CCSYSTM-023-RMA

CONTROL DATA
CORPORATION

Z-80 RELOCATABLE MACRO ASSEMBLER REFERENCE MANUAL

CONTROL DATA®
MP-32
COMPUTER SYSTEMS

Z80ASM Control Card Format

The Z80 Cross Assembler (Z80ASM) is envoked by the following Control Card:

```
*Z80ASM(I=10,L=20,R=22)
```

The table below describes the defaults and ranges of the various parameters. Parameters may be omitted, may stand alone, or may be equated to a numeric value in the range shown.

	ABSENT	ALONE	=XX	
I	63	56	1-63	INPUT
L	62	62	1-62	LISTING
R	4	4	1-60	RELOCATABLE OBJECT OUTPUT

All values above are logical unit numbers. The Relocatable Object Output is intended to become input for the Linking Cross Loader (Z80LDR).

CCS-A00X-01

PROFESSIONAL SERVICES DIVISION
a consulting service of
CONTROL DATA CORPORATION

MICROTEC

Z-80 RELOCATABLE MACRO ASSEMBLER

FLEET NUMERICAL WEATHER CENTRAL
CONSOLIDATED COMMUNICATIONS SYSTEM

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(10-01-79)	
Publication No. CCS-A00X-01	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning
this manual to:

CONTROL DATA CORPORATION
Professional Services Division
205 Montecito Avenue
Monterey, Ca. 93940

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision	Page	Revision	Page	Revision
Cover	–				
Title Page	–				
ii thru vi	A				
1-1	A				
1-2	A				
2-1 thru 2-7	A				
3-1 thru 3-7	A				
4-1 thru 4-20	A				
5-1 thru 5-9	A				
6-1 thru 6-11	A				
7-1 thru 7-9	A				
A-1 thru A-3	A				
B-1	A				
C-1	A				
D-1 thru D-6	A				

TABLE OF CONTENTS

INTRODUCTION

Microtec has developed a Relocatable Macro Assembler for the Z80 microprocessor that translates symbolic machine code into relocatable object code which may then be processed by Microtec's Linking Loader. The Assembler program is written in FORTRAN IV to achieve compatibility with most computer systems. It is modular and may be executed in an overlay mode should memory restrictions make that necessary. The program is approximately 4500 FORTRAN statements in length, 20% of which are comments. The program is written in ANSI standard FORTRAN IV and no facility peculiar to any one machine was utilized. This was done in order to eliminate FORTRAN compatibility problems.

The mnemonic Operation Codes as well as Directives are identical to those utilized by Zilog or MOSTEK in their literature and in their software products, except for the relocation directives. This has been done to eliminate any possible problems of program compatibility and to obviate the necessity of learning new assembly languages.

The assembler is a two pass program that builds a symbol table, issues helpful error messages, produces an easily read program listing and symbol table, and outputs a computer readable relocatable object (load) module.

The assembler features relocation, macro capability, conditional assembly, symbolic and relative addressing, forward references, complex expression evaluation, cross reference listing and a versatile set of directives.

These features aid the programmer/engineer in producing well documented, working programs in a minimum of time. Additionally, the assembler is capable of generating data in several number based systems as well as both ASCII and EBCDIC character codes.

Microtec does not present any information in this manual that will help the user understand the Z80 microprocessor, nor has any information been included to help the user write working programs. The reader is referred to the Zilog or MOSTEK manuals and specifications to achieve an understanding of their microprocessor. It is recommended that this be done before reading this manual.

ASSEMBLER LANGUAGE

The assembler language provides a means to create a computer program. The features of the Assembler are designed to meet the following goals:

- Programs should be easy to create
- Programs should be easy to modify
- Programs should be easy to read and understand
- A machine readable load module to be generated

This assembler language has been developed with the following features:

- Symbolic machine operation codes (opcodes, directives)
- Symbolic address assignments and reference
- Relative addressing
- Data creation statements
- Storage reservation statements
- Assembly listing control statements
- Addresses may be generated as constants
- Character codes may be specified as ASCII or EBCDIC
- Comments and remarks may be encoded for documentation
- Cross Reference Table listing
- Relocatable object format

An assembly language program is a program written in symbolic machine language. It is comprised of statements. A statement is either a symbolic instruction, a directive statement, a macro statement, or a comment.

The symbolic machine instruction is a written specification for a particular machine operation expressed by symbolic operation codes and sometimes symbolic addresses or operands.
Example:

```
          ISAM      LD         A,(HL)
```

where:

ISAM - is a symbol which will represent the memory address of the instruction.

LD - is a symbolic op-code which represents the bit pattern of the "load" instruction.

A - is a symbol, in this case a keyword, which represents the accumulator.

(HL) - is a symbol, another keyword, which represents memory accessed through registers H and L.

A directive statement is a statement which is not translated into a machine instruction, but rather is interpreted as a command to the assembler program.
Example:

```
          ABAT      DEFW       DELT
```

where:

ABAT - is a symbol. The assembler is to assign the memory address of the first byte of the two allocated bytes to this symbol.

DEFW - is a directive which directs the assembler program to allocate two bytes of memory.

DELT - is a symbol representing an address. The assembler is directed to place the equivalent memory address into the two allocated bytes.

Statements

Statements are always written in a particular format. This format is depicted below.

LABEL FIELD	OPERATION FIELD	OPERAND FIELD	COMMENT FIELD

The statement is always assumed to be written on an 80 column data processing card or as an 80 column card image.

The Label Field is provided to assign symbolic names to a byte of memory. If present, the label field may begin in any column if it is terminated by a colon. It may also begin in column one and not be terminated by a colon. A label may be the only field on the statement.

The Operation Field is provided to specify a symbolic operation code or a directive. If present, the Operation Field must either begin past column one or be separated from the Label Field by one or more blanks, tabs, or a colon.

The Operand Field is provided to specify arguments for the operation in the Operation Field. The Operand Field, if present, is separated from the Operation Field by one or more blanks or tabs. Arguments in the Operand Field may not be separated by blanks or more than one comma.

The Comment Field is provided to enable the assembly language programmer to optionally place an English message stating the purpose or intent of a statement or group of statements. The Comment Field must be separated from the preceding field by one or more blanks or tabs or by a semicolon.

Comment Statement

A Comment statement is a statement that is not processed by the assembler program. It is merely reproduced on the assembly listing. A comment statement is indicated by encoding an asterisk or a semicolon as the first non-blank character on a line. Care should be taken when using an asterisk to indicate a comment as it may be interpreted as an assembler directive (see section 4). It is recommended that a blank follow an asterisk if it indicates a comment. Only an asterisk in column one may be interpreted as a directive.

Example:

```
 ;   THIS IS A COMMENT STATEMENT
```

Logical columns 73-80 are never processed by the assembler. This field is a good place for sequence numbers, if desired.

Reserved Keywords and Symbols

Certain keywords have been defined internally by the assembler. This will save the user the trouble of defining them in each program. Twenty-six keywords have been defined by the assembler. These symbols are not stored in the symbol table and consequently they may be used in the Label Field of a statement. However, it is recommended that this practice be avoided. The keywords are as follows:

```
A       B       C       D
E       F       H       L
BC      DE      HL      SP
AF      AF'     IX      IY
I       R       Z       NZ
C       NC      PE      PO
P       M
```

In addition the following two symbols denote the "STACK" and "MEMORY" segments of a program (see Section 6). They are stored in the symbol table any thus may not be used in the Label Field of any statement.

```
     STACK                         MEMORY
```

Symbolic Addressing

When writing statements in symbolic machine language, i.e. assembler language, the machine operation code is usually expressed symbolically. For example, the machine instruction that moves data from register B into the memory location addressed by the contents of register pair H,L may be expressed as:

```
        LD      (HL),B
```

When translating this symbolic operation code and its arguments into machine language for the Z80, the assembler defines one byte containing 70H at the memory location in the current Assembly Program Counter. The address of the translated byte is known because the Assembly Program Counter is always set to hold the address of the byte currently being assembled.

The user can optionally attach a label to such an instruction. For example:

```
SAVR    LD      (HL),B
```

The assembler, upon seeing a valid symbol in the label field, assigns the equivalent address to the label. The equivalent address is the address contained in the Assembly Program Counter. In the given example, if the LD instruction is to be stored in the address 127, then the symbol SAVR would be made equivalent to the value 127 for the duration of the assembly.

The symbol could then be used anywhere in the source program to refer to the instruction location. The important concept is that the address of the instruction need not be known; only the symbol need be used to refer to the instruction location. Thus when jumping to the LD instruction, the user could write:

```
        JP    SAVR
```

When the jump instruction is translated by the assembler, the address of the LD instruction is placed in the address field of the jump instruction.

It is also possible to use symbolic addresses which are near other locations to refer to those locations without defining new labels. This may be done through use of the + and - operators. For example:

```
              JP    BEG
              JP    PE,BEG+4
        BEG   LD    A,B
              HALT
              LD    C,'B'
              INC   B
```

In the above example, the instruction "JP BEG" refers to the "LD A,B" instruction. The instruction "JP PE,BEG+4" refers to the "INC B" instruction.

BEG+4 means the address of BEG plus four bytes. This type of expression is called relative symbolic addressing and given a symbolic address such as "BEG" it can be used as a landmark to express several bytes before or after the symbolic address.

Assembly Program Counter

During the assembly process the assembler maintains a FORTRAN word that always contains the address of the next memory location to be assembled. This word is called the Assembly Program Counter. It is used by the assembler to assign addresses to the assembled bytes, but it is also available to the programmer.

The character "$" is the symbolic name of the Program Counter. It may be used like any other symbol, but it may not appear in the label field.

When using the "$", the programmer may think of it as expressing the idea; "$" = "address of myself." For example:

```
                    3F      JR    $
```

The jump instruction is in location 3FH. The instruction directs the microprocessor to "jump to myself." The Program Counter in this example contains the value 3FH and the instruction will be translated to a "JR 3FH". This could be used for example when waiting for an interrupt.

SYNTAX

The Assembler Language is a language like any other. That is, it has a character set, vocabulary, rules of grammar, and allows for individuals to define new words or elements. The rules that describe the language are termed the syntax of the language.

For an expression or statement in assembler language to be translated by the assembly program, it must be written correctly in accord with the rules of syntax.

Character Set

The following list of characters describes the characters that the assembler will recognize. They are the only valid characters. Use of any other characters will cause the assembler to generate an error message.

Alphabetic Characters

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Numeric Characters

0 1 2 3 4 5 6 7 8 9

Special Characters

b̸	blank	/	slash
>	greater than	$	dollar sign
<	less than	*	asterisk
'	single quote	(	left parenthesis
,	comma	)	right parenthesis
+	plus sign	@	commercial at
-	minus sign	.	period

&	ampersand
!	exclamation
"	double quote
#	sharp sign
%	percent
|	vertical bar
	tab character
:	colon
;	semi-colon
=	equal sign
?	question mark
_	underbar
\	back slash

Symbols

A symbol is a sequence of characters. The first character of a symbol may not be a numeric character. A symbol may consist of any alphanumeric character plus any of the following special characters: !,%,?,@,_. Imbedded blanks are not permitted. The user is cautioned not to use symbols that start with the ? character as the assembler generates "local" symbols starting with this character (see LOCAL directive).

Only the first six characters of a symbol are used by the assembler to define that symbol; the remaining characters are for documentation. The parameter that dictates the number of characters used to define a symbol may be changed in the Fortran source code.

The assembler's symbol table can contain up to 200 symbols. If more symbols are required, the symbol table may be increased in size by changing a parameter in the Fortran source program.

Symbols are used to represent arithmetic values, memory addresses, bit arrays (masks), etc. Examples of valid symbols:

LAB1
MASK_ONE
LOOPNUMS (symbol used is LOOPNU)

Examples of invalid symbols:

ABORT*	(contains nonallowed special character)
1LAR	(begins with a numeric)
PAN N	(embedded blank - symbol would be PAN)

Constants

A constant is an invariant quantity. It may be an arithmetic value or a character code. There are several ways of specifying constants in this assembler language.

Decimal constants may be defined as a sequence of numeric characters optionally preceded by a plus sign or a minus sign. If unsigned, the value is assumed to be positive.

All constants are evaluated modulo 65536. A one byte constant can contain an unsigned number with a value from 0 to +255. A two byte unsigned number can range from 0 to +65535. When a constant is negative, its equivalent two's complement representation is generated and placed in the field specified.

Whenever an attempt is made to place a constant in a field for which it is too large, an error message is generated by the assembler.

Other constants are defined by utilizing a descriptor after the value. The following list indicates the available descriptors and their meaning. If no descriptor is given, the number is assumed to be decimal. A leading Ø must be added to hexadecimal constants that start with A-F.

B - binary	(base 2)
O - octal	(base 8)
Q - octal	(base 8)
D - decimal	(base 10)
H - hexadecimal	(base 16)

Examples of these constants are:

```
    1ØØ11B    25    ØFFH    37Q    255D    1357O
```

An ASCII or EBCDIC character constant may be specified by enclosing a single character within quote marks and preceding it with a A for ASCII or an E for EBCDIC. If no descriptor is specified, the string is assumed to be ASCII. Examples of this constant form are:

```
LD    A,'1'
LD    A,E'Z'
OR    'Ø'
```

A character string may be specified by using the DEFB,DB, DATA, or DEFM directives. Character strings must follow the format described for these directives (see section 4). Characters may be specified as ASCII or EBCDIC in a manner similar to the character constant. Examples of the character string are:

```
A'TELETYPE CODES'
E'TERMINAL CODES'
 '   123.8'
```

Note that one byte of memory is required for each character in a string. When a string is specified in a DEFB, DB, DATA, or DEFM directive, characters are stored in sequential bytes of memory beginning at the first available byte.

To cause the code for a single quotation mark to be generated in the character constant or string, it must be specified as two single quote marks. Example:

```
'DON''T'
```

The character code for a single quotation mark will be generated once for every two marks that appear contiquously within the character string.

Expressions

An expression is a sequence of one or more symbols, constants or other expressions separated by arithmetic operators. Expressions are evaluated left to right subject to the precedence of operators shown below. Parenthesis may be used to establish the correct order of the arithmetic operators and it is recommended they be used in complex expressions involving operators such as SHR, AND, EQ, etc.

Precedence	Operator	
1	+	(unary plus)
	-	(unary minus)
2	**	(exponentation)
3	*	(multiplication)
	/	(division)
	.MOD.	(modulo)
	.SHR.	(logical shift right)
	.SHL.	(logical shift left)
4	+	(addition)
	-	(subtraction)
5	,.NOT.	(logical NOT)
6	&,.AND	(logical AND)
7	|,.OR.	(logical OR)
	.XOR.	(exclusive OR)
8	=,.EQ.	(equals)
	>,.GT.	(greater than)
	<,.LT.	(less than)
	.UGT.	(unsigned greater than)
	.ULT.	(unsigned less than)
9	.RES.	(result)
10	.LOW.	(low 8 bits)
	.HIGH.	(high 8 bits)

The comparison operators (.EQ.,.GT.,.LT.,.UGT.,.ULT.) return a logical True (all ones) if the comparison is true and a logical False (zero) if the comparison is not true. The operators .GT. and .LT. deal with signed numbers while .UGT. and .ULT. assume unsigned values. For .GT. and .LT. the high order bit of an expression is treated as a sign bit. Hence values greater than 32767 will be treated as negative numbers.

The Result operator (.RES.) does not perform any function but is supplied for compatibility.

The Shift operators (.SHR.,.SHL.) shift their first argument right or left the number of bits specified by the second argument. Zeros are shifted into the high or low order bits.

The .HIGH. and .LOW. operators have been provided to help the user define two byte addresses as individual bytes whenever that is desirable. The result of application of either of these operators is a one byte value. These operators are unary and may be used anywhere in an expression. When .HIGH. or .LOW. are used in a relocatable expression the result will remain relocatable. This enables the user to relocate 8 bit values. The following example demonstrates the utility of these operators.

```
        LD    HL,BUFF
LOOP    LD    A,(HL)
        CP    13
        JP    Z,MAIN
        INC   HL
        LD    L,A
        CP    .LOW.(BUFF+40)   ;CHECK FOR END
        JP    Z,MAIN
        JR    LOOP
```

An expression must resolve to a single unique value. Consequently, character strings are not permitted in expressions. All expressions are evaluated modulo 65536. Whenever an attempt is made to place an expression in a one byte field and the expression is too large, an error message is generated. Examples of valid expressions:

```
PAM+3
(PAM+45H)/CAL
IDAM.AND.255
LOOP+(ADDR.SHR.8)/2
VAL1.EQ.VAL2
```

Note: for certain opcodes, an expression enclosed in parenthesis indicates a memory address. A leading plus sign may be used to avoid any problems if the expression is actually an immediate value.

Relative Addressing

For those instructions that use relative addressing (JR, DJNZ), the program counter, "$" may or may not be subtracted from the relative address depending upon the option specified in the LIST/NLIST directive. Thus the user has the option of specifying the operand of a relative address in either of the following two ways:

```
DJNZ    MAIN                    DJNZ    MAIN-$
```

The default is that the "$" must be specified. It is recommended that the user let the assembler subtract the "$" from the relative address instead of explicitly doing so in the assembly statement. This allows certain error detection to be performed on relocatable program segments that cannot otherwise be done. (See section on Relocation)

DIRECTIVES

The directives or pseudo-operations are written as ordinary statements in the assembler language, but rather than being translated into equivalent machine language, they are interpreted as commands to the Assembler itself.

Through use of these directives the Assembler will reserve memory space, define bytes of data, control the listing, assign values to symbols, etc.

This section of the manual describes all directives and assembler commands except those primarily associated with macro assembly and relocation. Some directives such as ORG apply to both absolute and relocatable assembly.

Assembler Commands

Assembler commands are directives that begin with an asterisk in column one. Column two identifies the type of command. The user should be aware of these commands when denoting comments with an asterisk in column one. Depending upon the character in column two, it may be interpreted as a command. The Assembler Commands are equivalent to the following directives.

*EJECT		EJEC	
*HEADING	S	TITLE	'S'
*LIST	ON	LIST	S
*LIST	OFF	NLIST	S
*MACLIST	ON	LIST	M
*MACLIST	OFF	NLIST	M

The directives described in this section are:

ORG	Set Program Origin
END	End of Assembly
EQU	Equate a Symbol to an Expression
DEFL	Define a Label
DEFB	Define a Byte
DB	Define a Byte (same as DEFB)
DATA	Define a Byte (same as DEFB)
DEFW	Define a Word
DW	Define a Word (same as DW)
DDB	Define Double Byte
DEFS	Define Storage
DS	Define Storage (same as DEFS)
DEFM	Defime Message
EJEC	Advance Listing Form to next page
SPAC	Space lines on listing
TITLE	Set Program Heading
LIST	List the elements specified
NLIST	Suppress listing of elements specified
IF	Conditional Assembly Statement
COND	Conditional Assembly Statement (same as IF)
ELSE	Conditional Assembly Statement Converse
ENDIF	End Conditional Assembly Code
ENDC	End Conditional Assembly Code (same as ENDIF)

In the following descriptions, the brackets, { }, are used to indicate optionality, or if more than one item appears within the same pair of brackets, they indicate a choice.

ORG — Set Program Origin (non relocatable mode)

The ORG directive is used to inform the assembler of the memory address to which the next assembled byte should be assigned. All subsequent bytes will be assigned sequential addresses beginning with this address.

If the program does not have an ORG as the first statement, an ORG Ø is assumed and assembly will begin at location zero with absolute assembly.

Example:

```
ORG  1ØØH
```

{label}	ORG	expression

where:

label - is an optional label which if present will be equated to the given expression.

expression - a value which will replace the contents of the Assembly Program Counter and bytes subsequently assembled will be assigned memory addresses beginning with this value. Any symbols used in the expression must be previously defined.

END — End of Assembly

The END directive is used to inform the assembler that the last card of the source program has been read, as well as indicate that load module starting address. Any statements following the END directive will not be processed.

Example:

```
                    END    MAIN
```

```
            END | {expression}
```

where:

expression - is an address that is placed in the end record of the load module and informs the loader where program execution is to begin. If expression is not specified the load address is set to zero. Specifying a load address in this directive also implies that this is a main program to the loader. If multiple load modules are combined by the Linking Loader, only one module may specify a load address and hence be a main program.

EQU — Equate a Symbol to an Expression

The EQU directive is used to cause the assembler to assign a particular value to a new label. This value may be an absolute value or a relocatable value (see Section 6).

Example:

```
SEVEN  EQU   7
```

label	EQU	expression

where:

label - is a symbol defined by this statement

expression - is an expression whose value will be assigned to the given label for the duration of the assembly. An attempt to reequate the same label will result in an error. Any symbols used in the expression must be previously defined. An external symbol may not be used in the expression.

DEFL — Define a Label

The DEFL directive may be used to set a symbol equal to a value. Unlike the EQU directive, multiple DEFL directives may be encoded in the same source program for the same symbol. The most recent DEFL directive determines the value of the symbol at any given place in the source program.

Example:

```
                    GO  DEFL  5
                    GO  DEFL  GO+10
```

label	DEFL	expression

where:

label - is a symbol defined by this statement

expression - is a value that will be assigned to the given label until changed by another DEFL directive. Any symbols used in the expression must be previously defined. An external symbol may not be used in the expression.

DEFB — Data Definition
DATA
DB

The DEFB, DATA, and DB directives are used to define up to 70 bytes of data. The assembler will allocate one byte if an expression is given and will allocate several bytes if a character string is given. All expressions must evaluate to an one byte value or an error is generated. Negative values are stored using their two's complement representation. If an operand is a relocatable expression, it must be preceded by the .LOW. or .HIGH. operators. If neither operator is present, an error is generated and the .LOW. operator is assumed.

Example:

```
ITEM  DEFB   +122,17,.LOW.EXP1
      DATA   6,1FH,'A'+1,32Q
OUT2  DB     A'ERR 1',7
```

{label}	DEFB DATA DB	operand$_1$,{operand$_2$}, ...

where:

label - is an optional label which will be assigned the address of the first byte defined.

operand$_i$ - is an evaluatable expression contained in one byte, a character constant or an ASCII or EBCDIC character string of up to 70 characters.

DEFW — Define Word
DW

The DEFW or DW directive informs the assembler to allocate two bytes per operand. Each operand is stored in successive bytes. The operands are stored with the low order 8 bits in the first byte and the high order 8 bits in the second byte. Negative values are stored using their two's complement representation.

Example:

```
                    ADD1   DW     1BH,40
                           DEFW   1000,10000
```

{label}	DEFW DW	operand_1,$\{\text{operand}_2\}$, ...

where:

label - is an optional label which will be assigned the address of the first byte defined.

operand_i - is an evaluatable expression contained in two bytes. A total of 70 bytes may be allocated by this directive.

DDB — Define Double Byte

This directive is similar to the DEFW directive except for the order in which the 16 bit value of each operand is stored. The low order 8 bits of the operand are stored in the second byte of the double byte and the high order 8 bits are stored in the first byte. Negative values are stored using their two's complement representation.

Example:

```
                REV1    DDB     1000,10000
```

{label}	DDB	operand$_1$,{operand$_2$}, ...

where:

label - is an optional label which will be assigned the address of the first byte defined.

operand$_i$ - is an evaluatable expression contained in two bytes. A total of 70 bytes may be allocated by this directive.

DEFS — Define Storage
DS

The DEFS and DS directives are used to reserve a block of sequential bytes of storage. These directives merely cause the program counter to be advanced. Therefore, the contents of the reserved bytes are unpredictable.

Example:

```
                    PAT     DEFS    62H
```

{label}	DEFS DS	expression

where:

label - is an optional label which will be assigned the address of the first byte allocated.

expression - a value which specifies the number of bytes to be allocated by this directive. Any symbols used in this expression must be previously defined. This expression may not contain any relocatable symbols.

DEFM — Define Message

The DEFM directive is used to define up to 70 bytes as an ASCII or EBCDIC string. This is the same as using the DEFB directive with only the string as an operand.

Example:

```
                    DEFM   'MACRO ASSEMBLER'
```

{label}	DEFM	'string'

where:

label - is an optional label which will be assigned the address of the first byte allocated.

string - is a string of up to 70 characters. The string must be enclosed in quotes. A single quote within the string must be represented by two single quotes. The leading quote may be preceded by an A for ASCII or an E for EBCDIC. If no character precedes the quote ASCII is assumed.

EJEC — Advance Listing Form to next Page

This directive instructs the assembler to skip to the top of the next page on the listing form. Its purpose is to make program listings easier to read. Some programmers prefer to start each subroutine on a new page.

```
               EJEC
```

SPAC — Space lines on listing

The SPAC directive causes one or more blank lines to appear on the output listing. It enables the programmer to format the program listings for easier reading. The directive itself does not appear on the listing.

Example:

```
                SPAC    7
```

```
                SPAC | expression
```

where:

expression - evalues to a value that determines how many lines are to be skipped. This expression may not be relocatable.

TITLE — Set Program Heading

The TITLE directive is used to print a heading at the beginning of each page of the listing. The default heading defined by the assembler and used if the user does not specify one via this directive is "Z80 ASSEMBLER VER _._MR". For a user specified title to appear on the first page of the output listing, the TITLE directive must be the first statement in the program.

Example:

```
                    TITLE     'TEST PROGRAM'
```

```
             TITLE | heading
```

where:

heading - title which will be placed at the beginning of each page. The heading may be up to 50 characters, with any additional characters not appearing in the title. The heading is delimited by single quotes but if the terminating quote is not present the first 50 characters will be used as the title. Heading may contain no characters in which case the title will be set to blanks.

Note: The Assembler Command *HEADING S; is similar to the TITLE directive with the following differences:

- *HEADING also causes a page eject
- title displayed with the *HEADING command begins with the first non blank character in the operand
- *HEADING statement is not displayed on listing

LIST — List the Elements Specified

The LIST directive may be used to generate listings of the elements specified in the directive. The defaults are that the source text, symbol table, macro expansions, and conditional assembly statement not assembled are listed and in addition an object module is produced. The symbol table is not placed into the object module and system generated local symbols are not listed. Errors are always listed regardless of the elements specified.

Example:

```
                    LIST    X,B       produce cross reference
                                      table and put symbol table
                                      in object module
```

```
            LIST | B,G,I,M,O,R,S,T,X
```

where:

B - specifies that the symbol table will be placed into the object module and may be used for debugging.

G - specifies that system generated symbols (see Section 6) will be listed in the symbol table and object module.

I - specifies that the instructions not assembled due to conditional assembly statements will be listed.(default)

M - specifies that expanded macros will be listed in the source text.(default)

O - specifies that the object module will be produced. (default)

R - specifies that the user must subtract the program counter, "$", when using a relative addressing instruction. E.g. JR LABEL-$. See section on relative addressing. (default)

S - specifies that the source text will be listed.(default)

T - specifies that the symbol table will be listed. (default)

X - specifies that the cross reference table will be listed. This parameter overrides the T option if specified. Thus if T and X are both specified, a cross reference table will be generated. (see page 7-9)

Note: if the user specifies the B or G option, it must be done at the start of the program before the first instruction that generates any code.

NLIST — Suppress Listing of the Elements Specified

The NLIST directive instructs the assembler to suppress the listings of the elements specified. The listings may be enabled again by the LIST directive. Errors generated by the assembler are always listed regardless of the list flags. Thus to obtain an output listing of only errors the user should specify "NLIST S" at the beginning of the program.

Example:

```
                        NLIST     O      do not produce an
                                         object module
```

```
              NLIST | B,G,I,M,O,R,S,T,X
```

where:

B - specifies that the symbol table will not be placed into the object module.

G - specifies that system generated symbols will not be listed in the symbol table or object module.

I - specifies that the instructions not assembled due to conditional assembly statements not be listed.

M - specifies that expanded macros not be listed.

O - specifies that the object module will not be produced.

R - specifies that the program counter, "$", need not be subtracted from the address of a relative address instruction. See section on relative addressing.

S - specifies that the source text will not be listed. Only those statements with errors will be listed.

T - specifies that the symbol table will not be listed.

X - specifies that a cross reference table will not be produced or listed.

COND — Conditional Assembly Statement
IF

The COND or IF directive may be used to conditionally assemble source text between the IF or COND directive and the ELSE, ENDIF, or ENDC directive. If the expression in the operand field is evaluated to any non-zero value, the code will be assembled. If the expression evaluates to a value of zero the code will not be assembled. Conditional statements may be nested up to 16 levels and appear in the source text at any place.

Example:

```
          COND    SYSTEM
          IF      DATA.EQ.7FH
```

```
                    COND | expression
                    IF   |
```

where:

expression - evaluates to a value which determines whether or not the assembly between the IF and the following ELSE, ENDC, or ENDIF will take place. Any symbols used in this expression must be previously defined. The expression may not be relocatable.

ELSE — Conditional Assembly Statement Converse

The ELSE directive is used in conjuction with the IF directive and is the converse of the IF. If the expression in the operand field of the IF or COND directive was zero, all statement between the ELSE directive and the next ENDIF or ENDC directive are assembled. If the expression in the operand field of the IF or COND directive was non-zero, all statements between the ELSE directive and the next ENDIF or ENDC are not assembled.

The ELSE directive is optional and can appear only once within an IF-ENDIF block.

Example:

```
                    IF     MAIN
                    -
                    ELSE
                    -
                    ENDIF
```

ELSE

ENDC — End Conditional Assembly Code
ENDIF

The ENDIF or ENDC directive is used to inform the assembler where the source code subject to the conditional assembly statement ends. In the case of nested conditional statements, an ENDC or ENDIF is paired with the most recent COND of IF statement.

Example:

In the following code, if the expression SUM-4 is equal to zero, the instructions between the IF and ELSE directive will not be assembled and those between the ELSE and ENDIF will be assembled. If SUM-4 is non-zero the opposite occurs. To not list the non assembled instructions, the "NLIST I" directive may be used.

```
                         EIN
                         IF     SUM-4
assembled if             OR     200
SUM-4 is non-zero        ADD    A,VALUE
                         ELSE
assembled if             OR     07FH
SUM-4 is zero            ADD    A,C
                         ENDIF
```

```
          ENDC
          ENDIF
```

MACROS

A macro is a sequence of instructions that can be inserted in the assembly source text by encoding a single instruction, the macro call. The macro definition is written only once and can be called any number of times. The macro definition may contain parameters which can be changed for each call. The macro facility simplifies the coding of programs, reduces the chance of programmer error, and makes programs easier to understand, as the source code need only be changed in one location, the macro definition.

A macro definition consists of three parts; a heading, a body, and a terminator. This definition must precede any call to the macro being defined. A macro may be redefined at any time with the latest definition of a macro name applying to the macro call. A standard mnemonic (e.g. BIT) may also be redefined by defining a macro with the name BIT. In this case all subsequent uses of the mnemonic BIT in the program will cause the macro to be expanded and placed into the source program.

Macro Heading

The heading, which consists of the directive MACRO or MACR, gives the macro a name and defines any formal parameters.

Example:

```
                    GET    MACRO    #ADDR,#VALUE
```

label	MACRO	{parameter list}

Label specifies the macro name and may be any user defined symbol. This name may be the same as other program defined symbols since it has meaning only in the operation field. For example, TAB could be the name of a symbol as well as a macro.

If a macro name is identical to a machine instruction or an assembler directive, the mnemonic is redefined as the macro. Once a mnemonic has been redefined as a macro, there is no way of returning that name to be a standard mnemonic. A macro name may also be redefined as a new macro with a new body.

The operand field of the MACRO line contains the name of dummy formal parameters in the order in which they occur on the macro call. Each parameter is separated by commas and each begins with a sharp sign (#). The parameters may consist of any arbitrary text, e.g. #12XYZ. The parameter list is terminated by either a blank, tab, or semicolon after a parameter. Parameters are scanned from left to right for a match, so the user is cautioned not to use parameter names which are prefix substrings of later parameter names. E.g. #AB,#ABC. The scope of a formal parameter is limited to its specific macro definition.

Macro Body

The first line of code following the MACRO or MACR directive which is not a LOCAL directive is the start of the macro body. These statements are placed in a macro file for use when the macro is called. At expansion time an error will be generated if another macro is defined within a macro. No statements are assembled at definition time including Assembler directives and Assembler Commands.

Within the macro body, in any field, the name of a formal parameter listed on the MACRO or MACR line may appear. If a parameter exists, it is marked and the actual parameter from the macro call will be substituted when the macro is called. Formal parameters may exist anywhere in the macro body including in the comment field. A formal parameter in the macro body is indicated by a sharp sign (#) just as in the macro heading.

For every macro definition there is an internally defined macro parameter indicated by #$YM. This parameter may be referenced in the macro body but should not appear in the formal parameter list. When the macro is called, each occurrence of #$YM in the macro body is replaced by a string representing a 4 digit hexadecimal constant, e.g. 0001. The four digit string is constant over a given level of macro expansion and increases by one for each macro call. The typical usage of the #$YM string is to provide unique labels to a macro that is expanded multiple times so as to avoid a duplicate label error. This may also be done however, by use of the LOCAL directive.

Macro Terminator

The ENDM directive terminates the macro definition. During a Macro definition, an ENDM must be found before another MACRO or MACR statement may be used. an END statement that is found during a macro definition will terminate the macro definition as well as the assembly. The format of the ENDM is as follows:

```
      {label} | ENDM
```

where:

label - is an optional symbol which becomes the symbolic address of the first byte of memory following the inserted macro.

Macro Call

A macro may be called by encoding the macro name in the operation field of the statement. The format of the macro call is shown below.

```
    {label} | name | {parameter list}
```

where:

label - is an optional label which will be assigned a value equal to the address of the first instruction in the macro.

name - is the name of the macro called. This name should be defined by the MACRO or MACR directive or an error message will be generated.

parameter list - is a list of parameters separated by commas. These parameters may be constants, expressions, symbols, character strings or any other text separated by commas.

The parameters in the macro call are actual parameters and their names may be different than the formal parameters used in the macro definition. The actual parameters will be substituted for the formal parameters in the order in which they are written. Commas may be used to reserve a parameter position. In this case the parameter will be null. Any parameters not specified will also be null. The parameter list is terminated by a blank, tab, or a semicolon.

All actual parameters are passed as character strings into the macro definition statements. Thus symbols are passed by name and not by value. In other words, the parameters are not evaluated until the macro expansion is produced. Thus DEFL directives within a macro may alter the value of parameters passed to the macro.

During the macro expansion, the assembler recognizes certain characters to have special meaning. The ampersand, "&", is used to concatenate the text of the definition line and any actual parameters. During macro expansions, an ampersand immediately preceding or immediately following a formal parameter is removed and the substitution of the actual parameter occurs at that point. If the ampersand is not immediately adjacent to the parameter, the ampersand is not removed and remains part of the definition line.

Single quotes are used to delimit actual parameters that may contain other delimiters. All characters between the quotes are considered part of the parameter and the quotes are removed before being substituted for the formal parameters. Single quotes are the only way to pass a parameter that contains a blank, comma, tab, or other delimiter. For example, to use the instruction "LD HL,0" as an actual parameter, would require placing 'LD HL,0' in the actual parameter list. A null parameter may consist of the quotes with no intervening characters. A quote in the actual parameter is represented by two quotes in sequence.

An example of a macro call and its expansion is shown below. Note the use of concatenation and the special #$YM parameter. Expanded macro code is marked with plus signs.

```
Definition:        GET     MACRO    #X,#Y,#Z
                           LD       B,#X&.AND.0FH
                           #Y
                   #Z      JP       C,MAIN
                           ADD      HL,HL
                   L#$YM   SET      0,C
                           ADD      A,C
                           ENDM


Macro Call:                -
                           -
                           SCF
                   LOOP    GET      200,'INC     B',ENTRY
                           JR       NZ,GO
                           -
                           -


Source Code
Generated:                 -
                           -
                           SCF
                   LOOP    GET      200,'INC     B',ENTRY
                  +        LD       B,200.AND.0FH
                  +        INC      B
                  +ENTRY   JP       C,MAIN
                  +        ADD      HL,HL
                  +L0001   SET      0,C
                           ADD      A,C
                           JR       NZ,GO
                           -
                           -
```

LOCAL — Define Local Symbol

As all labels, including those within macros, are global to the complete program, a macro which contains a label and which is called more than once will cause a duplicate label error to be generated. To avoid this problem, the user may declare labels within macros to be "local" to the macro. Each time the macro is called the assembler assigns each local symbol a system generated symbol of the form ??nnnn. Thus the first local symbol will be ??0001, the second ??0002, etc. The assembler does not start at ??0001 for each macro but increases the count for each local symbol encountered. The symbols defined in the LOCAL directive are treated like formal macro parameters and hence may be used in the operand field of instructions. The operand field may not contain any formal parameters defined on the MACRO directive line. As many LOCAL directives as necessary may be included within a macro definition, but they must occur immediately after the MACRO or MACR directive and before the first line of the macro body. LOCAL directives will not appear in the output listing during a macro expansion. LOCAL directives that appear outside a macro definition will generate an error. To avoid duplicate labels within macros, the user may of course use the #$YM symbol.

Example:

Definition:

```
WAIT    MACRO   #R
        LOCAL   #LAB1
        LD      B,#R
#LAB1   DEC     B
        JR      NZ,#LAB1
        ENDM
```

```
First call        +          LD      B,5
with R = 5        +??0001    DEC     B
                  +          JR      NZ,??0001


Second call       +          LD      B,OFFH
with R = OFFH     +??0002    DEC     B
                  +          JR      NZ,??0002
```

```
            LOCAL | symbol list
```

where:

symbol list - is a list of parameters similar to those used on the MACRO directive that are to defined local to this macro. These local symbols must be separated by commas.

EXITM — Alternate Macro Exit

The EXITM directive provides an alternate method for terminating a macro expansion. During a macro expansion, an EXITM directive causes expansion of the current macro to stop and all code between the EXITM and the ENDM for this macro to be ignored. If macros are nested, EXITM causes code generation to return to the previous level of macro expansion. Note that an EXITM or an ENDM may be used to terminate a macro expansion, but only an ENDM may be used to terminate a macro definition.

In the following example the code following the EXITM will not be assembled if DATA is zero.

```
STORE    MACRO    #DATA
          -
          -
         IF       #DATA
         EXITM
          -
          -
         ENDM
```

```
    {label} | EXITM
```

where:

label - is an optional label which will be given the address of the instruction assembled after the macro terminates.

RELOCATION

The object module produced by this assembler is in a relocatable format. This allows users to write programs whose final addresses will be adjusted by Microtec's Linking Loader and which may also be changed without reassembling the complete program. It also allows separate object modules to be linked together into a final program.

Relocatable programming provides many advantages for the user. Actual memory addresses are of no concern until the final load time. Large programs may be easily separated into smaller segments, developed separately, and linked together. If one segment contains an error, only it need be reassembled. A library of routines may be used by many users once developed. The Loader will adjust addresses to meet each user's requirements.

To take advantage of relocatability, the user should understand the concept of program segments and how separate object modules are linked together. A program segment is that part of a program which contains its own program counter and is a logically distinct section of the program. At load time the addresses for each segment may be specified separately.

This assembler provides for four program segments. The CODE segment is typically the segment that contains the actual machine instructions. In a ROM/RAM system it would be the segment that would be placed into ROM. The data area of a program is typically placed into the DATA segment. This segment usually resides in RAM. This segment could contain actual machine instructions. The STACK segment is used to contain the program stack area and resides in RAM. Typically only the main program makes references to the STACK segment and specifies the STACK segment length. References are made to the stack segment with the reserved symbol STACK. The MEMORY segment is that portion of memory space not allocated to the other three segments. References are made to this segment with the reserved symbol MEMORY.

Although users may place actual code in the CODE or DATA segments, only references may be made to the STACK and MEMORY segments at assembly time.

As with non relocatable assemblers, users may also specify absolute addresses when assembling a program. In this case the object module will contain an absolute program designed to run in a particular memory location.

The object modules of the assembler are combined or linked together by a Linking Loader. The Loader converts all relocatable addresses into absolute addresses and resolves references from one module to another. Linkage between modules is provided by PUBLIC and EXTRN symbols. PUBLIC symbols are defined in one object module and made available to all other object modules via the Linking Loader. EXTRN symbols are symbols referenced in one module but defined in another module. The Linking Loader links the PUBLIC's from one module with the EXTRN's from other modules to resolve these references. A program may contain both PUBLIC and EXTRN symbols.

Relocatable Symbols

Each symbol in the assembler has associated with it a symbol type which denotes the symbol as absolute or relocatable, and the program segment to which the symbol belongs. Symbols whose values do not change value depending upon program origin are absolute symbols. Symbols whose value change when the program origin is changed by the Linking Loader are termed relocatable symbols. The reserved symbols STACK and MEMORY discussed above are special forms of relocatalbe symbols. EXTRN symbols are also relocatable. Absolute and relocatable symbols may both appear in an absolute or relocatable segment.

Absolute symbols are defined as follows:

1. A symbol is in the label field when the program is assembling an absolute segment of code.
2. A symbol is defined equal to an absolute expression by the EQU or DEFL directives. This occurs even if the program is assembling a relocatable segment.

Relocatable symbols are defined as follows:

1. A symbol is in the label field when the program is assembling a CODE or DATA segment of code.
2. A symbol is defined equal to a relocatable expression by the EQU or DEFL directives.
3. The reserved symbols STACK and MEMORY are relocatable.
4. External (EXTRN) symbols are relocatable
5. A reference to the program counter ($) while assembling a relocatable segment is relocatable.

Relocatable symbols are also classified as CODE, DATA, STACK, or MEMORY relocatable depending upon how they were defined.

Relocatable Expressions

The relocatability of an expression is determined by the relocation of the symbols that comprise the expression. All numeric constants are considered absolute. Relocatable expressions may be combined to produce an absolute expression, a relocatable expression or in certain instances illegal expressions. The following list shows those expressions whose result is relocatable. ABS denotes an absolute symbol or constant and REL denotes a relocatable symbol.

```
ABS+REL
REL+ABS
REL-ABS
.LOW.REL
.HIGH.REL
```

In addition the following expressions are valid and produce an absolute expression. Both relocatable expression must be relocatalbe in the same program segment.

```
REL-REL
REL.EQ.REL
REL.GT.REL
REL.LT.REL
REL.UGT.REL
REL.ULT.REL
```

Relocatable symbols that appear in expressions with any other operators will cause an error, e.g. REL*REL. Any combination of two relocatable symbols from different segments including externals (EXTRN) is an error condition.

Relocation Directives

The following pages describe those directives in the assembler that pertain primarily to relocation. The nomenclature is the same as for the directives described in Section 4. The directives are:

ASEG	Specify Absolute Segment
CSEG	Specify Code Segment
DSEG	Specify Data Segment
ORG	Specify Origin
PUBLIC	Specify PUBLIC symbols
EXTRN	Specify External symbols
NAME	Specify Module Name
STKLN	Specify Stack Length

ASEG — Specify Absolute Segment

The ASEG directive specifies to the assembler that the following statements should be assembled in the absolute mode. The ASEG remains in effect until a CSEG or DSEG directive is assembled. The starting address for the ASEG program counter is zero. At the start of the assembly, the program assumes an ASEG directive has been specified and assembly proceeds in the absolute mode.

```
    {label} | ASEG
```

where:

label - is an optional label that will be assigned the address of the next assembled instruction.

CSEG — Specify Code Segment

The CSEG directive specifies to the assembler that the following statements should be assembled in the relocatable mode using the CODE segment program counter. Initially the CODE segment program counter is set to zero. In addition, this directive may specify an operand which is passed to the Loader and has no effect on the assembly. The operand is described below.

Example:

```
                    CSEG    PAGE
```

{label}	CSEG	{blank,PAGE,INPAGE}

where:

label - is an optional label which will be assigned the address of the next instruction.

blank - a blank operand field specifies that the CODE segment may be relocated to the next available byte.

PAGE - specifies that the CODE segment must begin on a page boundary (i.e. 0,100H,200H,...) when relocated by the Linking Loader.

INPAGE - specifies that the CODE segment must fit within a single page when relocated. The Loader will start the segment at the next page boundary if the segment will not fit within the current page.

Note: if multiple CSEG directives are specified in the same assembly, each must specify the same operand.

DSEG — Specify Data Segment

The DSEG directive specifies to the assembler that the following statements should be assembled in the relocatable mode using the DATA segment program counter. Initially the DATA segment program counter is set to zero. In addition, this directive may specify an operand which is passed to the Loader and has no effect on the assembly. The operand is described below.

Example:

```
                         DSEG   INPAGE
```

{label}	DSEG	{blank,PAGE,INPAGE}

where:

label - is an optional label which will be assigned the address of the next instruction.

blank - a blank operand field specifies that the DATA segment may be relocated to the next available byte during Loading.

PAGE - specified that the DATA segment must begin on a page boundary (i.e. 0,100H,200H,...) when relocated by the Linking Loader.

INPAGE - specifies that the DATA segment must fit within a single page when relocated. The Loader will start the segment at the next page boundary if the segment will not fit within the current page.

Note: if multiple DSEG directives are specified in the same assembly, each must specify the same operand.

ORG — Set Program Origin (relocatable mode)

The ORG directive is used to inform the assembler of the memory address to which the next assembled byte should be assigned. This directive changes the program counter of the segment which is currently being assembled, absolute, code or data. When the ORG is in a relocatable program segment, the origin address must be an absolute expression of a relocatable expression which is relocatable within the current segment.

Example:

```
            ORG   $+30H
```

{label}	ORG	expression

where:

label - is an optional label which will be equated to the given expression.

expression - a value which will replace the contents of the current segment program counter. Any symbols used in the expression must be previously defined.

PUBLIC — Specify PUBLIC symbols

The PUBLIC directive specifies a list of symbols which will be given the PUBLIC attribute. These symbols will then be made available to other modules to establish the necessary linkage between modules. Only those symbols declared PUBLIC and defined in the assembly are placed in the object module and made available to other object modules.

The PUBLIC directive may appear anywhere in the program and each symbol may be declared in only one PUBLIC directive.

Example:

```
        PUBLIC   SCAN,LABEL,SYMBOL
```

{label}	PUBLIC	symbol list

where:

label - is an optional label which will be assigned the address of the next instruction.

symbol list - is a list of symbols separated by commas which specify the PUBLIC names available to other modules.

EXTRN — Specify External Symbols

The EXTRN directive specifies a list of symbols which will be given the EXTRN attribute. These are symbols that are referenced in this program module but defined within another program. This directive provides the linkage to those symbols through the Linking Loader.

The EXTRN directive may appear anywhere in the program and each symbol may be declared in only one EXTRN directive.

Example:

```
          EXTRN   INPUT,OUTPUT
```

{label}	EXTRN	symbol list

where:

label - is an optional label which will be assigned the address of the next instruction.

symbol list - is a list of symbols separated by commas which specify the EXTRN names available in other modules.

NAME — Specify Module Name

The NAME directive is used to assign a name to the object module produced by the assembly. Only one NAME directive may appear in a program. The module name is a handle used by the Linking Loader when combining programs.

If no NAME directive is specified by the user, the default name "MODULE" is used.

Example:

```
                    NAME  MULT
```

```
   {label} | NAME | name
```

where:

label - is an optional label which will be assigned the address of the next instruction.

name - is the name to be placed in the object module to denote the module name to the Loader. This name must follow all the rules of a symbol.

STKLN — Specify Stack Length

The STKLN directive allows the user to specify the length of the STACK segment generated by the Linking Loader. Typically this directive is only used in the main program, but other programs may also specify a stack length. The Loader combines all STACK segments into one segment.

If the user does not specify a STKLN directive, the assembler uses a default length of zero. More that one STKLN directive may be placed in a program, only the last one is used.

Example:

```
                    STKLN  20H
```

{label}	STKLN	expression

where:

label - is an optional label which will be assigned the address of the next instruction.

expression - an expression which indicates the length of the stack segment. This expression may not contain a relocatable symbol.

HOW TO USE THE ASSEMBLER

The Assembler

The Assembler program is usually supplied as an unlabeled unblocked magnetic tape with 80 character card image records. Other media may be requested.

The Assembler is written entirely in Fortran and is comprised of a main program and several subroutines. The main program appears first on the tape and the last subroutine is followed by a tape mark. The Assembler may be compiled from the tape.

The Assembler Installation Notes describe program installation and any modification that may have to take place for a particular computer. It is helpful to read these notes before installing the program.

Assembler Operation

The Assembler is a two pass Assembler wherein the source code is scanned twice. During the first pass the labels are examined and placed into a symbol table. Certain errors may be detected during Pass One; these will be displayed on the output listing.

During Pass Two, the object code is completed, symbolic addresses resolved, a listing and object module are produced. Certain errors, not detected during Pass One may be detected and displayed on the listing.

At the end of the Assembly process a symbol table or cross reference table may be displayed.

The following steps are taken to assemble a source program:

1. Write a program utilizing instruction mnemonics and directives. Encode the arguement fields with constants labels, symbolic addresses, etc.

2. Transfer the source program to some computer readable medium; cards, tape, etc. This medium should correspond to the input device expected by the Assembler. On some systems, device assignments may be changed during the course of an assembly by utilizing proper system control cards.

3. Include the source code as shown in the sequence in Illustration I.

4. Execute the Assembler Program.

5. Get listing and object module as output.

Assembler Listing

During Pass Two of the assembly process a program listing is produced. The listing displays all information pertaining to the assembled program; both assembled data and the users original source statements.

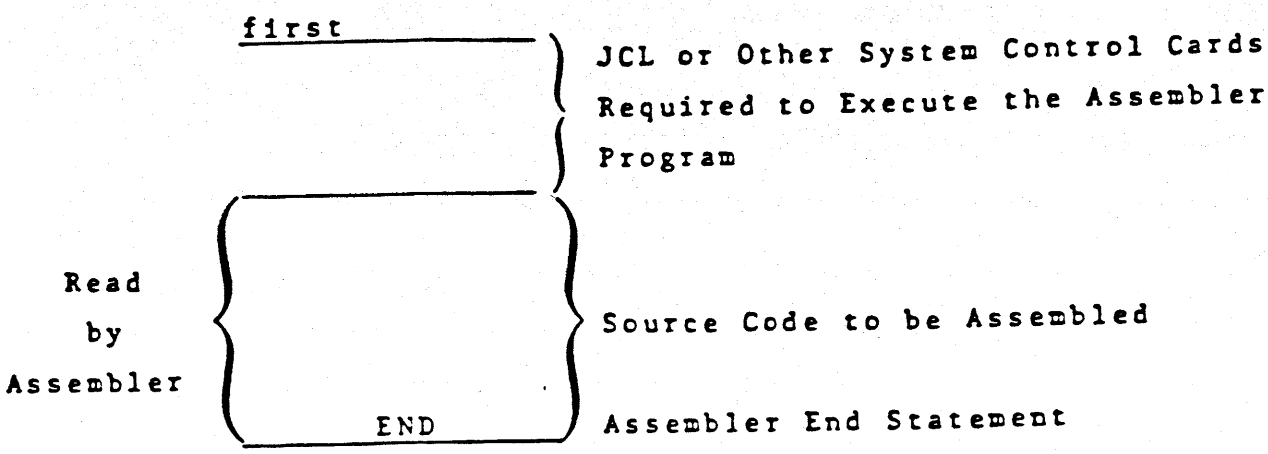
The listing may be used as a documentation tool through the inclusion of the comments and remarks that describe the function of the particular program segment.

The main purpose of the listing is to convey all pertinent information about the assembled program, i.e. the memory addresses and their contents. The load module, also produced during Pass Two, contains the address and content information but in a format that can be read only with great effort.

CARD ORDER

Illustration I

Read the Input Stream


first
JCL or Other System Control Cards Required to Execute the Assembler Program
Read by Assembler
Source Code to be Assembled
END
Assembler End Statement

The illustration on page 7-6 is a sample of a typical program listing. Referring to the listing illustration, the following information is pertinent:

- The assembler may detect error conditions during the assembly process. The column titled "ERR" will contain the error code(s) should the assembler detect one or more errors in the associated line or source code. An explanation of the individual error codes is given in Appendix A.

- The column titled "LINE" contains decimal numbers which are associated with the listing line numbers. The maximum number of lines is a source program is 9999.

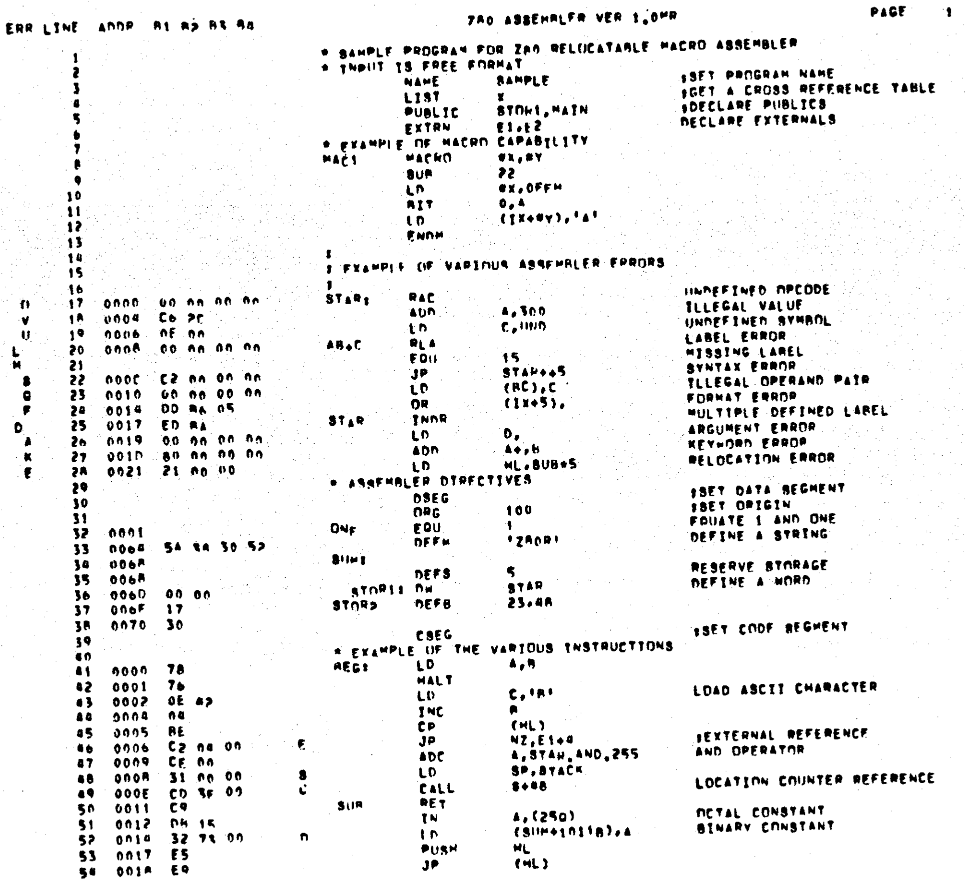
- The column titled "ADDR" contains a value which represents the first memory address of the data shown in bytes one to four on a given line or the value of an EQU or SET directive. The hexadecimal number under B1 represents one byte of data to be stored in the memory address. If there is a number under B2 it represents data to be stored in the given memory address plus one. Columns B3 and B4, if they contain a number, similarly represent data to be stored in the memory address plus two or three.

- To the right of the data bytes are the relocation types of any relocatable operands. The types are as follows: C - code, D - data, S - stack, M - memory, E - external.

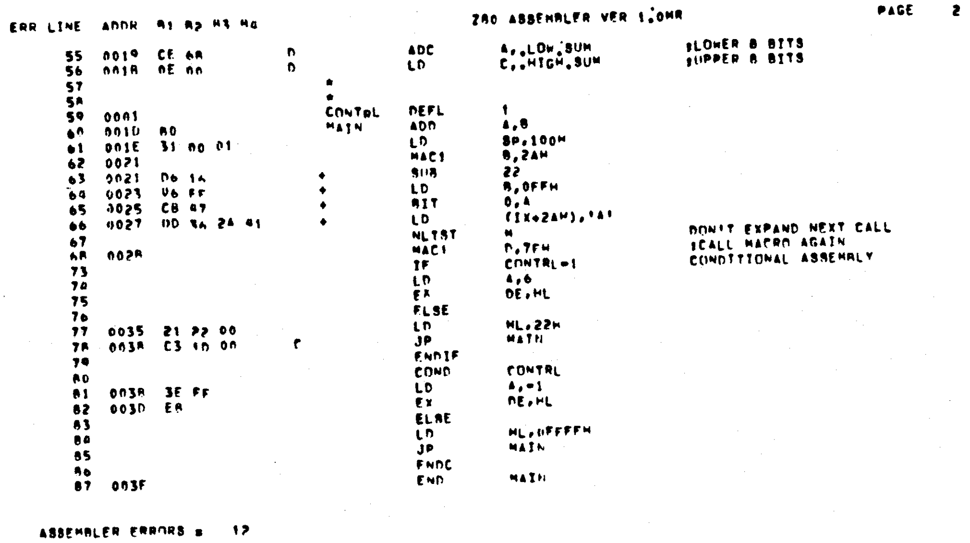
- The users original source statements are reproduced without alteration to the right of the above information. Macro expansions are preceded with a plus sign.

- At the end of the listing the assembler prints the message "ASSEMBLER ERRORS = " with a cumulative count of errors. The assembler substitutes four bytes of NOP's when it cannot translate a particular opcode and so provides room for patching the program if desired.

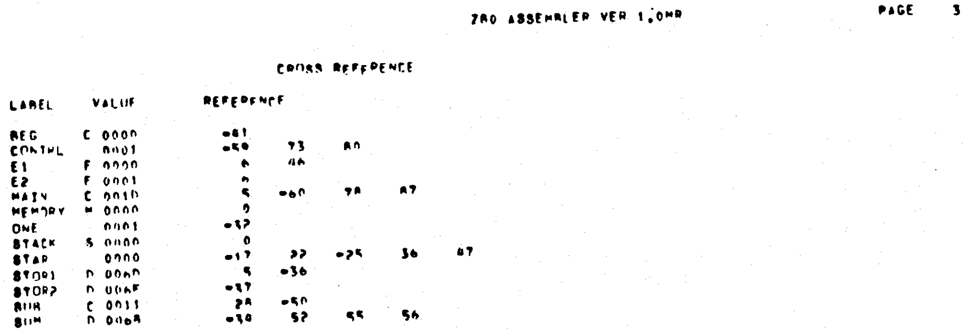
- A symbol table or cross reference table is generated at the end of each assembly listing. The table lists all symbols utilized in alphabetic order along with any relocation types as described above.



```
ERR LINE  ADDR  B1 B2 B3 B4                 Z80 ASSEMBLER VER 1.0MR                      PAGE    1

         1                           * SAMPLE PROGRAM FOR Z80 RELOCATABLE MACRO ASSEMBLER
         2                           * INPUT IS FREE FORMAT
         3                                   NAME      SAMPLE              ;SET PROGRAM NAME
         4                                   LIST      X                   ;GET A CROSS REFERENCE TABLE
         5                                   PUBLIC    STOR1,MAIN          ;DECLARE PUBLICS
         6                                   EXTRN     E1,E2               DECLARE EXTERNALS
         7                           * EXAMPLE OF MACRO CAPABILITY
         8                           MAC1    MACRO     #X,#Y
         9                                   SUB       22
        10                                   LD        #X,0FFH
        11                                   BIT       0,A
        12                                   LD        (IX+#Y),'A'
        13                                   ENDM
        14                           ;
        15                           ; EXAMPLE OF VARIOUS ASSEMBLER ERRORS
        16                           ;
 O      17  0000  00 00 00 00        STAR:   RAC                           UNDEFINED OPCODE
 V      18  0004  C6 2C                      ADD       A,300               ILLEGAL VALUE
 U      19  0006  0E 00                      LD        C,UND               UNDEFINED SYMBOL
L       20  0008  00 00 00 00        AB+C    RLA                           LABEL ERROR
M       21                                   EQU       15                  MISSING LABEL
  S     22  000C  C2 00 00 00                JP        STAR++5             SYNTAX ERROR
  Q     23  0010  00 00 00 00                LD        (BC),C              ILLEGAL OPERAND PAIR
  F     24  0014  DD B6 05                   OR        (IX+5),             FORMAT ERROR
 D      25  0017  ED BA              STAR    INDR                          MULTIPLE DEFINED LABEL
  A     26  0019  00 00 00 00                LD        D,                  ARGUMENT ERROR
  K     27  001D  80 00 00 00                ADD       A+,B                KEYWORD ERROR
  E     28  0021  21 00 00                   LD        HL,SUB*5            RELOCATION ERROR
        29                           * ASSEMBLER DIRECTIVES
        30                                   DSEG                          ;SET DATA SEGMENT
        31                                   ORG       100                 ;SET ORIGIN
        32  0001                     ONE     EQU       1                   EQUATE 1 AND ONE
        33  0064  5A 38 30 52                DEFM      'Z80R'              DEFINE A STRING
        34  0068                     SUM:
        35  0068                             DEFS      5                   RESERVE STORAGE
        36  006D  00 00                STOR1: DW       STAR                DEFINE A WORD
        37  006F  17                 STOR2   DEFB      23,48
        38  0070  30
        39                                   CSEG                          ;SET CODE SEGMENT
        40                           * EXAMPLE OF THE VARIOUS INSTRUCTIONS
        41  0000  78                 REG:    LD        A,B
        42  0001  76                         HALT
        43  0002  0E 42                      LD        C,'B'               LOAD ASCII CHARACTER
        44  0004  04                         INC       B
        45  0005  BE                         CP        (HL)
        46  0006  C2 04 00       E           JP        NZ,E1+4             ;EXTERNAL REFERENCE
        47  0009  CE 00                      ADC       A,STAR.AND.255      AND OPERATOR
        48  000B  31 00 00       S           LD        SP,STACK
        49  000E  CD 3F 00       C           CALL      $+48                LOCATION COUNTER REFERENCE
        50  0011  C9                 SUB     RET
        51  0012  DB 15                      IN        A,(25O)             OCTAL CONSTANT
        52  0014  32 73 00       D           LD        (SUM+1011B),A       BINARY CONSTANT
        53  0017  E5                         PUSH      HL
        54  0018  E9                         JP        (HL)
```



```
ERR LINE  ADDR  B1 B2 B3 B4                 Z80 ASSEMBLER VER 1.0MR                      PAGE    2

        55  0019  CE 68          D           ADC       A,.LOW.SUM          ;LOWER 8 BITS
        56  001B  0E 00          D           LD        C,.HIGH.SUM         ;UPPER 8 BITS
        57                           *
        58                           *
        59  0001                     CONTRL  DEFL      1
        60  001D  80                 MAIN    ADD       A,B
        61  001E  31 00 01                   LD        SP,100H
        62  0021                             MAC1      B,2AH
        63  0021  D6 16             +        SUB       22
        64  0023  06 FF             +        LD        B,0FFH
        65  0025  CB 47             +        BIT       0,A
        66  0027  DD 36 2A 41       +        LD        (IX+2AH),'A'
        67                                   NLIST     M                   DON'T EXPAND NEXT CALL
        68  002B                             MAC1      D,7FH               ;CALL MACRO AGAIN
        73                                   IF        CONTRL=1            CONDITIONAL ASSEMBLY
        74                                   LD        A,6
        75                                   EX        DE,HL
        76                                   ELSE
        77  0035  21 22 00                   LD        HL,22H
        78  0038  C3 1D 00       C           JP        MATN
        79                                   ENDIF
        80                                   COND      CONTRL
        81  003B  3E FF                      LD        A,-1
        82  003D  EB                         EX        DE,HL
        83                                   ELSE
        84                                   LD        HL,0FFFFH
        85                                   JP        MAIN
        86                                   ENDC
        87  003F                             END       MAIN

  ASSEMBLER ERRORS =    12
```



```
                                            Z80 ASSEMBLER VER 1.0MR                      PAGE    3


                           CROSS REFERENCE

 LABEL    VALUE       REFERENCE

 REG      C 0000        #41
 CONTRL     0001        #59     73     80
 E1       E 0000          6     46
 E2       E 0001          6
 MAIN     C 001D          5    #60     78     87
 MEMORY   M 0000          9
 ONE        0001        #32
 STACK    S 0000          0
 STAR       0000        #17     22    #25     36     47
 STOR1    D 006D          5    #36
 STOR2    D 006F        #37
 SUB      C 0011         28    #50
 SUM      D 0068        #34     52     55     56
```

The Object Module

As part of the Pass Two processing, the assembler produces an object module. The object module is a machine readable computer output in the form of punched cards, paper tape, etc. The output module contains specifications for loading the memory of the target microprocessor and provide the necessary linkage to link object modules together.

The object module is normally punched out on the device specified. However, through use of the LIST and NLIST directives, all or part of the output may be deleted.

The object module is produced as a series of card images on the output punch device. The object module is compatible with Intel's relocatable format although it is produced in a readable as opposed to a binary format.

The object module may be loaded into Microtec's Linking Loader which will then convert it to an absolute program in Intel's standard hexadecimal format. This may then be loaded into a development system or used to program a PROM.

A sample object module is shown on the following page. This is the object module of the sample program shown on the preceding pages.

```
22E0006SAMPLE00   013E00030271000303000030400000387
8160000E1****0000E2****00A3
61200011D0006MAIN**0007
61200026D0006STOR1*00F1
63A00000000000000000062C0E000000000000C2000000000000000DDB605EDDABF
61E00001900000000000000000000021000022
6100002E4005A36305270
61000026D000000173034
6300001000078760E4204BEC20400CE0031000C03E00C9DB1540
:20800030F00C4
:40A0003030C0CC0
:00C000300000700CA
61200011400327300E5E960
:40A0002031500B8
60C00011900CE689E
:40A0002011A00B5
63C000118000E0080310001D61606FFC847DD362A41DE1616FFC847DD367F417B
:40A0002021C00B2
61A00013500212200C31D0C3EFFEB5F
22080003390C9A
40A0001011D00D3
E0200F0
```

Cross Reference Format

The cross reference option is normally turned off. To turn it on use "LIST X", to turn it off again use "NLIST X" (see LIST and NLIST directives). The assembler will produce either a cross reference table or a symbol table. The cross reference table will be produced if "LIST X" has been specified. References may only be accumulated during particular portions of the program by turning the cross reference option on and off. However, to get the listing of cross references, the option must be turned on before the END statement. Typically the "LIST X" directive will be one of the first statements in the source and never turned off.

An example of the cross reference output is as follows:

```
LABEL     VALUE        REFERENCE

ABC       F45A         -4   15   35  -77
MAIN    C 0000         -1  104
MEMORY  M 0000          0
PRINT   E 0003         -5   23
```

LABEL and VALUE are self explanatory. Any flags on the left of the value are the relocation types of the symbols as explained under the Assembler Listing section. Under REFERENCE, a value preceded by a minus sign indicates that the symbol was defined on that line. A value of 0 as the only entry on the line indicates this is an internal system symbol (e.g. MEMORY, STACK). Line numbers not preceded by a minus sign indicate a reference to the svmbol on that line. For DEFL symbols, more than one definition may appear for a given symbol as in ABC above. Internal assembler keywords, e.g. A,HL, etc. are not shown on the cross reference listing.

APPENDIX A

ASSEMBLER ERROR CODES

If errors in the source code are detected during the assembly process, an indication of the type of error is printed on the listing on the same line as the statement in error.

The following list should serve as a guide to diagnose the error. The listing always displays a total error count.

A - Argument error. The argument is missing or contains an illegal character. Argument for CSEG or DSEG directive must match previous use of argument.

B - Branch error. A relative branch instruction is attempting to branch to a location which is out of range for the relative address.

C - Macro substitution error. When substituting actual macro parameters for formal parameters, the 80 column limit was exceeded.

D - Duplicate Label error. The label in the statement has previously appeared in the label field. A label on a DEFL directive previously appeared in a statement other than a DEFL or a label on a statement other than a DEFL statement now appears on a DEFL statement. A label appears more than once in an EXTRN or PUBLIC directive or a symbol defined in an EXTRN directive appears in the label field of some statement.

E - Relocation error. The instruction contains an operand that violates a rule of relocation. An operand that should be absolute is relocatable or an EQU or DEFL directive make reference to an external (EXTRN) symbol.

F - Format error. The instruction has been written in a format which is not permitted. This error usually indicates a trailing comma and the instruction is assembled properly.

K - Keyword error. A keyword has been found which does not have the proper syntax or should have parenthesis but does not or vice versa. E.g. LD (A),B

L - Label error. A label contains an invalid character of starts with a numeric character.

M - Missing Label. This statement requires a label.

N - Macro Nesting error. When nesting macros the tables used to hold the nesting information has become full.

O - Opcode error. The opcode mnemonic has not been recognized as a valid mnemonic, directive, or a macro call. Also a macro defined within another macro or conditional statements nested too deeply. ELSE, ENDIF, ENDC, ENDM, or EXITM used without preceding IF or MACRO statement. LOCAL directive used outside or MACRO body or more than one NAME directive in a program.

Q - Questionable operands. The combination of operands is not valid for the opcode. E.g. LD (HL),(HL).

S - Syntax error. A rule of syntax has been violated in the statement. Parenthesis are not nested properly or possibly two operators appear in sequence.

T - Table overflow. Symbol table is full - assembly continues. An attempt was made to define too many macros, or too many parameters in nested macro calls. Also too many formal parameters for a given macro definition.

U - Undefined symbol. There is a symbolic name in the operand field which has never been in the label field. The symbol should have been previously defined for certain directives and was not but may have been defined after the directive. Possibly the user is attempting to use an external symbol that was not defined in an EXTRN directive.

V - Value error. An evaluated expression or constant is out of range for the field of the actual machine instruction in which it is to be contained. A one byte value is relocatable but was not preceded by a .LOW. or .HIGH. operator. In this case it is forced to .LOW.

CROSS REFERENCE OVERFLOW AT ____. The cross reference file has been filled. Assembly continues and references are not accumulated past this line. This message appears in the cross reference table listing. Enlarge cross reference file space or turn reference off for sections of the program.

APPENDIX B

ASCII AND EBCDIC CODES

The Assembler will recognize only the following characters. The equivalent codes are expressed in hexadecimal notation.

CHARACTER	ASCII	EBCDIC
Ø	3Ø	FØ
1	31	F1
2	32	F2
3	33	F3
4	34	F4
5	35	F5
6	36	F6
7	37	F7
8	38	F8
9	39	F9
A	41	C1
B	42	C2
C	43	C3
D	44	C4
E	45	C5
F	46	C6
G	47	C7
H	48	C8
I	49	C9
J	4A	D1
K	4B	D2
L	4C	D3
M	4D	D4
N	4E	D5
O	4F	D6
P	5Ø	D7
Q	51	D8
R	52	D9
S	53	E2
T	54	E3
U	55	E4
V	56	E5

CHARACTER	ASCII	EBCDIC
W	57	E6
X	58	E7
Y	59	E8
Z	5A	E9
blank	2Ø	4Ø
!	21	5A
"	22	7F
#	23	7B
$	24	5B
%	25	6C
&	26	5Ø
'	27	7D
(	28	4D
)	29	5D
*	2A	5C
+	2B	4F
,	2C	6B
-	2D	6Ø
.	2E	4B
/	2F	61
:	3A	7A
;	3B	5E
<	3C	4C
=	3D	7E
>	3E	6E
?	3F	6F
@	4Ø	7C
\	5C	EØ
|	5E	4F
_	5F	6D

APPENDIX C

HEXADECIMAL NOTATION

Hexadecimal notation is a convenient way to express binary information. Each hexadecimal digit may be thought of as representing the information in four binary bits.

The assembled code is expressed in hexadecimal notation on the output listing. Hexadecimal is the name of the base 16 number system.

DECIMAL	HEXADECIMAL	BINARY
Ø	Ø	ØØØØ
1	1	ØØØ1
2	2	ØØ1Ø
3	3	ØØ11
4	4	Ø1ØØ
5	5	Ø1Ø1
6	6	Ø11Ø
7	7	Ø111
8	8	1ØØØ
9	9	1ØØ1
1Ø	A	1Ø1Ø
11	B	1Ø11
12	C	11ØØ
13	D	11Ø1
14	E	111Ø
15	F	1111

Appendix D

HEXADECIMAL-DECIMAL CONVERSION TABLE

This table allows conversions to be made between hexadecimal and decimal numbers. The table has a decimal range of 0 to 4095. To convert larger numbers add the following values to the table values.

Hexadecimal	Decimal
1000	4096
2000	8192
3000	12228
4000	16384
5000	20480
6000	24576
7000	28672
8000	32768
9000	36864
A000	40960
B000	45056
C000	49152
D000	53248
E000	57344
F000	61440

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont'd)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0331	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0301	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	.0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0212	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont'd)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont'd)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

HEXADECIMAL-DECIMAI INTEGER CONVERSION (Cont'd)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	4761	2762	2763	2764	2765	2766	2767
AD0	2768	2769.	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	3851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2866	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
B80	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont'd)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
D80	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
CC0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MN 55440
SALES OFFICE AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

CONTROL DATA CORPORATION