Z80 TO 8086 TRANSLATOR

The Seattle Computer Products Z80 to 8086 Translator 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 ASM, the Seattle Computer Products assembler.

To translate a file, simply type TRANS <filename>.<ext> . Regardless of the original extension, the ouput file will be named <filename>.ASM and will appear on the same drive as the input file.

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 ASM. 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 assembler does not have all the features found in some Z80 assemblers. For example, ASM can't handle logical operators in expressions. These errors can only be corrected by finding a way not use the missing feature.

ASM - The Small Assembler

This assembler is presented as an alternative to Microsoft's MASM. It is much smaller and faster, but has far fewer features as well (no macros, for example). It also has full 8087 opcode support, unlike the present version of MASM. It can be recommended for smaller assembly-language jobs or if 8087 opcodes are needed. Also, IO.ASM is in this syntax and the Z80-to-8086 translator produces this format.

This assembler uses a syntax slightly different from MASM and Intel's ASM-86. It is not strongly typed, which means that sometimes an explicit type designator is required. For the 8086-only subset, B is used for byte operations, W for word. When also programming the 8087, S is used for short (32-bit), L for long (64-bit), and T for ten-byte (temporary real). While almost all mnemonics are the same, the section on Operands should be carefully reviewed for other differences.

This manual attempts no explanation of the 8086 architecture or instruction set. Intel reference manuals will be required for this, such as the iAPX 86,88 User's Manual.

CALLING THE ASSEMBLER

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

The first variation of this form is to precede the file name with a drive specifier and a colon, such as ASM 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 default drive.

The most general form is ASM FILENAME.<DRIVE ASSIGNMENT>. The <DRIVE ASSIGNMENT> is a 3-letter extension not related to the actual extension to the source file, which is always ASM. 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 one of the following special characters:

P - Send the listing directly to the printer (TABs are NOT expanded). Lines with errors and their error messages are still displayed on the console.
X - Send the listing to the console.
Y - No listing file is produced, except that lines with errors and their error messages are displayed on the console.
Z - No listing file is desired. Error messages are still sent to the console, including address and line number in error, but the actual source text of the line in error is not listed. This option is much faster than any of the others since the source file is not read from disk a second time. Since it provides less diagonostics than the other options, it is normally used only if you are assembling a file which probably has no errors in it.

If a listing is selected, then an alphabetical symbol table dump may be appended to it by typing "S" after the file name and extension.

Examples:

```
ASM FILENAME.ABA
        Source - Drive A
        Object - Drive B
        Listing - Drive A

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

ASM FILENAME.BZX S
        Source - Drive B
        Object - None
        Listing (with symbol table dump) - Console
```

CALLING THE ASSEMBLER (Continued)

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"-"O" and possibly "P", "X", "Y" 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. A Register
2. A Value
3. An Address

---

1. REG - A register:

AX, BX, CX, DX, AL, AH, BL, BH, CL, CH, DL, DH, SI, DI, SP, BP, CS, DS, ES, SS are the 8086 registers; ST(0) through ST(7) refer to 8087 registers, where ST(0) may be referred to as simply ST. Most instructions have limitations on which registers may be used.

---

2. VALUE

An expression involving constants or labels. The operators may be "*", "/", "+", and "-" for multiplication, division, addition, and subtraction, respectively. "*" and "/" have their usual higher precedence over "+" and "-", but order of evaluation may be forced with parentheses. Terms of the expression maybe:

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. An undefined label is one which has not yet appeared in the label field as the source code is scanned from the beginning to the current line.

e) "$". This special symbol means the value of the location counter at the start of this instruction.

OPERANDS - Value (Continued)

f) "RET". This special symbol means "the address of a nearby RETurn instruction". The purpose of this is to allow conditional returns without requiring a label to be put on the RETurn instruction. For example,

```
CMP     AL,20H
JC      RET
```

The jump instruction effectively means "return if carry", yet no label named RET need appear--in fact, a label would be ignored. If no RETurn instruction appears within the range of the jump, then a "value error", number 65 hex, will occur. Only RETurn instructions with no operands will be the target of the jump (intra-segment return without adding to stack pointer).

---

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 ]        ;COUNT must have already been defined
        [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
```

PSEUDO-OPS

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.

PSEUDO-OPS (Continued)

```
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 be nested up to to 255 deep.

PSEUDO-OPS (Continued)

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.

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.

8086 OPCODE CLASSIFICATIONS

TWO OPERAND ALU

ADC, ADD, AND, CMP, DIV, IDIV, IMUL, MOV, MUL, 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.

For DIV, IDIV, MUL, and IMUL, the first operand must be AL or AX and the second operand may not be immediate.

8086 OPCODE CLASSIFICATIONS (Continued)

ONE OPERAND ALU

DEC, INC, NEG, NOT, POP, PUSH

Operand Forms:

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

Specific Notes:

POP and PUSH 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.

8086 OPCODE CLASSIFICATIONS (Continued)

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, JMPS, 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. It is retained for historical reasons only and should not be used for new code. Use JMPS instead.

JMPS is the unconditional short jump.

8086 OPCODE CLASSIFICATIONS (Continued)

LONG JUMPS/CALLS

CALL, JMP

Operand Forms:

VALUE	Intra-segment direct
VALUE,VALUE	Inter-segment direct (offset, segment)
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 JMPS and is included under "Short Jumps".

RETURN

RET

Operand Forms:

(no operand)	Intra-segment
L	Inter-segment ("Long")
VALUE	Intra-segment and add VALUE to SP
L,VALUE	Inter-segment and add VALUE to SP

8086 OPCODE CLASSIFICATIONS (Continued)

STRING OPERATIONS

CMPB, CMPSB, CMPSW, CMPW, LODB, LODSB, LODSW, LODW, MOVB, MOVSB, MOVSW, MOVW, SCAB, SCASB, SCASW, SCAW, STOB, STOSB, STOSW, STOW

No operand. Those mnemonics with an "S" as their 4th letter are Intel standard and should be used for all new code. Those operands without the "S" as the 4th letter are retained for historical reasons, and generate the same code as the corresponding Intel standard mnemonic. For example,

CMPB is the same as CMPSB

CMPW is the same as CMPSW

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.

8086 OPCODE CLASSIFICATIONS (Continued)

PROCESSOR ESCAPE OPERATION

ESC

Operand Forms:

VALUE,[ADDR]

VALUE,VALUE

Specific Notes:

The first value is a number in the range 0 to 63 which is internally represented by 6 bits. The leftmost 3 bits from the last part of the first byte of the ESC opcode (the first 5 bits are always 11011). The rightmost 3 bits form the middle section (bits 3,4,5) of the second byte of the ESC opcode. The rest of the second byte of the ESC opcode is determined by the second operand.

If the second operand is [ADDR], then the second byte of the ESC opcode is set up for the correct addressing mode and possibly a one or two byte displacement is appended to the two opcode bytes. If the second operand is a VALUE, it must be in the range 0 to 7, which is internally represented by 3 bits. These 3 bits are placed in bits 0,1,2 of the second byte of the ESC opcode and bits 6 and 7 are both set to 1.

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.

8086 OPCODE CLASSIFICATIONS (Continued)

ALL OTHER OPCODES

AAA, AAD, AAM, AAS, CBW, CLC, CLD, CLI, CMC, CWD, DAA, DAS, DI, DOWN, EI, HALT, 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.

HALT is the same as HLT.

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

8087 OPCODE CLASSIFICATIONS

All 8087 opcodes normally generate a WAIT instruction before the actual operation code. This wait may be suppressed on any instruction by adding a "N" to the mnemonic as the second letter, after the leading "F". For example, the no-wait form of FCLEX is FNCLEX.

TWO-OPERAND ARITHMETIC

FADD, FDIV, FDIVR, FMUL, FSUB, FSUBR

Operand Forms:

ST,ST(i)	ST := ST op ST(i)
ST(i),ST	ST(i) := ST(i) op ST
S,[ADDR]	ST := ST op Short Real
L,[ADDR]	ST := ST op Long Real
(no operand)	ST(1) := ST(1) op ST; pop stack

Specific Notes:

For FDIVR and FSUBR, operation is "reverse division" and "reverse subtraction", respectively. For example, FDIVR ST,ST(2) is ST := ST(2) / ST.

FADD is same as FADDP ST(1),ST. Likewise for other mnemonics.

TWO-OPERAND ARITHMETIC WITH POP

FADDP, FDIVP, FDIVRP, FMULP, FSUBP, FSUBRP

Operand Form:

ST(i),ST	ST(i) := ST(i) op ST; pop stack

Specific Notes:

For FDIVRP and FSUBRP, operation is "reverse division" and "reverse subtraction", respectively. For example, FDIVRP ST(2),ST is ST(2) := ST / ST(2); pop stack.

8087 OPCODE CLASSIFICATIONS (Continued)

LOAD/STORE

FLD, FST, FSTP

Operand Forms:

ST(i)

S,[ADDR] Short Real

L,[ADDR] Long Real

T,[ADDR] Temporary Real (illegal for FST)

Specific Notes:

FST may not be used to store the temporary real type.

INTEGER OPERATIONS

FIADD, FICOM, FICOMP, FIDIV, FIDIVR, FILD, FIMUL, FIST, FISTP, FISUB, FISUBR

Operand Forms:

W,[ADDR] Word Integer

S,[ADDR] Short Integer

L,[ADDR] Long Integer (FILD, FISTP only)

Specific Notes:

Long integer can only be used with FILD and FISTP.

8087 OPCODE CLASSIFICATIONS (Continued)

ONE ADDRESS OPERATIONS

FBLD, FBSTP, FLDCW, FLDENV, FRSTOR, FSAVE, FSTCW, FSTENV, FSTSW

Operand Form:

[ADDR]	Operand size is implicit in instruction

REAL COMPARE

FCOM, FCOMP

Operand Forms:

ST(i)	
S,[ADDR]	Short Real
L,[ADDR]	Long Real
(no operand)	Compare ST and ST(1)

8087 OPCODE CLASSIFICATIONS (Continued)

ONE REGISTER OPERATIONS

FFREE, FXCH

Operand Forms:

ST(i)

(no operand) ST and ST(1) (FXCH only)

Specific Notes:

Only FXCH has no operand form.

ALL OTHER OPCODES

FABS, FCHS, FCLEX, FCOMPP, FDECSTP, FDISI, FENI, FINCSTP, FINIT, FLDLG2, FLDLN2, FLDL2E, FLDL2T, FLDPI, FLDZ, FLD1, FNOP, FPATAN, FPREM, FPTAN, FRNDINT, FSCALE, FSQRT, FTST, FWAIT, FXAM, FXTRACT, FYL2X, FYL2XP1

No operand.

Specific Notes:

FWAIT is the same as WAIT. FNWAIT is not allowed.

HEX2BIN filespec [offset]

The specified file is assumed to be in Intel hex format, as produced by the SCP Assembler. If no extension is given, ".HEX" is assumed. A file of the same name but with an extension of ".COM" is produced which is the absolute binary equivalent of the hex input file, offset by the optional offset parameter or by the default offset of -100 hex if no offset is specified.

To use the offset, it is necessary to understand just what HEX2BIN does:

First, as large a segment as possible (limited to 64K and available memory) is set aside as the load segment. This segment is filled with zeros. Then the ".HEX" file is read piece by piece and decoded. Each time a load address is encountered in the ".HEX" file, the offset is added to it modulo 64K and loading continues at that point. The highest location loaded is also kept track of. The loading process is aborted if the load address (with offset added to it) exceeds available memory. After loading, the file is saved starting at location 0 in the segment, up to the highest location loaded.

Normally, a program intended to become a ".COM" file will be assembled with a "PUT" address of 100 hex, since this is where it will execute. However, the program must actually be written starting right at the beginning of the ".COM" file because the file itself is loaded at 100 hex. This requires a load offset of -100 hex, which is the default offset if none is specified. That is, if no offset is specified (no second parameter to HEX2BIN), the file will be loaded (and saved) at location 0. When the file is later loaded as a command, it will load at 100 hex which is its proper execution address.

The offset parameter is particularly useful when converting a file not intended as an MS-DOS ".COM" file. The parameter has an optional "+" or "-", then a hex number with one to four digits. Note that if an offset of 0 is desired, it must be specified explicitly since the default is -100 hex.