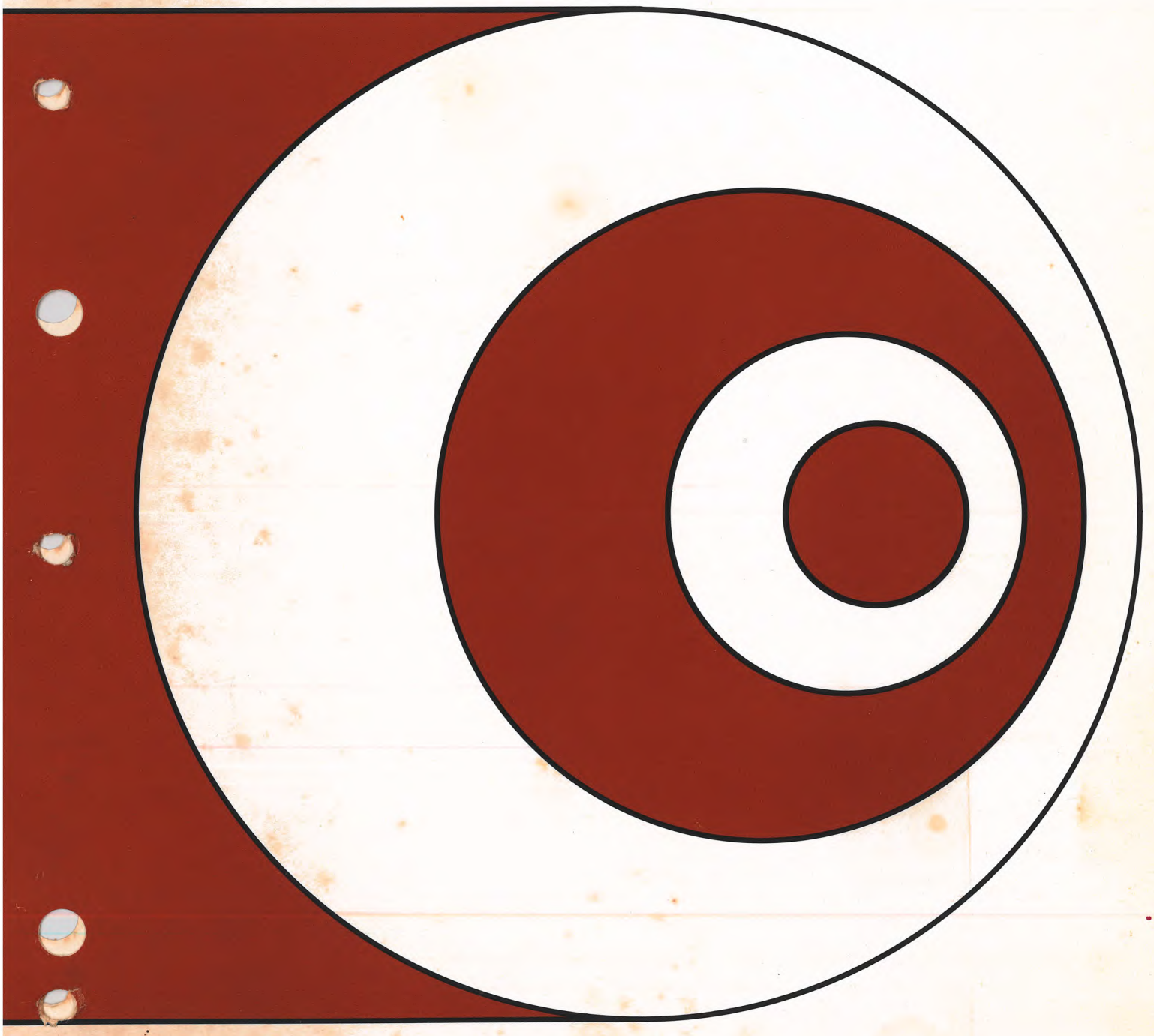
SIGNETICS TWIN 2650 RELOCATABLE ASSEMBLER MANUAL



$4.00

SIGNETICS TWIN 2650 RELOCATABLE ASSEMBLER MANUAL

signetics
a subsidiary of U.S. Philips Corporation

Signetics Corporation
P.O. Box 9052
811 East Arques Avenue
Sunnyvale, California 94086
Telephone 408/739-7700

REVISION RECORD

REVISION	DESCRIPTION
	Initial Release (1-15-79)
A	Revised instructions for use of ORG assembler directive (4-1-79)
TW09007000 JJ	

Signetics TWIN 2650 Relocatable Assembler Manual

Address comments concerning this manual to:

Applications Department
MOS Microprocessors
Signetics Corporation
811 E. Arques Avenue
Sunnyvale, California 94086

CONTENTS

SECTION V – DIRECTIVES AND EXTENDED INSTRUCTION SET

SECTION VI – CONDITIONAL ASSEMBLY

SECTION VII – RELOCATABILITY/RELOCATABLE ASSEMBLY

SECTION VIII – LINK EDITOR

SECTION IX – TWIN RESIDENT MACRO FACILITY

LIST OF ILLUSTRATIONS

FOREWORD

The TWIN Relocatable Assembler (RASM) provides features, designed for the development of larger software systems, which are not available with the TWIN Absolute Assembler (ASM). For RASM version 3.0, the list of features include:

1. A relocatable output is produced, so that locations in which a program is to be executed can be determined after assembling.

2. It is possible to introduce items which are defined in other programs so links between separately assembled programs can be established mechanically.

3. The mnemonic branch-instruction codes are extended by the inclusion of the condition in the mnemonic, increasing the readability of programs. For example, BER is equivalent to BCTR,0.

4. ASCII data constants and other types of data constants may be combined into a single statement.

5. Data definitions may be preceded by a multiplication factor.

6. The number of bytes in a character string (ASCII data constant) can be specified explicitly.

7. The new assembler accepts most correct ASM programs. Major differences are in the ORG and END definitions.

In addition to RASM, a link editor, called LINK, has been constructed to provide the capability of linking separately assembled programs. RASM produces an object module and LINK links a number of these object modules and edits the modules into one or more load modules. The modules can be loaded by means of the SDOS commands LOAD/GO and XEQ.

NOTE

> The object file output of RASM is not compatible with the hexadecimal output of RASM. The WHEX command can be used to reformat LINK output into hexadecimal format.

RASM assembles source code for the 2650B microprocessor and because the 2650A instruction set is a proper subset of 2650B assembly language, 2650A code may also be assembled with RASM.

CAUTION

> The 2650B instructions STPL and LDPL are not members of the 2650A instruction set. When these instructions appear in the program, the user is required to know on which CPU the program is to execute.

I. INTRODUCTION

The 2650 assembly language is a symbolic language which facilitates the writing of programs for the Signetics 2650 microprocessor. The 2650 Relocatable Assembler, RASM, is a program which accepts this symbolic source code as input and produces a listing and/or a relocatable program object module as output.

The assembler is a two-pass program that builds a symbol table, issues helpful error messages and produces an easily readable program listing. In conjunction with the link editor, LINK, a computer readable object (load) module is output. It features conditional assembly, symbolic and relative addressing, forward references, free format source code, self-defining constants, complex expression evaluation, relocatability, and a versatile set of Pseudo Operations. Additionally, the assembler is capable of generating data in several number-based systems, and ASCII character code. These features aid the programmer/engineer in producing well documented, working programs in minimum time.

ASSEMBLY LANGUAGE

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

A symbolic machine instruction is a written specification for a particular machine operation expressed by a symbolic operation code and, if required, a symbolic address or operand. For example:

```
LOC2    STRR,R0    SAV
```

Where:

LOC2	is a symbol representing the memory address of the instruction.
STRR	is a symbolic operation code representing the bit pattern of the "store relative" instruction.
R0	is a symbol that has been defined as register 0 by the "EQU" pseudo-op.
SAV	is a symbol representing the memory location into which the contents of register 0 are to be stored.

A pseudo-operation statement is a statement which is not translated into a machine instruction, but rather is interpreted as a directive to the assembler program. For example:

```
SCHD    ACON    REDY
```

Where:

SCHD	is a symbol. The assembler is to assign the memory address of the first byte of the two allocated to this symbol.
ACON	is a pseduo-op which directs the assembler program to allocate two bytes of memory.
REDY	is a symbol representing an address. The assembler is directed to place the equivalent memory address into the allocated bytes.

STATEMENTS

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

LABEL FIELD OPERATION FIELD OPERAND FIELD COMMENT FIELD

The statement is always assumed to be written as an 80-column image.

The Label Field is provided to assign symbolic names to bytes of memory. If present, the Label Field must begin in column one.

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

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.

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

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 in column one. For example:

```
*THIS IS A COMMENT STATEMENT
```

LOCATION COUNTER

During the assembly process, the assembler maintains a cell that always contains the address of the next memory location to be assembled. This cell is called the Location Counter. It is used by the assembler to assign addresses to assembled bytes, and its value is available to be used by the programmer.

The character "$" is a symbol that the assembler recognizes as the symbolic name of the Location 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:

```
        108₁₆    BCTR,3    $
```

The first byte of this branch instruction is in location 108_{16}. The instruction directs the microprocessor to "branch to myself." The Location Counter in this example contains the value 108_{16}.

SYMBOLIC ADDRESSING

As mentioned above, the user can attach a label to an instruction, as shown in the following example:

```
SAVR    STRR,RO    SAV
```

The assembler, upon seeing a valid symbol in the label field, assigns the equivalent address (the value of the Location Counter) to the label. In the given example, if the STRR instruction is to be stored in the address 0127, then the symbol SAVR would be made equivalent to the value 0127 for the duration of the assembly.

The symbol could then be used anywhere in the source program to refer to the address value or, more typically, it could be used 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 branching to the STRR instruction, one could write:

```
        BCTA,3    SAVR
```

When this three-byte branch instruction is translated by the assembler, the address of the STRR instruction is placed in the address field of the branch instruction. Similarly, in the STRR instruction above, the symbol SAV in the Operand Field is replaced by the address of the memory location whose label is SAV.

It is also possible to use symbolic addresses which are near other locations to refer to those locations without defining new labels. For example:

```
                BCTR,3      BEG
                BCTR,0      BEG+4
                ANDZ        3
                BSTR,3      S+48
BEG             LODA,2      PAL
                HALT
                SUBI,2      3
```

In the above example, the instruction "BCTR,3 BEG" refers to the "LODA,2 PAL" instruction. The instruction "BCTR,0 BEG+4" refers the the "SUBI,2 3" instruction.

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

```
                BCTR,3      PAL+23
                BSTA,0      STT-18
```

The arguments are evaluated like expressions and cannot exceed in value the maximum number that can be contained in an integer constant in the range of -32,768 to +32,767.

II. LANGUAGE ELEMENTS

Input to the assembler consists of a sequence of characters combined to form assembly language elements. These language elements include symbols, instruction mnemonics, constants and expressions which make up the individual program statements that comprise a source program.

CHARACTERS

Alphabetic: A through Z

Numeric: 0 through 9

Special characters:

blank	
(	left parenthesis
)	right parenthesis
+	add or positive value
-	subtract or negative value
*	asterisk
'	single quote
,	comma
/	slash
$	dollar sign
<	less than sign
>	greater than sign
#	pound sign
@	at sign
?	question mark
!	exclamation point
"	double quote
%	percent sign
=	equal
;	semicolon
:	colon
.	period

SYMBOLS

Symbols are formed from combinations of characters. Symbols provide a convenient means of identifying program elements so that they can be referenced by other elements.

1. Symbols may consist of 1 to 6 characters. Any of the alphabetic, numeric, or the special characters #,@,!," and % are valid.

2. The character $ is a special symbol which may be used in the argument field of a statement to represent the current value of the Location Counter.

3. The character * is a special symbol which is used as an indirect address indicator.

4. The characters + and - are also used as auto-increment/auto-decrement indicators.

The following are examples of valid symbols:

```
DOP1      RAV3      SEVEN%
AA        TEMZ      AT#12
```

The following are examples of invalid symbols:

```
ROUNDOFF      more than six characters
1LAR          begins with a numeric
PA N          imbedded blank
TEN$          unallowed special character
```

CONSTANTS

A constant is a self-defining language element. Unlike a symbol, the value of a constant is its own "face" value and is invariant. Internal numbers are represented in 2's complement notation. There are two forms in which constants may be written: the Self-Defining Constant and the General Constant.

Self-Defining Constant

The self-defining constant is a form of constant which is written directly in an instruction and defines a decimal value. For example:

```
LODA,R3     BUFF+65
```

In this example, 65 is a self-defining constant. The value of the integer constant expressed by a self-defining constant must be in the range -32,768 to +32,767.

General Constant

The general constant is also written directly in an instruction, but the interpretation of its value is dictated by a code character and delimited by quotation marks.

```
LODA,R3     BUFF+H'3E'
```

In this example, the code letter H specifies that 3E is a hexadecimal constant equivalent to decimal value 62.

The size of a number generated by a general constant form (B, O, D, H) must be in the range -32,768 to +32,767. However, the most important concept to

understand when using constant forms is that the final value of a resolved expression must fit the constraints of the actual field destined to contain the value. For example:

```
LODA,R2     PAL+H'3EE2"-H'3EE0'
```

In this case, the argument, when resolved, must fit into the 13 bits in the actual machine instruction. Even though each of the two hexadecimal constants is larger than can fit into 13 bits, the final value of the expression is containable in 13 bits and therefore the constants are permitted. Similarly, the statement DATA H'3FE' is not allowed, as the DATA statement defines one byte quantities and H'3FE' specifies more than 8 bits. Summarily, the size of the evaluated expressions must be less than or equal to their corresponding data fields.

There are five types of General Constants usable with 2650 Assembler:

Code	Type
B	Binary Constant
O	Octal Constant
D	Decimal Constant
H	Hexadecimal Constant
A	ASCII Character Constant

B: Binary Constant

A binary constant consists of an optionally signed binary number of up to 16 bits enclosed in single quotes and preceded by the letter B, e.g., B'1011011'. Binary information is stored right justified.

O: Octal Constant

An octal constant consists of an optionally signed octal number enclosed by single quotation marks and preceded by the letter O, e.g., O'352'. The value will be right justified.

D: Decimal Constant

A decimal constant consists of an optionally signed decimal number enclosed by single quotation marks and preceded by the letter D, e.g., D'249'. The value will be right justified.

H: Hexadecimal Constant

A hexadecimal constant consists of an optionally signed hexadecimal number enclosed in single quotation marks and preceded by the letter H, e.g., H'3F'. The value will be right justified.

A: ASCII Character Constant

An ASCII character constant consists of a string of ASCII characters enclosed by single quotation marks and preceded by the letter A. For example, A'HELLO THERE'. Each character will be encoded in 7-bit ASCII and stored in

successive bytes. The high-order bit is always set to zero in each allocated byte.

See Appendix D for permissible characters and their equivalent ASCII codes. To specify a single quotation mark as a character constant, it must appear twice in the character string, e.g., A'TYPE"HELP"NOW' will appear in storage as TYPE'HELP'NOW.

MULTIPLE CONSTANT SPECIFICATIONS

General constant forms, when used in DATA and ACON statements, allow multiple specifications within the constant expression. For example, D'52,21,208,27'. A comma separates each byte specification (except in the ASCII form) and successive specifications determine successive bytes of storage. The forms within a single constant expression may be mixed. Each byte may be optionally signed. For example:

```
H'03,-F2,+11,-8,33,0',0'271,133',255,H'F0,FF'0'271,133',A'X,H'0,A,B,C'
```

EXPRESSIONS

An expression is an assembly language element that represents a value. It consists of a single term or a combination of terms separated by arithmetic, logical or relational operators. A term may be a valid symbolic reference, a self-defining constant or a general constant.

The valid operators which may be used in an expression are:

+	for addition (or unary positive sign)
-	for subtraction (or unary negative sign)
*	for multiplication
/	for division
.MOD.	for remainder after division
.NOT.	for one's complement (unary operator)
.AND.	for logical AND
.OR.	for logic OR
.XOR.	for logical EXCLUSIVE OR
.SHR.	for logical shift right
.SHL.	for logical shift left
>	for retrieving the lower byte of a two-byte argument
<	for retrieving the upper byte of a two-byte argument
.EQ.	for equal (=) relation
.NE.	for not equal ($\neq$) relation
.GT.	for greater than (>) relation
.LT.	for less than (<) relation
.GE.	for greater than or equal to ($\geq$) relation
.LE.	for less than or equal to ($\leq$) relation

The relational operators are binary operators which yield a true (1) or false (0) value.

These operators have an implied priority which determines the order in which the operations are performed in multiple operator expressions. The operations specified by the operators of the highest priority are performed first, then those of the second priority, and so forth.

The priority of operators is as follows, with operators on the same line having equal priority:

```
* / .MOD. .SHR. .SHL.
+ -
.EQ. .NE. .GT. .LT. .GE. .LE.
.NOT.
.AND.
.OR. .XOR.
< >
```

Parentheses can be used to override the implied order of operations or to make the expression clearer. For example:

```
 A + B * C          is equivalent to A + (B * C)
.NOT.A-B.MOD.C      is equivalent to .NOT.(A-(B.MOD.C))
 A.AND.B-C          is equivalent to A.AND.(B-C)
```

The expression A.SHR.B shifts the argument A B places towards the right and fills the most-significant bits of A with zeroes. Similarly, A.SHL.B shifts A B places to the left and fills the least-significant bits with zeroes. For example:

```
 H'43'.SHR.1     is equivalent to H'21'
 'H'21'.SHL.1    is equivalent to H'42'
```

Examples of valid expressions are:

```
LOOP            PAL-$
LOOP+5          $-PAL+3
SAM+3-LOOP      BIT-3+H'3B'
```

NOTE

The special symbol '$' represents the current value of the Location Counter.

If an expression resolves to a two-byte value where only a single-byte value is required, the assembler will use the least-significant byte as the operand.

EVALUATION OF EXPRESSIONS

Due to the relocation facility and the ability to refer to external symbols, several types of operands in expressions can be distinguished:

1. Constant - A constant is an internally defined item with an invariant value. Constants can be defined by the assembler instruction EQU; for example:

```
CR      EQU     H'OD'
```

2. Address - An address is also an internally defined item, but its final value is assigned by the link editor when the program object code is located. Labels of machine instructions are examples of addresses.

3. External Reference - An external reference is a reference to an item which is defined in another program, where it is either an address or a constant. External references are introduced with the instruction EXTRN.

All operations are allowed on constants, but only a limited number of operations are useful on addresses and external references. When an operation, not giving a usable result, is performed on an address or an external reference, the result is a constant. The operations, types of operands and results are:

Where: A = address; E = external reference; C = constant.

Operation	Operands	Result
Addition:	A + C	A
	E + C	E
		C in all other cases
Subtraction:	A - C	A
	E - C	E
		C in all other cases
Any Other Operation:		C in all cases

Expressions are evaluated with 16-bit arithmetic, that is, division may have an 8-bit divisor and 16-bit dividend; multiplication may have a 16-bit multiplier and muliplicand. When an expression evaluates to larger than a 16-bit signed integer, an overflow error is reported and processing on the line aborts. As for constants, the final value of a resolved expression must fit the constraints of the actual field destined to contain the value.

SPECIAL OPERATORS

There are two special operators that are recognized by the assembler. They are:

< less than sign
> greater than sign

The assembler interprets these operators in a special way:

> perform a modulo 256 divide (use low-order byte)
< perform an integer divide by 256 (use high-order byte)

These special operators are intended to be used to access a two-byte address in one byte parts using a minimum of storage. For example, if it is desired to get the high-order bits of an address (ADDB) into register 2 and the low-order bits into register 1 it could be done as follows:

```
                    LODR,R2         APAL
                    LODR,R1         APAL+1
                    . . .
                    . . .
                    . . .
                    ACON            ADDB
APAL
```

or by utilizing the special operators, it could be done as follows:

```
                    LODI,R2         < ADDB
                    LODI,R1         > ADDB
```

The first method uses 6 bytes to accomplish what the second method can do in 4 bytes.

The special operators are most often used to facilitate the passing of an address in registers.

III. SYNTAX

Assembly language elements may be combined to express symbolically both 2650 instructions and assembler directives. There are specific rules for writing these instructions. This set of rules is known as the Syntax of the symbolic assembly language.

FIELDS

A statement prepared for processing by the assembler is divided into four fields: the Label Field, the Operation Field, the Operand Field and the Comment Field. Each field is separated by at least one blank character. Columns 1 through 80 of the input line are scanned by the assembler.

Label Field

The label field optionally contains a symbolic name which the assembler assigns to the instruction specified in the remaining part of the line. If a name is specified, it must begin in column 1. The assembler assumes that there is no label if column 1 is blank. The label field, if present, must contain only a valid symbol.

Operation Field

The operation field contains a nmemonic code which represents a 2650 processor operation or an assembly directive. The operation field must be present in every non-comment line. See Appendix B for a list of the valid mnemonic codes. Additionally, depending on the instruction type, the operation field may also specify a general purpose register or a condition code.

Operand Field

The operand (or argument) field contains one or more symbols, constants or expressions separated by commas. The argument field specifies storage locations, constants, register specifications and any other information necessary to specify completely a machine operation or an assembler directive. Embedded blanks are not permitted as they are considered field terminators.

Comment Field

The comment field contains any valid characters in any combination. The comment field is not processed by the assembler, but is merely reproduced on the listing next to the accompanying instruction. It is usually used to explain the purpose or intention of a particular instruction or group of instructions.

Comment Line

An entire 80 column line may be utilized to print comments by coding an asterisk (*) in column 1. This entire line is merely reproduced on the assembly listing without further processing by the assembler.

SYMBOLS

Symbols are used in the label field of a symbolic machine instruction to identify that particular instruction and to represent its address. Symbols may be used for other purposes, such as the symbolic representation of some memory address, the symbolic representation of a constant, the symbolic representation of a register, etc.

No matter how the symbol is used, it must be defined. A symbol is defined when the assembler knows what value the symbol represents. There is only one way to define a symbol. The symbol must at some time appear either in the label field of an instruction or of an assembler directive. The symbol will be assigned the current value of the Location Counter when it appears in the label field of a machine instruction, or of an ORG, ACON, DATA, or RES pseudo-op. It may be assigned some other value through use of the EQU or SET assembler directives. A symbol may not appear in the name field more than once in a program, because this would cause the assembler to try to redefine an already defined label. The assembler will not do this and will flag all definitions of the particular label as an error. The only exception is that a symbol assigned a value by a SET pseudo-op may have another value assigned by subsequent SET pseudo-ops.

SYMBOLIC REFERENCES

Symbols may be used to refer to storage designations, register assignments, constants, etc. For example:

```
Address         Label          Operation          Operand

9B              MAZE             DATA               H'F5'
9C                               LODA,3             MAZE
```

The symbolic label "MAZE" represents the address 9B. It is used in the machine instruction at address 9C to tell the assembler to build an instruction LODA,3 9B. The symbolic label, in this case, is a way for the programmer to specify an address without knowing exactly what the address should be when the program is written. In this example, assume there was a need to modify this sequence of code: a data statement was inserted between the original two statements.

```
Address         Label          Operation          Operand

99               MAZE            DATA               H'F5'
9A,9B                            DATA               H'FE,3A'
9C                               LODA,3             MAZE
```

Even though there was a program change which caused the data at MAZE to be located at address 99, the load instruction referencing the data did not have to be rewritten because the assembler could provide the proper physical address for the symbolic address MAZE. The instruction at address 9C will be assembled as LODA,3 99.

SYMBOLIC ADDRESSING

When writing instructions in the symbolic assembler language for the 2650, the addresses may be expressed through symbolic equivalents. The assembler will translate the symbolic address to its numeric equivalent during the assembly process.

It is good programming practice to make all address references symbolic, as this greatly eases the programmer's job in producing a working program. To make the register specification symbolic, one could equate a symbol to the register number:

```
RG3         EQU         3
            . . .
            . . .
            . . .
            . . .
            LODA,RG3    MAZE
```

Forward References

A previously defined symbol is one which has appeared in the name field before it is referenced (as above). In contrast, a forward reference is a symbolic reference to a line of code when the symbol has not yet appeared in the name field. For example:

```
            ADDA,2          COEF
            . . .
            . . .
            . . .
COEF        DATA            D'123'
```

Forward references may be used anywhere in a program with the exception of the operand fields of the EQU, SET, RES, IF, and ORG statements.

Relative Addressing

The programmer may reference a memory cell either directly or via relative addressing. To refer directly to a memory cell of symbolic address MAIN, one has merely to use the name MAIN in the argument field of the referencing instruction. For example:

```
BIRA,R2     MAIN
```

It is also possible to express the address of a memory cell symbolically if some nearby cell is symbolically assigned. For example, to load the memory cell which is 5 cells higher in memory than the cell named MAIN, one need only to refer to it as MAIN+5:

```
LODA,2      MAIN+5
```

This latter method is called relative addressing. The relative count must be in the range -32,768 to +32,767.

The Location Counter and Symbol '$'

There is one symbolic name, "$," which is automatically defined by the assembler. This single character name is always symbolically equated to the contents of the assembler's location counter. Since the location counter is used by the assembler during the assembly process and is usually equated to the address of the byte being assembled, it represents the address of the instruction or data currently being specified. For example: BCTR,3 $+5. The branch address will be interpreted by the assembler to be the address of the first byte of the branch instruction plus 5 bytes.

Hardware Relative Addressing

When using instructions which use "hardware relative addressing" (as distinguished from relative addressing discussed earlier in this section), it is important to realize the assembler will not only evaluate the expression which is given as an operand address, but will convert it to a hardware relative address (see the Hardware Specifications manual for a description of the addressing modes). For example:

```
Address          Name          Operation          Argument

100              SAM            LODA,R2             PAL
103                             SUBI,R2             -3
105                             BIRR,R3             SAM
107              next instruction
```

In this code, the BIRR instruction specifies hardware relative addressing. Even though the equivalent value of the symbolic address SAM is 100, the relative addressing instruction requires a displacement relative to the address of the next sequential instruction. Therefore, the operand SAM will be evaluated as = -(current location counter+length of BIRR instruction-SAM) = -(105+2-100) = -(+7) = -7. Remember, where the hardware instruction calls for "hardware relative addressing," the expression in the operand field will be evaluated as the displacement from the address of the next sequential instruction. The value of this displacement may range from -64 to +63.

Indirect Addressing

The symbol "*" is used to specify indirect addressing. For example:

```
                          BCTA,3        *SAM
                          . . .
                          . . .
                          . . .
                SAM       ACON          SUBR
```

In this sequence of instructions, the BCTA instruction specifies indirect addressing. The assembler will set the indirect bit (byte #1, bit #7) to '1' for this instruction.

Auto-Increment and Auto-Decrement

The symbols "+" and "-" are used to specify auto-increment and auto-decrement, respectively. For example:

```
LODA,RO     BUF,R3,+
```

In this instruction, which specifies auto-increment, the assembler sets bits #6 and #5 of byte #1 to "01." This option is specified in the instruction set tables as (,X).

IV. THE ASSEMBLY PROCESS

The 2650 assembler translates symbolic source code into machine language instructions. The assembler examines every source statement for syntactic validity and produces the equivalent machine code for the 2650 processor.

This is a two pass assembler, which means the entire source code is scanned twice by the assembler. On the first pass, all defined labels and their equivalent values are stored in a symbol table, the first byte of every instruction is fully determined, and some classes of errors are detected. During pass 2, symbolic address references are replaced by their values, all remaining errors are detected, and a listing and linkable object module are generated.

EXECUTION OF THE RELOCATABLE ASSEMBLER

The relocatable assembler is initiated by means of the SDOS command RASM. The format of the command is:

RASM,<ifile>,[<list>], [<ofile>], [<print>]

Where:

<ifile> denotes the file containing the assembler source text which is the user's source file if no macros are used. Otherwise, <ifile> is the output file of the MACRO preprocessor. This is the only required parameter.

<list> identifies the list output device. <list> may be a filename. If this parameter is omitted, no listing is produced. If CONO is specified, the listing lines are truncated after 79 characters.

<ofile> identifies the file that is to contain the object program module. This file does not contain executable code. No object program module is stored if this parameter is omitted.

<print> is a flag which determines whether or not macro expansions are to be printed. If <print> = G, the expansions will be printed. The default is no print. A parameter other than G or (blank) produces an error message.

Upon successful initiation, the relocatable assembler displays the message:

** 2650 RELOCATABLE ASSEMBER VER n.n **

where:

n.n identifies the RASM version being executed.

LOCATION COUNTER

The assembler maintains a memory cell which it uses as a location counter. This location counter keeps track of the address of the next byte of storage to be allocated by the assembler. During coding, the programmer may think of the location counter as containing the address of the first byte of the instruction being written. In this assembler, the location counter is also used to provide load information.

ERROR DETECTION

During an assembly, the source program is checked for syntax errors. If errors are found, appropriate notification is given and the assembly proceeds. Although an assembled program containing errors generally will not run properly, it is considered good practice to complete the assembly to locate all errors at one time, rather than terminate it when an error is encountered.

ERROR CODES

There is a column on the listing in which an error indication may appear. Sometimes, because an error causes the assembler to view a subsequent statement incorrectly, a valid statement may be flagged as an error. A good rule is to fix errors in a particular line of code as they are discovered.

The following alphabetic characters are printed in the error indicator column and imply the corresponding message.

A - Argument error. The argument has been coded in such a way that it cannot be resolved to a unique value.

L - Label error. The label contains too many characters, contains invalid characters, has been previously defined, or is an invalid symbol.

O - Op-code error. The op-code mnemonic has not been recognized as a valid mnemonic.

P - Paging error. A memory access instruction has attempted to address across a page boundary.

R - Register field error. The register field expression could not be evaluated, or when evaluated, was less than 0 or greater than 3, or the register field was not found.

S - Syntax error. The instruction has violated some syntax rule.

U - Undefined symbol. There is a symbol in the argument field which has not been previously defined.

V - Numeric or expression stack overflow. Expression evaluation has resulted in an integer outside the valid range of -32,768 to +32,767, or the expression is too complex to evaluate. Processing of the source line is aborted.

W - Warning. The assembler has detected a syntactically correct but unusual construction.

If more than one error occurs in a single statement, only the indicator for the first error encountered will be printed, and only the first error will be counted in the total errors.

I/O Errors

If an input/output error is detected, a message is displayed on the console and the assembly is terminated. The format of the message is:

** ERR <xx> <id> FILE **

where:

<xx>	donotes the SRB status returned for the I/O operation.
<id>	identifies the concerned file. The possible identifiers are:
	S - Source file, <ifile>
	L - Listing file, <list>
	O - Object file, <ofile>

Command Line Errors

If the optional parameter <print> is neither G nor a blank, the following message is displayed on CONO, and processing is aborted:

INVALID 4TH PARAMETER

Symbol Table Error

A symbol table overflow produces the message:

SYMBOL TABLE OVERFLOW

SYMBOL TABLE

An alphabetic symbol table is produced upon completion of RASM processing. This symbol table contains all program labels, their addresses, and attributes with which they are declared (i.e., as a label).

The maximum number of symbols allowed is 650 for a TWIN with 16K bytes of slave memory, and an additional 400 symbols for each 4K byte increment of memory.

The message SYMBOL TABLE OVERFLOW appears when the symbol table is full.

V. DIRECTIVES AND EXTENDED INSTRUCTION SET

ASSEMBLER DIRECTIVES

There are seventeen directives (pseudo-ops) which the assembler will recognize. These assembler directives, although written much like processor instructions, are simply commands to the assembler instead of to the processor. They direct the assembler to perform specific tasks during the assembly process, but have no meaning to the 2650 processor. These assembler directives are:

ORG
EQU
SET
ACON
DATA
RES
END
EJE
CEJE
PRT
SPC
TITL
PCH
REPRO
IF
ELSE
ENDIF

The use of the last three directives listed above (IF, ELSE, and ENDIF) is described in Section VI.

There are also three directives that relate to relocatability. These directives are:

CSECT
ENTRY
EXTRN

The use of these relocatable directives is described in Section VII.

ORG — Set Location Counter

The ORG directive sets the assembly time Location Counter to the location specified. The assembler assumes an ORG 0 at the beginning of the program if no ORG statement is given. The instruction ORG may not set the location counter to a value lower than the current value.

LABEL	OPERATION	OPERAND
{name}	ORG	expression

Where:

name — optionally provides a symbol whose value will be equated to the value in the Location Counter after the ORG is evaluated.

expression — when evaluated, results in a positive integer value. This value will replace the contents of the location counter, and bytes subsequently assembled will be assigned sequential memory addresses beginning with this value. Any symbols which appear in the argument must have been previously defined.

Examples:

```
LARR    ORG     YORD
STAR    ORG     H'100'
```

EQU — Specify a Symbol Equivalence

The EQU directive tells the assembler to equate the symbol in the label field with the evaluatable expression in the operand field. A symbol equivalence defined by an EQU directive cannot be redefined by a subsequent EQU or SET directive and cannot appear in the label field of a statement.

LABEL	OPERATION	OPERAND
name	EQU	expression

Where:

name — is the symbol which is to be assigned some value by the execution of this directive.

expression — is resolved to an integer value. If a symbol is used in the argument, it must have been previously defined.

Examples:

```
PAN     EQU     H'10F'
LOP2    EQU     PAL
RAMP    EQU     SLOP-3+PAL
REG1    EQU     1
```

SET — Specify a Symbol Equivalence

The SET directive tells the assembler to equate the symbol in the label field with the expression to be evaluated in the argument field.

The SET directive is identical to the EQU directive, except that the symbol defined by the SET directive may be redefined later in the program by another SET directive.

LABEL	OPERATION	OPERAND
name	SET	expression

ACON — Define Address Constant

The ACON directive tells the assembler to allocate two successive bytes of storage. The evaluated argument will be stored in the two bytes, the low-order 8 bits in the second byte and the high-order bits in the first byte. This directive is mainly intended to provide a double byte containing an address for use as the indirect address for any instruction executing in the indirect addressing mode. Any number of operands may be specified with one ACON directive. Each operand will be allocated two bytes of storage.

The data definitions in the instruction ACON may be preceded by a multiplication factor. This instruction cannot be used to define character strings.

NOTE

If an error is detected in the ACON instruction, two null bytes (00) are generated at the place where the error was detected. No further processing is done on the line. This is done to facilitate patching of the object code for debugging purposes.

LABEL	OPERATION	OPERAND
{name}	ACON	expression

Where:

name — is an optional label. If specified, the name becomes the symbolic address of the first byte allocated.

expression — is some expression which must resolve to a positive value or zero. If positive, the value should be no larger than that which can be contained in two bytes. Otherwise, only the two least-significant bytes are used.

Example:

```
ASUB    ACON    SUBR
        ACON    SUBR,H'AFF0'
        ACON    3H'000F'
```

DATA — Defines Memory Data

The DATA directive tells the assembler to allocate the exact number of bytes required to hold the data specified in the argument field of this directive. Any number of bytes can be specified with one DATA directive.

The data arguments in the instruction DATA may be preceded by a multiplication factor. Additionally, the length of character strings can be defined explicitly.

Two types of data constants are distinguished: character strings and other constants. Both types may be specified in the same data instruction. A character string occupies a variable number of bytes; this number is either implicitly or explicitly specified. Other constants always occupy a single byte.

NOTE

If an error is detected in the DATA instruction, two null bytes (00) are generated at the place where the error was detected. No further processing is done on the line. This is done to facilitate patching of the object code for debugging purposes.

LABEL	OPERATION	OPERAND
{name}	DATA	argument

Where:

name — is an optional label. If used, the name becomes the symbolic address of the first byte allocated by the directive.

argument — is a general constant, a self-declining constant or a symbolic address. A multiple constant specification in the argument field will be handled as if the corresponding number of data statements were input.

To specify explicitly the length of the generated argument, precede the argument with the ASCII character L followed by a length value. Length value can be either an expression or a constant. The argument string will be truncated or blank-filled as necessary to satisfy the length value.

Examples:

```
PAL       DATA      LOOP,LOOP+1
          DATA      H'03,22,FC,A1'
          DATA      +127
          DATA      D'28'
DEFINE    DATA      A'THIS IS'
          DATA      AL10'THIS IS'
          DATA      AL2'THIS IS'
          DATA      3A'THIS IS'
          DATA      3AL2'THIS IS'
```

NOTE

If the argument evaluates to a value between 0 and 255, the result is an eight bit absolute binary number. DATA +127 results in H'7F'. Also, if the argument evaluates to a value which is less than 0, the result is a 2's complement, binary number. DATA H'-5' results in H'FB'. If the argument resolves to a value which requires more than one byte, only the least-significant byte is used, unless a length is specified.

RES — Reserve Memory Storage

The RES directive tells the assembler to reserve contiguous bytes of storage. The number of bytes so reserved is determined by the argument. The reserved bytes are not set to a known value, but rather the effect of this directive is to increment the location counter.

LABEL	OPERATION	OPERAND
{name}	RES	expression

Where:

name — is an optional label. If used, the name becomes the symbolic address of the first byte allocated.

expression — is some evaluatable expression which must resolve to some positive integer or zero. If a symbol is specified, it must have been previously defined.

Example:

```
LOR     RES     23
MASK    RES     O'374'
        RES     H'1A'
```

END — End of Assembly

The END directive informs the assembler that the last statement to be assembled has been input. The END directive causes the assembler to communicate the program start address to the object module. The assembler will not generate this instruction.

The operand of an END instruction, if present, must resolve to an address, and that address is used as the entry point of the module. Otherwise, the module has no explicitly defined entry point.

NOTE

A PHASE, as produced by the link editor needs an entry point and the entry point of the last module edited in a PHASE is used. When a module has no explicitly defined entry point, the begin address of the PHASE is taken as the entry point.

LABEL	OPERATION	OPERAND
	END	{expression}

Where:

expression — is resolved to be the starting address of the program.

EJE — Eject the Listing Page

The EJE directive tells the assembler to advance the listing to the top of the next page regardless of the line position on the current listing page.

The directive is used primarily to organize listing for documentation purposes and does not appear in the listing.

LABEL	OPERATION	OPERAND
	EJE	

CEJE — Conditional Eject the Listing Page

This instruction is equivalent to a normal EJE instruction if the remaining number of lines on the current page is less than the value of the expression. If the remaining number of lines is equal to or greater than the value of the expression, the instruction has no effect.

LABEL	OPERATION	OPERAND
	CEJE	expression

Where:

expression — is an evaluatable expression which must resolve to a positive integer.

PRT — Printer Control

The PRT directive tells the assembler to resume or discontinue printing of the assembled program.

This directive is used primarily to shorten assembly time by listing only that portion of the program which the user needs to see. This directive does not appear in the listing.

LABEL	OPERATION	OPERAND
	PRT	{ON OFF GEN NOGEN}

Where:

ON — means print the assembled program

OFF — means discontinue print of the assembled program

GEN — means print generated macro expansions

NOGEN — means discontinue print of generated macro instruction expansions. When the G flag is entered in the RASM command line, it overrides all NOGEN flags.

NOTE

PRT is set to ON,NOGEN at the beginning of the assembly.

SPC — Space Control

The SPC directive tells the assembler to skip or space a number of lines.

This directive is used primarily to organize listings for documentation purposes and does not appear in the listing.

LABEL	OPERATION	OPERAND
	SPC	expression

Where:

expression — is an evaluatable expression which must resolve to a positive integer. If the value of this expression is equal to, or greater than, the number of lines remaining on the page, the effect is the same as the EJE directive.

Example:

```
        SPC     5
```

TITL — Title

The TITL directive tells the assembler to skip to the top of the next page and insert a given title into the main header.

This directive is used primarily for documentation purposes and does not appear in the listing.

LABEL	OPERATION	OPERAND
	TITL	expression

Where:

expression — is the title information not to exceed 38 character positions.

Example:

```
        TITL        MAIN PROGRAM
```

PCH — Punch Control

The PCH directive tells the assembler to selectively resume or discontinue the output of the object program module.

This directive is used primarily to shorten assembly time when an object program module is not desired or when only a portion of the object program module is desired.

LABEL	OPERATION	OPERAND
	PCH	{ON / OFF}

NOTE

PCH is set ON at the beginning of an assembly. When PCH OFF is specified, any prior object program module data is output.

REPRO — Copy Next Source Line

Often it is desirable to apply certain link editing commands to the link editor via the source file. Such commands are not automatically supplied by RASM. The REPRO directive is used to copy such commands from the source module. The instruction means copy the next source line into the object module and do not assemble that line. The instruction may not be labeled and has no operands.

The instruction REPRO is executed during pass 1 of an assembly run, so the reproduced statements appear at the beginning of the object module, before the external symbol dictionary (ESD).

LABEL	OPERATION	OPERAND
	REPRO	

Example:

```
REPRO
INCLUDE FILEA
```

directs the link editor to include FILEA in front of this program object module in creating the current phase.

EXTENDED INSTRUCTION SET

In order to simplify the writing of programs and to increase the readability of programs, the mnemonic branch instruction op codes have been extended. The additional branch instructions are intended for use after compare, load and arithmetic instructions and the instruction TMI. These instructions with their standard equivalents are:

Situation	Condition Code Value	Extended Instruction	2650 Equivalent Op Code
After Compare	Branch if higher	BHR	BCTR,1
		BHA	BCTA,1
	equal	BER	BCTR,0
		BEA	BCTA,0
	lower	BLR	BCTR,2
		BLA	BCTA,2
	not higher	BNHR	BCFR,1
		BNHA	BCFA,1
	not equal	BNER	BCFR,0
		BNEA	BCFA,0
	not lower	BNLR	BCFR,2
		BNLA	BCFA,2
After Load and Arithmetic Operations	Branch if positive	BPR	BCTR,1
		BPA	BCTA,1
	zero	BZR	BCTR,0
		BZA	BCTA,0
	minus	BMR	BCTR,2
		BMA	BCTA,2
	not positive	BNPR	BCFR,1
		BNPA	BCFA,1
	not zero	BNZR	BCFR,0
		BNZA	BCFA,0
	not minus	BNMR	BCFR,2
		BNMA	BCFA,2
After TMI	Branch if ones	BOR	BCTR,0
		BOA	BCTA,0
	mixed	BMR	BCTR,2
		BMA	BCTA,2

Also, the following unconditional extended branch instructions have been added:

Extended Instruction	2650 Equivalent Op Code
BR	BCTR,3
BA	BCTA,3
BSR	BSTR,3
BSA	BSTA,3
RET	RETC,3

VI. CONDITIONAL ASSEMBLY

The conditional assembly directives allow the programmer to vary the sequence of generated statements. Thus, the programmer can use these instructions to generate different sequences of statements from the same source program.

There are three conditional assembly directives. They are:

IF
ELSE
ENDIF

The format of the IF statement is:

LABEL	OPERATION	OPERAND
{name}	IF	expression

The standard expression rules apply with the addition of the following six relational operators which may be used in relational expressions:

.EQ. .NE. .LT. .GT. .LE. .GE.

The format for relational expression is

(expression.XX.expression)

Where:

.XX. is any one of the above relational operators.

The format of the ELSE statement is:

LABEL	OPERATION	OPERAND
	ELSE	

The format of the ENDIF statement is:

LABEL	OPERATION	OPERAND
	ENDIF	

Every IF statement must have a corresponding ENDIF statement. The use of the ELSE statement is optional, but if present, it must appear after the IF statement and before the ENDIF statement.

When an IF statement is encountered, the expression or relational expression is evaluated to be either true (not zero) or false (zero). If true, the following source statements are processed until an ENDIF is encountered. However, if an ELSE is encountered during this processing, the statements between the ELSE and the ENDIF are not processed. If false, the source statements following the IF are not processed until an ELSE or ENDIF is encountered, at which time normal processing resumes.

Conditional assembly constructs may be nested to eight levels but may not be overlapped. Therefore, the end of an inner IF construct (nested) must be encountered before the end of the outer IF construct is encountered.

The assembler listing will contain only those statements actually assembled. The statements between an IF and ENDIF (or ELSE) which do not produce object code do not appear on the listing.

If the optional label is included in the IF statement, the label will be assigned the value of the Location Counter when the next byte is actually assembled.

VII. RELOCATABILITY/RELOCATABLE ASSEMBLY

In order to allow links between separately assembled programs, each program object module begins with an external symbol dictionary (ESD). The ESD is printed at the beginning of the program listing. An ESD can contain the following types of records:

SD - Section definitions

The first record in each ESD is of the type SD. It contains the name of the section, its assembled origin and its length. The section name is specified by the CSECT instruction.

LD - Label definition

An ESD can contain several records of the type LD. They identify symbols declared as entry points into the program and their associated addresses. Other separately assembled programs may then refer to these symbols. The directive ENTRY is responsible for the generation of the LD records.

CD - Constant definitions

A CD record is similar to an LD record, except that the value is not an address but a constant. The CD records are also generated by the ENTRY directive.

ER - External reference

The ER records form the counterpart of the LD records and the CD records. They contain the symbols (with a sequence number) which are defined as external in the associated program. The directive EXTRN is used for this purpose.

At link editing time, a symbol table is constructed which contains all the symbols from the SD records, the LD records, and the CD records. Each symbol may occur only once. During the link editing of an object module, the values corresponding to its ER symbols are retrieved from this symbol table and edited into the object module.

The following RASM directives are available for the construction of an ESD.

CSECT — Identify Object Module

The instruction CSECT can be used to identify an object module. Each source program may contain at most one CSECT directive. Normally, this instruction is placed at the beginning of the program and format is:

LABEL	OPERATION	OPERAND
label	CSECT	

Where:

label — is optional; it simply assigns an identification to the object module for reference on the ESD. The address of the assembled origin of the module is assigned to the label. If there is no label, the ESD shows CSECT at the assembled origin of the object module without an ID. The directive has no operands.

ENTRY — Define Entry Points

The directive ENTRY is to be used to specify the symbols which may be referenced by other programs. The directive may not be labeled. The operand consists of a string of symbols separated by commas. The first blank encountered signals the end of the symbol string. These symbols must be previously defined as the label of some machine instruction or assembler instruction.

The format of the ENTRY statement is:

LABEL	OPERATION	OPERAND
	ENTRY	label list...

Where:

label list — is one or more labels separated by commas.

Example:

```
        ENTRY       LABEL1
        ENTRY       LABEL2,LABEL3
```

EXTRN — Define External Labels

The directive EXTRN is used to specify the symbols which are defined in some other program but which are referred to in this program. EXTRN may not be labeled. The operand consists of a string of symbols separated by commas. The first blank encountered signals the end of the instruction.

These symbols may not be defined in the module containing the EXTRN directive. The only exception is given by EXTRN directives which are generated by means of a macro call expansion (see Section IX). Then the attribute external reference generated via the macro is ignored if the symbol is also defined elsewhere in the program.

The format of EXTRN statement is:

LABEL	OPERATION	OPERAND
	EXTRN	label list...

Where:

label list — is one or more labels separated by commas.

Example:

```
        EXTRN       LABELA
        EXTRN       LABELB,LABELC
```

VIII. LINK EDITOR

The link editor, LINK, creates an executable load module from one or more object program modules generated by the relocatable assembler. Any object program module generated by RASM must be processed by LINK before it can be executed. The executable load module can be executed by the SDOS commands LOAD/GO or XEQ.

LINK is a two pass program that resolves inter-module references flagged by the relocatable assembler and converts relocatable addresses into absolute addresses. A load map is printed at the end of pass two that lists:

1. For each phase, the:

   - Phase name (if any)
   - Phase begin address
   - Phase end address
   - Phase entry point
   - All object modules contained in that phase

2. For each object module, the:

   - Module name (if such name exists)
   - Module begin address
   - All external symbols defined in the module with their value

3. Linkage editing error messages

When the link editor encounters an object program module ORGed at zero (either by default or by ORG 0) which will cross a 2K page boundary, it automatically relocates that module to the start of the next 2K page.

EXECUTION OF THE LINK EDITOR

The link editor is invoked by means of the SDOS command LINK. The format of the command is:

LINK,<input>,[<loadmap>],[<bfile>]

Where:

<input> contains commands to the link editor to locate the object program module or modules to be linked and identify the binary load module created. The input file may be the output from RASM containing link editor commands or a link editor command file.

<loadmap> specifies the device to which the load map is written. The default value for this parameter is no load map. <loadmap> may be a filename.

<bfile> is used to name the first binary load module when no PHASE command precedes the first INCLUDE command. When <bfile> is omitted and <input> contains no PHASE command, the edited output, i.e., the binary load module, is not stored.

After successful initiation, the program displays the message:

** LINK EDIT VER n.n **

Where:

n.n identifies the version of LINK being executed.

Commands to the link editor may be invoked as follows:

1. Include the commands in the RASM <ofile> file via the REPRO assembler directive in conjunction with LINK commands. Specify the RASM <ofile> as the LINK <input> .

2. Enter the LINK commands into a command file specified as <input> to LINK.

3. Combine 1 and 2 above, but the command file must be the one specified as LINK <input> .

ERROR MESSAGES

The link editor can display the following error messages: These error messages appear on the load map.

INVALID ASSEMBLED ORIGIN - The assembled origin of the program object module is an address less than the current location counter. The assembled origin is ignored and linking continues.

INVALID RECORD - An invalid command record is encountered. If the record is a null record (no characters in the line), processing aborts.

MULTIPLY DEFINED SYMBOL - An external symbol is defined in more than one object module

NESTING ERROR - Included modules are nested too deep. A nesting level up to 4 can be accepted. This means that the basic object program module may include object module (level 1). This object program module includes a next object program module (level 2), and so on to 4 levels.

PHASE COMMAND REQUIRED - The first two commands to LINK were INCLUDEs not preceded by a PHASE command. Processing is aborted

PAGING ERROR - An instruction with a 13-bit address refers to a location outside the current page.

TOO MANY PHASES - The object program file specified more than 20 phases.

UNDEFINED SYMBOL - A symbol is defined as EXTRN in one of the object modules, but it has not been defined as ENTRY in one of the other modules.

If an I/O error is detected, a message is displayed on CONO and processing aborts. The format of the message is:

** ERR <xx> <id> FILE **

Where:

<xx>	denotes the SRB status return for the I/O operation
<id>	identifies the concerned file. The possible identifiers are: I - Input file; LINK <input> , i.e., RASM <ofile> or LINK command file L - Load map file, <loadmap> B - Binary load module file, <bfile>

LINK EDITOR COMMANDS

Although the link editor is able to produce an executable load module from single object program modules, additional link editor commands can be useful. Except for the comment, all of the following link editor commands must be preceded by a blank.

PHASE

This command names a phase† to be created by the link editor. When two or more object programs are to be included in the phase, they must be preceded by a PHASE command. The format is:

PHASE<bfile>[,<origin>] [,<ID>]

Where:

_ (underscore)	is a required space
<bfile>	is the name of the binary phase module to be created.
<origin>	specifies the begin address of the phase. It is represented by a normal expression. The phase address works in conjunction with the assembled origin, as follows:

†Phase used here is defined as separately stored load module. Such a module is equivalent to the modules produced by SDOS MODULE command.

Phase	Assembled Origin	Resultant Address
none	0	Current location counter (LC)
X	0	X
none	Y	Y
X	Y (X< Y)	a. Phase module begins at X b. CSECT begins at Y c. Arbitrary data fills X≤ LC < Y
X	Y (X> Y)	a. Phase module begins at X b. CSECT begins at X c. Warning message issued

<ID> is the module identification. It is a character string of, at most, 21 characters. Larger character strings are truncated without warning.

This command is used to include a separately assembled object program module. INCLUDE commands may be nested 4 levels. The format of the command is:

_INCLUDE<name>

Where:

_ (underscore) is a required space.

<name> is the name of the program module to be included. The link editor replaces this command with the program object module.

Comment

The input of the link editor may comprise comment lines, which are only listed in the load map. The symbol * in column one flags a line as a comment. The format of the comment is:

*** [comment]**

Note that the comment must not be preceded by a space.

All PHASE and INCLUDE commands are listed on CONO for each pass as they are processed by LINK.

IX. TWIN RESIDENT MACRO FACILITY

When programs are constructed, often the same or nearly the same sequences of instructions are written. If this occurs, macro instructions can be used to reduce the number of written source code instructions. When a macro call is processed, it is replaced by a sequence of machine instructions from a corresponding macro definition. The macro definition contains the data to be generated.

A single level, or non-nested, macro processing facility is available with the relocatable assembler. It is implemented as a preprocessor; that is, from the source program, it produces a text file for use as the source input to the relocatable assembler, RASM.

EXECUTION OF THE MACRO PROCESSOR

The macro pre-processor is invoked with the SDOS command MAC, as follows:

MAC,<source>,<expanded source>, [<macro file>], [<error list>]

Where:

<source>	is the user program file containing the macro calls.
<expanded source>	is the output file of the macro preprocessor to be used as the input source file to RASM.
<macro file>	is the file containing user written macro definitions for inclusion in the source program.
<error list>	is the optional list device or filename to which error messages are listed. The default is CONO.

Macro Libraries

Two macro libraries may contain input to the macro processor:

1. <macro file> is the user's macro definition file and, when specified in the command, is searched first.

2. SYSMACRO is a default macro library. SYSMACRO is searched whenever a macro definition is not found in <macro file> or if a <macro file> is not specified. SYSMACRO must reside on disk drive 0.

The macro preprocessor displays the following sign-on message:

** MACRO PREPROCESSOR VER n.n **

Where:

n.n identifies the version of MAC being executed.

Then, the input files are opened and macro definitions from both macro definition files are read into a memory buffer. If no errors have been detected at this point, the <source> file is processed.

When an end-of-file (EOF) condition is detected in <source> , processing is terminated.

ERROR MESSAGES

Macro preprocessor error messages are written to CONO, or to the specified <error list> file.

Certain error conditions are detected in the macro definition file before the source file is processed. Checking is done on macro names and on the size of the macro definition files. Error messages issued for these error conditions are:

INVALID MACRO NAME

The macro name is more than five characters or the macro prototype contains no macro name.

TOO MANY MACRO DEFINITIONS

The internal macro storage table or macro name table has overflowed. Up to 64 macro names are allowed. This error causes processing to abort.

NO MACRO DEFS

No user macro definition library was specified, and SYSMACRO was not on the disk drive 0. Processing aborts.

When an error has been detected during the processing of a macro instruction or control instruction, an alphabetic character indicating error type is printed with the source file line and corresponding macro definition line at which the error occurred. The format of this two line message is:

<C><source macro call>

<macro definition line>

Where:

<C>	is the error indicator, as follows:
	C - Invalid control instruction
	P - Error in prototype
	D - Error in macro definition
	I - Error in macro instruction
	O - Output line becomes too long
<source macro call>	represents the concerned source file input line

If an input/output error occurs, a message is displayed and the program is terminated. The format of the message is:

** ERR <xx> <id>FILE **

Where:

<xx>	represents the SDOS SRB status returned for the I/O operation.
<id>	represents the concerned file. Possible values of ID are:
	I - Input file, <source>
	O - Output file, <expanded source>
	M - Macro definition file, <macro file> or SYSMACRO

<source> FILE CONTENTS

The <source> file can contain the following type of lines:

1. Control instructions. The first character of a control instruction is a period. The only recognized control instruction in the <source> file is:

.TABS N_1 [,N_2,N_3...N_7] [comment]

Where:

N_n	is a decimal number in the range 0 - 70 and specifies a tab stop for the output lines. At most, 7 tab stops are accepted. The default tab stops are 10, 20, 30, 40, 50, 60 and 70. To reset tab stops to the default values, .TABS may be entered without parameters.

2. Comment lines. The first character of a comment line is *. These lines are added to the output file without any processing.

3. Data lines. The contents of the opcode field of assembly language source code are compared with the <macroname> of all macro definition prototypes. Upon a match, the corresponding macro definition is interpreted. Otherwise, the data line is simply copied to the output file.

<macro file> CONTENTS

The <macro file> consists of commands to the macro preprocessor and macro definitions.

MACRO DEFINITIONS

Macro definitions are bracketed by the macro preprocessor control instructions .MACRO and .MEND. The commands must be the only text on the line, and the period must appear in column one. The macro definition consists of one prototype statement followed by the macro body.

Prototype

The prototype defines the macro name and the macro definition formal parameters. It must be the first statement of a macro definition following the .MACRO control instruction and has the form:

[&<label>] <macroname> [&<param>, &<param>,...]

The <macroname> can be at most five (5) characters long and may not begin in column 1.

Macro Body

The macro body follows the prototype and consists of assembly language statements and macro control instructions. Nesting of macros is not supported. Where formal parameters declared on the prototype line appear in the macro body, they are replaced by the actual parameter value passed in the macro call.

Macro control instructions do not appear in the macro expansion, but give directions to the macro processor as to how to do the expansion. All macro control instructions begin with a period and may be labeled with macro labels.

MACRO VARIABLES

During the processing of macro calls, the text corresponding to the macro body is generated. Macro labels, macro control instructions, and the prototype will not appear in the expanded output. Also, expressions and variables are replaced by their actual computed values.

Identifiers of up to six characters can be used as macro variables. Two types of variables are recognized:

- Binary variables can assume numeric values from -32,768 to +32,767.
- Character variables may be a string of up to 14 alphanumeric characters. Empty strings are valid.

These variables can be declared explicitly by the DECLARE macro command or implicitly by their appearance in a Prototype statement.

A leading & identifies macro variables.

Parameters are simply variables that are implicitly declared by their appearance in the prototype statement.

All special characters, except _ (underscore) are treated as normal data characters. The character _ (underscore) at the end of a variable is used as concatenation operator; it will not appear in the output line.

EXPRESSIONS

Expressions may occur in macro control instructions as well as in assembly language statements. Expressions are nearly identical to those used by RASM, but the operators < (less than) and > (greater than) do not exist. An expression is constructed with operators, parentheses and factors.

Constant

The following types of constants are possible:

H - Hexadecimal constants
O - Octal constants
B - Binary constants
D - Decimal constants
Self defining Constants

All constants, except self defining constants, are enclosed by single quotation marks and are preceeded by the appropriate code character, as defined previously in this manual.

A constant is internally expressed as 15 bits plus a sign bit and it can have a value in the range -32,768 through +32,767.

Character String

A character string is enclosed by quotation marks and is preceeded by the code character A. A character string may contain up to 14 characters. Any character, except a single quotation mark, may occur within a character string. A quotation mark in a character string is to be specified by two consecutive single quotation marks. These two characters are then interpreted as a single character and not as a string delimiter.

Variable

A variable is identified by & as the first character. The value of a variable is a character string or a constant.

Expression Between Parentheses

The operators, ordered on increasing priority, are:

.OR. .XOR.	Logical OR and Exclusive OR
.AND.	Logical AND
.NOT.	Logical NOT (1-Complement)
.EQ. .NE. .GE. .LE. .LT.	Relations
+ -	Arithmetic addition and subtraction
* / .SHL. .SHR.	Arithmetic multiplication, division and shifts

The requirements and results of the various operations are:

- Arithmetic operations - The operands are binary numbers or it must be possible to convert the operands into binary numbers. The result is a binary number which shall be in the range or -32,768 to +32,767.

- Relations - The operands are character strings or binary numbers. Negative numbers are preceded by a minus sign, and leading zeros are omitted. These character strings are compared from left to right and the result of the comparison is:

  1. Equal

     Both strings have the same length and contain the same characters in an identical sequence.

  2. Greater Than

     The number of characters of the second operand is less than the number of characters of the first operand or the binary number, which represents the current character of the first operand, is greater than the binary number which represents the current character of the second operand.

  3. Less Than

     The length of the second operand exceeds the length of the first operand or the binary number, corresponding to the current character of the first operand, is less than the binary number corresponding to the current character of the second operand.

The final result of a relation operation is a binary number conforming to the following matrix:

	EQ	NE	LE	LT	GE	GT
Equal	1	0	1	0	1	0
Greater Than	0	1	0	0	1	1
Less Than	0	1	1	1	0	0

Logical Operations

The operands of logical operations are binary numbers. Character strings are, if applicable, converted to binary numbers.

MACRO PARAMETER EXPANSION

A macro call is simply a reference to a macro definition. It may contain the actual values of any formal parameters defined in the macro prototype statement.

In the macro definition a leading & identifies formal parameters. Three types of parameters are distinguished:

1. Label parameters. A label may be attached to the macro expansion by use of the label parameter. The formal parameter &<label> in the prototype statement passes the label on the macro call to the macro expansion. The label parameter must begin in column one of the macro prototype, and is of the form:

   &<label>

2. Positional parameters. The values of positional parameters are given by the character strings at the associated positions in the macro prototype. Positional parameters appear in the operand field and have the form:

   &<param>

3. Keyword parameters. The actual values of keyword parameters are identified by the corresponding keywords in the macro calls and are declared in the operand field of the macro prototype. Keyword parameters do not have pre-defined positions, and have the form:

   &<param> = <value>

Where:

<value>	is the parameter default value when the parameter is not specified in the macro call. <value> may be blank.

When both keyword and positional parameters are used in one macro call, the positional parameters are to be listed first.

Example:

The following macro moves a string of specified length.

```
    .MACRO                              Begin macro
    &L          MOVE   &A,&B,&N         Prototype statement
                LODI,R1 &N
    &L          SUBI,R1 1
                LODA,RO &A,R1           Macro body
                STRA,RO, &B,R1
                BRNR,R1 &L
    .MEND                               End macro
```

The corresponding macro call contains a label parameter and the values of three positional parameters:

```
                LBL         MOVE        P,Q,3
```

The character string following the first blank in the call, MOVE, identifies the <macroname> . It corresponds to the operation code of a normal machine instruction. The character strings LBL, P, Q and 3 represent the actual values of formal parameters in the associated macro definition for MOVE.

Processing of the above macro call results in the following output:

```
                LODI,R1 3
    LBL         SUBI,R1 1
                LODA,RO P,R1
                STRA,RO Q,R1
                BRNR,R1 LBL
```

Note that the control instructions and the prototype do not appear in the output. The formal parameters have been replaced by their actual values in the expanded macro call.

Suppose most of the string moves in our program move one byte of data to Q. The macro may then be written:

```
    .MACRO
    &L          MOVE        &A,&B=Q,&N=1
                LODI,R1     &N
    &L          SUBI,R1     1
                LODA,R0     &A,R1
                STRA,R0     &B,R1
                BRNR,R1     &L
    .MEND
```

The macro call now need only specify the data source as follows:

```
                        LBL         MOVE        C
```

resulting in this macro expansion:

```
                LODI,R1     1
    LBL         SUBI,R1     1
                LODA,R0     C,R1
                STRA,R0     Q,R1
                BRNR,R1     LBL.
```

Note that in this second example the keywords for string length and destination are N and B respectively. To move a string of length 3, simply specify the "length" keyword and actual value in the macro call.

```
                        LBL         MOVE        P,N=3
```

The expansion of the above macro call results in code identical to that shown in the first example.

MACRO COMMANDS

Macro definitions can contain special instructions to the macro processor to allow greater flexibility at expansion time. Macro commands begin with a period and those whose first character is underscored in this manual can be abbreviated to the first letter of the command. Macro commands must appear only one per line of text.

.&<macrolabel>

This label, within the macro definition, labels macro commands. Note that this is not the same as the formal &<label> parameter.

.ASSIGN &<variable> = <expression>

A simple assignment statement. The value of the expression is assigned to the variable.

.DECLARE &<variable>,...

This declares variables for use by the macro processor. Upon declaration, the variables are initialized with the empty character string.

.GENERATE &<variable> = <expression1>,<expression2>
.END

The GENERATE - END pair delimits a group of statements to be processed iteratively. The GENERATE statement declares a new variable, which assumes the value of the first expression as its inititial value. The variable is incremented by one each time the END statement is encountered. When the value of the variable exceeds that of the second expression, the iteration stops. Variables declared with a GENERATE loop exist only inside that loop.

.IF <expression> .&<macrolabel>

This command conditionally includes a block of text into the macro expansion. If <expression> evaluates to true, the next statement to be considered for inclusion is at .&<macrolabel> . If <expression> is false, the next statement is the one following the IF statement. Representations of true and false are as follows: binary: 0 = false
not 0 = true

character: all blanks = false
at least 1 non-blank in the string = true

.JUMP .&<macrolabel>

The next statement to be scanned for inclusion is the one indicated by .&<macrolabel> .

.MACRO
⋮ } **macro definition**
.MEND

This pair of commands must bracket each macro definition and cannot be abbreviated. They must begin in column 1; that is, .MEND cannot be labeled.

.MESSAGE <text>

This command causes <text> to be displayed on the system console, CONO.

.NOP

No operation. This is used for defining macrolabels, since they cannot appear on a blank line, or for jumping to the end of a macro definition since .MEND cannot be labeled. For example,

```
.J .&LBL
 .
 .
 .
.&LBL .NOP
```

.SUBSTRING &<variable> = &<variable>(,<expression1>,<expression2>)

A substring of the second variable is assigned to the first variable. The first expression indicates the beginning character of the substring (starting with the value zero for the first character of the string), while the second expression gives the length of the substring.

.TAB <character>

This sets the tab character. The definition remains in effect only during the current macro definition. & is not a valid tab character.

***<text>**

This is a comment; the line is ignored.

X. SAMPLE PROGRAM SET

Included here are two 2650 assembly language source programs, a SYSMACRO file listing, a LINK command file listing and the loadmap resulting from linking the assembled program object code from the two source programs.

The sample programs consist of a main program and a subroutine for an arithmetic bubble sort. These programs are called MAIN;S and SUBR;S respectively. The SYSMACRO file contains two macro definitions:

1. Register value equates
2. Condition code value equates

To expand the macro calls, REGS and CODES, for the equate values run:

```
MAC MAIN;S MAIN;E
MAC SUBR;S SUBR;E
```

Since no user macro library is specified in the MAC command lines, MAC will automatically look in SYSMACRO for macro definitions. The expanded source file names are denoted by the ";E" suffix on the filenames.

Now, the source files with expanded macros can be assembled. To do this, enter:

```
RASM MAIN;E LPT1 MAIN
RASM SUBR;E LPT1 SUBR
```

We have directed the assembly list output to the line printer and the assembled object program code to files MAIN and SUBR, both of which may now be input to the link editor in order to create an executable binary load module. To link these object program files, enter:

```
LINK LNKSRT LPT1
```

LNKSRT is a small file containing only commands for the link editor. Note that the PHASE command specifies the name of the binary load module as BUBBLE and it begins at address H'0000'. The included main program, MAIN, itself contains an INCLUDE command for the subroutine object program file. This causes the subroutine object program file to be linked first, in front of the MAIN program. The module entry address, specified by the main program END statement, is resolved to H'00D8'. Refer to the loadmap listing for particulars regarding resolution of addresses by the link editor.

The binary load module, BUBBLE, may now be loaded and executed by entering the SDOS commands XEQ or LOAD and GO.

ASSEMBLY LISTING

Figures 10-1 through 10-5 are sample program listings produced by RASM. The following explanations are keyed to the listing.

1. Page heading - Displays the current versions and level of RASM and the title, if any, specified by the TITL assembler directive.

2. Page number - Every page of the listing is numbered sequentially.

3. Location column - The numbers in this column are equal to the value of the assembly Location Counter and indicate the address at which the first byte is located.

4. Object - This field describes the data bytes which are stored sequentially starting at the address in the Address Column.

5. Address column - The numbers in this column are the results of evaluating address expressions which are in the instruction operand field. For the EQU and SET directives, this column contains the hexadecimal value of the operand field.

6. Error column - This column may contain an error code as detailed elsewhere in this chapter.

7. Statement Number - This number corresponds to the line number of the source program file.

8. Source code - This area of the listing reproduces the source code as it was read by the assembler.

9. Total errors - This field indicates the total number of errors detected by the assembler during passes 1 and 2 of the assembly process.

```
 1 *
 2 * SYSMACRO FILE - CONTAINS STANDARD 2650 REGISTER AND CONDITION CODE EQUATES
 3 *
 4 *
 5 *  REGISTER EQUATES
 6 *
 7 .MACRO
 8  REGS
 9 .T /
10 R0/EQU/0
11 R1/EQU/1
12 R2/EQU/2
13 R3/EQU/3
14 .MEND
15 *
16 * CONDITION CODE EQUATES
17 *
18 .MACRO
19  CODES
20 .T /
21 EQ/EQU/0
22 GT/EQU/1
23 LT/EQU/2
24 Z/EQU/0
25 P/EQU/1
26 N/EQU/2
27 UN/EQU/3
28 .MEND
29 *
30 * END OF SYSMACRO FILE
31 *
```

Figure 10-1. SYSMACRO File

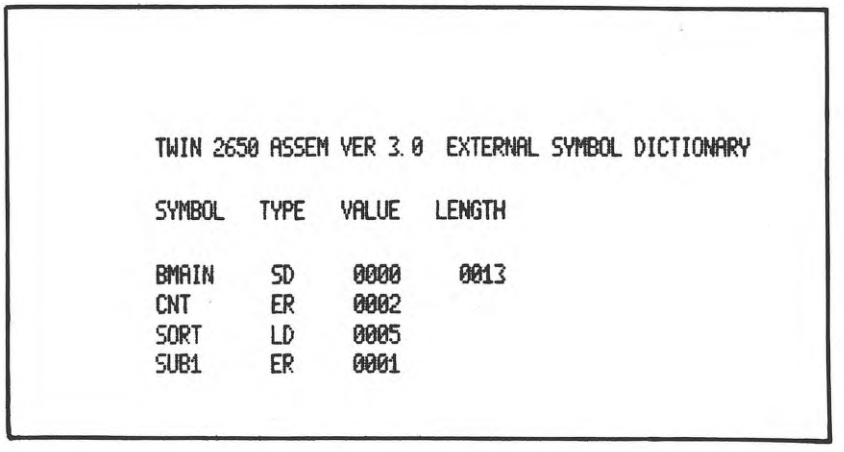


Figure 10-2. MAIN Program Assembly (Page 1)

```
TWIN 2650 ASSEM VER 3.0  BUBBLE SORT                                    PAGE    2

 LOC OBJECT   ADDR E STMT   SOURCE LINE

                      2   BMAIN   CSECT
                      3   *
                      4   * ARITHMATIC BUBBLE SORT PROGRAM
                      5   *
                      6   * GET REGISTER AND CONDITION CODE EQUATES FROM SYSMACRO.
                      7         PRT     NOGEN          THE "G" PARAM WILL OVERRIDE THIS FLAG.
                      8         REGS
                     13         CODES
                     21         PRT     GEN
                     22   *
                     23   *     MAIN PROGRAM
                     24   *
0000 0F0012   0012    25   STRT    LODA,R3 LEN            LOAD BUFFER LENGTH IN R3
0003 7502             26           CPSL    2              SET ARITHMATIC COMPARISONS
0005 A701             27   SORT    SUBI,R3  1              DECREMENT LOOP COUNTER
0007 CF0000   0000    28           STRA,R3 CNT            STORE LOOP COUNTER
000A 1A05     0011    29           BCTR,N  STOP           IF COUNT DECREMENT PAST 0, STOP
000C 3F0000   0000    30           BSA     SUB1
000F 1B74     0005    31           BR      SORT
0011 40               32   STOP    HALT
                      33   *
0012 0F               34   LEN     DATA    15             LENGTH OF BUFFER
                      35   *
                      36           REPRO                  COMMAND TO LINK EDITOR
                      37           INCLUDE SUBR
                      38           ENTRY   SORT
                      39           EXTRN   SUB1,CNT
              0000    40           END     STRT
```

Figure 10-2. MAIN Program Assembly (Page 2)

```
TWIN 2650 ASSEM VER 3.0   SYMBOL DICTIONARY                    PAGE    3

SYMBOL   TYPE   VALUE   LENGTH

BMAIN     SD     0000     0013
BMAIN     LD     0000
CNT       ER     0002
EQ        CD     0000
GT        CD     0001
LT        CD     0002
LEN       LD     0012
N         CD     0002
P         CD     0001
R0        CD     0000
R1        CD     0001
R2        CD     0002
R3        CD     0003
STRT      LD     0000
SORT      LD     0005
STOP      LD     0011
SUB1      ER     0001
UN        CD     0003
Z         CD     0000

  TOTAL NUMBER OF ERRORS    0
```

Figure 10-2. MAIN Program Assembly (Page 3)

```
TWIN 2650 ASSEM VER 3.0  EXTERNAL SYMBOL DICTIONARY                PAGE    1

SYMBOL   TYPE   VALUE   LENGTH

BSUBR     SD     0000     00D8
CNT       LD     00C8
LOOP      LD     0004
SUB1      LD     0001
```

Figure 10-3. SUBR Program Assembly (Page 1)

```
TWIN 2650 ASSEM VER 3.0                                                PAGE    2

 LOC OBJECT   ADDR E STMT   SOURCE LINE

                      1   *  SUBROUTINE FOR BUBBLE SORT.
                      2   *
                      3   *
                      4   * GET REGISTER AND CONDITION CODE EQUATES FROM SYSMACRO.
                      5   *
                      6          REGS
                     11          CODES



                     20   BSUBR CSECT
0000 00              21   ZERO   DATA    0
                     22   *
                     23   *  DOES ONE ITERATION THROUGH BUFFER
                     24   *
0001 0E0000   0000   25   SUB1   LODA,R2 ZERO          R2 COUNTS COMPARISONS
0004 EE00C8   00C8   26   LOOP   COMA,R2 CNT           IF =, ITERATION COMPLETE
0007 14              27          RETC,EQ
0008 0E60C9   00C9   28          LODA,R0 BUF,R2        LOAD FIRST # OF THE PAIR
000B EE20C9   00C9   29          COMA,R0 BUF,R2,+      COMPARE WITH SECOND #
000E 9974     0004   30          BCFR,GT LOOP          IF FIRST LT OR =, LOOP BACK
0010 C1              31          STRZ    R1            MOVE LARGER # TO R1
0011 0E60C9   00C9   32          LODA,R0 BUF,R2        LOAD SMALLER # INTO R0
0014 CE60C8   00C8   33          STRA,R0 BUF-1,R2      STORE SMALLER # IN FIRST
0017 01              34          LODZ    R1            MOVE LARGER # TO R0
0018 CE60C9   00C9   35          STRA,R0 BUF,R2        STORE LARGER # SECOND
001B 1B67     0004   36          BCTR,UN LOOP          LOOP BACK
                     37   *
                     38   *
00C8                 39          ORG     200
00C8                 40   CNT    RES     1
00C9                 41   BUF    RES     15            BUFFER TO BE SORTED
                     42   *
                     43          ENTRY   SUB1,LOOP,CNT
                     44          END
```

Figure 10-3. SUBR Program Assembly (Page 2)

```
TWIN 2650 ASSEM VER 3.0   SYMBOL DICTIONARY                    PAGE    3

SYMBOL   TYPE   VALUE   LENGTH

BSUBR     SD     0000     00D8
BSUBR     LD     0000
BUF       LD     00C9
CNT       LD     00C8
EQ        CD     0000
GT        CD     0001
LT        CD     0002
LOOP      LD     0004
N         CD     0002
P         CD     0001
R0        CD     0000
R1        CD     0001
R2        CD     0002
R3        CD     0003
SUB1      LD     0001
UN        CD     0003
Z         CD     0000
ZERO      LD     0000

  TOTAL NUMBER OF ERRORS    0
```

Figure 10-3. SUBR Program Assembly (Page 3)

```
TWIN 2650 LINK EDITOR VER 3.0   LOAD MAP

PHASE       BEGIN    END  ENTRY     SYMBOL  TYPE   VALUE  OFFSET    PAGE   1

*
* LNKSRT - LINK MAIN PROGRAM AND SUBROUTINE FOR BUBBLE SORT
*

BUBBLE       0000   00EA   00D8     BSUBR   CSECT   0000    0000
                                     CNT     ENTRY   00C8
                                   * LOOP    ENTRY   0004
                                     SUB1    ENTRY   0001

                                     BMAIN   CSECT   00D8    00D8
                                   * SORT    ENTRY   00DD
```

Figure 10-4. Load Map

```
1 *
2 * LNKSRT - LINK MAIN PROGRAM AND SUBROUTINE FOR BUBBLE SORT
3 *
4  PHASE BUBBLE,H'0000',BUBBLE SORT PROGRAM
5  INCLUDE MAIN
```

Figure 10-5. LINK Command File

APPENDIX A
MESSAGES

Error Message	Meaning	Recovery Procedure
ASSEMBLY		
A - Argument error	The argument has been coded in such a way that it cannot be resolved to a unique value.	Self-explanatory
L - Label error	The label contains too many characters, contains invalid characters, has been previously defined, or is an invalid symbol.	Self-explanatory
O - Op-code error	The op-code mnemonic has not been recognized as a valid mnemonic.	Self-explanatory
P - Paging error	A memory access instruction has attempted to address across a page boundary.	Self-explanatory
R - Register field error	The register field expression could not be evaluated, or when evaluated, was less than 0 or greater than 3, or the register field was not found.	Self-explanatory
S - Syntax error	The instruction has violated some syntax rule.	Self-explanatory
U - Undefined symbol	There is a symbol in the argument field which has not been previously defined.	Self-explanatory
V - Overflow	Numeric or expression stack overflow.	Self-explanatory
W - Warning	The assembler has detected a syntactically correct but unusual construction.	Self-explanatory

Error Message	Meaning	Recovery Procedure
INPUT/OUTPUT		
ERR<xx><ID>FILE	And I/O error has occurred and the program is terminated. <xx> denoted for SRB status returned for the I/O operation. <ID> identifies the concerned file. The possible identifiers are: S - Source file L - Listing file O - Object program file	Correct the problem in the named file according to the SRB status code (the list is contained in this appendix).
INVALID 4th PARAMETER	Optional macro expansion print parameter is neither G nor blank. Processing is aborted.	Reenter the RASM command with the correct parameter.
SYMBOL TABLE OVERFLOW	Symbol table is full.	Reduce the number of symbols in the program source file and reassemble.
LINK EDITOR		
NO PHASE	When the first two link commands are INCLUDE, a PHASE is required preceding them.	Insert a PHASE card in front of the LINK command stream and reLINK the files.
INVALID RECORD	An invalid command record is encountered.	Self-explanatory
MULTIPLY DEFINED SYMBOL	An external symbol is defined in more than one object program module.	Change the symbol name in one of the object program modules.
NESTING ERROR	Included modules are too deep. A nesting level up to 4 can be accepted.	Adjust INCLUDE nesting to 4 levels or less.
PAGING ERROR	An instruction with a 13-bit address refers to a location outside the current page.	Self-explanatory

Error Message	Meaning	Recovery Procedure
LINK EDITOR (Cont'd)		
TOO MANY PHASES	The object file specifies more than 20 phases.	Link to intermediate phases, then link these to the final load module.
UNDEFINED SYMBOL	A symbol is defined as EXTRN in one of the object modules, but it has not been defined as ENTRY in one of the other modules.	Non-fatal error.
ERR <xx> <ID> FILE	An I/O error has occurred and the program is terminated. <xx> represents the SDOS SRB status. <ID> represents the concerned file. Possible values of <ID> are: I - One of the input files L - Load map file/device B - Binary load module output	Correct the problem in the named file according to the SRB status (the list is contained in this appendix).
MACRO FACILITY		
<C> <source macro call> <macro definition>	An error has been detected during the processing of an instruction. where <C> : C - Invalid control instruction P - Error in prototype D - Error in macro definition I - Error in macro instruction O - Output line becomes too long	Correct the line in error as dictated by the <C> error flag.

Error Message	Meaning	Recovery Procedure
MACRO FACILITY (Cont'd)		
ERR <xx> <ID> FILE	An input/output error has occurred and the program is terminated. <xx> represent the SDOS SRB status <ID> represents the concerned file. Possible values of <ID> are: I - Input source file O - Output expanded source file M - One of the macro definition files	Correct the problem in the named file to the SRB status code (the list is contained in this appendix).
INVALID MACRO NAME	The macro name is more than five characters or the macro prototype contains no macro name.	Shorten the macro name to five characters or enter a valid prototype statement.
TOO MANY MACRO DEFINITIONS	The internal macro storage table or macro name table has overflowed. Up to 64 macro names are allowed. This error causes processing to abort.	Split the macro definition file and run the source file against each smaller macro definition file.
NO MACRO DEFS	No user library was specified and SYSMACRO was not found on drive 0. Processing is aborted.	Rerun MAC. Specify the correct macro library file.

SDOS SRB Status Codes

Status	Description
0	Function complete/no error
1	New file
2	Illegal channel number
3	Channel not assigned
4	Channel busy
5	Illegal function code
6	Short read
7	Short write
8	Illegal drive number
9	File in use
A	Device not operational
B	Device not available
C	Device not ready
D	Device in use
E	Directory read error
F	Directory write error
10	Directory full
11	Device read error
12	Device write error
13	Invalid address: Utility routine conflict or memory wrap-around
14	Unused

SDOS SRB Status Codes (Cont'd)

Status	Description
15	File name in use
16	Illegal file name
17	File in R/W progress
18	Channel already assigned
19	Incorrect diskette
7F	I/O in progress
FF	End of file/end of device

APPENDIX B
SUMMARY OF 2650 INSTRUCTION MNEMONICS

In these tables parentheses are used to indicate options. In no case are they coded in any instruction. The following abbreviations are used:

r— register expression, must evaluate to $0 \leqslant r \leqslant 3$.
v— value expression
*— indirect indicator
a— address expression
x— index register expression
X— index register expression with optional auto-increment or auto-decrement

Note

— The use of the indirect indicator is always optional.

— When an index register expression is specified, it can be followed by ',+' or ',–' which indicates use of auto-increment or auto-decrement of the index register. Example:

```
LODA,0      DPR,R3,+
```

BXA, BSXA are exceptions and do not permit auto-increment or auto-decrement.

— Even though an address expression is specified in a hardware relative addressing instruction, the assembler develops it into a value of $(-64 \leqslant v \leqslant +63)$.

— A memory reference instruction which requires indexing may use only register 0 as the destination of the operation.

— If an index register expression is used with either BXA or BSXA instructions, it must specify index register #3 (either register bank) for indexing. Any other value in the index field will produce an error during assembly. However, it is not necessary to use an index register expression with these instructions: a blank in this field will default to register 3.

LOAD/STORE INSTRUCTIONS			Length (bytes)
LODZ	r	Load Register Zero	1
LODI,r	v	Load Immediate	2
LODR,r	(*)a	Load Relative	2
LODA,r	(*)a(,X)	Load Absolute	3
STRZ	r	Store Register Zero	1
STRR,r	(*)a	Store Relative	2
STRA,r	(*)a(,X)	Store Absolute	3

ARITHMETIC INSTRUCTIONS			
ADDZ	r	Add to Register Zero	1
ADDI,r	v	Add Immediate	2
ADDR,r	(*)a	Add Relative	2
ADDA,r	(*)a(,X)	Add Absolute	3
SUBZ	r	Subtract from Register Zero	1
SUBI,r	v	Subtract Immediate	2
SUBR,r	(*)a	Subtract Relative	2
SUBA,r	(*)a(,X)	Subtract Absolute	3

LOGICAL INSTRUCTIONS			
ANDZ	r	And to Register Zero	1
ANDI,r	v	And Immediate	2
ANDR,r	(*)a	And Relative	2
ANDA,r	(*)a(,X)	And Absolute	3
IORZ	r	Inclusive or to Register Zero	1
IORI,r	v	Inclusive or Immediate	2
IORR,r	(*)a	Inclusive or Relative	2
IORA,r	(*)a(,X)	Inclusive or Absolute	3
EORZ	r	Exclusive or to Register Zero	1
EORI,r	v	Exclusive or Immediate	2
EORR,r	(*)a	Exclusive or Relative	2
EORA,r	(*)a(,X)	Exclusive or Absolute	3

COMPARISON INSTRUCTIONS			
COMZ	r	Compare to Register Zero	1
COMI,r	v	Compare Immediate	2
COMR,r	(*)a	Compare Relative	2
COMA,r	(*)a(,X)	Compare Absolute	3

ROTATE INSTRUCTIONS			
RRR,r		Rotate Register Right	1
RRL,r		Rotate Register Left	1

BRANCH INSTRUCTIONS			
BCTR,v	(*)a	Branch on Condition True Relative	2
BCFR,v	(*)a	Branch on Condition False Relative	2
BCTA,v	(*)a	Branch on Condition True Absolute	3
BCFA,v	(*)a	Branch on Condition False Absolute	3
BRNR,r	(*)a	Branch on Register Non-Zero Relative	2
BRNA,r	(*)a	Branch on Register Non-Zero Absolute	3
BIRR,r	(*)a	Branch on Incrementing Register Relative	2
BIRA,r	(*)a	Branch on Incrementing Register Absolute	3
BDRR,r	(*)a	Branch on Decrementing Register Relative	2
BDRA,r	(*)a	Branch on Decrementing Register Absolute	3
BXA	(*)a(,x)	Branch Indexed Absolute, Unconditional	3
ZBRR	(*)a	Zero Branch Relative, Unconditional	2

SUBROUTINE BRANCH/RETURN INSTRUCTIONS			Length (bytes)
BSTR,v	(*)a	Branch to Subroutine on Condition True, Relative	2
BSFR,v	(*)a	Branch to Subroutine on Condition False, Relative	2
BSTA,v	(*)a	Branch to Subroutine on Condition True, Absolute	3
BSFA,v	(*)a	Branch to Subroutine on Condition False, Absolute	3
BSNR,r	(*)a	Branch to Subroutine on Non-Zero Register, Relative	2
BSNA,r	(*)a	Branch to Subroutine on Non-Zero Register, Absolute	3
BSXA	(*)a(,x)	Branch to Subroutine, Indexed, Unconditional	3
RETC,v		Return From Subroutine, Conditional	1
RETE,v		Return From Subroutine and Enable Interrupt, Conditional	1
ZBSR	(*),a	Zero Branch to Subroutine Relative, Unconditional	2

PROGRAM STATUS INSTRUCTIONS			
LPSU		Load Program Status, Upper	1
LPSL		Load Program Status, Lower	1
SPSU		Store Program Status, Upper	1
SPSL		Store Program Status, Lower	1
LDPL[1]		**Load Program Status, Lower**	**3**
STPL[1]		**Store Program Status, Lower**	**3**
CPSU	v	Clear Program Status, Upper, Selective	2
CPSL	v	Clear Program Status, Lower, Selective	2
PPSU	v	Preset Program Status, Upper, Selective	2
PPSL	v	Preset Program Status, Lower, Selective	2
TPSU	v	Test Program Status, Upper, Selective	2
TPSL	v	Test Program Status Lower, Selective	2

INPUT/OUTPUT INSTRUCTIONS			
WRTD,r		Write Data	1
REDD,r		Read Data	1
WRTC,r		Write Control	1
REDC,r		Read Control	1
WRTE,r	v	Write Extended	2
REDE,r	v	Read Extended	2

MISCELLANEOUS INSTRUCTIONS			
HALT		Halt, Enter Wait State	1
DAR,r		Decimal Adjust Register	1
TMI,r	v	Test Under Mask Immediate	2
NOP		No Operation	1

[1] 2650B only

APPENDIX C
NOTES ABOUT THE 2650 MICROPROCESSOR

1. AUTO-INCREMENT, DECREMENT of index register. This feature is optional on any instruction which used indexing with the exception of BXA and BSXA. The increment or decrement occurs before the index register is added to the displacement in the instruction.

2. The contents of registers when used for indexing are considered to be unsigned absolute numbers. Consequently, index registers can contain values from 0 to 255. They "wrap-around" so that the number following 255 is 0.

3. Only absolute addressing instructions can be indexed.

4. The Branch on Incrementing Register or Decrementing Register instructions perform the increment or decrement before testing for zero. The only time the branch is not made is when the register contains zero.

5. All hardware relative addressing is implemented as modulo 8K and therefore relative addressing across the top of a page boundary will result in a physical address near the bottom of the page being accessed. For example:

   $1FFC_{16}$ LODR,R2 $+16

   This instruction results, during execution, in accessing the byte at location 000C in the same page as the instruction. Similarly, negative relative addresses from near the bottom of a page may result in an effective address near the top of the page.

6. Page boundaries cannot be indexed across.

7. Data can always be accessed across a page boundary through use of relative indirect or absolute indirect addressing modes.

8. The only way to transfer control to a program in some other page is to branch absolute or branch indirectly to the new page. Program execution cannot flow across a page boundary.

9. Unconditional branch or branch to subroutine instructions are coded by specifying a value of 3 in the register/value field of BSTA, BSTR, BCTA or BCTR. Example:

```
UN      EQU       3
        • • •
        • • •
        • • •
        BSTA,UN   PAL
        BCTR,3    LOOP
```

   Unconditional branches or subroutine branches on condition false (BCFA, BCFR, BSFA, BSFR) are not allowed.

APPENDIX D
ASCII CODE

This table presents the only characters that the assembler will recognize in an A type constant and their equivalent codes in hexadecimal.

VALID CHARACTERS	ASCII CODE	VALID CHARACTERS	ACSII CODE
0	30	V	56
1	31	W	57
2	32	X	58
3	33	Y	59
4	34	Z	5A
5	35	blank	20
6	36	.	2E
7	37	(	28
8	38	+	2B
9	39	|	7C
A	41	&	26
B	42	!	21
C	43	$	24
D	44	*	2A
E	45	)	29
F	46	;	3B
G	47	¬ or ~	7E*
H	48	–	2D
I	49	/	2F
J	4A	,	2C
K	4B	%	25
L	4C	_ or ←	5F*
M	4D	>	3E
N	4E	?	3F
O	4F	:	3A
P	50	#	23
Q	51	@	40
R	52	'	27
S	53	=	3D
T	54	"	22
U	55	<	3C

*may have different graphic symbols on different output devices.

APPENDIX E
ASCII CHARACTER SET

ACSII CHARACTER SET (7-BIT CODE)									
L.S. CHAR \ M.S. CHAR		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	–	=	M	]	m	}
E	1110	SO	RS	•	>	N	↑	n	~
F	1111	SI	US	/	?	O	← or _	o	DEL

APPENDIX F
POWERS OF TWO TABLE

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 45
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

APPENDIX G
HEXADECIMAL-DECIMAL CONVERSION TABLES

From hex: locate each hex digit in its corresponding column position and note the decimal equivalents. Add these to obtain the decimal value.

From decimal: (1) locate the largest decimal value in the table that will fit into the decimal number to be converted, and (2) note its hex equivalent and hex column position. (3) Find the decimal remainder. Repeat the process on this and subsequent remainders.

The table on pages 85-88 provides for direct conversion of hexadecimal and decimal numbers in these ranges:

Hexadecimal	Decimal
000 to FFF	0000 to 4095

In the table, the decimal value appears at the intersection of the row representing the most significant hexadecimal digits (16^2 and 16^1) and the column representing the least significant hexadecimal digit (16^0).

Example: $C21_{16} = 3105_{10}$

HEX	0	1	2
C0	3072	3073	3074
C1	3088	3089	3090
C2	3104	3105	3106
C3	3120	3121	3122

HEXADECIMAL COLUMNS												
6		5		4		3		2		1		
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	
0	0	0	0	0	0	0	0	0	0	0	0	
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1	
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2	
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3	
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4	
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5	
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6	
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7	
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8	
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9	
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10	
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11	
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12	
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13	
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14	
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15	
BYTE 1				BYTE 2				BYTE 3				

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	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	2761	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

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

COMMAND/DIRECTIVE SUMMARY

Command/Directive	Description	Page Number
{name} ACON expression	Assembler allocates two successive bytes	27
.ASSIGN &<variable> = <expression>	Value of expression assigned to variable	59
CEJE expression	Same as EJE if remaining number of lines is less than value of expression	33
label CSECT	Identify an object module	43
{name} DATA argument	Assembler allocates exact number of bytes	28
.DECLARE &<variable>,...	Declares variables for use by macroprocessor	59
EJE	Assembler advances listing to top of page	32
ELSE	Generate different sequences of statements	40
END {expression}	Tells assembler last statement has been input	31
ENDIF	End of conditional assembly construct	40
ENTRY label list ...	Specify symbols referenced by other programs	44
name EQU expression	Equate symbol in label field with evaluatable expression in operand field	25
EXTRN label list ...	Specify symbols defined in some other program which are referred to in this program	45
.GENERATE &<variable> = <expression1>,<expression2> .END	Statement pair delimits group of statements to be processed iteratively	59
{name} IF expression	Programmer can vary the sequence of statements	40
.IF <expression> .&<macrolabel>	Conditionally include block of text	59
_INCLUDE name	Include separately assembled object program module	49
.JUMP .&<macrolabel>	Indicate next statement to be scanned	59
LINK,<input>,[<loadmap>],[<bfile>]	Link editor command	46
MAC,<source>,<expanded source>, [<macro file>],[<error list>]	Macro preprocessor command	50
.MACRO ⋮ .MEND } macro definition	Command pair brackets each macro definition	60
.&<macrolabel>	Label macro command	59
.MESSAGE<text>	Display text on system console	60
.NOP	No op used for defining macrolabels	60
{name} ORG expression	Set asembly time location counter	24
PCH {ON / OFF}	Resume/discontinue output of object program module	37
PHASE<bfile>[,<origin>] [,<ID>]	Name a phase to be created by link editor	48
PRT {ON / OFF / GEN / NOGEN}	Resume/discontinue printing of assembled program	34
RASM,<ifile>,[<list>], [<ofile>], [<print>]	Relocatable assembler command	18
REPRO	Copy link editor commands from source module	38
{name} RES expression	Assembler reserves contiguous bytes of storage	30
name SET expression	Equate symbol in label field with expression in argument field	26
SPC expression	Assembler skips or spaces a number of lines	35
.SUBSTRING &<variable> = &<variable> (,<expression1>,<expression2>)	Second variable substring assigned to first variable	60
.TAB character	Sets the tab character	60
TITL expression	Specifies a title for the assembly listing	36

COMMENT SHEET

TITLE:

REVISION:

This form is not intended to be used as an order blank. Signetics Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

CUT ON THIS LINE

General comments:

FROM NAME: _____ POSITION: _____

COMPANY NAME: _____

ADDRESS: _____

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 166
SUNNYVALE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

signetics

a subsidiary of **U.S. Philips Corporation**

Signetics Corporation
P.O. Box 9052
811 East Arques Avenue
Sunnyvale, California 94086

Bin No. 038 MOS Microprocessor Division

CUT ON THIS LINE

FOLD

FOLD

STAPLE

STAPLE

signetics

a subsidiary of **U.S. Philips Corporation**

Signetics Corporation
P.O. Box 9052
811 East Arques Avenue
Sunnyvale, California 94086
Telephone 408/739-7700