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

Manual Order No. 9800642A

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

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 Assembler listings of source and output files.

Although the MCS-86 Assembler (version V1.0) does not support macro or conditional assemblies, Appendix F provides a method by example whereby 8080/8085 source files containing macros and conditionals can be converted to acceptable MCS-86 source 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
- *MCS-86 User's Manual,* Order No. 9800722
- *MCS-86 Assembly Language Reference Manual,* Order No. 9800640
- *MCS-86 Assembler Operating Instructions for ISIS-II Users,* Order No. 9800641
- *MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users,* Order No. 9800639
- *PL/M-86 Operator's Manual for ISIS-II Users,* Order No. 9800478

CONTENTS

TABLES

FIGURES and LISTINGS

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
   - 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).

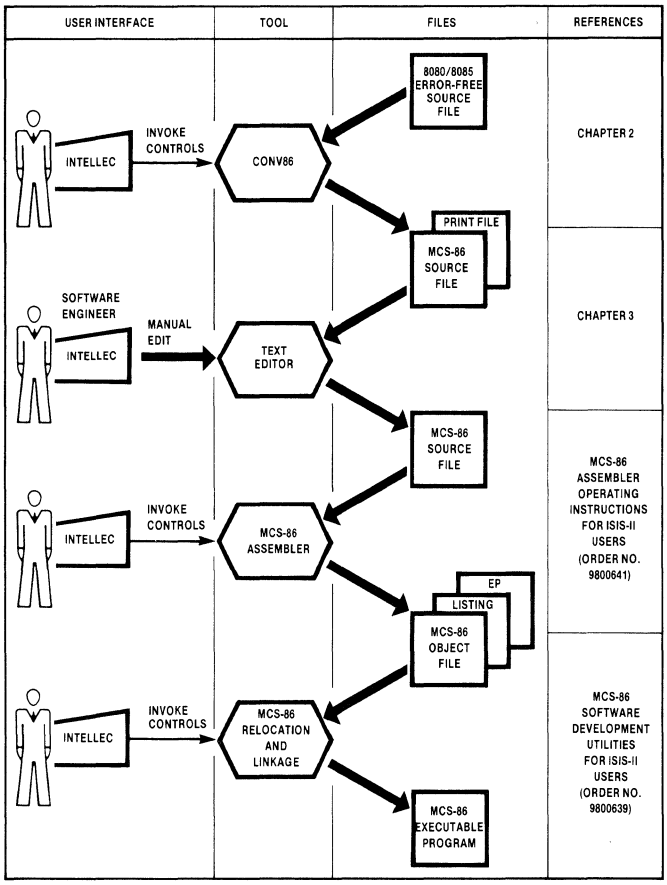


Figure 1-1. From 8080/8085 Assembly Language Source File to 8086 Execution.

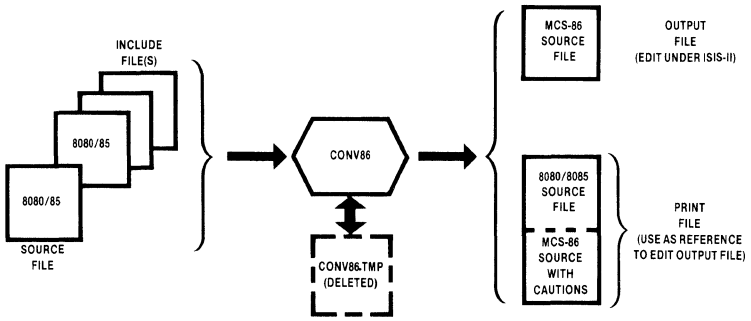


Figure 1-2. CONV86 Input and output Files (The MCS-86 Assembler (version V1.0) does not support the INCLUDE control.)

What About SETs, Macros and Conditional Assembly Directives?

The SET directive, macro definitions, macro calls, and conditional assembly directives are not supported by Version V1.0 of the MCS-86 Assembler. Table C-2 in Appendix C shows how Version V1.0 of CONV86 maps these statements. When CONV86 encounters a macro definition, macro call, or conditional assembly directive, the following caution message is issued to the PRINT file:

```
29  FEATURE NOT SUPPORTED FOR ASM86 V1.0
```

The caution message, however, should *not* be construed as an indication that the mapping shown in Table C-2 will be accepted by the MCS-86 Macro Assembler. If you want to convert your source programs containing macros and conditional directives, you can refer to Appendix F for instructions and examples regarding pre- conversion 8080/8085 assembly and editing procedures.

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.)

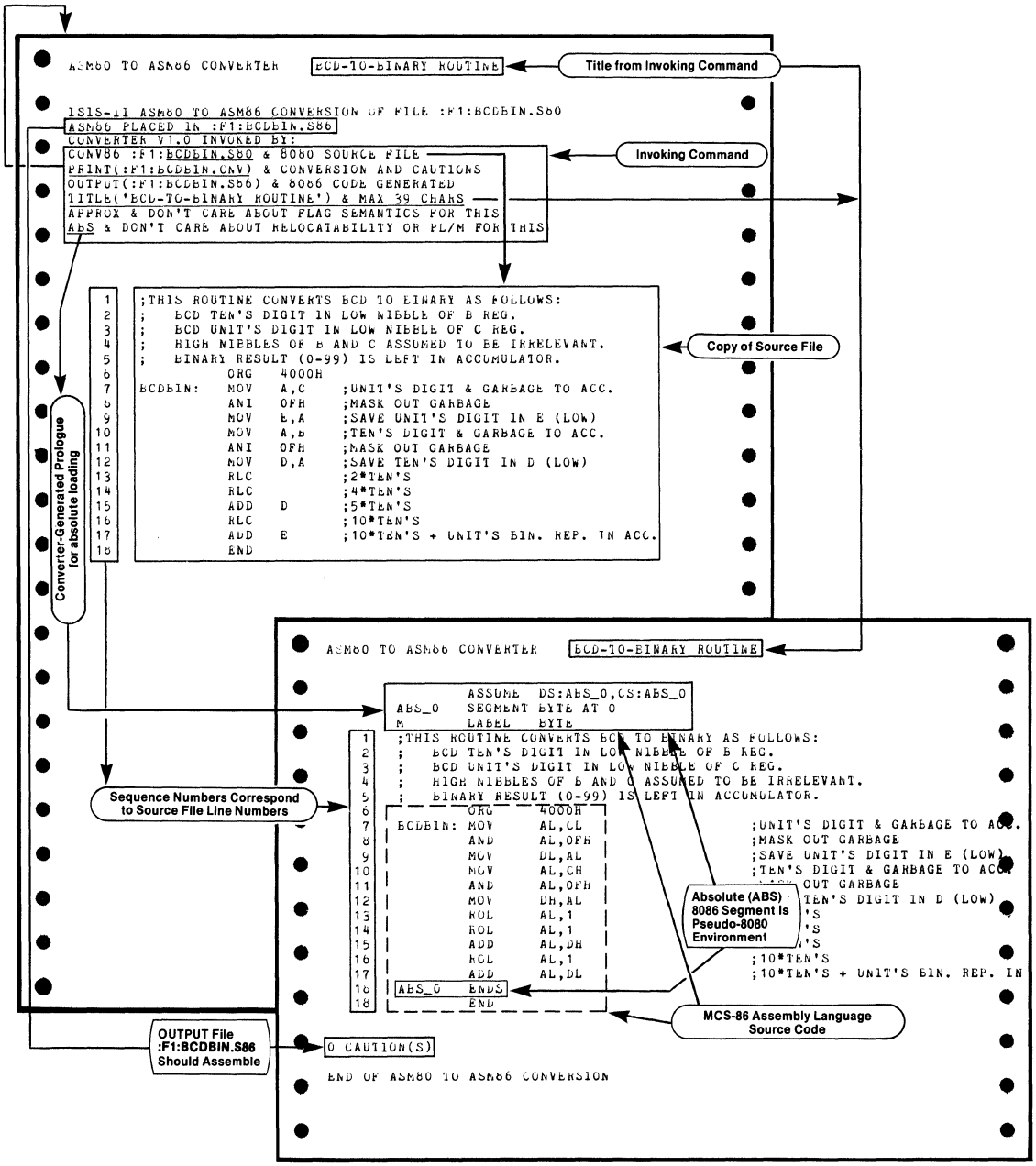


Figure 1-3. Sample PRINT File

```
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 82            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 A
                     18            END

ASSEMBLY COMPLETE,  NO ERRORS
```

```
MCS-86 ASSEMBLER    BCDBIN


ISIS-II MCS-86 ASSEMBLER V1.0 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   ;THIS ROUTINE CONVERTS BCD TO BIN
                             5   ;   BCD TEN'S DIGIT IN LOW NIBB
                             6   ;   BCD UNIT'S DIGIT IN LOW NIBB
                             7   ;   HIGH NIBBLES OF B AND C ASSUME
                             8   ;   BINARY RESULT (0-99) IS LEFT IN
4000                         9           ORG     4000H
4000 8AC1                   10   BCDBIN: MOV     AL,CL
4002 240F                   11           AND     AL,0FH
4004 8AD0                   12           MOV     DL,AL
4006 8AC5                   13           MOV     AL,CH
4008 240F                   14           AND     AL,0FH
400A 8AF0                   15           MOV     DH,AL
400C D0C0                   16           ROL     AL,1
400E D0C0                   17           ROL     AL,1
4010 02C6                   18           ADD     AL,DH
4012 D0C0                   19           ROL     AL,1
4014 02C2                   20           ADD     AL,DL
----                        21   ABS_0   ENDS
                            22           END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
MCS-86 ASSEMBLER    BCDMCS


ISIS-II MCS-86 ASSEMBLER V1.0 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
```

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
```

where N in the STACK segment 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 QRL86 and 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 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?

"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(LASZL O)        ;load BX with word at LASZLO
  .
  .
  .
JMP     NEAR PTR(LASZL O)           ;jump to symbolic location LASZLO
```

How Are Comments Mapped?

Comments are mapped unchanged.

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

Appendix C shows the assembler directive mapping. (Recall that the MCS-86 Assembler (version V1.0) does not support macro or conditional directives, or the SET directive.)

Table C-1 shows the mapping of directives supported by the MCS-86 Assembler (version V1.0).

Table C-2 shows a pseudo-mapping of directives not supported by version V1.0, and should *not* be construed as a specification of MCS-86 Macro Assembler directives.

Operands (expressions) of all directives (whether supported or not) are mapped according to Appendix B.

How Are 8080/8085 Assembler Controls Mapped?

CONV86 deletes the MOD85 and NOMACROFILE controls, and issues corresponding caution messages.

The MACROFILE (:Fn:) control is converted to WORKFILES(:Fs:, :Fn:), where :Fs: is the diskette on which the source file resides. All other 8080/8085 assembler controls are copied unchanged to the 8086 source 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.

NOTE

The MCS-86 Assembler (version V1.0) does not support the INCLUDE control. CONV86 supports the INCLUDE control as described above.

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) 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 line. 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.

What Features Are Not Implemented for the MCS-86 Assembler (version V1.0)?

These features are not implemented for the MCS-86 Assembler (version V1.0):

- The SET directive.
- Macros and/or conditional assembly directives (IF, ELSE, ELSEIF, ENDIF) can be successfully converted using CONV86, but the MCS-86 Assembler (version V1.0) does not support macro or conditional assembly.
- Programs using assembler controls can be converted successfully, but the MCS-86 Assembler (version V1.0) does not support assembler control statements. **(In particular, no INCLUDE files are permitted.)**

Appendix C shows directive mappings.

You can, however, convert 8080 source files containing macros, macro calls, and conditional assemblies by following the procedure and example given in Appendix F. SETs having constants as operands can be replaced by EQUs in your 8086 source file as described under Caution Message 26 in Chapter 3.

CHAPTER 2
OPERATING THE CONVERTER

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.

5. Your source file must not contain assembler controls or any of the following 8080 assembler directives:

   - The SET directive.
   - Macro definition or macro statements, including MACRO, NUL, LOCAL, REPT, IRP, IRPC, EXITM, ENDM, and macro calls.
   - Conditional assembly directives, including IF, ELSE, ENDIF.

   These statements are not supported by version V1.0 of the MCS-86 Assembler. Appendix F shows how to convert 8080/8085 source files that contain macros and conditionals.

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:)

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.

PAGEWIDTH(n)

specifies the number of characters per output line in the PRINT file. The miniumum 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.

NOPAGING

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

Examples

Example 1-1. 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.

CHAPTER 3
EDITING CONVERTER OUTPUT

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
- 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 Assembler.

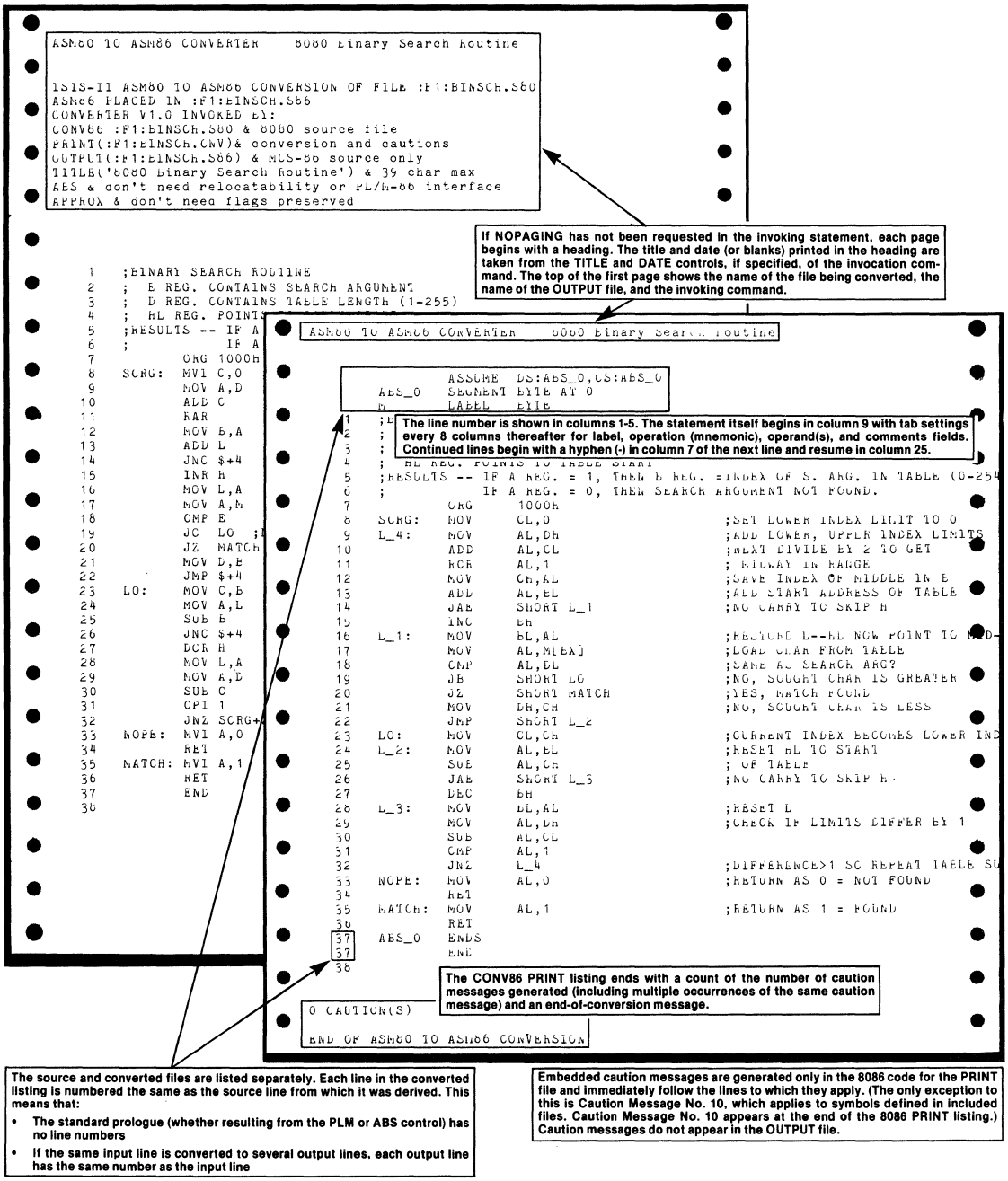


Figure 3-1. Annotated PRINT File

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 QRL86 and 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

```
2          MOV D,E
3          JMP $+4
4    LO:   MOV C,E
5          MOV A,L
```

MCS-86 (CONV86-Generated) PRINT File

```
2          MOV     DH,CH
3          JMP     SHORT L_1
4    LO:   MOV     CL,CH
5    L_1:  MOV     AL,BL
```

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

8080 Source

```
7          MOV A,C
8          JMP $+3*((3+2)*2-7)
9          DB  78h
10         DB  10111101B
11         DW  0EAEAh
12         DW  0BEACh
13         CMA
```

MCS-86 (CONV86-Generated) PRINT File

```
7          MOV     AL,CL
8          JMP     $+3*((3+2)*2-7)
CAUTION 017 *** ADDRESS EXPRESSION
9          DB      78h
10         DB      10111101B
11         DW      0EAEAh
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
JMP     $+3*((3+2)*2-7)
DB      78h
DB      10111101B
DW      0EAEAh
DW      0BEACh
NOT     AL
```

Before Your Edit

```
        MOV     AL,CL
        JMP     SHORT LASZLO
        DB      78H
        DB      10111101B
        DW      0EAEAH
        DW      0BEACH
LASZLO: NOT     AL
```

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 schema 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.

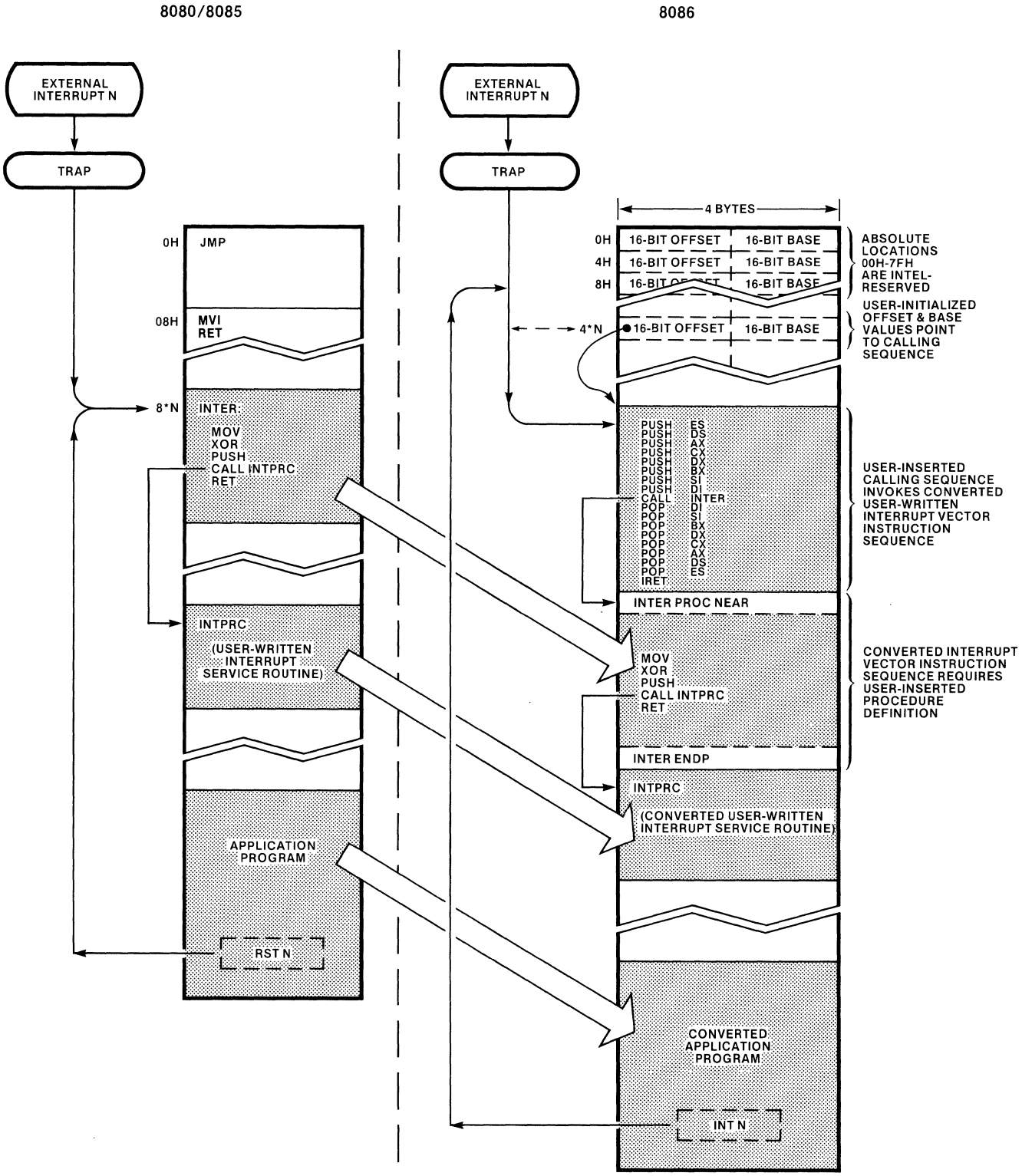


Figure 3-2. Converting Your Interrupt Procedures

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 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 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[1] of caution message is as follows:

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 OR LABEL

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

26 THIS SET DIRECTIVE INCOMPATIBLE WITH 8086

27 SYMBOL NAME TOO LONG

28 CONDITIONAL ASSEMBLY GENERATED

29 FEATURE NOT IMPLEMENTED FOR ASM86 V1.0

---

1. Caution messages 9 and 15 do not exist.

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 CONVERSION OF NOP MAY NOT YIELD DESIRED RESULTS

A 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 CONVERSION OF RST MAY NOT YIELD DESIRED RESULTS

A 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 CONVERTED

The 8086 has no counterpart for RIM or SIM. You should recode according to the 8086 interrupt scheme as described in the *MCS-86 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 OR LABEL

CONV86 has not been able to determine what type of expression is in this instruction. CONV86 has assumed that the expression is a variable or label. 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

The label for this line is unreferenced in this file; 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.

26 THIS SET DIRECTIVE INCOMPATIBLE WITH 8086

An 8086 assembler SET directive must have a constant as its operand. Thus, expressions of the form:

```
X  SET    X+Y
```

have no direct counterpart in 8086-AL. You can, however, use sequences of the form:

```
Z    EQU    X+Y
     PURGE  X
X    EQU    Z
     PURGE  Z
```

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 deskcheck 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 V1.0 of the MCS-86 Assembler.

29 FEATURE NOT IMPLEMENTED FOR ASM86 V1.0

The MCS-86 Assembler (V1.0) does not support IF, ELSE, ENDIF, MACRO, LOCAL, IRP, IRPC, REPT, SET, EXITM, or ENDM. Mappings of these directives are not intended to be assembled. Refer to Appendix F for a conversion procedure for these directives.

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 rw' EQ AX
	
	POP rw'
	XCHG AL, AH
	ELSE
	POP rw'
	ENDIF
		(Using APPROX when rw could be PSW)

POP rw	IF rw' EQ AX
	
	POP rw'
	XCHG AL, AH
	SAHF
	ELSE
	POP rw'
	ENDIF
		(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 rw' EQ AX
	
	XCHG AL, AH
	PUSH rw'
	XCHG AL, AH
	ELSE
	PUSH rw'
	ENDIF
		(Using APPROX Control when rw could be PSW)

PUSH rw	IF rw EQ AX
	
	LAHF
	XCHG AL, AH
	PUSH rw'
	XCHG AL, AH
	ELSE
	PUSH rw'
	ENDIF
		(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']

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

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

2. unknown generates Caution Message 17.

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").

For purposes of the MCS-86 Assembler (version V1.0), the mapping of 8080 assembler directives by CONV86 is here shown in two tables:

- Table C-1 shows the mapping of 8080 directives which convert to 8086 directives that are supported by the MCS-86 Assembler (V1.0).
- Table C-2 shows the mapping of 8080 directives which convert to 8086 pseudo-directives. Entries in Table C-2 are neither supported by the MCS-86 Assembler (version V1.0), nor are they intended to be construed as *bona fide* statements for any future versions of the MCS-86 Assembler.

Table C-1. Assembler Directives Mapping for Supported MCS-86 Directives

8080/8085	8086
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***[1]

1. If the REL control (a default) is used, STKLN converts to information in the prologue. Refer to Chapter 1, "Functional Mapping."

Table C-2 shows those 8080 assembler directives which map into *unsupported* (by version V1.0 of the MCS-86 Assembler) 8086 statements.

If you want to convert a source file containing any of these 8080 assembler directives, you can do it by pre-assembling your source file, and then manually editing (under ISIS-II) your program listing as outlined and illustrated by example in Appendix F.

Table C-2. Assembler Directive Mapping for Unsupported MCS-86 Directives

8080/8085	8086
ELSE	ELSE
ENDIF	ENDIF
ENDM	ENDM
EXITM	EXITM
IF ib	IF ib'
IRP parm,<list>	IRP parm',<list>
IRPC parm,string	IRPC parm',string
LOCAL name-list	LOCAL name-list'
name MACRO parm-list	name' MACRO parm-list'
macro-call arg-list	macro-call' arg-list'
REPT expr	REPT expr'
name SET constant-expr	name' SET constant-expr'
name SET nonconstant-expr	PURGE name' name' EQU nonconstant-expr'

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:

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

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
SAMPLE CONVERSION AND LISTINGS

This appendix consists of:

- Figure E-1. 8080 Listing of Sort Routine
- Figure E-2. PRINT File of Conversion of 8080 Sort Routine
- Figure E-3. MCS-86 Assembler Listing of Conversion of 8080 Sort Routine
- Figure E-4. MCS-86 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 36-39 in Figure E-3) was inserted as described in Chapter 3.
2. 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 E-2 and E-3.

```
ASM80 :F1:SORT80.A80 PRINT(:F1:SORT80.80L) OBJECT(:F1:SORT80.800)

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

  LOC  OBJ          SEQ         SOURCE STATEMENT

                      1 ;*************************************************
                      2 ; A PL/M callable subroutine:
                      3 ;       CALL SORT(.A1, .N)
                      4 ; Sorts the array A1, containing N words.
                      5 ; At entry BC points to the array A1, and
                      6 ; DE points to N.  Two pointers to elements of A1 are
                      7 ; kept in the DE and HL registers. These pointers are
                      8 ; incremented in two loops. The outer loop steps DE
                      9 ; through the elements of A1.  The inner loop steps
                     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 than 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
                     19         CSEG
                     20         PUBLIC  SORT
                     21 ; TEST = address of the last element of A1.
  0000 EB            22 SORT:   XCHG     ; TEST = (N - 1) * 2  +  .A1
  0001 5E            23         MOV      E,M
  0002 23            24         INX      H
  0003 56            25         MOV      D,M
  0004 EB            26         XCHG               ;(N
  0005 2B            27         DCX      H         ;   - 1)
  0006 29            28         DAD      H         ;        * 2
  0007 09            29         DAD      B         ;             +  .A1
  0008 220000    D   30         SHLD     TEST      ;                  = TEST
                     31
                     32 ; OUTER LOOP:   DO DE = .A1 TO TEST BY 2;
  000B 59            33         MOV      E,C       ; BC CONTAINS .A1
  000C 50            34         MOV      D,B
                     35
  000D 3A0000    D   36 OUTTST: LDA      TEST      ; IF DE > TEST THEN RETURN
  0010 93            37         SUB      E
  0011 3A0100    D   38         LDA      TEST + 1
  0014 9A            39         SBB      D
  0015 D8            40         RC
                     41
                     42 ; INNER LOOP:  DO HL = DE+2 TO TEST BY 2
  0016 6B            43         MOV      L,E
  0017 62            44         MOV      H,D
  0018 23            45         INX      H
  0019 23            46         INX      H         ; HL = DE+2
                     47
                     48 ; IF HL > TEST THEN GOTO OUTINC
  001A 3A0000    D   49 INTST:  LDA      TEST
  001D 95            50         SUB      L
  001E 3A0100    D   51         LDA      TEST + 1
  0021 9C            52         SBB      H
  0022 DA4300    C   53         JC       OUTINC
                     54
                     55 ; IF A1(HL) < A1(DE) THEN GOTO ININC
                     56 ; As a side effect, HL and DE are incremented by 1
                     57 ; to point to the high bytes of their array elements.
  0025 1A            58         LDAX     D
  0026 96            59         SUB      M
  0027 13            60         INX      D
  0028 23            61         INX      H
  0029 1A            62         LDAX     D
  002A 9E            63         SBB      M
  002B D23E00    C   64         JNC      ININC
                     65
                     66 ; Exchange A(DE) with A(HL). Leave HL and DE
                     67 ; pointing to HIGH bytes.
  002E 1A            68         LDAX     D         ; SWAP HIGH BYTES
  002F 4E            69         MOV      C,M
```

Figure E-1A

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

  LOC  OBJ          SEQ          SOURCE STATEMENT

  0030 77             70          MOV     M,A
  0031 EB             71          XCHG
  0032 71             72          MOV     M,C
  0033 EB             73          XCHG
                      74
  0034 1B             75          DCX     D       ; POINT HL AND DE TO LOW BYTES.
  0035 2B             76          DCX     H
                      77
  0036 1A             78          LDAX    D       ; SWAP LOW BYTES
  0037 4E             79          MOV     C,M
  0038 77             80          MOV     M,A
  0039 EB             81          XCHG
  003A 71             82          MOV     M,C
  003B EB             83          XCHG
                      84
  003C 13             85          INX     D       ; POINT HL AND DE TO HIGH BYTES.
  003D 23             86          INX     H
                      87
                      88 ; DE and HL point to HIGh bytes. For the next iteration,
                      89 ; set DE = Previous DE, HL = 2 + Previous HL.
  003E 1B             90 ININC:   DCX     D
  003F 23             91          INX     H
  0040 C31A00    C    92          JMP     INTST
                      93
                      94 ; End of outer loop. Set DE = DE + 2 and CONTINUE
  0043 13             95 OUTINC:  INX     D
  0044 13             96          INX     D
  0045 C30D00    C    97          JMP     OUTTST
                      98
                      99
                     100 ; Data area follows.
                     101          DSEG
  0002               102 TEST:    DS      2
                     103          END

PUBLIC SYMBOLS
SORT   C 0000

EXTERNAL SYMBOLS

USER SYMBOLS


ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0         MODULE   PAGE    3

ININC  C 003E    INTST  C 001A    OUTINC C 0043    OUTTST C 000D    SORT   C 0000    TEST   D 0000

ASSEMBLY COMPLETE,  NO ERRORS
```

Figure E-1B

```
ASM80 TO ASM86 CONVERTER


ISIS-II ASM80 TO ASM86 CONVERSION OF FILE :F1:SORT80.A80
ASM86 PLACED IN :F1:SORT80.A86
CONVERTER V1.0 INVOKED BY:
CONV86 :F1:SORT80.A80 PRINT(:F1:SORT80.CVL)


    1   ;*******************************************************
    2   ; A PL/M callable subroutine:
    3   ;       CALL SORT(.A1, .N)
    4   ; Sorts the array A1, containing N words.
    5   ; At entry BC points to the array A1, and
    6   ; DE points to N.  Two pointers to elements of A1 are
    7   ; kept in the DE and HL registers. These pointers are
    8   ; incremented in two loops. The outer loop steps DE
    9   ; through the elements of A1.  The inner loop steps
   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 than 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
   19           CSEG
   20           PUBLIC  SORT
   21   ; TEST = address of the last element of A1.
   22   SORT:   XCHG     ; TEST = (N - 1) * 2  +  .A1
   23           MOV      E,M
   24           INX      H
   25           MOV      D,M
   26           XCHG             ;(N
   27           DCX      H       ;   - 1)
   28           DAD      H       ;         * 2
   29           DAD      B       ;              +  .A1
   30           SHLD     TEST    ;                   = TEST
   31
   32   ; OUTER LOOP:   DO DE = .A1 TO TEST BY 2;
   33           MOV      E,C     ; BC CONTAINS .A1
   34           MOV      D,B
   35
   36   OUTTST: LDA      TEST    ; IF DE > TEST THEN RETURN
   37           SUB      E
   38           LDA      TEST + 1
   39           SBB      D
   40           RC
   41
   42   ; INNER LOOP:  DO HL = DE+2 TO TEST BY 2
   43           MOV      L,E
   44           MOV      H,D
   45           INX      H
   46           INX      H       ; HL = DE+2
   47
   48   ; IF HL > TEST THEN GOTO OUTINC
   49   INTST:  LDA      TEST
   50           SUB      L
   51           LDA      TEST + 1
   52           SBB      H
   53           JC       OUTINC
   54
   55   ; IF A1(HL) < A1(DE) THEN GOTO ININC
   56   ; As a side effect, HL and DE are incremented by 1
   57   ; to point to the high bytes of their array elements.
   58           LDAX     D
   59           SUB      M
   60           INX      D
   61           INX      H
   62           LDAX     D
   63           SBB      M
   64           JNC      ININC
   65
   66   ; Exchange A(DE) with A(HL). Leave HL and DE
   67   ; pointing to HIGH bytes.
   68           LDAX     D       ; SWAP HIGH BYTES
```

Figure E-2A

```
ASM80 TO ASM86 CONVERTER

   69           MOV     C,M
   70           MOV     M,A
   71           XCHG
   72           MOV     M,C
   73           XCHG
   74
   75           DCX     D       ; POINT HL AND DE TO LOW BYTES.
   76           DCX     H
   77
   78           LDAX    D       ; SWAP LOW BYTES
   79           MOV     C,M
   80           MOV     M,A
   81           XCHG
   82           MOV     M,C
   83           XCHG
   84
   85           INX     D       ; POINT HL AND DE TO HIGH BYTES.
   86           INX     H
   87
   88   ; DE and HL point to HIGH bytes. For the next iteration,
   89   ; set DE = Previous DE, HL = 2 + Previous HL.
   90   ININC:  DCX     D
   91           INX     H
   92           JMP     INTST
   93
   94   ; End of outer loop. Set DE = DE + 2 and CONTINUE
   95   OUTINC: INX     D
   96           INX     D
   97           JMP     OUTTST
   98
   99
  100   ; Data area follows.
  101           DSEG
  102   TEST:   DS      2
  103           END
```

Figure E-2B

```
ASM86 TO ASM86 CONVERTER

        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
   1    ;*****************************************************
   2    ; A PL/M callable subroutine:
   3    ;       CALL SORT(.A1, .N)
   4    ; Sorts the array A1, containing N words.
   5    ; At entry BC points to the array A1, and
   6    ; DE points to N.  Two pointers to elements of A1 are
   7    ; kept in the DE and HL registers.  These pointers are
   8    ; incremented in two loops.  The outer loop steps DE
   9    ; through the elements of A1.  The inner loop steps
  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 than 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
  19    ABS_0   ENDS
  19    CODE    SEGMENT WORD PUBLIC 'CODE'
  20            PUBLIC  SORT
  21    ; TEST = address of the last element of A1.
  22    SORT:   XCHG    BX,DX                   ; TEST = (N - 1) * 2  +  .A1
  23            MOV     DL,M[BX]
  24            LAHF
  24            INC     BX
  24            SAHF
  25            MOV     DH,M[BX]
  26            XCHG    BX,DX                   ;(N
  27            LAHF
  27            DEC     BX
  27            SAHF                            ;   - 1)
  28            LAHF
  28            ADD     BX,BX
  28            RCR     SI,1
  28            SAHF
  28            RCL     SI,1                    ;        * 2
  29            LAHF
  29            ADD     BX,CX
  29            RCR     SI,1
  29            SAHF
  29            RCL     SI,1                    ;             +  .A1
  30            MOV     WORD PTR(TEST_),BX      ;                  = TEST
  31
  32    ; OUTER LOOP:   DO DE = .A1 TO TEST BY 2;
  33            MOV     DL,CL                   ; BC CONTAINS .A1
  34            MOV     DH,CH
  35
  36    OUTTST: MOV     AL,TEST_                ; IF DE > TEST THEN RETURN
  37            SUB     AL,DL
  38            MOV     AL,TEST_+1
  39            SBB     AL,DH
  40            JNB     SHORT L_1
  40            RET
  40    L_1:
  41
  42    ; INNER LOOP:  DO HL = DE+2 TO TEST BY 2
  43            MOV     BL,DL
  44            MOV     BH,DH
  45            LAHF
  45            INC     BX
```

Figure E-2C

```
ASM80 TO ASM86 CONVERTER

   45           SAHF
   46           LAHF
   46           INC     BX
   46           SAHF                            ; HL = DE+2
   47
   48   ; IF HL > TEST THEN GOTO OUTINC
   49   INTST:  MOV     AL,TEST_
   50           SUB     AL,BL
   51           MOV     AL,TEST_+1
   52           SBB     AL,BH
   53           JB      SHORT OUTINC
   54
   55   ; IF A1(HL) < A1(DE) THEN GOTO ININC
   56   ; As a side effect, HL and DE are incremented by 1
   57   ; to point to the high bytes of their array elements.
   58           MOV     SI,DX
   58           LODS    DS:M[SI]
   59           SUB     AL,M[BX]
   60           LAHF
   60           INC     DX
   60           SAHF
   61           LAHF
   61           INC     BX
   61           SAHF
   62           MOV     SI,DX
   62           LODS    DS:M[SI]
   63           SBB     AL,M[BX]
   64           JAE     SHORT ININC
   65
   66   ; Exchange A(DE) with A(HL). Leave HL and DE
   67   ; pointing to HIGH bytes.
   68           MOV     SI,DX
   68           LODS    DS:M[SI]                ; SWAP HIGH BYTES
   69           MOV     CL,M[BX]
   70           MOV     M[BX],AL
   71           XCHG    BX,DX
   72           MOV     M[BX],CL
   73           XCHG    BX,DX
   74
   75           LAHF
   75           DEC     DX
   75           SAHF                            ; POINT HL AND DE TO LOW BYTES.
   76           LAHF
   76           DEC     BX
   76           SAHF
   77
   78           MOV     SI,DX
   78           LODS    DS:M[SI]                ; SWAP LOW BYTES
   79           MOV     CL,M[BX]
   80           MOV     M[BX],AL
   81           XCHG    BX,DX
   82           MOV     M[BX],CL
   83           XCHG    BX,DX
   84
   85           LAHF
   85           INC     DX
   85           SAHF                            ; POINT HL AND DE TO HIGH BYTES.
   86           LAHF
   86           INC     BX
   86           SAHF
   87
   88   ; DE and 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
   91           SAHF
   92           JMP     INTST
   93
   94   ; End of outer loop. Set DE = DE + 2 and CONTINUE
   95   OUTINC: LAHF
   95           INC     DX
```

Figure E-2D

```
ASM80 TO ASM86 CONVERTER

   95            SAHF
   96            LAHF
   96            INC     DX
   96            SAHF
   97            JMP     OUTTST
   98
   99
  100    ; Data area follows.
  101    CODE    ENDS
  101    DATA    SEGMENT WORD PUBLIC 'DATA'
  102    TEST_   DB      2 DUP (?)
  103    DATA    ENDS
  103            END

0 CAUTION(S)

END OF ASM80 TO ASM86 CONVERSION
```

Figure E-2E

```
MCS-86 ASSEMBLER    SORT80


ISIS-II MCS-86 ASSEMBLER V1.0 ASSEMBLY OF MODULE SORT80
OBJECT MODULE PLACED IN :F1:SORT80.860
ASSEMBLER INVOKED BY: ASM86 :F1:SORT80.A86 PRINT(:F1:SORT80.86L) OBJECT(:F1:SORT80.860)

LOC  OBJ                  LINE   SOURCE

                             1   CGROUP  GROUP   ABS_0,CODE,CONST,DATA,STACK,MEMORY
                             2   DGROUP  GROUP   ABS_0,CODE,CONST,DATA,STACK,MEMORY
                             3           ASSUME  DS:DGROUP,CS:CGROUP,SS:DGROUP
----                         4   CONST   SEGMENT WORD PUBLIC 'CONST'
----                         5   CONST   ENDS
----                         6   STACK   SEGMENT WORD STACK 'STACK'
0000                         7   STACK_BASE      LABEL   BYTE
----                         8   STACK   ENDS
----                         9   MEMORY  SEGMENT WORD MEMORY 'MEMORY'
0000                        10   MEMORY_ LABEL   BYTE
----                        11   MEMORY  ENDS
----                        12   ABS_0   SEGMENT BYTE AT 0
0000                        13   N       LABEL   BYTE
                            14   ;*************************************************
                            15   ; A PL/M callable subroutine:
                            16   ;       CALL SORT(.A1, .N)
                            17   ; Sorts the array A1, containing N words.
                            18   ; At entry BC points to the array A1, and
                            19   ; DE points to N.  Two pointers to elements of A1 are
                            20   ; kept in the DE and HL registers. These pointers are
                            21   ; incremented in two loops. The outer loop steps DE
                            22   ; through the elements of A1.  The inner loop steps
                            23   ; HL through the elements of A1 that follow DE.  At
                            24   ; each step of the inner loop, the items at HL and DE
                            25   ; are exchanged, if required, so that at the end of
                            26   ; the inner loop, the item at DE is larger than all
                            27   ; the items that follow it. The item at DE is then in
                            28   ; its proper position, so DE is incremented to
                            29   ; complete one iteration of the outer loop.
                            30   ;*************************************************
                            31
----                        32   ABS_0   ENDS
----                        33   CODE    SEGMENT WORD PUBLIC 'CODE'
                            34           PUBLIC  SORT
                            35   ; TEST = address of the last element of A1.
0000 5B                     36   SORT:   POP     BX      ; **** CODE INSERTED TO
0001 5A                     37           POP     DX      ; **** RETRIEVE PL/M-86
0002 59                     38           POP     CX      ; **** STACK PARAMETERS
0003 53                     39           PUSH    BX      ; **** (CHAPTER 3)
0004 87DA                   40           XCHG    BX,DX                   ; TEST = (N - 1) * 2  +  .A1
0006 8A970000         R     41           MOV     DL,M[BX]
000A 43                     42           INC     BX
000B 8AB70000         R     43           MOV     DH,M[BX]
000F 87DA                   44           XCHG    BX,DX                   ;(N
0011 4B                     45           DEC     BX                      ;   - 1)
0012 03DB                   46           ADD     BX,BX                   ;         * 2
0014 03D9                   47           ADD     BX,CX                   ;               + .A1
0016 891E0000         R     48           MOV     WORD PTR(TEST_),BX      ;                     = TEST
                            49
                            50   ; OUTER LOOP:   DO DE = .A1 TO TEST BY 2;
001A 8AD1                   51           MOV     DL,CL                   ; BC CONTAINS .A1
001C 8AF5                   52           MOV     DH,CH
                            53
001E A00000           R     54   OUTTST: MOV     AL,TEST_                ; IF DE > TEST THEN RETURN
0021 2AC2                   55           SUB     AL,DL
0023 A00100           R     56           MOV     AL,TEST_+1
0026 1AC6                   57           SBB     AL,DH
0028 7301                   58           JNB     SHORT L_1
002A C3                     59           RET
002B                        60   L_1:
                            61
                            62   ; INNER LOOP:  DO HL = DE+2 TO TEST BY 2
002B 8ADA                   63           MOV     BL,DL
002D 8AFE                   64           MOV     BH,DH
002F 43                     65           INC     BX
0030 43                     66           INC     BX                      ; HL = DE+2
                            67
```

Figure E-3A

```
MCS-86 ASSEMBLER    SORT80


LOC  OBJ                  LINE   SOURCE

                            68   ; IF HL > TEST THEN GOTO OUTINC
0031 A00000          R      69   INTST:  MOV     AL,TEST_
0034 2AC3                   70           SUB     AL,BL
0036 A00100          R      71           MOV     AL,TEST_+1
0039 1AC7                   72           SBB     AL,BH
003B 7242                   73           JB      SHORT OUTINC
                            74
                            75   ; IF A1(HL) < A1(DE) THEN GOTO ININC
                            76   ; As a side effect, HL and DE are incremented by 1
                            77   ; to point to the high bytes of their array elements.
003D 8BF2                   78           MOV     SI,DX
003F AC                     79           LODS    DS:M[SI]
0040 2A870000        R      80           SUB     AL,M[BX]
0044 9F                     81           LAHF                    ; **** THE UNNECCESSARY 'EXACT'
                            82                                   ; **** MAPPED CODE WAS REMOVED
0045 42                     83           INC     DX
0046 43                     84           INC     BX
0047 9E                     85           SAHF    ; **** THIS 'EXACT' CODE IS ALSO NEEDED
0048 8BF2                   86           MOV     SI,DX
004A AC                     87           LODS    DS:M[SI]
004B 1A870000        R      88           SBB     AL,M[BX]
004F 732A                   89           JAE     SHORT ININC
                            90
                            91   ; Exchange A(DE) with A(HL). Leave HL and DE
                            92   ; pointing to HIGH bytes.
0051 8BF2                   93           MOV     SI,DX
0053 AC                     94           LODS    DS:M[SI]                ; SWAP HIGH BYTES
0054 8A8F0000        R      95           MOV     CL,M[BX]
0058 88870000        R      96           MOV     M[BX],AL
005C 87DA                   97           XCHG    BX,DX
005E 888F0000        R      98           MOV     M[BX],CL
0062 87DA                   99           XCHG    BX,DX
                           100
0064 4A                    101           DEC     DX                      ; POINT HL AND DE TO LOW BYTES.
0065 4B                    102           DEC     BX
                           103
0066 8BF2                  104           MOV     SI,DX
0068 AC                    105           LODS    DS:M[SI]                ; SWAP LOW BYTES
0069 8A8F0000        R     106           MOV     CL,M[BX]
006D 88870000        R     107           MOV     M[BX],AL
0071 87DA                  108           XCHG    BX,DX
0073 888F0000        R     109           MOV     M[BX],CL
0077 87DA                  110           XCHG    BX,DX
                           111
0079 42                    112           INC     DX                      ; POINT HL AND DE TO HIGH BYTES.
007A 43                    113           INC     BX
                           114
                           115   ; DE and HL point to HIGH bytes. For the next iteration,
                           116   ; set DE = Previous DE, HL = 2 + Previous HL.
007B 4A                    117   ININC:  DEC     DX
007C 43                    118           INC     BX
007D EBB2                  119           JMP     INTST
                           120
                           121   ; End of outer loop. Set DE = DE + 2 and CONTINUE
007F 42                    122   OUTINC: INC     DX
0080 42                    123           INC     DX
0081 EB9B                  124           JMP     OUTTST
                           125
                           126   ; Data area follows.
                           127
----                       128   CODE    ,ENDS
----                       129   DATA    SEGMENT WORD PUBLIC 'DATA'
0000 (2                    130   TEST_   DB      2 DUP (?)
     ??
     )
----                       131   DATA    ENDS
                           132           END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Figure E-3B

```
MCS-86 ASSEMBLER    SORT86


ISIS-II MCS-86 ASSEMBLER V1.0 ASSEMBLY OF MODULE SORT86
OBJECT MODULE PLACED IN :F1:SORT86.860
ASSEMBLER INVOKED BY: ASM86 :F1:SORT86.A86 PRINT(:F1:SORT86.86L) OBJECT(:F1:SORT86.860)

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
                           11   ; each step of the inner loop, the items at DI and SI
                           12   ; are exchanged, if required, so that at the end of
                           13   ; the inner loop, the item at SI is larger than all
                           14   ; the items that follow it. The item at SI is then in
                           15   ; its proper position, so SI is incremented to
                           16   ; complete one iteration of the outer loop.
                           17   ;*******************************************************
                           18
                           19   CGROUP  GROUP   CODE
                           20   ; No DS ASSUME is needed, since this routine
                           21   ; doesn't reference a DATA segment.
                           22           ASSUME  CS:CGROUP
----                       23   CODE    SEGMENT PUBLIC 'CODE'
                           24           PUBLIC  SORT
0000                       25   SORT    PROC    NEAR
  0006[]                   26   ADDR_A1 EQU      WORD PTR [BP+6]        ; first parameter
  0004[]                   27   ADDR_N  EQU      WORD PTR [BP+4]        ; second parameter
                           28
0000 55                    29           PUSH    BP      ; use BP to access parameters
0001 8BEC                  30           MOV     BP,SP
0003 8B7606                31           MOV     SI,ADDR_A1
                           32
                           33   ; Outer loop: DO SI = .A1 BY 2 WHILE SI < CX
0006 8B5E04                34           MOV     BX,ADDR_N
0009 8B0F                  35           MOV     CX,[BX] ; CX = N
000B 03C9                  36           ADD     CX,CX   ;        * 2
000D 03CE                  37           ADD     CX,SI   ;              +  .A1
                           38
000F 3BF1                  39   OUTTST: CMP     SI,CX   ; IF SI >= CX THEN RETURN
0011 731B                  40           JAE     EXIT
                           41
                           42   ; Inner loop: DO DI = SI+2 BY 2 WHILE DI < CX
0013 8D7C02                43           LEA     DI,[SI+2]       ; DI = SI+2
0016 3BF9                  44   INTST:  CMP     DI,CX   ;IF DI >= CX
0018 730F                  45           JAE     OUTINC  ;   THEN exit inner loop
                           46
001A 8B04                  47           MOV     AX,[SI] ; IF A1[SI]
001C 3B05                  48           CMP     AX,[DI] ;           < A1[DI]
001E 7304                  49           JNB     ININC
                           50
0020 8705                  51           XCHG    AX,[DI] ; THEN EXCHANGE A1[DI]
0022 8904                  52           MOV     [SI],AX ; WITH A1[SI]
                           53
0024 83C702                54   ININC:  ADD     DI,2    ; END INNER LOOP
0027 EBED                  55           JMP     INTST
                           56
0029 83C602                57   OUTINC: ADD     SI,2    ; END OUTER LOOP
002C EBE1                  58           JMP     OUTTST
                           59
002E 5D                    60   EXIT:   POP     BP
002F C20400                61           RET     4
                           62   SORT    ENDP
----                       63   CODE    ENDS
                           64           END


ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Figure E-4

APPENDIX F
CONVERTING MACROS AND CONDITIONAL ASSEMBLIES

Because version V1.0 of the MCS-86 Assembler does not support macros (including the directives MACRO, IRP, IRPC, LOCAL, REPT, macro call, EXITM, or ENDM) or conditional assembler directives (including IF, ELSE, ENDIF), this Appendix provides a method of converting these constructs. The method is as follows:

1. Assemble your 8080/8085 source file using the ISIS-II 8080/8085 Macro Assembler, version 2.0, using the following controls:
   - NOPAGING
   - MACROFILE
   - NOCOND
   - GEN
   - NOMACRODEBUG
2. Edit your 8080/8085 program list file as follows:
   a. Delete the header and trailer information.
   b. Delete the first 24 columns (location, object, sequence numbers, and macro-generated plus (+) signs, where applicable) of every remaining line.
   c. Delete (or convert to comments) all macro skeletons (definitions), macro calls, and other (non-comment) lines which result in no object code.
3. Submit the resulting file to CONV86 as described in Chaper 2, and treat the converter output as described in Chapter 3.

The remainder of this Appendix traces the evolution of an 8080 source file containing macros and conditional assembler directives through the following steps:

- F-1. 8080 Macro Assembler Listing (MACROS.L80) and Editing Procedure
- F-2. Edited 8080 Macro Assembler listing (MACROS.E80)
- F-3. PRINT file from conversion of edited listing (MACROS.CNV)
- F-4. MCS-86 Macro Assembler (V1.0) listing of converted file (MACROS.L86)

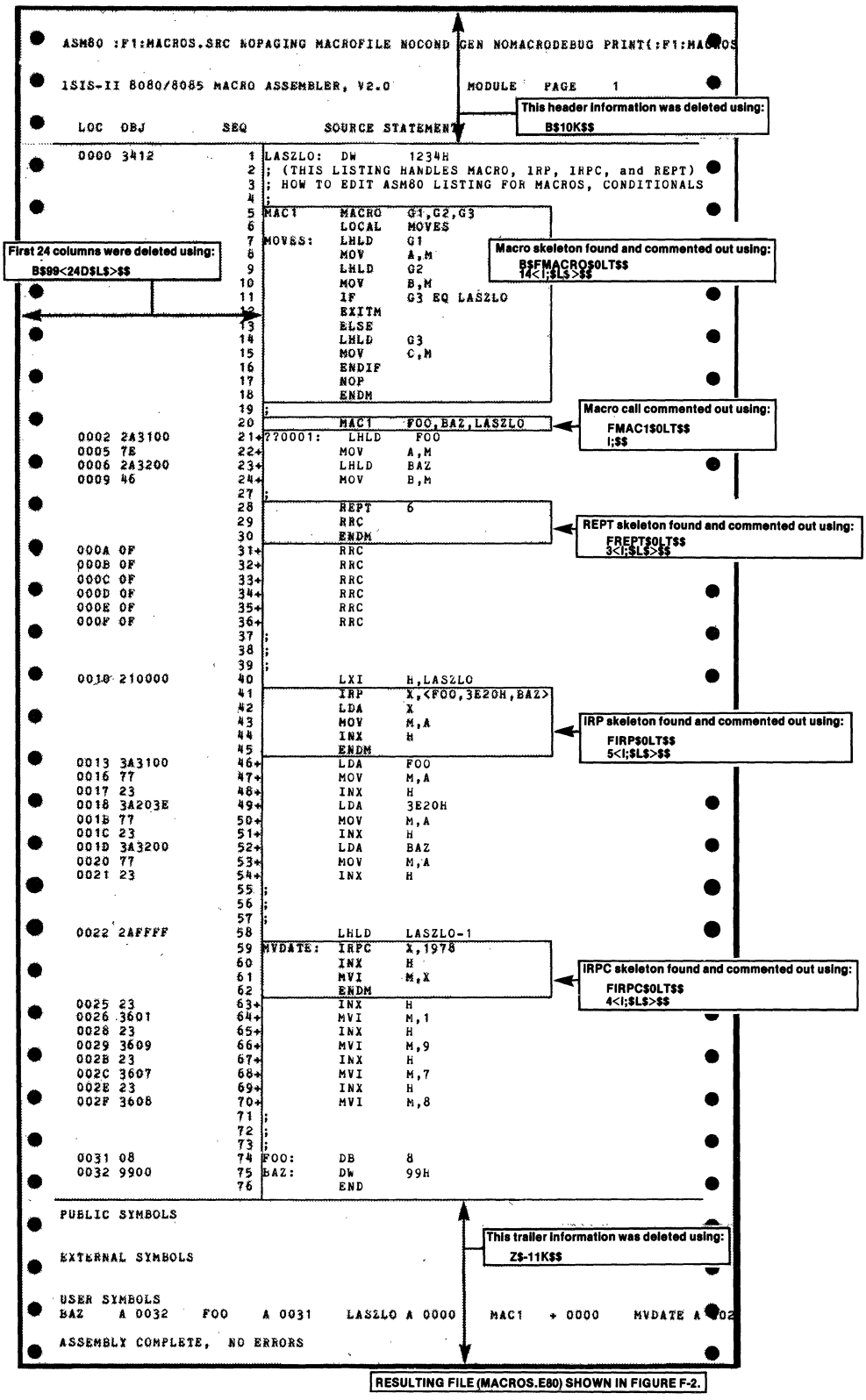
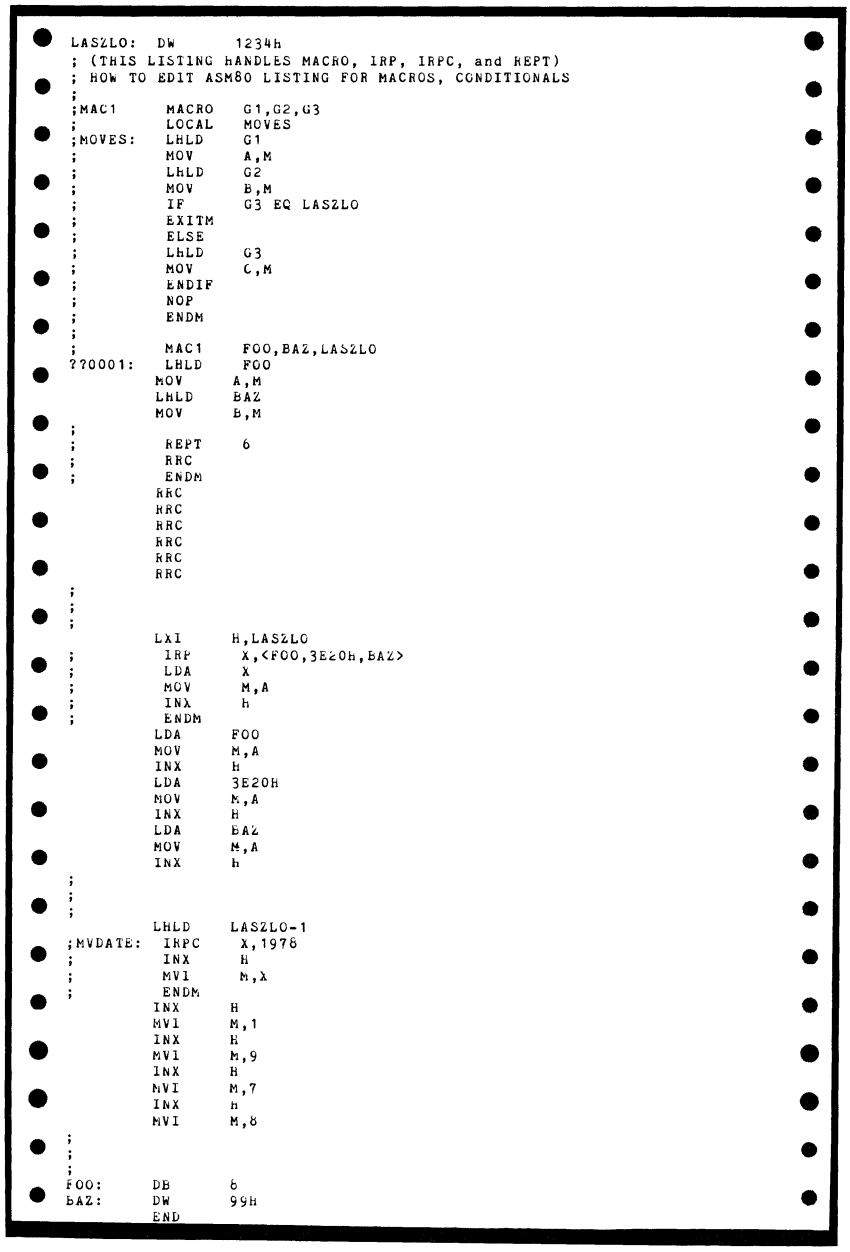


Figure F-1. Annotated 8080 Macro Assembler Listing (MACROS.L80)



```
LASZLO:  DW      1234h
; (THIS LISTING HANDLES MACRO, IRP, IRPC, and REPT)
; HOW TO EDIT ASM80 LISTING FOR MACROS, CONDITIONALS
;
;MAC1     MACRO   G1,G2,G3
;        LOCAL   MOVES
;MOVES:  LHLD    G1
;        MOV     A,M
;        LHLD    G2
;        MOV     B,M
;        IF      G3 EQ LASZLO
;        EXITM
;        ELSE
;        LHLD    G3
;        MOV     C,M
;        ENDIF
;        NOP
;        ENDM
;
;        MAC1    FOO,BAZ,LASZLO
??0001:  LHLD    FOO
         MOV     A,M
         LHLD    BAZ
         MOV     B,M
;
;        REPT    6
;        RRC
;        ENDM
         RRC
         RRC
         RRC
         RRC
         RRC
         RRC
;
;
;
         LXI     H,LASZLO
;        IRP     X,<FOO,3E20H,BAZ>
;        LDA     X
;        MOV     M,A
;        INX     H
;        ENDM
         LDA     FOO
         MOV     M,A
         INX     H
         LDA     3E20H
         MOV     M,A
         INX     H
         LDA     BAZ
         MOV     M,A
         INX     H
;
;
;
         LHLD    LASZLO-1
;MVDATE:  IRPC    X,1978
;        INX     H
;        MVI     M,X
;        ENDM
         INX     H
         MVI     M,1
         INX     H
         MVI     M,9
         INX     H
         MVI     M,7
         INX     H
         MVI     M,8
;
;
;
FOO:     DB      6
BAZ:     DW      99H
         END
```

Figure F-2. Edited 8080 Macro Assembler Listing (MACROS.E80)

```
ASM80 TO ASM86 CONVERTER    Converting Macros and Conditionals


ISIS-II ASM80 TO ASM86 CONVERSION OF FILE :f1:macros.e80
ASM86 PLACED IN :f1:macros.A86
CONVERTER V1.0 INVOKED BY:
conv86 :f1:macros.e80 & edited listing of macro assemoly
print(:f1:macros.cnv) & conversion and cautions
title('Converting Macros and Conditionals') & See App'dixF-1
abs & Don't care about relocatability or PL/M-86
approx & Don't care about saving flags



    1   LASZLO:  DW      1234H
    2   ; (THIS LISTING HANDLES MACRO, IRP, IRPC, and REPT)
    3   ; HOW TO EDIT ASM80 LISTING FOR MACROS, CONDITIONALS
    4   ;
    5   ;MAC1     MACRO   G1,G2,G3
    6   ;         LOCAL   MOVES
    7   ;MOVES:   LHLD    G1
    8   ;         MOV     A,M
    9   ;         LHLD    G2
   10   ;         MOV     B,M
   11   ;         IF      G3 EQ LASZLO
   12   ;         EXITM
   13   ;         ELSE
   14   ;         LHLD    G3
   15   ;         MOV     C,M
   16   ;         ENDIF
   17   ;         NOP
   18   ;         ENDM
   19   ;
   20   ;         MAC1    FOO,BAZ,LASZLO
   21   ??0001:   LHLD    FOO
   22             MOV     A,M
   23             LHLD    BAZ
   24             MOV     B,M
   25   ;
   26   ;         REPT    6
   27   ;         RRC
   28   ;         ENDM
   29             RRC
   30             RRC
   31             RRC
   32             RRC
   33             RRC
   34             RRC
   35   ;
   36   ;
   37   ;
   38             LXI     H,LASZLO
   39   ;         IRP     X,<FOO,3E20H,BAZ>
   40   ;         LDA     X
   41   ;         MOV     M,A
   42   ;         INX     H
   43   ;         ENDM
   44             LDA     FOO
   45             MOV     M,A
   46             INX     H
   47             LDA     3E20H
   48             MOV     M,A
   49             INX     H
   50             LDA     BAZ
   51             MOV     M,A
   52             INX     H
   53   ;
   54   ;
   55   ;
   56             LHLD    LASZLO-1
   57   ;MVDATA:  IRPC    X,1976
   58   ;         INX     H
   59   ;         MVI     M,X
   60   ;         ENDM
   61             INX     H
   62             MVI     M,1
   63             INX     H
   64             MVI     M,9
   65             INX     H
   66             MVI     M,7
   67             INX     H
   68             MVI     M,8
   69   ;
   70   ;
   71   ;
   72   FOO:     DB      6
   73   BAZ:     DW      99H
   74             END
```

Figure F-3A. Conversion of Edited Macro File (8080 Source Shown)

```
ASM80 TO ASM86 CONVERTER    Converting Macros and Conditionals

                    ASSUME  DS:ABS_0,CS:ABS_0
          ABS_0     SEGMENT BYTE AT 0
          M         LABEL   BYTE
     1    LASZLO    DW      1234H
     2    ; (THIS LISTING HANDLES MACRO, IRP, IRPC, and REPT)
     3    ; HOW TO EDIT ASM80 LISTING FOR MACROS, CONDITIONALS
     4    ;
     5    ;MAC1      MACRO   G1,G2,G3
     6    ;          LOCAL   MOVES
     7    ;MOVES:    LHLD    G1
     8    ;          MOV     A,M
     9    ;          LHLD    G2
    10    ;          MOV     B,M
    11    ;          IF      G3 EQ LASZLO
    12    ;          EXITM
    13    ;          ELSE
    14    ;          LHLD    G3
    15    ;          MOV     C,M
    16    ;          ENDIF
    17    ;          NOP
    18    ;          ENDM
    19    ;
    20    ;          MAC1    FOO,BAZ,LASZLO
    21    ??0001:    MOV     BX,WORD PTR(FOO)
    22               MOV     AL,M[BX]
    23               MOV     BX,BAZ
    24               MOV     CH,M[BX]
    25    ;
    26    ;          REPT    6
    27    ;          RRC
    28    ;          ENDM
    29               ROR     AL,1
    30               ROR     AL,1
    31               ROR     AL,1
    32               ROR     AL,1
    33               ROR     AL,1
    34               ROR     AL,1
    35    ;
    36    ;
    37    ;
    38               LEA     BX,LASZLO
    39    ;          IRP     X,<FOO,3E20H,BAZ>
    40    ;          LDA     X
    41    ;          MOV     M,A
    42    ;          INX     H
    43    ;          ENDM
    44               MOV     AL,FOO
    45               MOV     M[BX],AL
    46               INC     BX
    47               MOV     AL,M[3E20H]
    48               MOV     M[BX],AL
    49               INC     BX
    50               MOV     AL,BYTE PTR(BAZ)
    51               MOV     M[BX],AL
    52               INC     BX
    53    ;
    54    ;
    55    ;
    56               MOV     BX,LASZLO-1
*** CAUTION 017 ***  ADDRESS EXPRESSION MAY BE INVALID FOR 8086
    57    ;MYDATE:   IRPC    X,1978
    58    ;          INX     H
    59    ;          MVI     M,X
    60    ;          ENDM
    61               INC     BX
    62               MOV     M[BX],1
    63               INC     BX
    64               MOV     M[BX],9
    65               INC     BX
    66               MOV     M[BX],7
    67               INC     BX
    68               MOV     M[BX],8
    69    ;
    70    ;
    71    ;
    72    FOO        DB      8
    73    BAZ        DW      99H
    74    ABS_0      ENDS
    74               END

1 CAUTION(S)

END OF ASM80 TO ASM86 CONVERSION
```

NOTE

Caution 17 does not require manual editing here; LASZLO-1 is the expression that we want

Figure F-3B. Conversion of Edited 8080 Macro File (MCS-86 Source Shown)

```
MCS-86 ASSEMBLER    MACROS


ISIS-II MCS-86 ASSEMBLER V1.0 ASSEMBLY OF MODULE MACROS
OBJECT MODULE PLACED IN :f1:macros.OBJ
ASSEMBLER INVOKED BY: asm86 :f1:macros.a86 print(:f1:macros.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
0000 3412                    4   LASZLO DW      1234h
                             5   ; (THIS LISTING HANDLES MACRO, IRP, IRPC, and R
                             6   ; HOW TO EDIT ASM80 LISTING FOR MACROS, CONDITI
                             7   ;
                             8   ;MAC1     MACRO   G1,G2,G3
                             9   ;         LOCAL   MOVES
                            10   ;MOVES:   LHLD    G1
                            11   ;         MOV     A,M
                            12   ;         LHLD    G2
                            13   ;         MOV     B,M
                            14   ;         IF      G3 EQ LASZLO
                            15   ;         EXITM
                            16   ;         ELSE
                            17   ;         LHLD    G3
                            18   ;         MOV     C,M
                            19   ;         ENDIF
                            20   ;         NOP
                            21   ;         ENDM
                            22   ;
                            23   ;         MAC1    FOO,BAZ,LASZLO
0002 8B1E4400               24   ??0001: MOV     BX,WORD PTR(FOO)
0006 8A07                   25           MOV     AL,M[BX]
0008 8B1E4500               26           MOV     BX,BAZ
000C 8A2F                   27           MOV     CH,M[BX]
                            28   ;
                            29   ;         REPT    6
                            30   ;         IRC
                            31   ;         ENDM
000E D0C8                   32           ROR     AL,1
0010 D0C8                   33           ROR     AL,1
0012 D0C8                   34           ROR     AL,1
0014 D0C8                   35           ROR     AL,1
0016 D0C8                   36           ROR     AL,1
0018 D0C8                   37           ROR     AL,1
                            38   ;
                            39   ;
                            40   ;
001A 8D1E0000               41           LEA     BX,LASZLO
                            42   ;         IRP     X,<FOO,3E20H,BAZ>
                            43   ;         LDA     X
                            44   ;         MOV     M,A
                            45   ;         INX     H
                            46   ;         ENDM
001E A04400                 47           MOV     AL,FOO
0021 8807                   48           MOV     M[BX],AL
0023 43                     49           INC     BX
0024 A0203E                 50           MOV     AL,M[3E20H]
0027 8807                   51           MOV     M[BX],AL
0029 43                     52           INC     BX
002A A04500                 53           MOV     AL,BYTE PTR(BAZ)
002D 8807                   54           MOV     M[BX],AL
002F 43                     55           INC     BX
                            56   ;
                            57   ;
                            58   ;
0030 8B1EFFFF               59           MOV     BX,LASZLO-1
                            60   ;MVDATE:  IRPC    X,1978
                            61   ;         INX     H
                            62   ;         MVI     M,X
                            63   ;         ENDM
0034 43                     64           INC     BX
0035 C60701                 65           MOV     M[BX],1
0038 43                     66           INC     BX
0039 C60709                 67           MOV     M[BX],9
003C 43                     68           INC     BX
003D C60707                 69           MOV     M[BX],7
0040 43                     70           INC     BX
0041 C60708                 71           MOV     M[BX],8
                            72   ;
                            73   ;
                            74   ;
0044 08                     75   FOO    DB      8
0045 9900                   76   BAZ    DW      99H
----                        77   ABS_0  ENDS
                            78           END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Figure F-4. MCS-86 Assembler (V1.0) Listing of Converted File (MACROS.L86)

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
QRL86	ERROR 9	ABS_0 HAS INCOMPATIBLE ATTRIBUTES IN *modname* AND *modname*
	ERROR 11	ABS_0 AT 00000H PRECEDES LC= *addr*
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

REQUEST FOR READER'S COMMENTS

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

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of 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 for improvement.

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

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

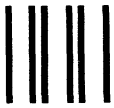
5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. _____

NAME _____ DATE _____
TITLE _____
COMPANY NAME/DEPARTMENT _____
ADDRESS _____
CITY _____ STATE _____ ZIP CODE _____

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS . . .

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



First Class
Permit No. 1040
Santa Clara, CA

BUSINESS REPLY MAIL

No Postage Stamp Necessary if Mailed in U.S.A.

Postage will be paid by:

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Attention: MCD Technical Publications