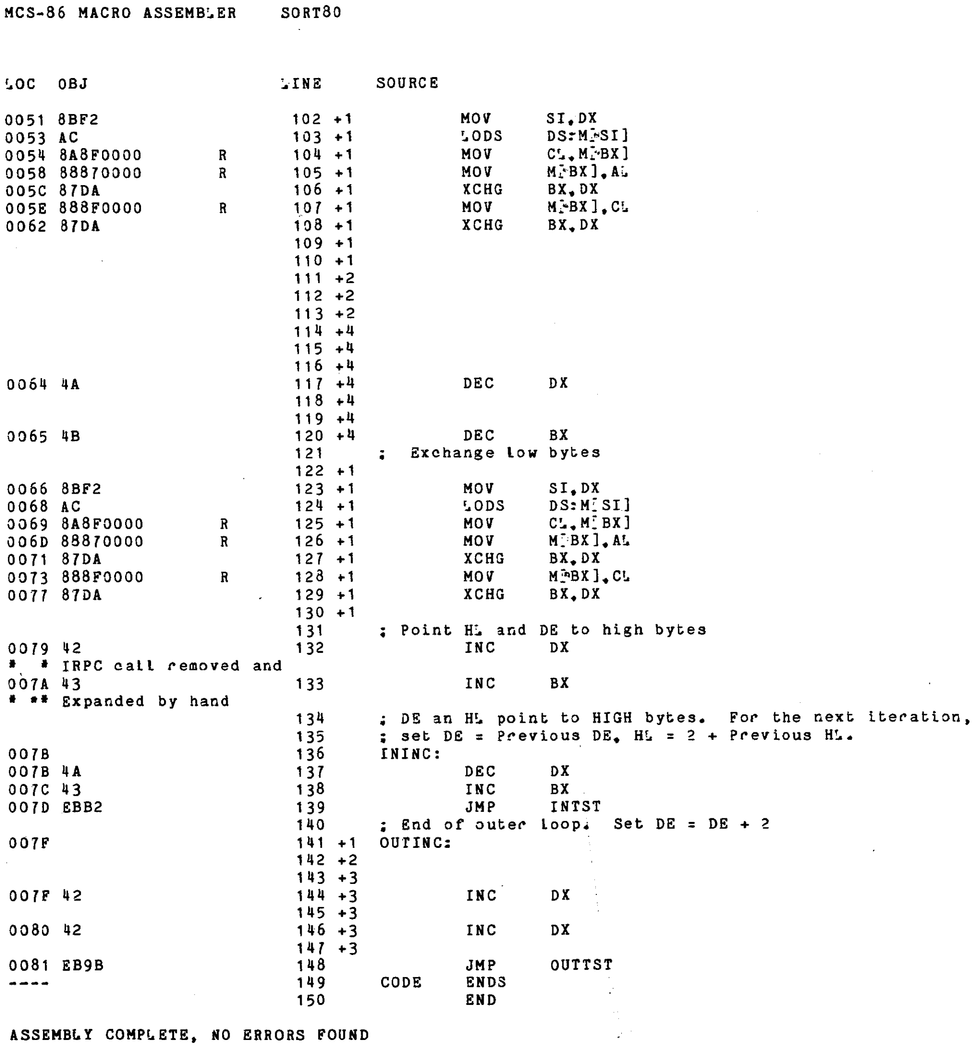
Figure F-2D. CONV86 PRINT File Conversion of 8080 Sort Routine

```
MCS-86 MACRO ASSEMBLER    SORT80


ISIS-II MCS-86 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE SORT80
NO OBJECT MODULE REQUESTED
ASSEMBLER INVOKED BY: :F3:ASM86 :F1:SORT80.A86

LOC  OBJ                    LINE     SOURCE

                              1      $ WORKFILES(:F1:,:F1:) NOOBJECT
                              2      CGROUP  GROUP   ABS_0,CODE,CONST,DATA,STACK,MEMORY
                              3      DGROUP  GROUP   ABS_0,CODE,CONST,DATA,STACK,MEMORY
                              4              ASSUME  DS:DGROUP,CS:CGROUP,SS:DGROUP
----                          5      CONST   SEGMENT WORD PUBLIC 'CONST'
----                          6      CONST   ENDS
----                          7      STACK   SEGMENT WORD STACK 'STACK'
0000                          8      STACK_BASE      LABEL   BYTE
----                          9      STACK   ENDS
----                         10      MEMORY  SEGMENT WORD MEMORY 'MEMORY'
0000                         11      MEMORY_ LABEL   BYTE
----                         12      MEMORY  ENDS
----                         13      ABS_0   SEGMENT BYTE AT 0
0000                         14      M       LABEL   BYTE
                             15
                             16 +1
                             17 +1
                             18      ;*********************************************************
                             19      ; A PL/M callable subroutine:
                             20      ;               CALL SORT(.A1,.N)
                             21      ; Sorts the array A1, containing N words.
                             22      ; At entry BC points to the array A1, and
                             23      ; DE points to N.  Two pointers to elements of A1 are
                             24      ; incremented in two loops.  The outer loop steps DE
                             25      ; through the elements of A1.  The inner loop steps
                             26      ; HL through the elements of A1 that follow DE.  At
                             27      ; each step of the inner loop, the items at HL and DE
                             28      ; are exchanged, if required, so that at the end of
                             29      ; the inner loop, the item at DE is larger tha all
                             30      ; the items that follow it.  The item at DE is then in
                             31      ; its proper position, so DE is incremented to
                             32      ; complete one iteration of the outer loop.
                             33      ;**********************************************************
                             34      ;   Data area follows
----                         35      ABS_0   ENDS
----                         36      DATA    SEGMENT WORD PUBLIC 'DATA'
0000 (2                      37      TEST_   DB      2 DUP (?)
     ??
     )
                             38      ;     Begin code area
----                         39      DATA    ENDS
----                         40      CODE    SEGMENT WORD PUBLIC 'CODE'
                             41              PUBLIC  SORT
                             42
                             43      ; Test = address of the last element of A1.
0000                         44      SORT:
0000 5B                      45              POP     BX                        ; **** CODE INSERTED TO
0001 59                      46              POP     CX                        ; **** RETRIEVE PL/M-86
0002 5A                      47              POP     DX                        ; **** STACK PARAMETERS
```

Figure F-3A. MCS-86™ Macro Assembler Listing of Conversion of 8080 Sort Routine

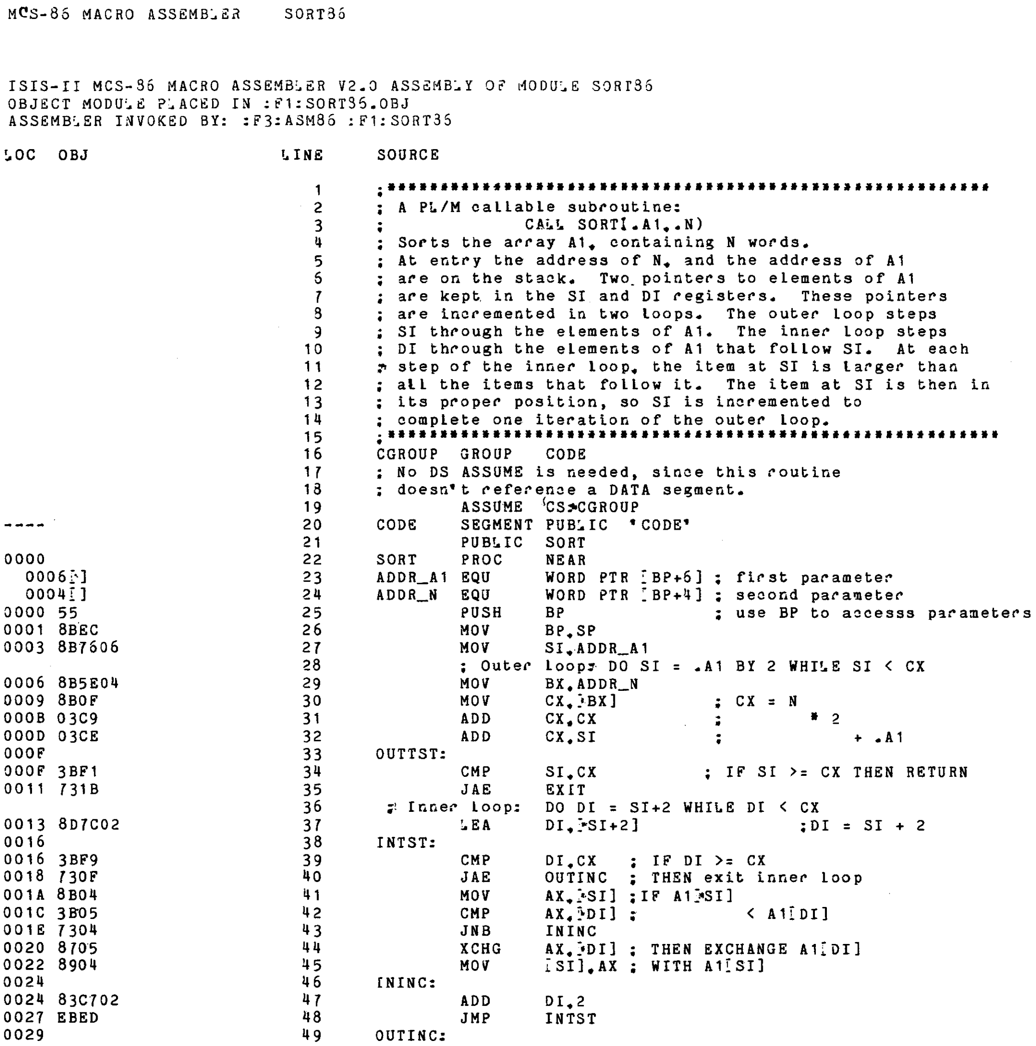
```
M S-86 MACRO ASSEMBLER    SORT80


LOC  OBJ                  LINE     SOURCE

0003 53                    48             PUSH     BX                       ; **** (CHAPTER 3)
0004 87DA                  49             XCHG     BX,DX                  ; TEST = [N - 1) * 2 + .A1
0006 8A970000         R    50             MOV      DL,M[BX]
000A 43                    51             INC      BX
000B 8AB70000         R    52             MOV      DH,M[BX]
000F 87DA                  53             XCHG     BX,DX                  ;    [N
0011 4B                    54             DEC      BX                         ;       - 1)
0012 03DB                  55             ADD      BX,BX                  ;             * 2
0014 03D9                  56             ADD      BX,CX                  ;                 + .A1
0016 891E0000         R    57             MOV      WORD PTR[TEST_),BX     ;                        = TEST
                           58         OUTER LOOP: DO DE = .A1 TO TEST BY 2;
001A 8AD1                  59             MOV      DL,CL                  ; BC CONTAINS .A1
001C 8AF5                  60             MOV      DH,CH
001E A00000           R    61     OUTTST: MOV      AL,TEST_               ; IF DE > TEST THEN RETURN
0021 2AC2                  62             SUB      AL,DL
0023 A00100           R    63             MOV      AL,TEST_+1
0026 1AC6                  64             SBB      AL,DH
0028 7301                  65             JNB      SHORT L_1
002A C3                    66             RET
002B                       67     L_1:
                           68     ; INNER LOOP>  DO HL = DE+2 TO TEST BY 2
002B 8ADA                  69             MOV      BL,DL
002D 8AFE                  70             MOV      BH,DH
                           71 +1
                           72 +2
                           73 +3
002F 43                    74 +3          INC      BX
                           75 +3
0030 43                    76 +3          INC      BX
                           77 +3
                           78                                             ;HL = DE + 2
                           79     ; IF HL > TEST THEN GOTO OUTINC
0031 A00000           R    80     INTST>  MOV      AL,TEST_
0034 2AC3                  81             SUB      AL,BL
0036 A00100           R    82             MOV      AL,TEST_+1
0039 1AC7                  83             SBB      AL,BH
003B 7242                  84             JB       SHORT OUTINC
                           85     ; IF A1[HL) < A1[DE) THEN GOTO ININC
                           86     ; As a side effect, HL and DE are incremented by 1
                           87     ; to point to the nigh bytes of their array elements.
003D 8BF2                  88             MOV      SI,DX
003F AC                    89             LODS     DS>M[SI]
0040 2A870000         R    90             SUB      AL,M[BX]
0044 9F                    91             LAHF               ; ****  The IRPC invocation requires manual editing
0045 42                    92             INC      DX        ; ****  The LAHF and SAHF exact mapping is required
0046 43                    93             INC      BX
0047 9E                    94             SAHF
0048 8BF2                  95             MOV      SI,DX
004A AC                    96             LODS     DS>M[SI]
004B 1A870000         R    97             SBB      AL,M[BX]
004F 732A                  98             JAE      SHORT ININC
                           99     ; Exchange A[DE) with A[HL).  Leave HL and DE
                          100     ; pointing to HIGH bytes.
                          101 +1
```

Figure F-3B. MCS-86™ Macro Assembler Listing of Conversion of 8080 Sort Routine



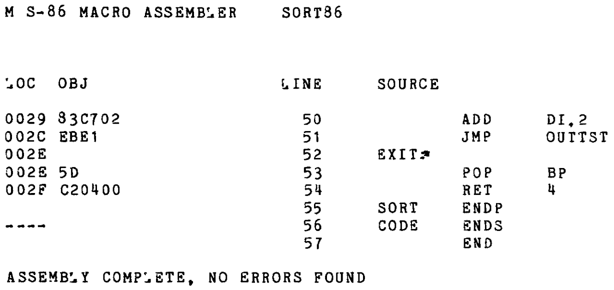
```
MCS-86 MACRO ASSEMBLER    SORT80


LOC  OBJ                   LINE     SOURCE

0051 8BF2                   102 +1          MOV      SI,DX
0053 AC                     103 +1          LODS     DS:M[SI]
0054 8A8F0000        R      104 +1          MOV      CL,M[BX]
0058 88870000        R      105 +1          MOV      M[BX],AL
005C 87DA                   106 +1          XCHG     BX,DX
005E 888F0000        R      107 +1          MOV      M[BX],CL
0062 87DA                   108 +1          XCHG     BX,DX
                            109 +1
                            110 +1
                            111 +2
                            112 +2
                            113 +2
                            114 +4
                            115 +4
                            116 +4
0064 4A                     117 +4          DEC      DX
                            118 +4
                            119 +4
0065 4B                     120 +4          DEC      BX
                            121      ;  Exchange low bytes
                            122 +1
0066 8BF2                   123 +1          MOV      SI,DX
0068 AC                     124 +1          LODS     DS:M[SI]
0069 8A8F0000        R      125 +1          MOV      CL,M[BX]
006D 88870000        R      126 +1          MOV      M[BX],AL
0071 87DA                   127 +1          XCHG     BX,DX
0073 888F0000        R      128 +1          MOV      M[BX],CL
0077 87DA                   129 +1          XCHG     BX,DX
                            130 +1
                            131      ; Point HL and DE to high bytes
0079 42                     132             INC      DX
*  * IRPC call removed and
007A 43                     133             INC      BX
* ** Expanded by hand
                            134      ; DE an HL point to HIGH bytes.  For the next iteration,
                            135      ; set DE = Previous DE, HL = 2 + Previous HL.
007B                        136      ININC:
007B 4A                     137             DEC      DX
007C 43                     138             INC      BX
007D EBB2                   139             JMP      INTST
                            140      ; End of outer loop.  Set DE = DE + 2
007F                        141 +1   OUTINC:
                            142 +2
                            143 +3
007F 42                     144 +3          INC      DX
                            145 +3
0080 42                     146 +3          INC      DX
                            147 +3
0081 EB9B                   148             JMP      OUTTST
----                        149      CODE    ENDS
                            150             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Figure F-3C. MCS-86™ Macro Assembler Listing of Conversion of 8080 Sort Routine



```
MCS-86 MACRO ASSEMBLER    SORT86


ISIS-II MCS-86 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE SORT86
OBJECT MODULE PLACED IN :F1:SORT86.OBJ
ASSEMBLER INVOKED BY: :F3:ASM86 :F1:SORT86

LOC  OBJ                  LINE     SOURCE

                             1     ;*********************************************************
                             2     ; A PL/M callable subroutine:
                             3     ;               CALL SORT(.A1,.N)
                             4     ; Sorts the array A1, containing N words.
                             5     ; At entry the address of N, and the address of A1
                             6     ; are on the stack.  Two pointers to elements of A1
                             7     ; are kept in the SI and DI registers.  These pointers
                             8     ; are incremented in two loops.  The outer loop steps
                             9     ; SI through the elements of A1.  The inner loop steps
                            10     ; DI through the elements of A1 that follow SI.  At each
                            11     ; step of the inner loop, the item at SI is larger than
                            12     ; all the items that follow it.  The item at SI is then in
                            13     ; its proper position, so SI is incremented to
                            14     ; complete one iteration of the outer loop.
                            15     ;*********************************************************
                            16     CGROUP  GROUP   CODE
                            17     ; No DS ASSUME is needed, since this routine
                            18     ; doesn't reference a DATA segment.
                            19             ASSUME  CS:CGROUP
----                        20     CODE    SEGMENT PUBLIC  'CODE'
                            21             PUBLIC  SORT
0000                        22     SORT    PROC    NEAR
  0006[]                    23     ADDR_A1 EQU     WORD PTR [BP+6] ; first parameter
  0004[]                    24     ADDR_N  EQU     WORD PTR [BP+4] ; second parameter
0000 55                     25             PUSH    BP              ; use BP to accesss parameters
0001 8BEC                   26             MOV     BP,SP
0003 8B7606                 27             MOV     SI,ADDR_A1
                            28             ; Outer Loop: DO SI = .A1 BY 2 WHILE SI < CX
0006 8B5E04                 29             MOV     BX,ADDR_N
0009 8B0F                   30             MOV     CX,[BX]         ; CX = N
000B 03C9                   31             ADD     CX,CX           ;        * 2
000D 03CE                   32             ADD     CX,SI           ;            + .A1
000F                        33     OUTTST:
000F 3BF1                   34             CMP     SI,CX          ; IF SI >= CX THEN RETURN
0011 731B                   35             JAE     EXIT
                            36      ; Inner Loop:  DO DI = SI+2 WHILE DI < CX
0013 8D7C02                 37             LEA     DI,[SI+2]                ;DI = SI + 2
0016                        38     INTST:
0016 3BF9                   39             CMP     DI,CX   ; IF DI >= CX
0018 730F                   40             JAE     OUTINC  ; THEN exit inner loop
001A 8B04                   41             MOV     AX,[SI] ;IF A1[SI]
001C 3B05                   42             CMP     AX,[DI] ;          < A1[DI]
001E 7304                   43             JNB     ININC
0020 8705                   44             XCHG    AX,[DI] ; THEN EXCHANGE A1[DI]
0022 8904                   45             MOV     [SI],AX ; WITH A1[SI]
0024                        46     ININC:
0024 83C702                 47             ADD     DI,2
0027 EBED                   48             JMP     INTST
0029                        49     OUTINC:
```

Figure F-4A. MCS-86™ Macro Assembler Listing of Originally Coded 8086 Sort Routine



```
M S-86 MACRO ASSEMBLER    SORT86


LOC  OBJ                 LINE     SOURCE

0029 83C702                50              ADD     DI,2
002C EBE1                  51              JMP     OUTTST
002E                       52     EXIT:
002E 5D                    53              POP     BP
002F C20400                54              RET     4
                           55     SORT     ENDP
----                       56     CODE     ENDS
                           57              END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Figure F-4B. MCS-86™ Macro Assembler Listing of Originally Coded 8086 Sort Routine

APPENDIX G
RELOCATION AND LINKAGE ERRORS AND WARNINGS

Because of the way CONV86 sets up multiple segments beginning at absolute location 0 (as described in Chapter 1 under "Functional Mapping"), MCS-86 linkage and relocation tools will issue warnings/errors as shown in Table G-1. You can safely ignore these warnings/errors when they specifically apply to intentional segment overlap.

Table G-1. MCS-86™ Relocation and Linkage Warnings/Errors for Segment Overlap

R & L Tool	Message ID	Message Text
MCS-86 LINKER	WARNING 14	GROUP ENLARGED FILE: *filename* GROUP: *groupname* MODULE: *modname*
	WARNING 28	POSSIBLE OVERLAP FILE: *filename* MODULE: *modname* SEGMENT: ABS_0 CLASS:

INDEX

Notes:

Notes:

Notes:

Notes:

Notes:

intel®

MCS-86™ Assembly Language Conve
Operating Instructions for ISIS-II U
980064

REQUEST FOR READER'S COMMENTS

The Microcomputer Division Technical Publications Department attempts to provide documents that m the needs of all Intel product users. This form lets you participate directly in the documentation proces

Please restrict your comments to the usability, accuracy, readability, organization, and completenes this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other type documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. ____________

NAME ______________________________ DATE ____________

TITLE ______________________________

COMPANY NAME/DEPARTMENT ______________________________

ADDRESS ______________________________

CITY ______________ STATE ____________ ZIP CODE ____________

Please check here if you require a written reply. □

) LIKE YOUR COMMENTS ...

locument is one of a series describing Intel products. Your comments on the back of this form will us produce better manuals. Each reply will be carefully reviewed by the responsible person. All ıents and suggestions become the property of Intel Corporation.

Intel Corporation
Attn: Technical Publications M/S 6-2000
3065 Bowers Avenue
Santa Clara, CA 95051

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 987-8080

Printed in U.S.A.