Z80/8086 Cross Assembler

Release 1

PRELIMINARY

REV. A

Seattle Computer Products, Inc.
1114 Industry Drive, Seattle, WA. 98188
(206) 575-1830

CONTENTS

Introduction

The Seattle Computer Products Z80/8086 Cross Assembler runs on the Z80 processor under CP/M or its derivatives. It accepts as input a text file (as would be produced by CP/M's ED) of 8086 source code using mnemonics similar or identical to Intel's ASM-86. It produces an Intel hex format object file and a listing.

This manual is meant to be used in conjunction with the Intel MCS-86 User's Manual which describes the opcodes common to this cross assembler and to Intel. The Opcode Index at the back of this manual is cross referenced to the Intel manual.

Copyright

Rights Being Offered

The rights being offered to purchasers of this software are limited to the right to use the software at the purchaser's place of business (or residence) and the right to make copies solely for the purpose of "backup". Specifically not being offered is the right to sell, give, disclose, or distribute the software to persons outside the purchaser's place of business (or residence).

Registration — Non-Disclosure

The software described in this manual is sold only on the condition that the enclosed "Registration - Non-Disclosure Agreement" is executed by the purchaser and returned to Seattle Computer Products Inc.

Problem Reporting

This manual includes "Problem Reporting Forms" (located in the back) for use by the purchaser to notify Seattle Computer Products of any problems discovered while using the software. It is urged that any problems discovered be reported in order that they may be corrected and other purchasers notified.

Updates

This cross assembler is in its evolutionary infancy. It is expected that it will be updated and enhanced from time to time. Registered purchasers will be notified when updates are available and they may purchase them for a nominal fee not to exceed $25.

Why a Cross Assembler?

During the early life of a new processor, a cross assembler offers the user certain advantages over a resident assembler. Chief among these is the ready availability of low cost software for the older host computer. For the Z-80, this software includes editors, operating systems, and utility packages. Using a cross assembler also allows the user to write maching language code for the new processor even before the hardware is available.

Hardware Requirements

To use this cross assembler, your computer must have a CP/M compatible operating system and a minimum of 16K of memory. To assemble the 8086 monitor program (the sample assembly referenced below), 13K of free memory is required which would be provided by a 24K CP/M system.

Format

The SCP Z-80/8086 Cross Assembler is available in 5" soft-sectored, 5" North Star, and 8" soft-sectored (IBM) formats. When ordering, please specify the format desired on the registration sheet.

Sample Assembly

Included on the disk with the cross assembler is the source file for the monitor used in our 8086 CPU card set. Assembling this source file will give the user an example of how the cross assembler input and output files should appear.

Calling the Assembler

The assembler is invoked with the command ASM86 FILENAME , which will assemble the 8086 source file named FILENAME.A86. The extension "A86" is always assumed and may not be overridden. This is the simplest form of the command. It assumes FILENAME.A86 resides on the current drive, and will write the Intel hex object file, named FILENAME.HEX, and the assembler listing, FILENAME.PRN, to the current drive.

The first variation of this form is to precede the file name with a drive specifier and a colon, such as ASM86 B:FILENAME , which will cause the specified drive to be searched for the source file, but the object and listing files will still be written to the current drive.

The most general form is ASM86 FILENAME.<DRIVE ASSIGNMENT>. <DRIVE ASSIGNMENT> is a 3-letter extension not related to the actual extension to the source file, which is always A86. Instead, it is used as follows:

1. The first letter is the name of the drive on which the source file will be found. This overrides a disk specifier which precedes the file name ("B:").

2. The second letter is the name of the drive to which the hex object file will be written, or "Z" if no object file is desired.

3. The third letter is the name of the drive to which the listing file will be written, or "X" to send the listing to the console, or "Z" if no listing file is desired. Assembling with no listing is much faster since the source file will not be read from disk a second time.

Examples:

ASM86 FILENAME.ABA
- Source - Drive A
- Object - Drive B
- Listing - Drive A

ASM86 FILENAME.AAZ
- Source - Drive A
- Object - Drive A
- No Listing

ASM86 FILENAME.BZX
- Source - Drive B
- Object - None
- Listing - Console

Several errors will cause the assembler to print an error message and abort:

FILE NOT FOUND - The source file was not found on the specified disk. Probably a misspelling or wrong disk.

BAD DISK SPECIFIER - The file name's extension contained an illegal character. Only "A"-"W" and possibly "X" or "Z" are legal.

NO DIRECTORY SPACE - The object or listing file could not be created.

DISK WRITE ERROR - Probably insufficient space on disk for object or listing files.

INSUFFICIENT MEMORY - Memory requirements increase with source program size due to storage required by the symbol table and by the intermediate code. Requirements can be reduced by using shorter labels, by defining labels before they are used, and by reducing the total number of program lines.

Source Program Format

Input to the assembler is a sequence of lines, where each line is terminated with ASCII carriage return and linefeed characters. The assembler accepts lines of any length, but does no list formatting so line length may be limited by your list device. Upper and lower case characters are completely equivalent and may be mixed freely.

Each line may include up to four fields, which may be separated from each other by any number of spaces or tabs (control-I). Fields must appear in order, as follows:

1. Label field (optional) - If present, it must either begin with the first character on the line or be followed immediately by a colon. A label begins with a letter and may be followed by any number of letters or digits, up to a total length of 80 characters, all of which are significant.

2. Opcode field (optional) - If present, it must begin AFTER the first character on the line (otherwise it would be mistaken for a label).

3. Operand field - This field is present only as required by the opcode field.

4. Comment field (optional) - If present, it must begin with a semicolon (;).

Since all fields are optional, lines may be blank, may have labels only, may have comments only, etc.

Bus lock (LOCK), string repeat (REP), and segment override (SEG) prefixes are treated as separate opcodes and must appear on the line preceding the opcode they are to prefix.

Operands

Each operand is one of the following types:

1. REG - A register: AX, BX, CX, DX, AL, AH, BL, BH, CL, CH, DL, DH, DI, SI, DI, SP, BP, CS, DS, ES, SS. Most instructions have limitations on which registers may be used.

2. VALUE - An expression involving additon or subtraction of constants or labels. Terms of the expression may be:

a) A decimal constant ("486").

b) A hex constant, which must begin with a digit from 0 to 9, and end with an "H" ("0F9H").

c) A string constant. In general, this is any number of characters enclosed by either single (') or double (") quotes. Since the opening and closing quotes must be the same, the other type may appear in the string freely. If the same quote as opened the string needs to appear within it, it must be given as two adjacent quotes. Examples:

"TEST" is the same as 'TEST'

"" is the same as ''''

Control characters except control-Z (1AH) may appear in the string, but this may have a strange effect on the listing.

Note that multi-character strings are meaningful only for the DB, DM, and DW pseudo-ops. All other expressions are limited to one character strings.

d) A label. No more than one undefined label may appear in an expression, and undefined labels may only be added, not subtracted. An undefined label is one which has not yet appeared in the label field as the souce code is scanned from the beginning to the current line.

Generally, a VALUE consists of:

a) An optional leading + or -.

b) A term.

c) Zero or more additional terms, each preceded by a + or -.

An exception to this is that no terms may precede a multi-character string in an expression. Terms may follow the string, in which case whey will be added to or subtracted from the value of the last character. Examples:

Legal: "Time" + 80H

Illegal: -"Time" or 80H + "Time"

3. [ADDR] - A valid 8086 address expression enclosed within brackets. The address expression may be:

a) A VALUE, as defined above.

b) A base register (BP or BX).

c) An index register (SI or DI).

d) The sum of any of the above, as limited by valid 8086 addressing modes.

EXAMPLES OF OPERANDS

Legal:

```
-3+ 17H

SCOPE+4

[ bx + COUNT-2 ]

[SI+ARRAY+BX-OFFSET]    ;OFFSET must have already been defined

[DI]

[ NEXT]
```

Illegal:

```
12+BX   ;Register not allowed in VALUE

9C01    ;Needs trailing "H"

[Count - BX]    ;Can't subtract register

[BX+BP]   ;Only one base register at a time

[ARRAY+BX+OFFSET]    ;Both labels are forward referenced (Note 1)

COUNT-DIF   ;DIF is forward referenced (Note 2)
```

Note 1. This problem could be corrected like this:

```
        MOV     AX,[BX + ARRAYPLUSOFFSET]
        .
        .
        .
ARRAY:
        .
OFFSET:
        .
        .
        .
ARRAYPLUSOFFSET:  EQU   ARRAY + OFFSET
```

Note 2. This problem could be corrected like this:

```
        MOV     AX,COUNT + MINUSDIF
        .
        .
        .
DIF:
        .
        .
        .
MINUSDIF:EQU    -DIF
```

ALIGN

ALIGN assures that the next location counter address is even, i.e., aligned on a word boundary. If the location counter is currently odd, both it and the PUT address are incremented; otherwise they are unchanged. See PUT and ORG for an explanation of these terms.

```
DB      VALUE
DB      VALUE, VALUE, VALUE, . . ., VALUE
```

DB (Define Byte) is used to tell the assembler to reserve one or more bytes as data in the object code. Each value listed is placed in sequence in object code, where a multi-character string is equivalent to a sequence of one-character strings. Values must be in the range -256 to +255. Example:

```
DB      "Message in quotes",0DH,0AH,-1
```

```
DM      VALUE
DM      VALUE, VALUE, VALUE, . . ., VALUE
```

DM (Define Message) is nearly identical to DB, except that the most significant bit (bit 7) of the last byte is set to one. This can be a convenient way to terminate an ASCII message since this bit would not otherwise be significant. Example:

```
        DM      'Message in quotes',0DH,0AH
```
is equivalent to
```
        DB      'Message in quotes',0DH,0AH+80H
```

```
DS      VALUE
```

DS (Define Storage) is used to tell the assembler to reserve VALUE bytes of the object code as storage. Any labels appearing in the expression for VALUE must have already been defined.

```
DW      VALUE
DW      VALUE, VALUE, VALUE, . . ., VALUE
```

DW (Define Word) is used to tell the assembler to reserve one or more 16-bit words as data in the object code. It is very similar to DB, except that each value occupies two bytes instead of one. Since a multi-character string is equivalent to a sequence of one-character strings,
```
        DW      'TEST'
```
is equivalent to
```
        DB      'T',0,'E',0,'S',0,'T',0
```
because the high byte of the 16-bit constant represented by 'T' is always zero.

LABEL: EQU VALUE

EQU (Equate) assigns the VALUE to the label. The label MUST be on the same line as the EQU. Three common uses of this operation are:

1. To assign a name to a constant, for convenience and documentation. For example:

```
CR:   EQU   13
LF:   EQU   10
```

The program could now refer to ASCII carriage return and linefeed with symbols CR and LF, respectively.

2. To "parameterize" a program. I/O ports and status bits, for example, could be set by equates at the beginning of the program. Then to reassemble the program for a different I/O system would require editing only these few lines at the beginning.

3. To bypass expressions that would have two or more undefined labels or that would subtract an undefined label. See examples under OPERANDS.

IF VALUE
ENDIF

IF allows portions of the source code to be assembled only under certain conditions. Specifically, that portion of the source code between the IF and ENDIF will be assembled only if the operand is NOT zero. This is particularly useful when producing different versions of the same program. IFs may not occur within an IF/ENDIF pair.

ORG VALUE

ORG sets the assembler's location counter, which is subsequently incremented for each byte of code produced or space allocated. The value of the location counter should always be equal to the displacement from the beginning of the segment to the next byte of code or data, since it is used to establish the value of labels. ORG may be used any number of times in a program. Any labels appearing in the expression for VALUE must have already been defined.

PUT VALUE

The assembler writes object code to the disk in Intel hex format. This format includes information which specifies the addresses at which the object file will be later loaded into memory by a hex loader such as DDT.

PUT is used to specify this load address. Intially, the load address is 100H, that is, "PUT 100H" is assumed before assembly begins. Each time a PUT occurs, all subsequent code would be loaded starting at the specified address until the next PUT is encountered. This allows modules to be placed in specific areas of memory. Note that the load address is not related to the location counter (see ORG), although PUTs and ORGs will often occur together. Any labels appearing in the expression for VALUE must have already been defined.

Opcode Classifications

TWO OPERAND ALU

ADC, ADD, AND, CMP, MOV, OR, SBB, SBC, SUB, TEST, XCHG, XOR

Operand Forms:

REG,REG	Register to register
[ADDR],REG	Register to memory
REG,[ADDR]	Memory to register
REG,VALUE	Immediate to register
B,[ADDR],VALUE	Byte immediate to memory
W,[ADDR],VALUE	Word immediate to memory
[ADDR],VALUE	Immediate to memory defaults to word

Specific Notes:

SBC is the same as SBB.

The order of operands for TEST and XCHG is irrelevant.

XCHG may not use immediate operands.

ONE OPERAND ALU

DEC, DIV, ESC, IDIV, IMUL, INC, MUL, NEG, NOT, POP, PUSH

Operand Forms:

REG	Register
B,[ADDR]	Memory byte
W,[ADDR]	Memory word
[ADDR]	Default to word

Specific Notes:

POP, PUSH, and ESC only operate on words.

INPUT/OUTPUT

IN, INB, INW, OUT, OUTB, OUTW

Operand Forms:

VALUE	Input/output to fixed port
DX	Input/output to port number in DX

Specific Notes:

IN, INB, OUT, OUTB transfer bytes.

INW, OUTW transfer words.

SHIFT/ROTATE

RCL, RCR, ROL, ROR, SAL, SAR, SHL, SHR

Operand Forms:

REG	Shift/rotate register one bit
REG,CL	Shift/rotate register CL bits
B,[ADDR]	Shift/rotate memory byte
B,[ADDR],CL	
W,[ADDR]	Shift/rotate memory word
W,[ADDR],CL	
[ADDR]	Default to word
[ADDR],CL	

Specific Notes:

SHL and SAL are the same.

SHORT JUMPS

JA, JAE, JB, JBE, JC, JCXZ, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ, LOOP, LOOPE, LOOPNE, LOOPNZ, LOOPZ

Operand Form:

VALUE	Direct jump

Specific Notes:

VALUE must be within -126 to +129 of instruction pointer, inclusive.

JP is NOT Jump on Parity. JP is the unconditional short direct jump.

JC, JNC are Jump on Carry and Jump on Not Carry, respectively.

LONG JUMPS/CALLS

CALL, JMP

Operand Forms:

VALUE	Intra-segment direct
VALUE,VALUE	Inter-segment direct
REG	Intra-segment indirect through register
[ADDR]	Intra-segment indirect through memory
L,[ADDR]	Inter-segment indirect through memory ("Long")

Specific Notes:

JMP does NOT include the short direct jump. Its mnemonic is JP and is included under "Short Jumps".

RETURN

RET

Operand Forms:

(none)	Intra-segment
L	Inter-segment ("Long")
VALUE	Intra-segment and add VALUE to SP
L,VALUE	Inter-segment and add VALUE to SP

STRING OPERATIONS

CMPB, CMPW, LODB, LODW, MOVB, MOVW, SCAB, SCAW, STOB, STOW

No operand. These mnemonics replace Intel's CMPS, LODS, SCAS, STOS. The ending "B" or "W" distinguishes between byte and word operations, respectively.

INTERRUPT

INT

Operand Form:

VALUE

ADDRESS MANIPULATION

LDS, LEA, LES

Operand Form:

REG,[ADDR] Put effective address in register

SEGMENT OVERRIDE PREFIX

SEG

Operand Form:

REG Must be a segment register (CS, DS, ES, SS)

Specific Notes:

This opcode should appear on the line immediately preceding the line to be prefixed.

STRING REPEAT PREFIXES

REP, REPE, REPNE, REPNZ, REPZ

No operand. Conditional repeats should be read as "Repeat while . . .", e.g., REPE is Repeat While Equal. For those string operations which affect the flags, REP, REPE, REPZ, all repeat while the zero flag is set; REPNZ, REPNE repeat while the zero flag is clear. This opcode should appear on the line immediately preceding the string operation to be prefixed.

ALL OTHER OPCODES

AAA, AAD, AAM, AAS, CBW, CLC, CLD, CLI, CMC, CWD, DAA, DAS, DI, DOWN, EI, HLT, INTO, IRET, LAHF, LOCK, NOP, POPF, PUSHF, SAHF, STC, STD, STI, UP, WAIT, XLAT

No operand.

Specific Notes:

DI is the same as CLI.

EI is the same as STI.

UP is the same as CLD.

DOWN is the same as STD.

NOP is the same as XCHG AX,AX.

LOCK is treated as a separate opcode and should appear on the line immediately preceding the opcode it is to prefix.

Error List

When a non-fatal error occurs in the source code, the next line of the listing will have an error message which will include a error number in hex. The following table lists the cause associated with the given error number.

01 Register field not allowed
02 Only BP, BX, SI, DI allowed
03 Only one base register (BP, BX) allowed
04 Only one index register (SI, DI) allowed
05 Subtraction of register or undefined label not allowed
06 Only one undefined label per expression allowed
07 Illegal digit in hex number
08 Illegal digit in decimal number
0A Illegal character in label or opcode
0B Double defined label
0C Opcode not recognized
14 Invalid operand
15 "," expected
16 Register mismatch
17 Immediate not allowed here
18 "]" expected
19 Memory-to-memory not allowed
1A Immediate may not be destination
1B Register-to-register not allowed here
1C Must specify segment register
1D Load only
1E Constant must be defined
1F Value error
20 Flag must be set only once
21 Label never defined
22 "EQU" must have label on same line
23 Zero length string illegal
24 ENDIF without IF
25 One-character strings only
26 Expression may not precede multi-character string
64 Undefined label
65 Value error

INDEX TO OPCODES

This list includes all opcodes recognized by the assembler plus those used by Intel but not used by Seattle Computer Products (SCP). Each has the page number on which it will be found in this manual, where a † denotes an Intel opcode NOT recognized by the cross assembler. Also listed is the page number on which a description of the operation will be found in the Intel MCS-86 User's Manual, where the * means Intel uses a different mnemonic for that operation. Opcodes with no entry under GROUP will be found under "All Other Opcodes".

OPCODE	GROUP	MANUAL PAGE	INTEL PAGE	REMARKS
AAA		15	4-10	
AAD		15	4-15	
AAM		15	4-13	
AAS		15	4-12	
ADC	Two Operand ALU	11	4-9	
ADD	Two Operand ALU	11	4-9	
AND	Two Operand ALU	11	4-18	
CALL	Long Jumps/Calls	13	4-23	
CBW		15	4-15	
CLC		15	4-29	
CLD		15	4-29	
CLI		15	4-30	
CMC		15	4-29	
CMP	Two Operand ALU	11	4-12	
CMPB	String Operations	14	4-22*	Intel uses CMPS
CMPS	String Operations	14†	4-22	Use CMPB, CMPW
CMPW	String Operations	14	4-22*	Intel uses CMPS
CWD		15	4-15	
DAA		15	4-10	
DAS		15	4-12	
DEC	One Operand ALU	11	4-11	
DI		15	4-30*	Intel uses CLI
DIV	One Operand ALU	11	4-14	
DOWN		15	4-29*	Intel uses STD
EI		15	4-30*	Intel uses STI
ESC	One Operand ALU	11	4-30	
HLT		15	4-30	
IDIV	One Operand ALU	11	4-14	
IMUL	One Operand ALU	11	4-13	
IN	Input/Output	12	4-7	SCP/Intel different
INB	Input/Output	12	4-7*	Intel uses IN
INC	One Operand ALU	11	4-10	
INT	Interrupt	14	4-28	
INTO		15	4-28	
INW	Input/Output	12	4-7*	Intel uses IN
IRET		15	4-29	
JA	Short Jumps	13	4-26	
JAE	Short Jumps	13	4-26	
JB	Short Jumps	13	4-25	
JBE	Short Jumps	13	4-25	

OPCODE	GROUP	MANUAL PAGE	INTEL PAGE	REMARKS
JC	Short Jumps	13		Intel does not use
JCXZ	Short Jumps	13	4-28	
JE	Short Jumps	13	4-24	
JG	Short Jumps	13	4-26	
JGE	Short Jumps	13	4-26	
JL	Short Jumps	13	4-25	
JLE	Short Jumps	13	4-25	
JMP	Long Jumps/Calls	13	4-23	SCP/Intel different
JNA	Short Jumps	13	4-25	
JNAE	Short Jumps	13	4-25	
JNB	Short Jumps	13	4-26	
JNBE	Short Jumps	13	4-26	
JNC	Short Jumps	13		Intel does not use
JNE	Short Jumps	13	4-26	
JNG	Short Jumps	13	4-25	
JNGE	Short Jumps	13	4-25	
JNL	Short Jumps	13	4-26	
JNLE	Short Jumps	13	4-26	
JNO	Short Jumps	13	4-27	
JNS	Short Jumps	13	4-27	
JNZ	Short Jumps	13	4-26	
JO	Short Jumps	13	4-25	
JP	Short Jumps	13	4-25	SCP/Intel different
JPE	Short Jumps	13	4-25	
JPO	Short Jumps	13	4-27	
JS	Short Jumps	13	4-26	
JZ	Short Jumps	13	4-24	
LAHF		15	4-8	
LDS	Address Manipulation	14	4-7	
LEA	Address Manipulation	14	4-7	
LES	Address Manipulation	14	4-8	
LOCK		15	4-31	
LODB	String Operations	14	4-22*	Intel uses LODS
LODS	String Operations	14†	4-22	Use LODB, LODW
LODW	String Operations	14	4-22*	Intel uses LODS
LOOP	Short Jumps	13	4-27	
LOOPE	Short Jumps	13	4-27	
LOOPNE	Short Jumps	13	4-28	
LOOPNZ	Short Jumps	13	4-28	
LOOPZ	Short Jumps	13	4-27	
MOV	Two Operand ALU	11	4-5	
MOVB	String Operations	14	4-21*	Intel uses MOVS
MOVS	String Operations	14†	4-21	Use MOVB, MOVW
MOVW	String Operations	14	4-21*	Intel uses MOVS
MUL	One Operand ALU	11	4-13	
NEG	One Operand ALU	11	4-12	
NOP		15	4-31	
NOT	One Operand ALU	11	4-15	
OR	Two Operand ALU	11	4-19	
OUT	Input/Output	12	4-7	SCP/Intel different
OUTB	Input/Output	12	4-7*	Intel uses OUT
OUTW	Input/Output	12	4-7*	Intel uses OUT
POP	One Operand ALU	11	4-6	

Z80 TO 8086 TRANSLATOR

The Seattle Computer Products Z80 to 8086 Translator runs on the Z80 under CP/M. It accepts as input a Z80 source file written using Zilog/Mostek mnemonics and converts it to an 8086 source file in a format acceptable to our 8086 Cross Assembler.

To translate a file, simply type TRANS86 <filename>.<ext> . Regardless of the original extension, the ouput file will be named <filename>.A86 and will appear on the same drive as the input file. A file named TRNTEST.Z80 is included to demonstrate the translator.

The entire Z80 assembly language is not translated. The following opcodes will result in an "opcode error":

```
CPD
CPI
IM
IND
INDR
INI
INIR
LDD
LDI
OTDR
OTIR
OUTD
OUTI
RLD
RRD
```

Only the following pseudo-ops are allowed:

```
DB
DM
DS
DW
EQU
IF/ENDIF
ORG
```

Any others will generate an "opcode error".

TRANSLATION NOTES

IX, IY, and the auxillary register set are mapped into memory locations but these locations are not defined by the translator. If a file using these registers is translated and assembled, "undefined label" errors will result. The file must be edited and the memory locations defined as follows:

```
IX:     DS      2
IY:     DS      2

BC:     DS      2       ;Auxillary register set definition
DE:     DS      2
HL:     DS      2
```

Since IX and IY are mapped into memory locations [IX] and [IY], a memory load or store of IX or IY will translate into a memory-to-memory move. LD IX,(LOC) would become MOV [IX],[LOC]. This is easily corrected by editing and using a register: MOV DI,[LOC]; MOV [IX],DI.

All references to the I (interrupt) and R (refresh) registers will generate an error when the translated file is assembled. The "I" and "R" designations are passed straight through, so that LD I,A becomes MOV I,AL, which would appear to be an attempt to move AL into an undefined immediate.

Blank spaces must not occur within operands. Blanks are equivalent to commas in separating operands.

The input file is assumed to assemble without errors with a Z80 assembler. Errors in input may cause incorrect translation without an error or warning message.

The BIT, SET, and RES instructions require the bit number to be a single digit, 0-7. Use of a label for a bit number, for example, will result in "cannot determine bit number" error.

DJNZ is translated into a decrement followed by jump-if-not-zero. DJNZ, however, does not affect the flags while the decrement does. This is flagged as a warning in the output file and may require special action in some instances.

The parity flag of the 8086 will always be set according to 8080 rules and therefore may not be correct for the Z80. Any jump on parity is flagged with this warning.

ASSEMBLY NOTES

It is likely that a translated program will be flagged with some errors when assembled by our 8086 Cross Assembler. These errors are usually caused by out-of-range conditional jumps. Since all 8086 conditional jumps must be to within 128 bytes, this type of error is corrected by changing the conditional jump to a reverse-sense conditional jump around a long jump to the target. For example:

```
        JZ      FARAWAY
```

becomes

```
        JNZ     SKIP1
        JMP     FARAWAY
SKIP1:
```

Other assembly errors may occur because the cross assembler does not have all the features found in some Z80 assemblers, particularly in expression handling, where the only operations are + and -. These errors can only be corrected by finding a way not use the missing feature.

OPCODE	GROUP	MANUAL PAGE	INTEL PAGE	REMARKS
POPF		15	4-9	
PUSH	One Operand ALU	11	4-5	
PUSHF		15	4-8	
RCL	Shift/Rotate	12	4-17	
RCR	Shift/Rotate	12	4-18	
REP	String Repeat Prefixes	14	4-21	Not fully defined by Intel
REPE	String Repeat Prefixes	14	4-21	Not fully defined by Intel
REPNE	String Repeat Prefixes	14	4-21	Not fully defined by Intel
REPNZ	String Repeat Prefixes	14	4-21	Not fully defined by Intel
REPZ	String Repeat Prefixes	14	4-21	Not fully defined by Intel
RET	Return	13	4-24	
ROL	Shift/Rotate	12	4-17	
ROR	Shift/Rotate	12	4-17	
SAHF		15	4-8	
SAL	Shift/Rotate	12	4-16	
SAR	Shift/Rotate	12	4-16	
SBB	Two Operand ALU	11	4-11	
SBC	Two Operand ALU	11	4-11*	Intel uses SBB
SCAB	String Operations	14	4-22*	Intel uses SCAS
SCAS	String Operations	14†	4-22	Use SCAB, SCAW
SCAW	String Operations	14	4-22*	Intel uses SCAS
SEG	Segment Override Prefix	14	4-3	Intel uses no opcode
SHL	Shift/Rotate	12	4-16	
SHR	Shift/Rotate	12	4-16	
STC		15	4-29	
STD		15	4-29	
STI		15	4-30	
STOB	String Operations	14	4-22*	Intel uses STOS
STOS	String Operations	14†	4-22	Use STOB, STOW
STOW	String Operations	14	4-22*	Intel uses STOS
SUB	Two Operand ALU	11	4-11	
TEST	Two Operand ALU	11	4-19	
UP		15	4-29*	Intel uses CLD
WAIT		15	4-30	
XCHG	Two Operand ALU	11	4-6	
XLAT		15	4-7	
XOR	Two Operand ALU	11	4-20	