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

Manual Order No. 9800642A
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
### TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>8080/8086 Flags Correspondence</td>
<td>1-8</td>
</tr>
<tr>
<td>1-2</td>
<td>Flag Settings That Change If APPROX Is Specified</td>
<td>1-11</td>
</tr>
<tr>
<td>1-3</td>
<td>CONV86 Output Files</td>
<td>2-2</td>
</tr>
<tr>
<td>2-1</td>
<td>CONV86 Controls and Defaults</td>
<td>A-1</td>
</tr>
<tr>
<td>A-1</td>
<td>Instruction Mapping</td>
<td>B-1</td>
</tr>
<tr>
<td>B-1</td>
<td>Operand Mapping</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-1</td>
<td>Directives Supported by MCS-86 Assembler (V1.0)</td>
<td>C-1</td>
</tr>
<tr>
<td>C-2</td>
<td>Directives Not Supported by MCS-86 Assembler (V1.0)</td>
<td>C-2</td>
</tr>
<tr>
<td>D-1</td>
<td>Reserved Names</td>
<td>D-1</td>
</tr>
<tr>
<td>G-1</td>
<td>Relocation and Linkage Errors and Warnings</td>
<td>G-1</td>
</tr>
</tbody>
</table>

### FIGURES and LISTINGS

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>From 8080/8085 Assembly Language Source File to 8086 Execution</td>
<td>1-2</td>
</tr>
<tr>
<td>1-2</td>
<td>CONV86 Input and Output Files</td>
<td>1-2</td>
</tr>
<tr>
<td>1-3</td>
<td>Sample PRINT File</td>
<td>1-4</td>
</tr>
<tr>
<td>1-4</td>
<td>Program Listings: Original 8080, Converted 8086, Original 8086</td>
<td>1-5</td>
</tr>
<tr>
<td>3-1</td>
<td>Annotated PRINT File</td>
<td>3-1</td>
</tr>
<tr>
<td>3-2</td>
<td>Converting Your Interrupt Procedures</td>
<td>3-4</td>
</tr>
<tr>
<td>E-1</td>
<td>Program Listing of 8080 Sort Routine</td>
<td>E-2</td>
</tr>
<tr>
<td>E-2</td>
<td>PRINT File of Conversion of 8080 Sort Routine</td>
<td>E-4</td>
</tr>
<tr>
<td>E-3</td>
<td>Program Listing (MCS-86) of Converted 8080 Sort Routine</td>
<td>E-9</td>
</tr>
<tr>
<td>E-4</td>
<td>Program Listing (MCS-86) of Sort Routine</td>
<td>E-11</td>
</tr>
<tr>
<td>F-1</td>
<td>Annotated 8080 Macro Assembler Listing of 8080 Macro Source File</td>
<td>F-2</td>
</tr>
<tr>
<td>F-2</td>
<td>Edited 8080 Macro Assembler Listing</td>
<td>F-3</td>
</tr>
<tr>
<td>F-3</td>
<td>PRINT File from Conversion of Edited 8080 Macro Assembler Listing</td>
<td>F-4</td>
</tr>
<tr>
<td>F-4</td>
<td>MCS-86 Assembler (V1.0) Listing of Converted 8080 Macro Source File</td>
<td>F-6</td>
</tr>
</tbody>
</table>
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.)
Overview of Conversion

1. This routine converts BCD to binary as follows:
   1. BCD unit's digit in low nibble of A and B.
   2. BCD unit's digit in low nibble of C and D.
   3. High nibbles of B and C assumed to be irrelevant.
   4. Binary result (0-99) is left in accumulator.

   MOV A,C  ;UNIT'S DIGIT & GARBAGE TO ACC.
   MOV B,A  ;SAVE UNIT'S DIGIT IN E (LOW)
   MOV A,E  ;UNIT'S DIGIT & GARBAGE TO ACC.
   MOV B,A  ;SAVE UNIT'S DIGIT IN D (LOW)
   ADD B,A  ;ADD TO ACC.
   ADD D,A  ;ADD TO ACC.
   ADD E,A  ;ADD TO ACC.

   Figure 1-3. Sample PRINT File
Overview of Conversion

CONV86: F1:BCDBIN.L86

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

LOC OBJ

1 ; THIS ROUTINE CONVERTS BCD TO BINARY AS FOLLOWS:
2 ; BCD TEN'S DIGIT IN LOW NIBBLE OF A REG.
3 ; BCD UNIT'S DIGIT IN LOW NIBBLE OF B REG.
4 ; 4bN MIDDLE OF B AND C ASSUMED TO BE IRRELEVANT.
5 ; BINARY RESULT (0-99) IS LEFT IN ACCUMULATOR.

4000
5 6 ORG 4000H
6
7 BCDBIN: MOV A,C ; UNIT'S DIGIT & GARBAGE TO ACC
8
9 MOV A,B ; BCD UNIT'S DIGIT IN A (LOW)
10
11 MOV A,C ; BCD UNIT'S DIGIT IN C (LOW)
12
13 MOV D,A ; 2bN TH'S DIGIT IN D (LOW)
14
15 ADD D,A ; 5bN TEN'S
16
17 ADD A,C ; 10bN TEN'S = UNIT'S BIN. REP.
18
19 AND
20
21
22

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
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:

<table>
<thead>
<tr>
<th>8080/8085</th>
<th>8086</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>AL</td>
</tr>
<tr>
<td>B</td>
<td>CH</td>
</tr>
<tr>
<td>C</td>
<td>CL</td>
</tr>
<tr>
<td>D</td>
<td>DH</td>
</tr>
<tr>
<td>E</td>
<td>DL</td>
</tr>
<tr>
<td>H</td>
<td>BH</td>
</tr>
<tr>
<td>L</td>
<td>BL</td>
</tr>
</tbody>
</table>

Word registers are mapped as follows:

<table>
<thead>
<tr>
<th>8080/8085</th>
<th>8086</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSW</td>
<td>AX</td>
</tr>
<tr>
<td>B</td>
<td>CX</td>
</tr>
<tr>
<td>D</td>
<td>DX</td>
</tr>
<tr>
<td>H</td>
<td>BX</td>
</tr>
<tr>
<td>SP</td>
<td>SP</td>
</tr>
</tbody>
</table>
How Are the 8080 FlagsMapped into the 8086 Flags?

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

<table>
<thead>
<tr>
<th>Flag Name</th>
<th>8080 Designation</th>
<th>8086 Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auxiliary-carry</td>
<td>AC</td>
<td>AF</td>
</tr>
<tr>
<td>Carry</td>
<td>C</td>
<td>CF</td>
</tr>
<tr>
<td>Zero</td>
<td>Z</td>
<td>ZF</td>
</tr>
<tr>
<td>Sign</td>
<td>S</td>
<td>SF</td>
</tr>
<tr>
<td>Parity</td>
<td>P</td>
<td>PF</td>
</tr>
</tbody>
</table>

Table 1-1. 8080-8086 Flag Correspondence

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV AL, LASZLO</td>
<td>; load AL with byte at LASZLO</td>
</tr>
<tr>
<td>MOV BX, WORD PTR(LASZLO)</td>
<td>; load BX with word at LASZLO</td>
</tr>
<tr>
<td>JMP NEAR PTR(LASZLO)</td>
<td>; jump to symbolic location LASZLO</td>
</tr>
</tbody>
</table>

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 MOD5 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
Overview of Conversion

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 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.

---

*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

<table>
<thead>
<tr>
<th>Source 8080 Instruction</th>
<th>8080 Flags Affected</th>
<th>Equivalent 8086 Instruction</th>
<th>8086 Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAD</td>
<td>CY</td>
<td>ADD BX, CY</td>
<td>AF, CF, PF, SF, ZF</td>
</tr>
<tr>
<td>INX</td>
<td>none</td>
<td>INC</td>
<td>AF, PF, SF, ZF</td>
</tr>
<tr>
<td>DCX</td>
<td>none</td>
<td>DEC</td>
<td>AF, PF, SF, ZF</td>
</tr>
<tr>
<td>PUSH PSW</td>
<td>none; saved in stack</td>
<td>PUSH AX</td>
<td>none</td>
</tr>
<tr>
<td>POP PSW</td>
<td>Z, S, P, CY, AC</td>
<td>POP AX</td>
<td>[SEE NOTE 1]</td>
</tr>
</tbody>
</table>

[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:

\[
\text{8080 INX B (APPROX) 8086 INC CX}
\]

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:

\[
\text{8080 INX B (EXACT) 8086 LAHF INC CX SAHF (COMMENTS)}
\]

; load flags into AH

; 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.
Overview of Conversion

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>
<thead>
<tr>
<th>File Designation in Invoking Command</th>
<th>Default File-Name</th>
<th>Contents and Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT :Fs:source.A86</td>
<td></td>
<td>Machine-readable 8086 source file; to be manually edited according to caution messages in PRINT file.</td>
</tr>
<tr>
<td>PRINT :Fs:source.LST</td>
<td></td>
<td>1) Copy of 8080/8085 source.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Human-readable 8086 source file with embedded caution messages for manually editing OUTPUT file.</td>
</tr>
</tbody>
</table>

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.
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:

\[ \text{:Fn:CONV86 source controls} \]

where \text{source} is the name of the file to be converted, and \text{controls} are as described in Table 2-1.
Table 2-1. CONV86 Controls and Defaults

<table>
<thead>
<tr>
<th>CONTROLS</th>
<th>DEFAULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINT(path-name) / NOPRINT</td>
<td>PRINT(:Fs:source.LST)</td>
</tr>
<tr>
<td>OUTPUT(path-name) / NOOUTPUT</td>
<td>OUTPUT(:Fs:source.A86)</td>
</tr>
<tr>
<td>DATE('date')</td>
<td>DATE(' ')</td>
</tr>
<tr>
<td>TITLE('title')</td>
<td>TITLE(' ')</td>
</tr>
<tr>
<td>PAGELENGTH(n) / NOPAGING</td>
<td>PAGELENGTH(60)</td>
</tr>
<tr>
<td>PAGEWIDTH(n)</td>
<td>PAGEWIDTH(120)</td>
</tr>
<tr>
<td>EXACT / APPROX</td>
<td>EXACT</td>
</tr>
<tr>
<td>INCLUDED / NOTINCLUDED</td>
<td>NOTINCLUDED</td>
</tr>
<tr>
<td>ABS/REL</td>
<td>REL</td>
</tr>
<tr>
<td>WORKFILES(:Fn:)</td>
<td>WORKFILES(:Fs:)</td>
</tr>
</tbody>
</table>

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 minimum is 60 characters per line; there is no effective maximum.

EXACT

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

APPROX

specifies that full flag-setting semantics are not to be preserved for the instructions DAD, DCX, INX, POP PSW, and PUSH PSW. Refer to Chapter 1, “Functional Equivalence,” for a description of flag preservation.

INCLUDED

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

NOTINCLUDED

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

REL

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

ABS

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

WORKFILES(:Fn:)

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

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

<table>
<thead>
<tr>
<th>8080 Source</th>
<th>MCS-86 (CONV86-Generated) PRINT File</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>MOV D, E</td>
</tr>
<tr>
<td>3</td>
<td>JNR $3+4</td>
</tr>
<tr>
<td>4</td>
<td>LC: MOV C, B</td>
</tr>
<tr>
<td>5</td>
<td>MOV A, L</td>
</tr>
<tr>
<td>7</td>
<td>MOV A, C</td>
</tr>
<tr>
<td>6</td>
<td>JNR $3*((3+2)*2-7)</td>
</tr>
<tr>
<td>9</td>
<td>DB 76h</td>
</tr>
<tr>
<td>10</td>
<td>DE 10111101b</td>
</tr>
<tr>
<td>11</td>
<td>Dw OEAEEAh</td>
</tr>
<tr>
<td>12</td>
<td>Dw OBEACH</td>
</tr>
<tr>
<td>13</td>
<td>CALL 0</td>
</tr>
</tbody>
</table>

| 2          | MOV Dh, Ch                      |
| 3          | JMP SHORT L+1                  |
| 4          | LC: MOV CL, Ch                 |
| 5          | MOV AL, EL                     |
| 7          | MOV AL, CL                     |
| 6          | JNR $3*((3+2)*2-7)              |
| 9          | DB 76h                         |
| 10         | DE 10111101b                   |
| 11         | Dw OEAEEAh                     |
| 12         | Dw OBEACH                      |
| 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 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

<table>
<thead>
<tr>
<th>Before Your Edit</th>
<th>After Your Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV AL, CL</td>
<td>MOV AL, CL</td>
</tr>
<tr>
<td>JNR $3*((3+2)*2-7)</td>
<td>JNR SHORT LAEAEH</td>
</tr>
<tr>
<td>DB 76h</td>
<td>DB 10111101E</td>
</tr>
<tr>
<td>DE 10111101b</td>
<td>Dw OEAEEAh</td>
</tr>
<tr>
<td>Dw OBEACH</td>
<td>Dw OBEACH</td>
</tr>
<tr>
<td>hot AL</td>
<td>LASTZLC: NOT AL</td>
</tr>
</tbody>
</table>

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
   DD CGROUP:INTSEQ ;INTSEQ used here for example above
   DD 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] ;nothing special about the word INTER
   INTER ENDP

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:
     \[
     \text{POPCX} ; \text{Retrieve only PL/M-86 Argument}
     \]
     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:
     \[
     \text{POPDX} \quad \text{POPCX} ; \text{Retrieve Second PL/M-86 Argument} \quad ; \text{Retrieve First PL/M-86 Argument}
     \]
     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:
   \[
   \text{PUSH BX} ; \text{Replace Return Address On Stack}
   \]
   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 CALLeed 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 CALLeed routine, those arguments were passed in the (B,C) register-pair, in the (D,E) register-pair, and remaining arguments on the stack. Hence you need to insert:

```
PUSH CX ;push (B,C) argument on stack
PUSH DX ;push (D,E) argument on stack
```
immediately before the CALL. The remaining arguments (if any) are already on the stack in the correct order. PL/M-86 return values are placed in AX or AL as described in Case 1.
Caution Messages

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

The entire list of caution 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
9. TYPE ASSIGNED TO INCLUDED SYMBOL MAY NOT AGREE WITH DEFINITION
10. TRANSLATION OF NOP MAY NOT YIELD DESIRED RESULTS
11. TRANSLATION OF RST MAY NOT YIELD DESIRED RESULTS
12. 8085-SPECIFIC INSTRUCTION CANNOT BE TRANSLATED
13. FORWARD REFERENCE TO A SYMBOL WHICH IS A REGISTER OR [BX] CANNOT BE CORRECTLY ASSEMBLED
14. EXPRESSION ASSUMED TO BE A VARIABLE OR LABEL
15. ADDRESS EXPRESSION MAY BE INVALID FOR 8086
16. INSTRUCTION AS OPERAND CANNOT BE TRANSLATED
17. REGISTER USED IN UNKNOWN CONTEXT
18. OUTPUT LINE TOO LONG; TRUNCATED
19. LABEL ASSUMED TO BE NEAR
20. NOMACROFILE CONTROL DELETED
21. MOD85 CONTROL DELETED
22. SOURCE LINE TOO LONG; IGNORED
23. CURRENT SEGMENT UNKNOWN; CANNOT GENERATE ENDS
24. THIS SET DIRECTIVE INCOMPATIBLE WITH 8086
25. SYMBOL NAME TOO LONG
26. CONDITIONAL ASSEMBLY GENERATED
27. 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:

<table>
<thead>
<tr>
<th>Before Your Editing</th>
<th>After Your Editing</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW(expr)</td>
<td>LOW DGROUP:(expr')</td>
</tr>
<tr>
<td>HIGH(expr)</td>
<td>HIGH DGROUP:(expr')</td>
</tr>
</tbody>
</table>

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:

\[ X \text{ SET } X + Y \]

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

\[ Z \text{ EQU } X + Y \]
\[ \text{PURGE } X \]
\[ X \text{ EQU } Z \]
\[ \text{PURGE } Z \]

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 desk-check or single-step your source program until you are able to make that determination. Otherwise, the conditional assembly statements placed by CONV86 in your OUTPUT file will not assemble under version 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.
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
- iw = word immediate
- mb = byte memory
- mw = word memory
- mn = near memory
- rb = byte register
- rw = word register

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.

<table>
<thead>
<tr>
<th>8080/8085</th>
<th>8086</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACI ib</td>
<td>ADC AL,ib'</td>
<td></td>
</tr>
<tr>
<td>ADC rb</td>
<td>ADC AL,rb'</td>
<td></td>
</tr>
<tr>
<td>ADD rb</td>
<td>ADD AL,rb'</td>
<td></td>
</tr>
<tr>
<td>ADI ib</td>
<td>ADD AL,ib'</td>
<td></td>
</tr>
<tr>
<td>ANA rb</td>
<td>AND AL,rb'</td>
<td></td>
</tr>
<tr>
<td>ANI rb</td>
<td>AND AL,ib'</td>
<td></td>
</tr>
<tr>
<td>CALL mn</td>
<td>CALL mn'</td>
<td>(L_n inserted as label for instruction following CALL)</td>
</tr>
<tr>
<td>CC mn</td>
<td>JNB SHORT L_n CALL mn'</td>
<td>(L_n inserted as label for instruction following CALL)</td>
</tr>
<tr>
<td>CM mn</td>
<td>JNS SHORT L_n CALL mn'</td>
<td>(L_n inserted as label for instruction following CALL)</td>
</tr>
<tr>
<td>CMA</td>
<td>NOT AL</td>
<td></td>
</tr>
<tr>
<td>CMC</td>
<td>CMC</td>
<td></td>
</tr>
<tr>
<td>CMP rb</td>
<td>CMP AL,rb'</td>
<td></td>
</tr>
<tr>
<td>CNC mn</td>
<td>JNAE SHORT L_n CALL mn'</td>
<td>(L_n inserted as label for instruction following CALL)</td>
</tr>
<tr>
<td>CNZ mn</td>
<td>JZ SHORT L_n CALL mn'</td>
<td>(L_n inserted as label for instruction following CALL)</td>
</tr>
<tr>
<td>CP mn</td>
<td>JS SHORT L_n CALL mn'</td>
<td>(L_n inserted as label for instruction following CALL)</td>
</tr>
<tr>
<td>CPE mn</td>
<td>JNP SHORT L_n CALL mn'</td>
<td>(L_n inserted as label for instruction following CALL)</td>
</tr>
<tr>
<td>CPI ib</td>
<td>CMP AL,ib'</td>
<td></td>
</tr>
<tr>
<td>CPO mn</td>
<td>JP SHORT L_n CALL mn'</td>
<td>(L_n inserted as label for instruction following CALL)</td>
</tr>
<tr>
<td>CZ mn</td>
<td>JNZ SHORT L_n CALL mn'</td>
<td>(L_n inserted as label for instruction following CALL)</td>
</tr>
<tr>
<td>8080/8085</td>
<td>8086</td>
<td>Remarks</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>DAA</td>
<td>DAA</td>
<td></td>
</tr>
<tr>
<td>DAD rw</td>
<td>ADD BX, rw'</td>
<td>(Using APPROX Control)</td>
</tr>
<tr>
<td>DAD rw</td>
<td>LAHF</td>
<td>(Using EXACT Control)</td>
</tr>
<tr>
<td></td>
<td>ADD BX, rw'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RCR SI, 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAHF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RCL SI, 1</td>
<td></td>
</tr>
<tr>
<td>DCR rb</td>
<td>DEC rb'</td>
<td></td>
</tr>
<tr>
<td>DCX rw</td>
<td>DEC rw'</td>
<td>(Using APPROX Control)</td>
</tr>
<tr>
<td>DCX rw</td>
<td>LAHF</td>
<td>(Using EXACT Control)</td>
</tr>
<tr>
<td></td>
<td>DEC rw'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAHF</td>
<td></td>
</tr>
<tr>
<td>DI</td>
<td>CLI</td>
<td></td>
</tr>
<tr>
<td>EI</td>
<td>STI</td>
<td></td>
</tr>
<tr>
<td>HLT</td>
<td>HLT</td>
<td></td>
</tr>
<tr>
<td>IN ib</td>
<td>IN AL, ib'</td>
<td></td>
</tr>
<tr>
<td>INR rb</td>
<td>INC rb'</td>
<td></td>
</tr>
<tr>
<td>INX rw</td>
<td>INC rw'</td>
<td>(Using APPROX Control)</td>
</tr>
<tr>
<td>INX rw</td>
<td>LAHF</td>
<td>(Using EXACT Control)</td>
</tr>
<tr>
<td></td>
<td>INC rw'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAHF</td>
<td></td>
</tr>
<tr>
<td>8080/8085</td>
<td>8086</td>
<td>Remarks</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>JC mn</td>
<td>JB SHORT mn'</td>
<td>(for forward short branch)</td>
</tr>
<tr>
<td>JC mn</td>
<td>JB mn'</td>
<td>(for backward short branch)</td>
</tr>
<tr>
<td>JC mn</td>
<td>JAE SHORT L_n JMP mn'</td>
<td>(otherwise)</td>
</tr>
<tr>
<td>JM mn</td>
<td>JS SHORT mn'</td>
<td>(for forward short branch)</td>
</tr>
<tr>
<td>JM mn</td>
<td>JS mn'</td>
<td>(for backward short branch)</td>
</tr>
<tr>
<td>JM mn</td>
<td>JNS SHORT L_n JMP mn'</td>
<td>(otherwise)</td>
</tr>
<tr>
<td>JMP mn</td>
<td>JMP SHORT mn'</td>
<td>(for forward short branch)</td>
</tr>
<tr>
<td>JMP mn</td>
<td>JMP mn'</td>
<td>(otherwise)</td>
</tr>
<tr>
<td>JNC mn</td>
<td>JAE SHORT mn'</td>
<td>(for forward short branch)</td>
</tr>
<tr>
<td>JNC mn</td>
<td>JAE mn'</td>
<td>(for backward short branch)</td>
</tr>
<tr>
<td>JNC mn</td>
<td>JNAE SHORT L_n JMP mn'</td>
<td>(otherwise)</td>
</tr>
<tr>
<td>JNZ mn</td>
<td>JNZ SHORT mn'</td>
<td>(for forward short branch)</td>
</tr>
<tr>
<td>JNZ mn</td>
<td>JNZ mn'</td>
<td>(for backward short branch)</td>
</tr>
<tr>
<td>JNZ mn</td>
<td>JZ SHORT L_n JMP mn'</td>
<td>(otherwise)</td>
</tr>
<tr>
<td>JP mn</td>
<td>JNS SHORT mn'</td>
<td>(for forward short branch)</td>
</tr>
<tr>
<td>JP mn</td>
<td>JNS mn'</td>
<td>(for backward short branch)</td>
</tr>
<tr>
<td>JP mn</td>
<td>JS SHORT L_n JMP mn'</td>
<td>(otherwise)</td>
</tr>
<tr>
<td>JPE mn</td>
<td>JP SHORT mn'</td>
<td>(for forward short branch)</td>
</tr>
<tr>
<td>JPE mn</td>
<td>JP mn'</td>
<td>(for backward short branch)</td>
</tr>
<tr>
<td>JPE mn</td>
<td>JNP SHORT L_n JMP mn'</td>
<td>(otherwise)</td>
</tr>
<tr>
<td>JPO mn</td>
<td>JNP SHORT mn'</td>
<td>(for forward short branch)</td>
</tr>
<tr>
<td>JPO mn</td>
<td>JNP mn'</td>
<td>(for backward short branch)</td>
</tr>
<tr>
<td>JPO mn</td>
<td>JP SHORT L_n JMP mn'</td>
<td>(otherwise)</td>
</tr>
<tr>
<td>JZ mn</td>
<td>JZ SHORT mn'</td>
<td>(for forward short branch)</td>
</tr>
<tr>
<td>JZ mn</td>
<td>JZ mn'</td>
<td>(for backward short branch)</td>
</tr>
<tr>
<td>JZ mn</td>
<td>JNZ SHORT L_n JMP mn'</td>
<td>(otherwise)</td>
</tr>
<tr>
<td>Instruction Mapping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>8080/8085</strong></td>
<td><strong>8086</strong></td>
<td><strong>Remarks</strong></td>
</tr>
<tr>
<td>LDA mb</td>
<td>MOV AL, mb'</td>
<td></td>
</tr>
<tr>
<td>LDAX rw</td>
<td>MOV SI, rw'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LODS DS:[SI]</td>
<td></td>
</tr>
<tr>
<td>LHLD mw</td>
<td>MOV BX, mw'</td>
<td></td>
</tr>
<tr>
<td>LXI rw, iw</td>
<td>MOV rw', iw'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(when 2nd operand immed. or near)</td>
<td></td>
</tr>
<tr>
<td>LXI rw, iw</td>
<td>LEA rw', iw'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(when 2nd operand is byte or word)</td>
<td></td>
</tr>
<tr>
<td>MOV rb1, rb2</td>
<td>MOV rb1', rb2'</td>
<td></td>
</tr>
<tr>
<td>MOV M, rb</td>
<td>MOV M[BX], rb'</td>
<td></td>
</tr>
<tr>
<td>MVI rb, ib</td>
<td>MOV rb', ib'</td>
<td></td>
</tr>
<tr>
<td>MVI M, ib</td>
<td>MOV M[BX], ib'</td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>NOP</td>
<td>XCHG AX, AX (1 byte, 3 clocks)</td>
</tr>
<tr>
<td>ORA rb</td>
<td>OR AL, rb'</td>
<td></td>
</tr>
<tr>
<td>ORI ib</td>
<td>OR AL, ib'</td>
<td></td>
</tr>
<tr>
<td>OUT ib</td>
<td>OUT ib', AL</td>
<td></td>
</tr>
<tr>
<td>PCHL</td>
<td>JMP BX</td>
<td></td>
</tr>
<tr>
<td>POP rw</td>
<td>POP rw'</td>
<td>(for EXACT or APPROX when rw is definitely not PSW)</td>
</tr>
<tr>
<td>POP PSW</td>
<td>POP AX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XCHG AL, AH</td>
<td>(Using APPROX Control)</td>
</tr>
<tr>
<td>POP PSW</td>
<td>POP AX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XCHG AL, AH</td>
<td>(Using EXACT Control)</td>
</tr>
<tr>
<td></td>
<td>SAHF</td>
<td></td>
</tr>
<tr>
<td>POP rw</td>
<td>IF rw' EQ AX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP rw'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XCHG AL, AH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ELSE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP rw'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENDIF</td>
<td></td>
</tr>
<tr>
<td>POP rw</td>
<td>IF rw' EQ AX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP rw'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XCHG AL, AH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAHF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ELSE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>POP rw'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENDIF</td>
<td></td>
</tr>
<tr>
<td>8080/8085</td>
<td>8086</td>
<td>Remarks</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>PUSH rw</td>
<td>PUSH rw'</td>
<td>(for EXACT or APPROX when rw is definitely not PSW)</td>
</tr>
<tr>
<td>PUSH PSW</td>
<td>LAHF XCHG AL, AH PUSH AX XCHG AL, AH</td>
<td>(Using EXACT Control)</td>
</tr>
<tr>
<td>PUSH PSW</td>
<td>XCHG AL, AH PUSH AX XCHG AL, AH</td>
<td>(Using APPROX Control)</td>
</tr>
<tr>
<td>PUSH rw</td>
<td>IF rw' EQ AX XCHG AL, AH PUSH rw' XCHG AL, AH ELSE PUSH rw' ENDIF</td>
<td>(Using APPROX Control when rw could be PSW)</td>
</tr>
<tr>
<td>RAL</td>
<td>RCL AL,1</td>
<td></td>
</tr>
<tr>
<td>RAR</td>
<td>RCR AL,1</td>
<td></td>
</tr>
<tr>
<td>RC</td>
<td>JNB SHORT L_n RET</td>
<td>(L_n inserted as label for instruction following RET)</td>
</tr>
<tr>
<td>RET</td>
<td>RET</td>
<td></td>
</tr>
<tr>
<td>RIM</td>
<td><em><strong>error</strong></em></td>
<td></td>
</tr>
<tr>
<td>RLC</td>
<td>ROL AL,1</td>
<td></td>
</tr>
<tr>
<td>RM</td>
<td>JNS SHORT L_n RET</td>
<td>(L_n inserted as label for instruction following RET)</td>
</tr>
<tr>
<td>RNC</td>
<td>JNAE SHORT L_n RET</td>
<td>(L_n inserted as label for instruction following RET)</td>
</tr>
<tr>
<td>RNZ</td>
<td>JZ SHORT L_n RET</td>
<td>(L_n inserted as label for instruction following RET)</td>
</tr>
<tr>
<td>RP</td>
<td>JS SHORT L_n RET</td>
<td>(L_n inserted as label for instruction following RET)</td>
</tr>
<tr>
<td>RPE</td>
<td>JNP SHORT L_n RET</td>
<td>(L_n inserted as label for instruction following RET)</td>
</tr>
<tr>
<td>RPO</td>
<td>JP SHORT L_n RET</td>
<td>(L_n inserted as label for instruction following RET)</td>
</tr>
<tr>
<td>RRC</td>
<td>ROR AL,1</td>
<td></td>
</tr>
<tr>
<td>RST ib</td>
<td>INT ib'</td>
<td></td>
</tr>
<tr>
<td>RZ</td>
<td>JNZ SHORT L_n RET</td>
<td>(L_n inserted as label for instruction following RET)</td>
</tr>
<tr>
<td>8080/8085</td>
<td>8086</td>
<td>Remarks</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>SBB rb</td>
<td>SBB AL,rb'</td>
<td></td>
</tr>
<tr>
<td>SBI ib</td>
<td>SBB AL,ib'</td>
<td></td>
</tr>
<tr>
<td>SHLD mw</td>
<td>MOV mw',BX</td>
<td></td>
</tr>
<tr>
<td>SIM</td>
<td><em><strong>error</strong></em></td>
<td></td>
</tr>
<tr>
<td>SPHL</td>
<td>MOV SP,BX</td>
<td></td>
</tr>
<tr>
<td>STA mb</td>
<td>MOV mb',AL</td>
<td></td>
</tr>
<tr>
<td>STAX rw</td>
<td>MOV DI,rw' MOV DS:[DI],AL</td>
<td></td>
</tr>
<tr>
<td>STC</td>
<td>STC</td>
<td></td>
</tr>
<tr>
<td>SUB rb</td>
<td>SUB AL,rb'</td>
<td></td>
</tr>
<tr>
<td>SUI ib</td>
<td>SUB AL,ib'</td>
<td></td>
</tr>
<tr>
<td>XCHG</td>
<td>XCHG BX,DX</td>
<td></td>
</tr>
<tr>
<td>XRA rb</td>
<td>XOR AL,rb'</td>
<td></td>
</tr>
<tr>
<td>XRI ib</td>
<td>XOR AL,ib'</td>
<td></td>
</tr>
<tr>
<td>XTHL</td>
<td>POP SI XCHG BX,SI PUSH SI</td>
<td></td>
</tr>
<tr>
<td>unknown expr</td>
<td>unknown' expr'</td>
<td></td>
</tr>
</tbody>
</table>
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 CONVS6 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:
     - 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:
     - 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.
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

<table>
<thead>
<tr>
<th>8080/8085</th>
<th>8086</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASEG</td>
<td>prev-seg ENDS</td>
</tr>
<tr>
<td></td>
<td>ABS_0  SEGMENT BYTE AT 0</td>
</tr>
<tr>
<td>CSEG</td>
<td>prev-seg ENDS</td>
</tr>
<tr>
<td></td>
<td>CODE  SEGMENT WORD PUBLIC 'CODE'</td>
</tr>
<tr>
<td>DB expr-list</td>
<td>DB expr-list'</td>
</tr>
<tr>
<td>DS expr</td>
<td>DB expr' DUP(?)</td>
</tr>
<tr>
<td>DSEG</td>
<td>prev-seg ENDS</td>
</tr>
<tr>
<td></td>
<td>DATA  SEGMENT WORD PUBLIC 'DATA'</td>
</tr>
<tr>
<td>DW expr-list</td>
<td>DW expr-list'</td>
</tr>
<tr>
<td>END [mn]</td>
<td>prev-seg ENDS</td>
</tr>
<tr>
<td></td>
<td>END [mn']</td>
</tr>
<tr>
<td>name EQU expr</td>
<td>name'  EQU expr'</td>
</tr>
<tr>
<td>EXTRN name-list</td>
<td>EXTRN name:usage-list'</td>
</tr>
<tr>
<td>NAME name</td>
<td>NAME name'</td>
</tr>
<tr>
<td>ORG mn</td>
<td>ORG mn'</td>
</tr>
<tr>
<td>PUBLIC name-list</td>
<td>PUBLIC name-list'</td>
</tr>
<tr>
<td>STKLN expr</td>
<td><em><strong>deleted</strong></em></td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>8080/8085</th>
<th>8086</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELSE</td>
<td>ELSE</td>
</tr>
<tr>
<td>ENDF</td>
<td>ENDF</td>
</tr>
<tr>
<td>ENDM</td>
<td>ENDM</td>
</tr>
<tr>
<td>EXITM</td>
<td>EXITM</td>
</tr>
<tr>
<td>IF Ib</td>
<td>IF Ib'</td>
</tr>
<tr>
<td>IRP parm,&lt;list&gt;</td>
<td>IRP parm',&lt;list&gt;</td>
</tr>
<tr>
<td>IRPC parm,string</td>
<td>IRPC parm',string</td>
</tr>
<tr>
<td>LOCAL name-list</td>
<td>LOCAL name-list'</td>
</tr>
<tr>
<td>name MACRO parm-list</td>
<td>name' MACRO parm-list'</td>
</tr>
<tr>
<td>macro-call arg-list</td>
<td>macro-call' arg-list'</td>
</tr>
<tr>
<td>REPT expr</td>
<td>REPT expr'</td>
</tr>
<tr>
<td>name SET constant-expr</td>
<td>name' SET constant-expr'</td>
</tr>
<tr>
<td>name SET nonconstant-expr</td>
<td>PURGE name' name' EQU nonconstant-expr'</td>
</tr>
</tbody>
</table>
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 PREFIX SEGFIX
ASSUME DIV JBE LEA PROC SEGMENT
AT DL JCZ LENGTH PROCLEN SHORT
AX DUP JE LES PIR SI
BH DWORD JG LOCK PURGE SIZE
BL DX JGE LDS PUSHF SS
BP ELSE JL LOOP RCL STD
BX ELSEIF JLE LOOPE RCR STI
BYTE ENDM JNA LOOPNE RECORD STOS
CBW ENDM JNAE LOOPNZ RELB STRUC
CH ENDP JNR LOOPZ RELW TEST
CL ENDS JNBE MASK REP THIS
CLC ESC 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.
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.
**ISIS-II HObO/bOb5 MACRO ASSEMBLER, 2.0**

**Sample Conversion and Listings**

**CONV86**

**LOC Obj SW**

**SOURCE STATEMENTS**

1:******************************************************************************
2: A PL/cpu 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 A. 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: 
20: PUBLIC SORT
21: TEST = address of the last element of A1.

| 0000 | KB | 22 SORT: XCHG ; TEST = (n - 1) * 2 + .A1 |
| 0001 | 5b | 23 MOV E,M |
| 0002 | 23 | 24 INX H |
| 0003 | 96 | 25 MOV D,M |
| 0004 | Eb | 26 XCHG ;(W |
| 0005 | 2b | 27 LCX H ; - 1 |
| 0006 | 29 | 28 LAD H ; |
| 0007 | 09 | 29 DAD B ; + .A1 |
| 0008 | 220000 | D 30 SMLD TEST ; |

31: OUTER LOOP: DO DE = .A1 TO TEST by 2;
32: |
33: |
34: |
35: 3A0000 | D 36 OUTST: LDA TEST ; IF DE > TEST THEN RETURN |
36: 0010 | 93 | 37 SUB E |
37: 0011 | 3A0100 | D 38 LDA TEST + 1 |
38: 0014 | 9a | 39 Sbb D |
39: 0015 | D8 | 40 RC |
40: 41: INHER LOOP: DO HL = DE+2 TO TEST by 2 |
41: |
42: |
43: 0016 | 6b | 43 MOV L,H |
44: 0017 | 62 | 44 MOV H,D |
45: 0018 | 23 | 45 INX h |
46: 0019 | 23 | 46 INX h ; HL = DE+2 |
47: 48: IF HL > TEST THEN GOTO OUTINC |
48: |
49: |
50: A0000 | D 51 INTST: LDA TEST |
51: 001a | 95 | 52 SUB L |
52: 001b | 3A0010 | D 53 LDA TEST + 1 |
53: 0021 | 9c | 54 Sbb H |
54: 0022 | DA4300 | C 55 JC OUTINC |
56: 57: As a side effect, HL and DE are incremented by 1 |
57: 58: to point to the high bytes of their array elements. |
59: |
60: 0025 | 1a | 60 LDA X |
61: 0026 | 96 | 61 SUB N |
62: 0027 | 13 | 62 INX D |
63: 0028 | 23 | 63 INX h |
64: 0029 | 1a | 64 LDA D |
65: 002a | 9b | 65 Sbb M |
66: 002b | D23E00 | C 66 JNC ININC |
67: 67: Exchange A(DE) with A(HL). Leave HL and DE |
68: 68: pointing to high bytes. |
69: 69: MOV C,M |

**Figure E-1A**
ISIS-II 80b0/80b5 MACH5 ASSEMBLER, V2.0

MODULE

LOC OBJ SEG SOURCE STATEMENT

0030 77 70 MOV M,A
0031 78 71 XCHG
0032 71 72 MOV M,C
0033 EB 73 XCHG

0034 1b 75 DCA D ; POINT HL AND DE TO LOW BYTES.
0035 7b 76 DCX H

0036 1a 76 LDAA D ; SWAP LOW BYTES
0037 47 79 MOV C,A
0038 77 60 MOV M,A
0039 6b 81 XCHG
003a 71 82 MOV M,C
003b BB 83 XCHG

003c 13 85 INX D ; POINT HL AND DE TO HIGH BYTES.
003d 23 86 INX H
003e 2b ; D and HL point to high bytes. For the next iteration,
003f 33 ; set DE = Previous DE, HL = 2 + Previous HL.
0040 1a 50 INHC: DCA D
0041 23 91 INX H

0042 31A00 C E2 JMP INTST

0043 13 95 OUTNC: INX D
0044 13 96 INX D
0045 C30000 C 97 JMP OUTTST

100 ; Data area follows.
101 DSEG
102 TEST: DS 2
103 END

PUBLIC SYMBOLS
SORT  C 0000

EXTERNAL SYMBOLS

USER SYMBOLS

ISIS-II 80b0/80b5 MACH5 ASSEMBLER, V2.0

MODULE PAGE 3

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

ASSEMBLY COMPLETE, NO ERRORS

Figure E-1B
ASM60 TO ASM66 CONVERTER

**ISIS-II** ASM60 TO ASM66 CONVERSION OF FILE :F1:SORT60.A60
ASM66 PLACED IN :F1:SORT60.A66
CONVERTER V1.0 INVOKED by:
CONV66 :F1:SORT60.A60 PRINT1 :F1:SORT60.CVL)

---

1 ;******************************************************************************
2 ; A PL/8 callable subroutine:
3 ; CALL Sort(.A1,.h)
4 ; Sorts the array .A1, containing h words.
5 ; At entry BC points to the array .A1, and
6 ; De points to h. 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 CSEG
19 PUBLIC SORT
20 ; TEST = address of the last element of .A1.
21 SORT: XCHG ; TEST = (N - 1) * 2 + .A1
22 MOV b,h
23 INX h
24 MOV D,M
25 XCHG
26 DGX h ; (h)
27 BAD h ; + 1
28 BAD B ; + .A1
29 SHLD TEST ; = TEST
30 ; OUTER LOOP: DO DE = .A1 TO TEST BY 2;
31 MOV E,C ; BC CONTAINS .A1
32 MOV D,B
33 OUTST: LDA TEST ; IF DE > TEST THEN RETURN
34 SUB E
35 LDA TEST + 1
36 SBB D
37 RC
38 ; INNER LOOP: DO HL = DE+2 TO TEST BY 2
39 MOV L,E
40 MOV h,D
41 INX h
42 INX h ; HL = DE+2
43 ; IF HL > TEST THEN GOTO OUT1AC
44 INTST: LDA TEST
45 SUB L
46 LDA TEST + 1
47 SBB H
48 JC OUT1AC
49 ; IF A(HL) < A(DE) THEN GOTO IN1AC
50 ; As a side effect, hl and De are incremented by 1
51 ; to point to the high bytes of their array elements.
52 LDAX D
53 SUB H
54 INX D
55 INX h
56 LDAX D
57 SBB M
58 JNC IN1AC
59 ; Exchange A(DE) with A(HL). Leave HL and DE
60 ; pointing to high bytes.
61 LDAX D ; SWAP HIGH BYTES

---

Figure E-2A
ASH80 TO ASM86 CONVERTER

69  MOV  C,M
70  MOV  M,A
71  XCHG
72  MOV  M,C
73  XCHG
74  DCX  D    ; POINT HL AND DE TO LOW BYTES.
75  DCX  h
76  LDA  D    ; SWAP LOW BYTES
77  MOV  C,h
78  MOV  M,A
79  XCHG
80  MOV  M,C
81  XCHG
82  INC  D    ; POINT HL AND DE TO HIGH BYTES.
83  INX  h
84  ; DE and HL point to HIGH bytes. For the next iteration,
85  ; set DE = Previous DE, HL = 2 + Previous HL.
86  INC:  DCX  D
87  INX  h
88  JMP  INC
89  ; End of outer loop. Set DE = DE + 2 and CONTINUE
90  OUTINC:  INX  D
91  INX  D
92  JMP  OUTINC
93  ; Data area follows.
94  DSEG
95  TEST:  DS  2
96  END
ASM60 TO ASHb6 CONVERTeR

- CGROUP GROUP AB S,DATA,STACK,MEMORY
- DGROUP GROUP AB S,DATA,STACK,MEMORY
- SEGMENT word public 'CGROUP'
- SEGMENT word public 'DGROUP'
- CONST word STACK 'STACK'
- STACK BASE LABEL BYTE
- STACK WORD STACK 'STACK'
- MEMORY SEGMENT word MEMORY 'MEMORY'
- MEMORY LABEL BYTE
- MEMORY WORD MEMORY 'MEMORY'
- MEMORY SEGMENT BYTE AT 0
- LABEL BYTE

1; -------------------------------
2; A PL/8 callabie subroutine:
3; CALL SORT(.A1, .DE)
4; Sorts the array A1, containing N words.
5; At entry BC points to the array A1, and
6; DE points to A1. Two pointers to elements of A1 are
7; kept in the BL and DL registers. These pointers are
8; incremented in two loops. The outer loop steps BL
9; through the elements of A1. The inner loop steps
10; BL through the elements of A1 that follow DL. At
11; each step of the inner loop, the items at BL and DE
12; are exchanged, if required, so that at the end of
13; the inner loop, the item at BL is larger than all
14; the items that follow it. The item at BL is then in
15; its proper position, so DE is incremented to
16; complete one iteration of the outer loop.
17; -------------------------------

18; AB S ENDS
19; CODE SEGMENT word public 'CODE'
20; PUBLIC SORT
21; TEST = address of the last element of A1.
22; SORT: XCHG BA,DA ; TEST = (n - 1) * 2 + .A1
23; MOV DL,[DL]
24; LAHF
25; IAC BA
26; SAHF
27; MOV DL,[DL]
28; XCHG BA,DA ; (n
29; LAHF
30; DEC BA
31; SAHF
32; LAHF
33; ADD BA,BA
34; RCR SI,1
35; SAHF
36; RCL SI,1 ; * 2
37; LAHF
38; ADD BX,CX
39; RCR SI,1
40; SAHF
41; RCL SI,1 ; + .A1
42; MOV WORD PIR(7),BX ; = TEST
43;
44; OUTER LOOP: DO DL = .A1 TO TEST BY 2;
45; MOV DL,CL ; BC CONTAINS .A1
46; MOV DL,CR
47; OUTTEST: MOV AL,TEST_ ; IF DL > TEST THEN RETURN
48; SUB AL,DL
49; MOV AL,TEST_+1
50; JNB SHORT L_1
51; RET
52; L_1:
53; INNER LOOP: DO DL = .DE+2 TO TEST BY 2
54; MOV DL,CL
55; MOV DL,CR
56; LAHF
57; IAC BL

Figure E-2C
Sample Conversion and Listings

ASM60 to ASH66 CONVERTER

45           SAHF
46           LAHF
47           INC   bx                  ; HL = D8 + 2
48           ; IF HL > TEST Then GOTO OUTINC
49       INIT:    MOV   AL, TEST_
50           SUB    AL, BL
51           MOV    AL, TEST_+1
52           JBE    AL, bh
53           Jb    SHORT GUTINC
54           ; IF A1(hL) < A1(DE) Then GOTO ININC
55           ; As a side effect, HL and BL are incremented by 1
57           ; to point to the high bytes of their array elements.
58           MOV    SI, DL
59           LODS    DS:[SI]
60           SUB    AL, [BL]
61           LAHF
62           INC    DL
63           SAHF
64           INc   bx
65           SAHF
66           MOV    SI, DX
67           LODS    DS:[SI]                  ; SWAP HIGH BYTES
68           MOV    CL, [BL]
69           MOV    BL, AL
70           XCHG   DL, DX
71           MOV    M[BL], CL
72           XCHG   DL, DX
73           ; Exchange A(DE) with A(hL). Leave HL and DE
74           ; pointing to high bytes.
75           MOV    SI, DL
76           LODS    DS:[SI]                  ; SWAP HIGH BYTES
77           MOV    CL, [BL]
78           MOV    BL, AL
79           XCHG   DL, DX
80           MOV    h[BL], CL
81           XCHG   DL, DX
82           ; Exchange A(DE) with A(hL). Leave HL and DE pointing to low bytes.
83           MOV    SI, DL
84           LODS    DS:[SI]
85           INC    DL
86           SAHF
87           INC   bx
88           SAHF
89           INC   bx                  ; POINT HL AND DE TO HIGH BYTES.
90           LAHF
91           INC    DL
92           SAHF
93           INC   bx
94           ; DE and HL point to high bytes. For the next iteration,
95           ; set DE = previous DE, HL = 2 + previous HL.
96           ININC:  LAHF
97           DEC    DX
98           SAHF
99           LAHF
100          INC   bx
101          SAHF
102          INc   bx
103          Jmp    INIT
104          ; End of outer loop. Set DE = DX + 2 and CONTINUE
105          OUTINC:  LAHF
106          INC    DX

Figure E-2D
ASH80 TO ASM86 CONVERTER

95       SAHF
96       LAHF
96       INC DL
97       SAHF
97       JHF OUTTEST
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 ASH80 TO ASM86 CONVERSION

Figure E-2E
OUTER LOOP: DO DB = .A1 TO TEST BY 2;  

OUTIST: MOV AL,TEST_1;  

OUTER LOOP: DO HL = DB+2 TO TEST BY 2;
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>Lihe</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0031</td>
<td>A00000</td>
<td>69</td>
<td>MOV AL,TEST_</td>
</tr>
<tr>
<td>0034</td>
<td>2AC3</td>
<td>70</td>
<td>SUB AL,HL</td>
</tr>
<tr>
<td>0036</td>
<td>A00100</td>
<td>71</td>
<td>MOV AL,TEST_+1</td>
</tr>
<tr>
<td>0039</td>
<td>1AC7</td>
<td>72</td>
<td>SBB AL,DE</td>
</tr>
<tr>
<td>003B</td>
<td>T242</td>
<td>73</td>
<td>JB SHORT GOUTINC</td>
</tr>
<tr>
<td>003D</td>
<td>6FF2</td>
<td>76</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>003F</td>
<td>AC</td>
<td>79</td>
<td>LODS DS:SI[SI]</td>
</tr>
<tr>
<td>0040</td>
<td>CAB70000</td>
<td>80</td>
<td>SUB AL,ALIBX</td>
</tr>
<tr>
<td>0044</td>
<td>6F</td>
<td>81</td>
<td>LAHF ; THE UNNECESSARY 'EXACT'</td>
</tr>
<tr>
<td>0047</td>
<td>9A</td>
<td>85</td>
<td>SARF ; THIS 'EXACT' CODE IS ALSO NEEDED</td>
</tr>
<tr>
<td>004B</td>
<td>6F62</td>
<td>86</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>004A</td>
<td>AC</td>
<td>87</td>
<td>LODS DS:SI[SI]</td>
</tr>
<tr>
<td>004E</td>
<td>7670000</td>
<td>88</td>
<td>SBB AL,ALIBX</td>
</tr>
<tr>
<td>004F</td>
<td>732A</td>
<td>89</td>
<td>JAE SHORT GOUTINC</td>
</tr>
<tr>
<td>0051</td>
<td>6BF2</td>
<td>93</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0053</td>
<td>AC</td>
<td>94</td>
<td>LODS DS:SI[SI]</td>
</tr>
<tr>
<td>0054</td>
<td>6AF0000</td>
<td>95</td>
<td>MOV CL,ALIBX</td>
</tr>
<tr>
<td>0056</td>
<td>6BF7000</td>
<td>96</td>
<td>MOV CL,ALIBX</td>
</tr>
<tr>
<td>0057</td>
<td>87BA</td>
<td>97</td>
<td>XCHG bx,DX</td>
</tr>
<tr>
<td>0059</td>
<td>8BF0000</td>
<td>98</td>
<td>MOV CL,ALIBX</td>
</tr>
<tr>
<td>0062</td>
<td>87DA</td>
<td>99</td>
<td>XCHG bx,DX</td>
</tr>
<tr>
<td>0064</td>
<td>4A</td>
<td>100</td>
<td>LAC DX ; POINT BL AND DE TO LOW BYTES.</td>
</tr>
<tr>
<td>0065</td>
<td>4B</td>
<td>101</td>
<td>LAC DX ; POINT BL AND DE TO LOW BYTES.</td>
</tr>
<tr>
<td>0066</td>
<td>BFF2</td>
<td>103</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0068</td>
<td>AC</td>
<td>104</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0069</td>
<td>6BF0000</td>
<td>105</td>
<td>LODS DS:SI[SI]</td>
</tr>
<tr>
<td>006B</td>
<td>6BF7000</td>
<td>106</td>
<td>MOV CL,ALIBX</td>
</tr>
<tr>
<td>0071</td>
<td>87DA</td>
<td>107</td>
<td>MOV CL,ALIBX</td>
</tr>
<tr>
<td>0073</td>
<td>8BF0000</td>
<td>108</td>
<td>MOV CL,ALIBX</td>
</tr>
<tr>
<td>0077</td>
<td>87DA</td>
<td>109</td>
<td>XCHG bx,DX</td>
</tr>
<tr>
<td>0079</td>
<td>4A</td>
<td>110</td>
<td>XCHG bx,DX</td>
</tr>
<tr>
<td>007A</td>
<td>43</td>
<td>111</td>
<td>XCHG bx,DX</td>
</tr>
<tr>
<td>007B</td>
<td>4A</td>
<td>112</td>
<td>XCHG bx,DX</td>
</tr>
<tr>
<td>007C</td>
<td>43</td>
<td>113</td>
<td>XCHG bx,DX</td>
</tr>
<tr>
<td>007D</td>
<td>EBF2</td>
<td>114</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>007F</td>
<td>42</td>
<td>115</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0080</td>
<td>42</td>
<td>116</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0081</td>
<td>6BF9</td>
<td>117</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0082</td>
<td>42</td>
<td>118</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0083</td>
<td>42</td>
<td>119</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0084</td>
<td>42</td>
<td>120</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0085</td>
<td>42</td>
<td>121</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0086</td>
<td>42</td>
<td>122</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0087</td>
<td>42</td>
<td>123</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0088</td>
<td>42</td>
<td>124</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0089</td>
<td>42</td>
<td>125</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>008A</td>
<td>42</td>
<td>126</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>008B</td>
<td>42</td>
<td>127</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>008C</td>
<td>42</td>
<td>128</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>008D</td>
<td>42</td>
<td>129</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>008E</td>
<td>42</td>
<td>130</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>008F</td>
<td>42</td>
<td>131</td>
<td>MOV SI,DX</td>
</tr>
<tr>
<td>0090</td>
<td>42</td>
<td>132</td>
<td>MOV SI,DX</td>
</tr>
</tbody>
</table>

ASSEMBL COMPLETE, NO ERRORS FOUND.

Figure E-3B
Figure E-4
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 CONVS6 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)
Converting Macros

Figure F-1. Annotated 8080 Macro Assembler Listing (MACROS.L80)
Figure F-2. Edited 8080 Macro Assembler Listing (MACROS.E80)
Figure F-3A. Conversion of Edited Macro File (8080 Source Shown)
Figure F-3B. Conversion of Edited 8080 Macro File (MCS-86 Source Shown)
Converting Macros

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

<table>
<thead>
<tr>
<th>R &amp; L Tool</th>
<th>Message ID</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>QRL86</td>
<td>ERROR 9</td>
<td>ABS_0 HAS INCOMPATIBLE ATTRIBUTES IN modname AND modname</td>
</tr>
<tr>
<td></td>
<td>ERROR 11</td>
<td>ABS_0 AT 00000H PRECEDES LC= addr.</td>
</tr>
<tr>
<td>MCS-86 LINKER</td>
<td>WARNING 14</td>
<td>GROUP ENLARGED FILE: filename GROUP: groupname MODULE: modname *</td>
</tr>
<tr>
<td></td>
<td>WARNING 28</td>
<td>POSSIBLE OVERLAP FILE: filename MODULE: modname SEGMENT: ABS_0 CLASS:</td>
</tr>
</tbody>
</table>
ABS control (CONV86), 1-6, 2-3
absolute address, 3-2
APPROX control (CONV86), 1-10, 2-3
cautions message, 1-12, 3-8
comments, mapping of, 1-9
conditional assembler directives, 1-3, F-1
conditional assembly, 1-3
continuation lines,
in CONV86 command, 2-5
in PRINT file, 3-1
controls (ASM80) mapping, C-1
controls (CONV86), 2-2
conversions, sample, 1-3, 3-1, E-2, F-3
cross-development (8080/8085-
to-8086), 1-2
DATE control (CONV86), 2-2
directives mapping, C-1
displaced reference, 3-2, 3-3, 3-10
EXACT control (CONV86), 1-10, 2-3
expressions, conversion of, B-1
files, CONV86, 1-2, 1-12
files, cross-development, 1-2
flags, mapping of, 1-8
flag semantics, 8080-8086 differences, 1-11
functional equivalence, 1-10
functional mapping, 1-6
INCLUDED control (CONV86), 2-3
instruction mapping, A-1
instruction queue (8086), 1-10
interrupts, 3-3
label insertion by CONV86, 3-2, A-1
label insertion by user, 3-3
macro call, F-1
macro conversion, F-1
macro definition, F-1
MACROFILE control (ASM80), 1-9
manual editing, 1-3, 1-12, 3-1, E-1, F-1
MOD85 control (ASM80), 1-9
NOMACROFILE control (ASM80), 1-9
NOOUTPUT control (CONV86), 2-2
NOPAGING control (CONV86), 2-4
NOPRINT control (CONV86), 2-2
NOTINCLUDED control (CONV86), 2-3
operand mapping, B-1
OUTPUT control (CONV86), 2-2
overriding controls (CONV86), 2-5
overriding symbol types, 1-8, 3-9,
3-10, 3-11
PAGELENGTH control (CONV86), 2-3
PAGEWIDTH control (CONV86), 2-3
pipeline (8086), 1-10
PL/M linkage conventions (8080 &
8086), 3-6
PL/M parameter passing (8080 &
8086), 3-6
PRINT control (CONV86), 2-2
PRINT file, sample, 1-4, 3-1
program listings, 1-5, E-2, E-9, E-11, F-6
prologues (8086), 1-6
prompting, 2-5
register initialization (8086), 3-2
register mapping, 1-7
REL control (CONV86), 1-6, 2-3, 3-2, 3-11
relative addressing, 3-2
relocation & linkage (8086)
errors/warnings, 1-6, G-1
requirements for conversion, 1-1, 1-3, 3-1
reserved names, 1-9, D-1
stack, CONV86 handling of, 1-7
stack segment (8086), 1-6
STKLN directive (8080), 1-6, C-1
symbol typing, 1-8
timing delays, software, 1-10
TITLE control (CONV86), 2-2
WORKFILES control (CONV86), 2-3
WORKFILES control (ASM80), 1-9
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 ten 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.

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