Signetics

MCCAP Microcontroller Cross Assembler Program

MICROCONTROLLER CROSS ASSEMBLER PROGRAM (MCCAP)

Development Systems Products Group

Bipolar LSI Division

January 1983

DSPG Document No. 79-101

Revised January 1983

FORTRAN MCCAP ERROR

Summary: An error has been identified in the FORTRAN version of the MCCAP CROSS ASSEMBLER (8X300 AS1-* *SS). This error can be corrected by adding one statement to the MCCAP SOURCE PROGRAM.

Error: Cannot write extension code to a statement which selects a right bank data field variable using the SEL statement.

```
EXAMPLE - EXT     EQ     100H
          DATA    RIV    123H, 7, 8
                  SEL    DATA/EXT
```

Solution: Add one statement, "GO TO 7500", right after the statement with the label 3240 in subroutine PASS2 of the FORTRAN MCCAP source program as shown below.

```
                0
                0       PROCESS SEL STATEMENT
                0
0351            3200    CALL SCAN (IARG,IVAL)
0352                    MODE = 3
0353                    LEN = 1
0354                    GO TO (3210,9100,9000,3210,9300,9300,9300,3210,9300),IERR
0355            3210    IF(IVIND) 9600,9600,3200
0356            3220    L = IVVAL/8
0357                    ITYPE = 7
0358                    IBYTE = IVAL
0359                    IF(L-2) 9600,3230,3240
0360            3230    IBIN = 1792+IBYTE
0361                    GO TO 7500
0362            3240    IBIN = 3840+IBYTE
0363                    GO TO 7500 <---------- new statement
                0
                0       PROCESS MACRO STATEMENT
                0
0364            3300    GO TO 8000
```

* = 1, 2, 3 or 4 depending on density or encoding

PREFACE

The MicroController Cross Assembler Program (MCCAP) has been developed to support the Signetics 8X300/8X305 MicroController. MCCAP provides many powerful features including macros, automatic subroutine handling, conditional assembly and extended instructions. These features significantly reduce the time required to compose and assemble MicroController programs. When combined with standard assembler features such as mnemonic op-codes and address labels, these extended features make MCCAP a powerful programming tool.

As input, MCCAP accepts source code written according to the rules presented in this manual. After assembling the source input, MCCAP produces an assembly listing and machine-readable object module.

MCCAP is written in ANSI standard FORTRAN IV and is available on the more popular timesharing services. MCCAP is also available as a fully supported product from Signetics for use on a user's in-house system.

This manual assumes a familiarity with the 8X300/8X305 Micro-Controller and its instruction set. Those unfamiliar with the 8X300/8X305 should read Appendix D before reading the main body of this manual.

CONTENTS

SECTION 1

INTRODUCTION

The MicroController Cross Assembler Program (MCCAP) translates symbolic statements into object code that can be executed by the 8X300/8X305 MicroController. The assembler consists of two passes which build a symbol table, issue helpful error messages, produce a detailed program listing, and output a machine-readable object module.

MCCAP is written in ANSI standard FORTRAN and runs on large-scale computers, (MCCAP is also written in Intel 8080 assembly language and runs on the Intel Intellec Microcomputer Development System.) It requires a direct access mass storage device such as disk, an input device for source code, and two output devices--one to output the assembly listing and one to output the object module. The program uses a minimum of memory but is modularized and may be linked to execute in overlays if memory restrictions require.

1.1 MCCAP FEATURES

The MCCAP assembler language has been developed with the following features:

- Free format source code
- Reserved symbols for registers
- ASCII character set
- Symbolic address assignments
- Forward referencing
- Address arithmetic
- Bit or byte manipulation
- Macros nested to three levels
- Conditional assembly
- Automatic procedure/subroutine handling
- Symbolic data field references
- Comments for self-documenting code
- Cross reference of the symbol table
- MicroController instruction words to 32 bits wide
- Versatile object file format specification

1.2 MCCAP ASSEMBLY PROCESS

When a program is assembled with MCCAP, two types of information are produced: an assembly listing and an object module.

The main purpose of the assembly listing is to convey all pertinent information about the assembled program; that is, memory addresses and their machine code contents, the original source code, and any error indications. The listing may also be used as a documentation tool through the inclusion of comments. The assembly listing may be displayed on a CRT or printed on a line printer.

The **object module** is the executable machine code produced from the source code. The object module is produced in a computer-readable format.

1.3 MCCAP OPERATION

MCCAP is a two-pass assembler. This means that the source code is scanned twice. During the first pass, symbols are examined and placed in a symbol table. Certain errors may be detected during this pass and will be retained for display in the assembly listing. In the second pass, symbolic addresses are resolved, the object code is generated, and the assembly listing and object module are produced. Errors detected during the second pass are included on the assembly listing with those errors detected during the first pass.

1.4 SOURCE PROGRAM ASSEMBLY

The following steps are used in assembling source programs:

1. Write a source program.
2. Transfer the source program to a computer-readable medium.
3. Assemble the source code program using the MCCAP assembler.
4. Obtain an assembly listing and an object module.

SECTION 2

SYNTAX AND FORMAT RULES

The assembler language has a character set, vocabulary, rules of grammar and allows individuals to define new elements. The rules that describe the language are called the syntax of the language. Likewise, an assembler language also has rules of format. For a MCCAP program to be translated properly, it must be written in accordance with the rules of syntax and format.

2.1 CHARACTER SET

A MCCAP program statement may not be more than 80 characters long and must include only valid MCCAP characters. These characters consist of all alphabetic characters (the letters A-Z), all numbers (0-9) and the special symbols shown in Table 2-1. The use of any other characters will result in errors.

Table 2-1. Valid Special Symbols in MCCAP Programs

Character	Description
b	blank character (space)
'	single quote
,	comma
+	plus sign
-	minus sign
/	slash
$	dollar sign
*	asterisk
(	left parenthesis
)	right parenthesis
>	greater than
<	less than
@	commercial "at" sign
.	period
&	ampersand
"	double quote
#	sharp
%	percent
:	colon
;	semi-colon
=	equal
?	question mark
!	exclamation point

2.2 SYMBOLS

A symbol is a sequence of characters that may be used to represent a register, an arithmetic value, a memory address, or an I/O data field. Only the first six characters of a symbol are scanned by MCCAP. Any remaining characters in a symbol will be treated

as documentation. **The first character of a symbol must be alphabetic.** Any other character may be alphabetic or numeric. The use of special characters or imbedded blanks within a symbol will result in an error indication.

The number of characters in a symbol and the number of symbols in the symbol table (usually 500) may be modified during the installation of MCCAP. (Refer to MCCAP Installation and Maintenance Manual for details.)

Example 2-1. Examples of Symbols

```
LOOP1
GOBACK
TABPTRS    (MCCAP recognizes this as TABPTR)
LOOP#      (invalid: special character used)
2NDTRY     (invalid: starts with a numeric)
GO BACK    (invalid: imbedded blank)
```

2.3 RESERVED SYMBOLS

As shown in Table 2-2, the assembler has 18 symbols that are internally defined to save the user the necessity of defining them in each program. Typically, these symbols are used quite frequently, but they are not required.

Table 2-2. Reserved Symbols and Their Values

Symbols	Register	Octal Value	8X300 Usage	8X305 Usage
AUX	Auxiliary Reg.	0	S,D	S,D
R1	Register 1	1	S,D	S,D
R2	Register 2	2	S,D	S,D
R3	Register 3	3	S,D	S,D
R4	Register 4	4	S,D	S,D
R5	Register 5	5	S,D	S,D
R6	Register 6	6	S,D	S,D
IVL or R7	Left Bank Address Reg.	7	D Only	S,D
OVF	Overflow Reg.	10	S Only	S Only
R11	Register 11	11	S,D	S,D
R12	Register 12	12	---	S,D
R13	Register 13	13	---	S,D
R14	Register 14	14	---	S,D
R15	Register 15	15	---	S,D
R16	Register 16	16	---	S,D
IVR or R17	Right Bank Address Reg.	17	D Only	S,D

S= Source, D= Destination, ---= Invalid

2.4 CONSTANTS

MCCAP recognizes four types of **numeric constants:** decimal, octal, binary and hexadecimal. These are defined as a sequence of numeric characters optionally preceded by a plus or a minus sign and followed

by an alphabetic descriptor that indicates the type. If unsigned, the value is assumed to be positive. If no descriptor is given, the number is assumed to be decimal. The available descriptors are **B for binary, H for octal,** and **X for hexadecimal.**

Hexadecimal constants, in order not to be confused with symbols, must begin with a numeric character.

The size of a constant is limited to the field size specified by the format of the machine instruction being assembled. When a constant is negative, its two's complement representation is generated and placed in the field specified.

Example 2-2. Numeric Constants

300	-1	+52	-1000
100111B	+00111B	-11110B	
7755H	+513H	-5724H	
+0BCX	4F9X	-2CFX	0ACEX

An ASCII character may be specified as a **character constant** by enclosing the character within single quote marks. (To cause the ASCII code for the single quote mark to be generated, it must be specified by four single quote marks, that is, ''''='.) Each ASCII character converts to an 8-bit value with the high-order bit set to zero.

Example 2-3. Character Constants

```
'G'
'O'
'?'
```

2.5 EXPRESSIONS

An expression is a sequence of operands (symbols, constants, or other expressions) separated by one of the operators shown in Table 2-3.

Table 2-3. Recognized Operators

Operator	Function
+	**Plus:** produces the sum of its operands.
-	**Minus:** produces the difference between its operands, or produces the negative value of its operand when used as a unary minus.
$	**Logical AND:** produces the bit-by-bit logical product of its operands.
.R.	**Right shift:** shifts the first operand right the number of bit positions specified by the second operand. Zeros are shifted into the high-order bit and bits are "dropped off" the low-order bit.
.L.	**Left shift:** shifts the first operand left the number of bit positions specified by the second operand. Zeros are shifted into the low-order bit, and bits are "dropped off" the high-order bit.

These operators are evaluated within an expression in two levels of hierarchy. Level 1 operators, $, .R., and .L., are evaluated first, from left to right as they are encountered. Level 2 operators, + and -, are then evaluated left to right as they are encountered. There is **no way** to alter the level of hierarchy. That is, parentheses are **not legal delimiters** to define an order of precedence that is not as described above.

Example 2-4. Expression Evaluation

Expression	Algebraic Equivalent
IN.L.A.-B$C	(IN.L.A.)-(B$C)
-A.R.4+B.L.3$C	-(A.R.4)+((B.L.3)$C)
A.R.B.L.C.+D$E	((A.R.B).L.C)+(D$E)

2.6 PROGRAM FORMAT

A complete program is composed of one or more program segments. The first program segment must be the main program, the one in which execution begins.

Procedures (subroutines) are program segments which perform a specific function and which may be executed from several points within the main program or other procedures. By creating the required function as a procedure, the statements associated with that function need be coded only once and then called out as needed.

To transfer to a procedure for execution and then return to the original program, "call" and "return" statements are provided in MCCAP.

The main program starts with a program title statement (PROG) and ends with the appearance of a procedure title statement (PROC) or a program END statement, if no procedures exist. Procedures begin with a procedure title statement (PROC) and are terminated by a procedure END statement. The complete program must be terminated by a program END statement. Only listing control and comment statements may appear

1. before the program title statement (PROG),
2. between a procedure END statement and the next procedure title statement (PROC), or
3. between a procedure END statement and the program END statement.

Example 2-5. General Program Organization

```
PROG name1     (Program Title Statement)
    .
    .
(directives, declarations, executable statements, call statements,
 and comments)
    .
    .
PROC name2     (Procedure Title Statement)
    .
    .
(directives, declarations, executable statements, call and return
 statements, and comments)
    .
    .
END name2      (Procedure End Statement)
PROC name3     (Procedure Title Statement)
    .
    .
(directives, declarations, executable statements, call and return
 statements, and comments)
    .
    .
END name3      (Procedure End Statement)
END name1      (Program End Statement)
```

2.7 STATEMENT FORMAT

MCCAP statements will always follow the general format:

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
field	field	field	field	field

In this manual each statement is explained in terms of this format, showing what data, if any, must appear in the various fields.

Each statement is written as an 80-column free-form image without regard for spacing other than that required to delimit one field from another. Logical columns 73-80 are simply reproduced on the assembly listing without processing. If desired, these columns may be used for sequence numbers, as shown in Appendix B.

The **label field** generally assigns values to symbols. If present, the label field must begin in logical column one.

The **operation field** specifies an executable statement, an assembler declaration, or an assembler directive. The operation field must either begin after column one or be separated from the label field by one or more blanks.

The **operand field** specifies operands for the code in the operation field. The operand field, if present, is separated from the operation field by at least one blank.

The **extension field** specifies code to be generated for parts of the microprocessor system other than the 8X300/8X305. The extension field, if present, is separated from the operand field by a slash.

The **comment field** enables the programmer to enter a message stating the purpose or intent of a statement or a group of statements. The comment field must be separated from the last **required** field of a statement by one or more blanks.

2.8 COMMENT STATEMENTS

A comment statement is a complete line dedicated to a message solely for documentation purposes. It is not processed by the assembler program but is merely reproduced on the assembly listing. A comment statement is indicated by beginning the line with an asterisk in the first column.

Example 2-6. Comment Statements

```
* DATA AND ADDRESS DECLARATIONS
* MACRO DEFINITIONS
* MAIN PROGRAM
```

(Taken from lines 10, 61, and 77 of Appendix B.)

SECTION 3

SYMBOLIC REFERENCES

When writing programs in MCCAP assembler language, symbolic references may be used to relieve the programmer of keeping track of absolute addresses and values, therefore reducing programming time. MCCAP recognizes four different types of symbolic references:

1. Location counter
2. Program storage
3. Data fields (typically divided into working storage addresses and I/O addresses)
4. Values (constants and variables)

Figure 3-1 depicts the two areas which may be addressed symbolically in a typical 8X300/8X305 system: program storage and I/O data fields (I/O ports, RAMs, peripherals, etc.).

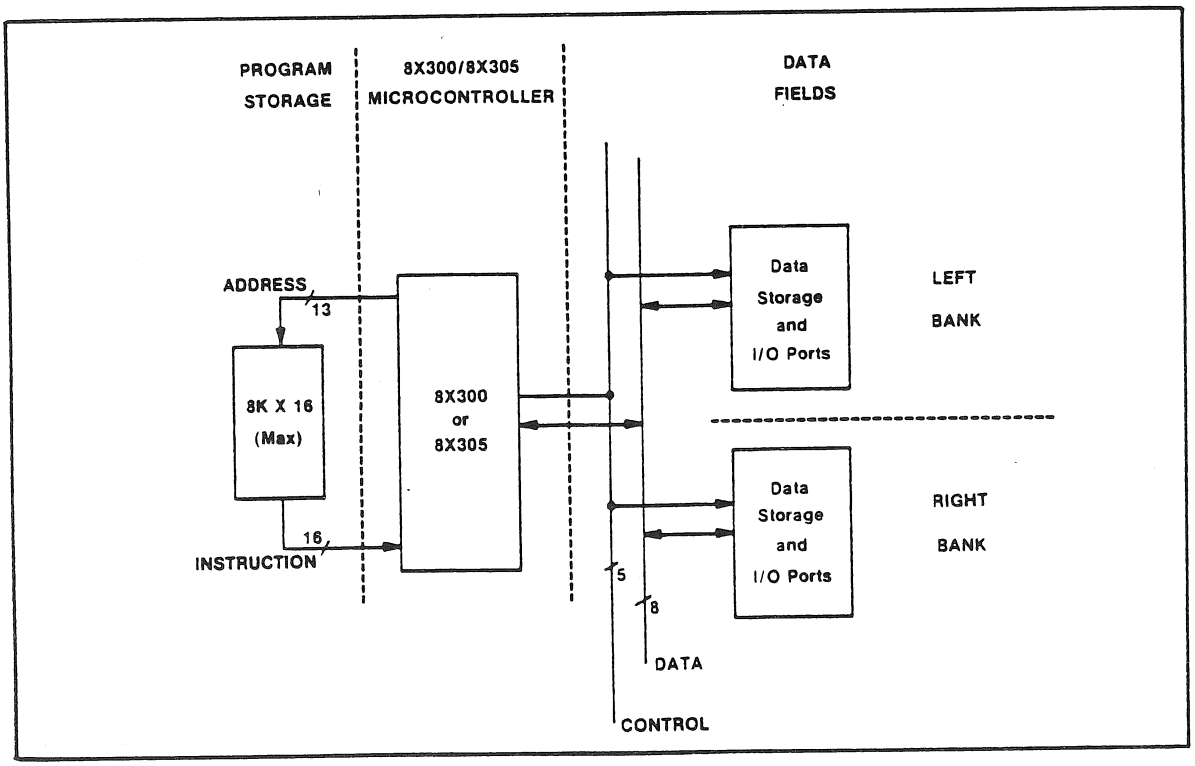


Figure 3-1. Typical 8X300/8X305 Configuration

3.1 ASSEMBLY LOCATION COUNTER

During the assembly process, MCCAP maintains a counter which always contains the address of the current program storage location for which machine code is being assembled. This counter is called the **location counter.** The special character * is the symbolic name

of the location counter and it may be used like any other symbol, except that it may **not** be used as such in the label field (an * in column 1 represents a comment). When using the *, the programmer may think of it as expressing the idea "*" = "this location".

The symbol * is the **only** valid symbol containing a special character that the assembler recognizes.

Example 3-1. Non-Labeled Reference

The use of a * in a program is shown as follows:

```
NZT      23H,4,*+6
RTN
LOOPCT   R5
SEL      DISP1
LOOPCT   DISP1
SEL      DISP2
LOOPCT   DISP2
```

(Taken from lines 229 through 235 of Appendix B.)

The *+6 in line 229 refers to "this location plus six", which is line 235.

3.2 PROGRAM STORAGE SYMBOLIC ADDRESSES

When writing a program, the programmer can optionally place a symbol in the label field of any of the executable statements. The assembler, upon detecting the symbol, assigns the value of the location counter to that symbol. The symbol can then be used in the operand field of any instruction in that program segment to reference the address of the statement in whose label field it appears. The important concept is that the absolute program storage address of an executable statement need not be known when writing in MCCAP; only a symbol is needed to reference the location of that statement.

Example 3-2. Labeled Reference

```
LAST     HALT
```

(Taken from line 91 of Appendix B.)

MCCAP also recognizes **relative addressing** of program locations, which is the use of label field symbols as "landmarks" to other executable statements nearby.

Forward referencing, referring to a symbol prior to its appearance in source code, is also valid in MCCAP but **only** when referencing symbols in label fields of executable statements. All other forward references will result in error indications.

Example 3-3. Relative Addressing

```
START   NOP
        XMIT    0,R1
        XMIT    0,R2
        LOOK    DSTAT,R1
STC     CALL    ARITH
        CALL    MOVMNT
        CALL    TRNSMT
        CALL    EXECT
        LOOPCT  R6
        NZT     OVF,START+3
```

(Taken from lines 81 through 90 of Appendix B.)

The expression START+3 in line 90 refers to three instructions after the statement with a label field of START, which would point to line 84.

3.3 DATA FIELD SYMBOLIC ADDRESSES

In addition to recognizing program storage address symbols, MCCAP recognizes data field address symbols as the operands of executable statements. Whereas program storage symbols are defined and recognized by their appearance in the label field of an executable statement, data field address symbols **must be defined separately** by the programmer in declaration statements prior to being used in any other source statements.

Further explanation is provided in Section 5 under LIV and RIV declaration statements.

3.4 SYMBOLIC VALUES

Assigning constant or variable values to symbols is another type of symbolic referencing in MCCAP. These constants or variables are declared in a fashion similar to that described above for data field symbols. Value symbols **must be defined prior** to being used in any executable statement.

Further explanation is provided in Section 5 under EQU and SET declaration statements.

3.5 GENERAL RULES

The following are additional rules which apply to symbolic references in MCCAP programs. Failure to adhere to these will result in error indications.

1. The program name must appear **only** in the program title and end statements.
2. Procedure and entry point names are global to the entire program.
3. Symbols declared within the main program segment are global to the entire program.
4. Symbols declared within a procedure by any declaration except SET are local to that procedure.
5. Symbols declared **anywhere** in a program by a SET declaration are global.
6. Control storage symbolic addresses are local to the program segment in which they appear.

Table 3-1. Accessibility of Symbol References

Statement Type	Main Program	Procedure (Subroutine)	Main Program Macro Call	Procedure Macro Call
EQU, LIV, RIV	Global	Local	Global	Local
SET	Global	Global	Global	Global
Directives	Error	Error	Error	Error
Executable	Local	Local	Local	Local

EXTENDED INSTRUCTIONS

The MCCAP Assembler assists the user with generating not only the 16-bit 8X300/8X305 instructions, but also an additional sixteen bits of code. These extensions specify code to be used for controlling parts of the microprocessor system other than the 8X300/8X305 instructions. They are addressed simultaneously with the 8X300/8X305 instructions and are used for the hardware selection of I/O ports or working storage. This technique reduces the program length and increases throughput of the system. Further descriptions of their usage are found in the 8X300/8X305 applications literature; here we will simply discuss the generation of these instruction extensions by MCCAP.

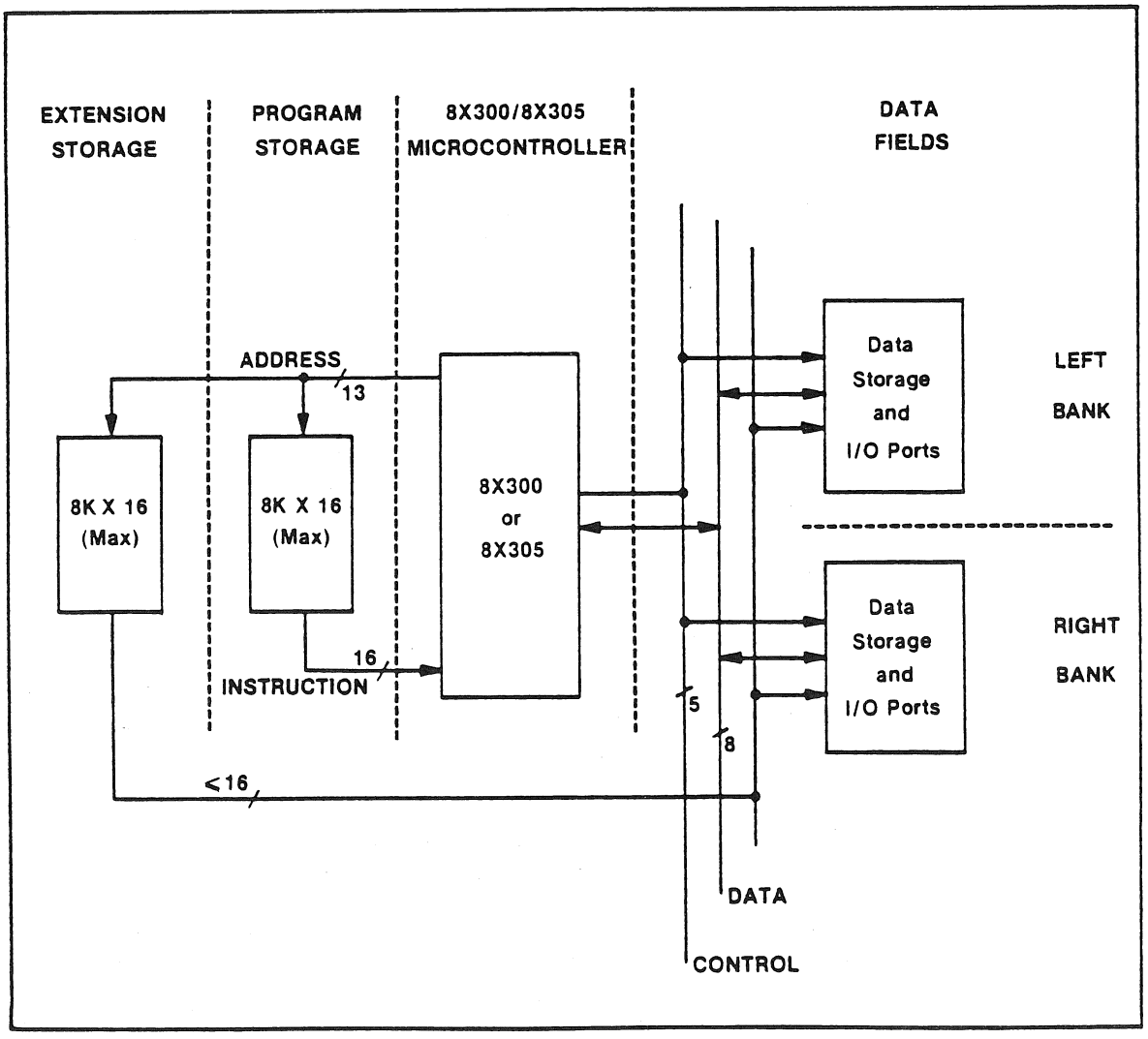


Figure 4-1. Typical Extended 8X300/8X305 Configuration

The DEF directive allows the user to specify as many as sixteen additional fields with a total extension size of sixteen bits. During assembly these fields may be filled with any expression allowable by the assembler. The result will be an object module containing bit patterns which may then be placed into a PROM.

The sample DEF directive shown in Example 4-1 specifies three additional fields. The first is four bits in length and the remaining two are three bits each. Note that the second field has a default value of 2. Because there are ten bits defined, the actual object module will contain twelve bits since only 4- and 8-bit object modules may be produced. The two right-most bits will be set to zero. Also the first field will be truncated to four bits.

Example 4-1. Use of Instruction Extensions

```
        DEF     -4,3(2),3
IVI     LIV     35
        MOVE    AUX,RI/IVI,1,7
          .
          .
          .
        HALT    /2,4,4
```

The instruction extension of the MOVE command will consist of a 4-bit field containing the lower four bits of the address of IVI, a 3-bit field with the value 1, and another 3-bit field with a value of 7. The actual bit pattern would be $33C_{16}$ (0011001111_2 plus the two low-order zeros). If the user had specified

```
MOVE AUX,RI/IVI,,7
```

the extended bit pattern would be $35C_{16}$ (0011010111_2). In this case the second field which was not specified in the source statement assumes the value of 2 which was specified as the default in the DEF directive.

The statement

```
MOVE AUX,RI/IVI
```

would use the default value of 2 for the second field and the default value of 0 for the third field, giving a bit pattern of 340_{16} (0011010000_2) for the extension.

To define instruction extensions during assembly, the user merely places a slash after the standard 8X300/8X305 instruction and specifies the values to be placed into the fields as specified in the DEF directive. For those instructions that do not contain an operand field, the instruction extension will follow the operator directly (with the intervening slash, of course). The user need not specify each field of an extension or even specify any fields. Source

statements without an explicit extension field or with only the slash following the 8X300/8X305 instruction will generate an extension that consists of only the default values specified in the DEF statement.

Only standard 8X300/8X305 instructions that generate code (executable statements) may have instruction extensions attached. They may not be specified for directives. An exception to this is the END statement. An instruction extension specified on the END statement for the program will be used as the extension for any instructions in the return jump table.

SECTION 5

ASSEMBLER DECLARATIONS

Declaration statements are used to assign values or addresses to symbols. References to the symbol so declared use the assigned values or addresses as required by the context in which the symbol is used. Assembler declarations do not generate any object code.

The declarations are EQU, SET, LIV, and RIV.

5.1 EQU — DEFINE A CONSTANT

The EQU statement assigns a value to the symbol in the label field, which may subsequently be used in the operand field of any other statement.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	EQU	expression	none	statement

Where:

"symbol". . . . is any valid symbol not previously defined as local to this program segment or global to the entire program.

"expression" . . is any valid expression which uses only pre-defined symbols.

Example 5-1. Use of EQU Statements

```
DEC     EQU     -1
SINMSK  EQU     10000000B
OEMASK  EQU     1B
LSMASK  EQU     7H
SSMASK  EQU     LSMASK.L.3
MSMASK  EQU     LSMASK.R.1.L.6
ROT     EQU     3
```

(Taken from lines 16 through 22 of Appendix B.)

5.2 SET — DEFINE OR REDEFINE A CONSTANT

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. Any attempt to redefine a symbol defined by the SET statement in any manner **other than by another SET statement** will result in an error indication.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	SET	expression	none	statement

Where:

"symbol". . . . is any valid symbol not previously defined as local to this program segment or global to the entire program.

"expression" . . is any valid expression which uses only pre-defined symbols.

Example 5-2. Use of SET Statements

```
VAL1    SET     0
VAL2    SET     1
VAL3    SET     2
VAL4    SET     3
VAL1    SET     VAL1+5
VAL2    SET     VAL2+5
VAL3    SET     VAL3+5
VAL4    SET     VAL4+5
```

(Taken from lines 24 through 27 and 217 through 220 of Appendix B.)

5.3 LIV — DEFINE A LEFT BANK DATA FIELD VARIABLE

The LIV declaration assigns a symbolic name to a **left bank** data field and defines the address, position, and precision (length) of that variable.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	LIV	byte,bit,length	none	statement

Where:

"symbol". . . . is any valid symbol not previously defined as local to this program segment or global to the entire program.

"byte", "bit", and "length" . . are constants, symbols or expressions. Any symbols used in an expression must be previously defined. "byte" represents the address, and must evaluate to less than 256; "bit" represents the least significant bit of the variable, and must evaluate to less than 8; and "length" represents the number of bits in the variable, and must evaluate to less than or equal to 8. Values greater than these will result in an error indication. It is also required that "length" be less than or equal to "bit"+1.

If "length" is not specified, it has a default value of 1. If "bit" is not specified, "bit" has a default value of 7 and "length" has a default value of 1. For example: INI LIV 10 is the same as INI LIV 10,7,1. The use of an expression for "byte" allows data field variables to be defined relative to each other, for example, if A LIV 10,7,8, then B LIV A+1,7,8 is equivalent to B LIV 11,7,8.

When "symbol" is used in a subsequent statement as a source or destination address, the appropriate information "bit" and "length" are used. However, when "symbol" is used as part of the expression, **it has only the value given by "byte".**

Example 5-3. Use of LIV Statements

```
DISC1    LIV      11H,7,8
DSTAT    LIV      DISC1,0
DSCLOK   LIV      DISC1,5
DRDWR    LIV      DISC1,6
DRDAT    LIV      DISC1
```

(Taken from lines 29 through 33 of Appendix B.)

5.4 RIV — DEFINE A RIGHT BANK DATA FIELD VARIABLE

The RIV declaration assigns symbolic names to **right bank** data field variables, but is otherwise identical to the LIV declaration.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	RIV	byte,bit,length	none	statement

Where:
"symbol", "byte", "bit" and "length", have identical meaning to those used in the LIV declaration.

Example 5-4. Use of RIV Statements

```
DATA1    RIV      100H,7,8
D1SIGN   RIV      DATA1,0
D1ODEV   RIV      DATA1
DATA2    RIV      DATA1+1,7,8
D2SIGN   RIV      DATA2,0
D2ODEV   RIV      DATA2
```

(Taken from lines 36 through 41 of Appendix B.)

SECTION 6

ASSEMBLER DIRECTIVES

An assembler directive is a statement that is not translated into object code, but rather is interpreted as a command to the assembler program to perform some action during the assembly process. By using directives, the programmer may divide the program into logical segments, format the output listing, or specify the format of the object module. The directives are:

PROG	ORG	LIST	IF
PROC	OBJ	NLIST	ENDIF
ENTRY	DEF	EJCT	8X300
END	PROM	SPAC	8X305

6.1 PROG — PROGRAM TITLE STATEMENT

The PROG statement introduces and names the main program. With the exception of listing control directives and comments, it **must be the first statement** of a program and may appear only once.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	PROG	name	none	statement

Where:

"name". . . . . is any valid symbol. It must not appear in any other assembler statement except the main program END statement.

Example 6-1. Use of PROG Statement

```
PROG        SAMPLE
```

(Taken from line 7 of Appendix B.)

6.2 PROC — PROCEDURE TITLE STATEMENT

The PROC directive begins and names a procedure. A PROC directive may only appear after another procedure has been terminated, or after the last executable or declaration statement of the main program segment. The main program segment is considered to be ended upon the occurrence of the first PROC directive.

Since other segments may call this procedure name, it is a global name known to the entire program. Use of the PROC name in the operand field of a procedure CALL statement calls the procedure into execution.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	PROC	name	none	statement

Where:
"name". . . . . is any valid symbol which **must not** be used anywhere else in the program, except as the operand of the procedure CALL and END statements.

Example 6-2. Use of PROC Statements

```
PROC    ARITH
PROC    EXECT
```

(Taken from lines 98 and 163 of Appendix B.)

6.3 ENTRY — SECONDARY ENTRY POINT INTO A PROCEDURE

The ENTRY directive specifies an additional entry point to a procedure. Calls to the procedure by additional names cause execution to start at the first executable statement following the ENTRY directive which defined that additional name. A procedure may contain more than one additional entry point.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	ENTRY	name	none	statement

Where:
"name". . . . . is any valid symbol that **must not** be used anywhere else in the program, except as the operand of the procedure CALL statements.

Example 6-3. Use of ENTRY Statements

```
ENTRY   MOVMNT
ENTRY   TRNSMT
```

(Taken from lines 132 and 143 of Appendix B.)

6.4 END — END THE PROGRAM OR A PROCEDURE

The END directive is required to terminate a procedure or the complete program. If an extension field is added to the END statement of the main program, the extension code will be added to the return jump table that is generated at the end of the program.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	END	name	/code	statement

Where:

"name" . . . . is the same name as was used in the title statement for this program segment (PROG or PROC).

"code" . . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 6-4. Use of END Statements

```
END         NONZXF
END         SAMPLE
```

(Taken from lines 237 and 239 of Appendix B.)

```
END         MAIN/0,0,0,0
```

(Taken from line 284 of Appendix D.)

CAUTION

If "name" is not used in the PROC or PROG statement, or no name appears, **an error will be indicated** and the program terminated.

6.5 ORG — SET LOCATION COUNTER

The ORG directive changes the value of the location counter either conditionally or unconditionally. The first form of the ORG directive **unconditionally** changes the value of the location counter to the value indicated by "address".

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	ORG	address	none	statement

Where:

"address" . . . is any constant, valid symbol, or valid expression which evaluates to a value between 0 and 8191. If the value is outside this range, an error is indicated and the location counter is not changed.

Example 6-5. Use of the Unconditional ORG Statement

```
        ORG 0
```

(Taken from line 80 of Appendix B.)

The second form of the ORG directive **conditionally** sets the location counter to the next page or segment boundary if there are insufficient locations in the current page or segment. This is determined by evaluating the operands "space" and "page size".

A conditional ORG may be necessary when using NZT or XEC instructions. If adding the value "space-1" to the location counter would move the location counter into the next page or segment, then the location counter will be set to the beginning of the next page. If the location counter would not move into the next page, then this statement will have no effect on the location counter.

If the location counter is moved to the next page, a jump instruction to that address is inserted in the program at the point where the ORG statement appeared. This added instruction assures the sequential flow of the program.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	ORG	space,page size	none	statement

Where:

"space" . . . . is any constant, valid symbol or valid expression. "space" specifies the number of program locations which **must remain** in this page or segment.

"page size" . . is a constant or valid symbol, or valid expression that evaluates to 256 or 32 (the page and segment sizes of control storage). If "space" is equal to "page size", this statement is an unconditional alignment to the next boundary of length "page size".

Example 6-6. Use of Conditional ORG Statements

```
ORG     256,256
ORG     5,32
ORG     7,32
ORG     16,256
```

(Taken from lines 100, 184, 189, and 221 of Appendix B. Reference also the results of assembling these lines in Appendix C.)

CAUTION

It is the **programmer's responsibility** to avoid setting the location counter to an address which already contains a previously assembled instruction, since **no error is indicated** if this is done.

6.6 OBJ — SPECIFY AN OBJECT FORMAT

The OBJ directive is used to specify to the assembler the format of the object module for both standard 8X300/8X305 instructions and for any instruction extensions. In addition, this directive allows the user to fill any unused addresses in the program.

The output format for an object module may be specified as blocked or unblocked. A **blocked format** implies that when the object module is produced, all words, including unused locations, will be output. The size of the block is specified by the PROM directive. A gap in the program due to an ORG directive will cause the object module to be output if the address is moved beyond the range of the block. If the program is smaller than the PROM size, a complete block will be output.

An **unblocked format** will produce an object module only when the module size specified in the PROM directive has been satisfied, if a program gap occurs, and/or when the program ends.

> **NOTE**
>
> Object modules produced in the MCSIM format are always unblocked even if the user specifies otherwise. If the ASCII-Hex word format is specified for either the 8X300/-8X305 instructions or the instruction extensions, it will be used for both modules.

A complete description of each format is given in Section 9.3. In the absence of a given specifications, there are three independent defaults:

1. 8X300/8X305 instructions are output in MCSIM format,
2. extensions are output in ASCII-Hex (Space) format, and
3. output is blocked and filled in with NOP's.

If MCSIM format is the case, the output will be unblocked.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	OBJ	format,type	/format,type	statement

Where:

"format" . . . . is the object module format required. This may be specified as any of the following characters:
- M for MCSIM format,
- N for BNPF format,
- R for ASCII-Hex (Quote) format,
- D for ASCII-Hex (Space) format,
- S for ASCII-Hex Word format, or
- Z to suppress output of the object module.

"type" . . . . . is optional and determines the type of blocking for the object module. One of the following characters can be specified:
- H for block format filled out with HALT's,
- B for blocked format filled out with NOP's (all zeros),
- O for blocked format filled out with all one's, or
- U for unblocked format.

Example 6-7. Use of OBJ Statements

```
        OBJ         M
```

(Taken from line 57 of Appendix B.)

```
        OBJ         R,H/R
```

(Taken from line 12 of Appendix D.)

6.7 IF, ENDIF — CONDITIONAL ASSEMBLY

The conditional assembly statement, IF, allows the programmer to control whether or not certain source statements are assembled. When an IF statement is encountered, the associated 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. If false, the source statements following the IF are **not processed** until an ENDIF is encountered, at which point normal processing resumes.

Conditional assembly constructs may be nested but may not be overlapped; that is, the end of an inner IF construct must be encountered before the end of the outer IF construct is encountered.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	IF	expression	none	statement

Where:

"expression" . . is a constant, a valid symbol, or a valid expression. Any symbols used in "expression" must be previously defined.

The ENDIF directive terminates the source statements subject to conditional assembly. In the case of the nested IF statements, ENDIF is paired with the most recent IF statement.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	ENDIF	none	none	statement

Example 6-8. Use of IF and ENDIF Statements

```
        IF          FINAL
        LIST        I,M,S,O
        OBJ         M
        ENDIF
```

(Taken from lines 55 through 58 of Appendix B.)

6.8 LIST — LIST THE SPECIFIED ELEMENTS

The LIST directive causes files to be generated for listing or punching according to the options specified.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	LIST	options	none	statement

Where:

"options". . . . indicate the output required. The following characters specify the necessary combinations.

- S for listing source statements (not including macro expansions and unassembled conditionals).
- O for producing output object code.
- M for listing statements generated by macro calls.
- I for listing statements which would not be assembled due to conditional assembly (IF).
- T for listing symbol table.
- X for listing cross reference table.
- A For printing the addresses and object code in absolute hexadecimal numbers instead of MCSIM format. (Four digits for address and four digits for object code.)

The default options are S, O, and T. If both X and T options are specified, the X option will override.

Example 6-9. Use of the LIST Statement

```
        LIST        I,M,S,O
(Taken from line 56 of Appendix .)
```

6.9 NLIST — SUPPRESS LISTING OF ELEMENTS

The NLIST directive is the same as the LIST directive, except the specified options **are not produced** for listing or punching. This directive is not printed on the listing if S is an option.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	NLIST	options	none	statement

Where:

"options". . . . indicate the output to be suppressed. The following characters are used for specifying the necessary combinations.

- S for not listing source statements (not including macro expansions and unassembled conditionals).
- O for not producing output object code.
- M for not listing statements generated by macro calls.
- I for not listing statements which would not be assembled due to conditional assembly (IF).
- T for not listing symbol table.
- X for not listing cross reference table. (This option overrides the T option if both are specified.)
- A For printing the addresses and object code in MCSIM format instead of absolute hexadecimal.

Example 6-10. Use of the NLIST Statement

```
NLIST       S,O
```

(Taken from line 52 of Appendix B.)

NOTE

Assembly lines with errors are **always listed**, regardless of the options specified by an NLIST. Also, an NLIST of the source (S) overrides any LIST of macros or unassembled conditionals (M and I), but only until a LIST S is executed.

6.10 EJCT — EJECT THE LISTING PAGE

EJCT is a listing control directive which causes the output listing to be advanced to the next page, thus making it possible to format the assembly listing. For example, each procedure could start on a new page of the assembly listing. EJCT is **not printed** on the assembly listing.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	EJCT	none	none	statement

6.11 SPAC — LINE FEED THE LISTING

SPAC is a listing control directive which inserts blank lines in the assembly listing. The SPAC statement is not printed on the assembly listing.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	SPAC	expression	none	statement

Where:

"expression" . . is a constant, a valid symbol, or a valid expression (constants typically are used). "expression" is evaluated to determine the number of blank lines to insert in the listing. There is **no default** "expression".

Example 6-11. Use of SPAC Statements

```
SPAC        1
SPAC        12
```

(Taken from lines 6 and 238 of Appendix B.)

6.12 PROM — SPECIFY PROM SIZE

The PROM directive is used to specify the widths and depth of the PROMs used for the assembled object code. Only those PROMs specified by this directive will be included in the object module. Thus if 8 bits are specified for a PROM and the DEF directive defines an extension to contain 16 bits, only 8 bits will be included in the object module.

This directive must appear prior to any executable statements.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	PROM	d,w,w...	/d,w,w...	statement

Where:

"d" . . . . . specifies the depth of the PROM used and hence the size of the object module format that will be used. "d" may be specified as 128, 256, 512, 1024, 2048. Any other value will generate an error indication. The depth of extension PROMs may differ from that the 8X300/8X305 instruction PROMs.

"w" . . . . . specifies the width of the PROM and hence the size of the object module format that will be used. A width should be specified for each PROM used and may be specified as either 4 bits or 8 bits. Note that some object module formats, e.g. MCSIM, will always work with a 16-bit value regardless of the width specified in this directive. The total width of all PROMs used for 8X300/8X305 instructions must be exactly 16 bits. The width of all PROMs used for the instruction extensions may be any value.

NOTE

The default PROM sizes are a depth of 512 and a width of 8. The user need only specify the extended instruction PROM sizes if desired.

Example 6-12. Use of the PROM Statement

```
PROM        128,8,8/256,4,8,4
```

(Taken from line 11 of Appendix D.)

6.13 DEF — DEFINE INSTRUCTION EXTENSION FIELDS

This directive is used to specify operand fields and default values for instruction extensions. The fields define output module bit positions in order from left to right (bit 0 to 15). This directive may define up to 16 fields with a total length of 16 bits. The length in bits of each field is specified along with an optional default field value and an error checking flag.

No extensions can be generated unless the format has been specified by this directive. This directive must appear before any executable instructions.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	DEF	l(v),l(v),...	none	statement

Where:

"l" . . . . . specifies the number of bits in the field. Any values placed in this field during assembly will be checked to ensure that it fits into the number of bits specified. If not, the value will be truncated and an error indication output. If "l" is preceded by a minus sign, **error checking will not take place.** This is useful when a field will contain the low-order bits of a program address.

"v" . . . . . specifies the default value for the field. The default value must fit within the number of bits specified or an error will be indicated. If no default value is specified, it is assumed to be zero.

Example 6-13. Use of the DEF Statement

```
DEF         4(5),-8(2),2,2
```

(Taken from line 10 of Appendix D.)

6.14 8X300/8X305 MICROCONTROLLER SPECIFIERS

The 8X300 directive specifies assembly of 8X300 instructions. The 8X305 directive specifies assembly of 8X305 instructions. (Use of XML, XMR or R12-R16 cause error diagnostics if 8X305 is not specified.)

NOTE

If neither is specified, 8X300 will be the default option.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	8X300	none	none	statement

Example 6-14. Use of MicroController Specifiers

```
8X300
8X305
```

(Taken from lines 246 and 243 of Appendix D.)

SECTION 7

EXECUTABLE STATEMENTS

The statements described in this section result in object code that is executable by the 8X300/8X305 MicroController. There are fifteen:

MOVE	NZT	XMIT	CALL
ADD	XEC	HALT	RTN
AND	JMP	NOP	SEL
XOR	XML	XMR	

It is not intended in this section to describe the operation or execution of the 8X300/8X305 machine codes, but rather to describe the MCCAP formats. Machine code information is contained in Appendix D for reference; however, more specific information about operation and execution may be obtained from the available 8X300/8X305 and other peripheral technical and applications literature.

NOTE

There are certain notes generally applicable to each of these statements. They are listed here and subsequently referenced whenever relevant.

1. If a source "s" or destination "d" field is specified by a constant, the value of that constant is evaluated as follows:

   a. Registers are designated by values less than 17_8.
   b. Left bank data fields are designated by values between 20_8 and 27_8.
   c. Right bank data fields are designated by values between 30_8 and 37_8.

   If the value is greater than 37_8, an error is indicated and the value is treated as modulo 40_8.

2. If the value of the expression in the operand field is too large to fit in the 8-bit immediate field (in the case of a register) or 5-bit immediate field (in the case of an I/O data field) of the object code, an error is indicated and the value is truncated (high-order bits dropped) to fit into the appropriate field length.

3. If the high-order five bits (in the case of a register) or eight bits (in the case of an I/O data field) of an indexed value (expression+index) are not equal to the corresponding bits of the location counter, a paging error is indicated.

4. If an optional value within a field is omitted, the associated punctuation must also be eliminated to prevent errors. For example, if a length or size is omitted, the comma preceding it must also be omitted from the statement.

7.1 MOVE, ADD, AND, XOR — DATA MANIPULATION

The MOVE, ADD, AND, and XOR symbolic codes may be written in any of three formats, as required.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	op	s,d	/code	statement

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	op	s(r),d	/code	statement

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	op	s,len,d	/code	statement

Where:

"symbol". . . . is any valid symbol that is not defined as local to the current program segment or global to the entire program.

"op". . . . . is one of the four data manipulation commands: MOVE, ADD, AND, XOR.

"s" . . . . . is an I/O data field variable or any of the 8X300/8X305 internal registers that may be used as **source operands.** "s" may be a symbol predefined by a declaration or it may be a constant. (See note 1.)

"d" . . . . . is an I/O data field variable or any of the 8X300/8X305 internal registers that may be used as **destination operands.** "d" may be a symbol predefined by a declaration or it may be a constant. (See note 1.)

"len" . . . . . is an **optional** value that specifies the explicit length of an I/O data field. This may be used to override the "length" of a LIV or RIV declaration. More typically, it is used when no LIV or RIV declaration is made and the source or destination operand is given by a constant. If "len" is greater than 8, an error is indicated and the value is taken as modulo 8. A "len" of 8 generates a value of 0. The default value of "len" is 0 (full byte). (See note 4.)

"r" . . . . . is an **optional** value that specifies the number of bit positions to right rotate the source register **when both source and destination operands are registers.** The default value for "r" is 0.

"code". . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-1. Data Manipulation Commands

```
STMV    MOVE    DSTAT,R1
        MOVE    24H,LEN,R2
        MOVE    DRDAT,LEN,R3
```

(Taken from lines 134 through 136 of Appendix B.)

```
STAD    ADD     R1,R1
        ADD     2,2
        ADD     R3(ROT),R3
```

(Taken from lines 107 through 109 of Appendix B.)

```
STAND   AND     R1,DATA1
        AND     R2,LEN,DATA2
        AND     R3,LEN,37H
        AND     4(4),AUX
```

(Taken from lines 112, 115, 118, and 121 of Appendix B.)

```
STOR    XOR     DATA1,DATA1
        XOR     DATA2,3,DATA2
        XOR     37H,LEN,37H
        XOR     33H,LEN,37H
```

(Taken from lines 124, 126, 128, and 130 of Appendix B.)

```
        MOVE    AUX,AUX/7
        MOVE    R1,R11/1,2,,3
        MOVE    R1,R5/1111B,-1,2
        MOVE    R2,R1/1V3,77H
```

(Taken from lines 40, 43, 47 and 51 of Appendix D.)

CAUTION

Addressing the I/O data fields which are allocated to different locations in the **same** I/O bank is **not valid.** Any attempt to do so will be detected by the assembler and indicated as an error. Also, data movement between data fields allocated to different I/O banks must move full bytes or an error is indicated.

7.2 XMIT — LOAD IMMEDIATE

The XMIT instruction may be written in either of two formats, as follows:

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	XMIT	exp8,reg	/code	statement

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	XMIT	exp5,df,len	/code	statement

Where:

"symbol". . . . is any valid symbol that is **not** defined as local to the current program segment or global to the entire program.

"exp8". . . . . is any valid epxression. "exp8" is evaluated and used as the 8-bit immediate field of the object code. (See note 2.)

"exp5". . . . . is any valid expression. "exp5" is evaluated and used as the 5-bit immediate field of the object code.

"reg" . . . . . is any of the 8X300/8X305 internal registers usable as a **destination operand.** This may be a symbol or a constant. (See note 1.)

"df" . . . . . is an I/O data field variable used as the **destination operand.** This may be a symbol defined by a LIV or RIV declaration, or it may be a constant. (See note 1.)

"len" . . . . . is an **optional** value that specifies the explicit length of an I/O data field. This may be used to override the "length" operand of a LIV or RIV declaration. More typically, it is used when the destination I/O data field variable is written as a constant. The default value of "len" is 0. (See note 4.)

"code". . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-2. Use of the XMIT Statement

```
XMIT    '!',R5
XMIT    VAL1,DISP1,LEN
XMIT    VAL2,23H,4
```

(Taken from lines 151, 153, and 154 of Appendix B.)

```
XMIT    2,R0/1,*,0
```

(Taken from line 148 of Appendix D.)

7.3 XEC — EXECUTE

The XEC instruction may be written in either of two formats, as required.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	XEC	exp8(reg),size	/code	statement

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	XEC	exp5(df,len),size	/code	statement

Where:

"symbol" . . . . is any valid symbol that is not defined as local to the current program segment or global to the entire program.

"exp8" . . . . . is any valid expression. "exp8" is evaluated, and is placed in the 8-bit immediate field of the object code. (See note 3.)

"exp5" . . . . . is any valid expression. "exp5" is evaluated, and is placed in the 5-bit immediate field of the object code. (See note 3.)

"reg" . . . . . is any of the 8X300/8X305 internal registers usable as a **source operand**. "reg" is used as an index to "exp8" and may be represented symbolically or by a constant. (See note 1.)

"df" . . . . . represents an I/O data field variable used as an index to "exp5". "df" may be a symbol defined by a LIV or RIV declaration, or it may be a constant. (See note 1.)

"len" . . . . . is an **optional** value that specifies the length of an I/O data field. This may be used to override the "length" operand of a LIV or RIV declaration. More typically it is used when the "df" operand is written as a constant. The default value of "len" is 0. (See note 4.)

"size" . . . . . is the **optional** table length size if the XEC is used with a jump table. The assembler checks to ensure that an XEC and its associated jump table are on the same program storage page. "size" has a default value of 1. The user specifying an XEC preceeding his jump table can obtain error checking on the table size by specifying this operand. (See note 4.)

"code". . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-3. Use of the XEC Statement

```
STXC    XEC     *+1(R6),5
TAB1    XEC     *+1(R1)
        XEC     *+1(T2PTR)
        XEC     *+1(T3PTR,2)
        XEC     *+1(T4PTR),2
        XEC     *+1(T5PTR,2),4
```

(Taken from lines 171, 178, 185, 190, 197 and 202 of Appendix B.)

```
        XEC     *+1(IV1,3)/,LI2,0
```

(Taken from line 76 of Appendix D.)

7.4 NZT — NON-ZERO TRANSFER

The NZT symbolic code may be written in either of two formats, as follows:

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	NZT	reg,exp8	/code	statement

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	NZT	df,len,exp5	/code	statement

Where:

"symbol". . . . is any valid symbol that is not defined as local to the current program segment or global to the entire program.

"reg" . . . . . is any of the 8X300/8X305 internal registers usable as a **source operand.** "reg" may be a symbol or a constant. (See note 1.)

"df". . . . . is an I/O data field variable used as a **source operand.** "df" may be a symbol defined by a LIV or RIV declaration, or it may be a constant. (See note 1.)

"exp8". . . . . is any valid expression. "exp8" is evaluated and used as the low-order 8-bit immediate field of the object code. (See note 2.)

"exp5". . . . . is any valid expression. "exp5" is evaluated and used as the low-order 5-bit immediate field of the object code. (See note 2.)

"len" . . . . . is an **optional** value that specifies the length of an I/O data field. This may be used to override the "length" operand of a LIV or RIV declaration. More typically, it is used when the I/O data field variable is written as a constant. The default value of "len" is 0. (See note 4.)

"code". . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-4. Use of the NZT Statement

```
STNT    NZT     R5,*+8
        NZT     DISP1,*+7
        NZT     23H,4,*+6
```

(Taken from lines 223, 226, and 229 of Appendix B.)

```
        NZT     IV1,3,*/0,IV1,1,0
```

(Taken from line 110 of Appendix D.)

7.5 JMP — UNCONDITIONAL JUMP

The JMP symbolic code is written in the following format:

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	JMP	address	/code	statement

Where:

"symbol". . . . is any valid symbol that is not defined as local to the current program segment or global to the entire program.

"address" . . . is any valid address in the range 0 to 8191. If "address" is outside this range, an error is indicated and the address field of the object code will be set to the **current value** of the location counter.

"code". . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-5. Use of the JMP Statement

```
        JMP     TAB1
        JMP     DONE
```

(Taken from lines 172 and 179 of Appendix B.)

```
        JMP     8191/0,0,0,0
```

(Taken from line 85 of Appendix D.)

7.6 SEL — I/O DATA FIELD SELECTION

The SEL statement generates code which, upon execution, places the address of an I/O data field into the IVL and IVR register, as appropriate. The generated object code is equivalent to one of the following executable statements.

```
        XMIT    "df", IVL
or      XMIT    "df", IVR
```

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	SEL	df	/code	statement

Where:

"symbol" . . . . is any symbol that is not defined as local to the current program segment or global to the entire program.

"df" . . . . . is a symbol that has been defined by a LIV or RIV declaration. If "df" is not so defined, an error is indicated.

"code" . . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-6. Use of SEL Statements

```
STAR    SEL     TEMP1
TAB2    SEL     TABPTRS
```

(Taken from lines 101 and 183 of Appendix B.)

```
        SEL     LI2/17H,0,3,3
```

(Taken from line 87 of Appendix D.)

NOTE

In an executable statement, if an I/O data field is referenced, the address of that port must have been already placed in the IVL or IVR select register. This can be accomplished by a SEL or a XMIT. Since the assembler **cannot detect** whether or not a data field has been selected at the time it is addressed, **it is the responsibility of the programmer** to select I/O data fields before they are referenced.

7.7 CALL — PROCEDURE (SUBROUTINE) CALL

The 8X300/8X305 MicroController does not have a provision for storing the program counter before jumping to a subroutine. However, an equivalent technique is used by MCCAP to permit the use of subroutines (or procedures as they are called). Each CALL statement generates a return jump index which is loaded into register R11, then control is transferred to the subroutine. When execution reaches a RTN (return) statement, control passes to a "return jump table" which uses the value in R11 as an index to jump back to the calling program. The entries in the return jump table, which match the CALL statements, are provided automatically by MCCAP. The programmer needs only to call the subroutine and return as he would with any subroutine arrangement. Of course if he wishes to use R11 within the subroutine, he must restore it before returning. For techniques in nesting subroutines, see the 8X300/8X305 Programming Manual. For subroutine call macros using the 8X310, see Section 8.3.

Example 7-7 illustrates the expansion of the source statements and the position of the return jump table.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	CALL	name	/code	statement

Where:

"symbol". . . . is any valid symbol that is not defined as local to the current program segment or global to the entire program.

"name" . . . . is a procedure name defined by a PROC or an ENTRY statement.

"code". . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-7. Equivalent Code for Procedure CALL's and RTN's

	SOURCE STATEMENTS		EQUIVALENT CODE
	PROG MAIN		
	.		.
	.		.
	.		.
	CALL SUB		XMIT 0,R11
ONE	.		JMP SUB
	.		.
	.		.
	CALL SUB		XMIT 1,R11
TWO	.		JMP SUB
	.		.
	.		.
	.		.
	.		.
	PROC SUB		.
	.		.
	.		.
	.		.
	RTN		JMP TABL
	.		.
	.		.
	.		.
	END MAIN		.
		TABL	XEC *+1(R11)
			JMP ONE
			JMP TWO

For actual usage of procedures, see the programs listed in the Appendices.

A program is limited to a maximum of 255 CALL statements; more will result in a "Table Overflow" error indication.

Example 7-8. Use of the CALL Statement

```
STC     CALL    ARITH
        CALL    MOVMNT
        CALL    TRNSMT
        CALL    EXECT
```

(Taken from lines 85 through 88 of Appendix B.)

```
        CALL    PROC3/0,0
```

(Taken from line 126 of Appendix D.)

CAUTION

The programmer is responsible for saving and restoring the value of R11, if it is used within a procedure. **No error** will be indicated from a failure to do so.

7.8 RTN — PROCEDURE RETURN

The RTN statement terminates the **execution** of a procedure and causes a return of control to the calling program segment. The assembler generates a return jump table at the end of the entire program to allow for this return of control. A RTN statement causes a jump to the return jump table, which in turn contains a XEC (with an index value in R11) followed by one jump instruction for each CALL statement in the program. RTN statements are **not valid** in the main program, but **at least one** must appear in each procedure. Example 7-7 shows the equivalence between the source code and the code produced by the assembler for RTN and the return jump table.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	RTN	none	/code	statement

Where:

"symbol". . . . is any valid symbol that is not defined as local to this program segment or global to the entire program.

"code". . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-9. Use of the RTN Statement

```
DONE        RTN
```

(Taken from line 208 of Appendix B.)

```
            RTN/,,3
```

(Taken from line 277 of Appendix D.)

```
    RETURN TABLE
        0173  8974
        0174  E008
        0175  E00A
        0176  E00C
        0177  E00E
        0178  E104
```

(Taken from page 8 of Appendix C.)

CAUTION

The programmer is responsible for saving and restoring the value of R11, if it is used within a procedure. **No error** will be indicated from a failure to do so.

7.9 NOP — NO OPERATION

The NOP instruction generates code which commands the 8X300/8X305 to advance to the next instruction without performing any other operation. It typically serves as a time delay. The NOP actually generates a "MOVE AUX,AUX" instruction.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	NOP	none	/code	statement

Where:

"symbol". . . . is any valid symbol that is not defined as local to the current program segment or global to the entire program.

"code". . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-10. Use of the NOP Statement

```
START   NOP
```

(Taken from line 81 of Appendix B.)

```
        NOP/0,WS2
```

(Taken from line 38 of Appendix D.)

7.10 HALT — STOP PROCESSING

The HALT instruction generates code which causes the 8X300/8X305 to stop processing and remain at the current address. The HALT instruction actually generates a "JMP *" instruction. The RESET signal of the 8X300/8X305 must be pulsed to restart the program after the execution of a HALT instruction.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	HALT	none	/code	statement

Where:

"symbol". . . . is any valid symbol that is not defined as local to the current program segment or global to the entire program.

"code". . . . . is an **optional** series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-11. Use of the HALT Statement

```
LAST    HALT
```

(Taken from line 91 of Appendix B.)

```
STOP    HALT/17H,377H,3,3
```

(Taken from line 249 of Appendix D.)

7.11 XML, XMR - LOAD IMMEDIATE TO LEFT OR RIGHT BANK

NOTE

XML or XMR cause opcode error diagnostics if 8X305 is not specified. These statements are only valid for use with the 8X305.

The XML statement generates code which, upon execution, transmits an 8-bit constant to the left bank. (Right bank for XMR.) The generated object code is equivalent to a XMIT immediate to R12 or R13 respectively.

```
XML     =     XMIT     IMMED, R12
XMR     =     XMIT     IMMED, R13
```

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	XML,XMR	immed	/code	statement

Where:

"symbol" . . . . is any valid symbol that is not defined as local to the current program segment or global to the entire program.

"immed" . . . . is an 8-bit constant.

"code" . . . . is an optional series of symbols, constants, or expressions specifying the bit patterns to be generated for placement into the extension field.

Example 7-12. Use of the XML Statement

```
XML     2
XMR     0FX
```

(Taken from lines 244 and 245 of Appendix D.)

MACROS

A macro is a predefined sequence of source statements that can be inserted into a MCCAP program by coding only one statement, the "macro call". The predefined sequence is written in the same program and is called a "macro definition". The macro definition statements must be written prior to a call to the macro. When a macro call is encountered during assembly, the assembler locates the saved macro definition statements, copies them into the source program immediately after the macro call statement, and then assembles them normally. Each macro, once defined, may be called any number of times within a program.

The number of macros that can be defined (initially 500) may be modified during the installation of MCCAP. (Refer to MCCAP Installation and Maintenance Manual and/or the 8X300/8X305 Cross Assembler Installation Guide for 8080 based systems for details.)

Upon encountering a macro definition statement, the assembler saves the body of the definition **as it is.** The statements are not assembled, nor are they checked for errors. When the macro is called, the assembler obtains the saved definition body, places it in-line with the source code immediately following the macro call, and substitutues the actual parameters for the formal parameter symbols. At this point, MCCAP assembles these statements as if they had originally been coded in that position. Therefore, all rules for statements, expressions, and symbols are enforced **only at the point of expansion,** not at the point of definition.

Example 8-1. Macro Usage

The following demonstrates the statements used to implement macros within a program.

Macro definition as it would appear in the source code:

```
INPUT   MACRO   R,S,T,LAB        (Title Statement)
        MOVE    R,S
        XMIT    1,T
        ADD     R,S
LAB     RIV     22,3,1
        MOVE    LAB,S
        ENDM                     (Terminator)
```

Macro call as it would appear in the source code:

```
        LIST    M
        XMIT    -1,R3
        MOVE    OVF,AUX
LOOP    INPUT   R1,R2,IV1,IV2    (Macro Call)
        JMP     GO
```

Expansion of the macro call as it would appear in the assembly listing:

```
          LIST    M
          XMIT    -1,R3
          MOVE    OVF,AUX
 LOOP     INPUT   R1,R2,IV1,IV2      (Macro Call)
+         MOVE    R1,R2
+         XMIT    1,IV1
+         ADD     R1,R2
+IV2      RIV     22,3,1
+         MOVE    IV2,R2
          JMP     GO
```

NOTE

The LIST M directive is necessary to produce the expansion of the macro in the listing.

8.1 THE MACRO DEFINITION

The macro definition consists of a title statement, a body of assembly statements, and a terminator statement, as shown in Example 8-1.

8.1.1 The Title Statement

The title statement marks the beginning of a macro definition, it names the macro and provides a list of format parameters to be passed to the macro.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
name	MACRO	p1,p2,...pn	none	statement

Where:

"name" . . . . is a valid symbol that defines the name of the macro. This "name" must not be used as a symbol anywhere else in the program except in the **operation** fields of macro call statements.

"p1,p2,...pn" . . is an **optional** formal parameter list. These formal parameters can be any valid symbols. The number of formal parameters per definition is a factor of the number of characters per source code line. If no formal parameters are included in the title statement, each call of the macro captures an exact copy of the macro definition body.

8.1.2 The MACRO Body

The body of a macro definition is composed of any number of assembler statements (and comments). These statements perform the function defined by the macro, and may include any valid assembler statements, **with the following limitations:**

1. The PROG directive are not valid.

2. Definitions of other macros, or calls to the same macro are not valid.

3. Nesting of macros is valid to three levels, but the innermost macros **must** be defined first.

4. Symbols that appear as statement labels within a macro are local to that macro.

5. Symbolic references to labels outside the macro are not valid.

6. Symbols defined by declarations within a macro are local if the macro call is made from a procedure, and global if made from the main program.

CAUTION

The assembler operation codes PROC, ENTRY, END, ORG, EQU, LIV and RIV are valid in a macro, but are difficult to implement without error.

8.1.3 The Terminator Statement

The terminator statement marks the end of a macro definition.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
none	ENDM	none	none	statement

8.2 THE MACRO CALL

The macro call statement marks a point in the program where the saved macro definition is expanded.

LABEL	OPERATION	OPERAND	EXTENSION	COMMENT
symbol	name	p1,p2,...pn	none	statement

Where:

"symbol". . . . is any valid symbol that is not defined as local to this program segment or global to the entire program.

"name" . . . . is the name of a macro which has appeared in the label field of a MACRO statement.

"p1,p2,...pn" . . is a list of the actual parameters to be substituted for the formal parameters that appear in the macro definition. These parameters may be constants, symbols, or expressions. Any symbols must be either previously defined, or defined within the macro definition prior to their use there. (Refer to the parameters IV1 and IV2 of Example 8.1.)

Example 8-2. Substitution of Macro Parameters

MACRO definition showing dummy parameters REPL1 and RX:

```
LOOK    MACRO   REPL1,RX
        ORG     4,256
        SEL     DISCI
        MOVE    REPL1,RX
        NZT     RX,*-2
```

MACRO call with the actual parameters substituted for the dummy parameters:

```
                LOOK    DSTAT,R1
+               ORG     4,256
+               SEL     DISCI
+               MOVE    DSTAT,R1
+               NZT     R1,*-2
```

(Taken from lines 63 through 67 and line 84 of Appendix C.)

8.3 MACRO EXAMPLES FOR USE WITH 8X310 Interrupt Control Coprocessor (ICC)

```
*8X310 CLEAR INTERRUPT

CLRI            MACRO
                MOVE R2, R2
                ENDM

*8X310 SUBROUTINE CALL (ONLY AT ODD LOCATION)

JSR             MACRO
                IF * $ 1 - 1
                NOP
                ENDIF
                MOVE R3, R3
                ENDM

*8X310 CLEAR MASK

CLRM            MACRO
                MOVE R4, R4
                ENDM

*8X310 SET MASK

SETM            MACRO
                MOVE R5, R5
                ENDM

*8X310 RETURN FROM SUBROUTINE OR INTERRUPT

RETN            MACRO
                MOVE R6, R6
                ENDM
```

SECTION 9

ASSEMBLY PROCESS

There are three results of assembling a program with MCCAP: the assembly listing, an object module, and the error codes.

9.1 THE ASSEMBLY LISTING

The most important function of the listing is to provide a record of all that occurred during the assembly process: source codes, object codes with addresses, and error codes. Typically, the assembly listing also serves as a documentation tool through the inclusion of descriptive comments with the source statements.

The following description refers to the partial assembly listing provided in Figure 9-1.

1. The first field, if present, contains alphabetic characters to indicate any errors during assembly.

2. The second field contains decimal numbers which are the listing line numbers. The maximum line number is 9999.

3. The third field contains a 5-digit octal number or a 4-digit hexadecimal number which represents the program memory address of the instruction generated.

4. The fourth field represents the code that was assembled or the value assigned in a symbol declaration. The code is written in the MCSIM object module format or in absolute hexadecimal format.

5. If extended instructions have been defined by the user, the fifth field will contain the instruction extension. Otherwise, the field is blank.

6. The sixth field contains the user's original source statements, without alteration.

7. A "+" in the sixth field indicates that the line was generated by a MACRO call and is the expansion of the macro.

8. After the END statement for the complete program, the return jump table is listed if any procedure calls were made.

9. After the return jump table the assembler prints the message "TOTAL ASSEMBLY ERRORS =", followed by a cumulative count of the errors.

10. The final part of the output is the symbol table or cross reference listing.

```
        200              000002          ABCDEF EQU 2
D       201              000003          ABCDEF EQU 3
        202              000005          ABCDEG EQU 5
        203  01061  0 02002  5020   J2 MOVE 2,2
D       204  01062  0 03002  5020   J2 MOVE 3,2
 X      205                          LAB10 ORG *
D       206  01063  6 00376  5020   S1  XMIT  -2,AUX
 U      207  01064  0 01000  5020     MOVE R1,R7
        208                          * REGISTER ERRORS
 R      209  01065  0 17001  5020     MOVE IVR,R1
 R      210  01066  0 01010  5020     MOVE R1,OVF
 R      211  01067  0 00012  5020     MOVE 0,10
        212  01070  0 00037  5020     MOVE 0,31
 R      213  01071  0 00000  5020     MOVE 0,32
 R      214  01072  6 10000  5020     XMIT 0,OVF
        215  01073  0 27027  5020     MOVE IV1,8,IV1
 R      216  01074  0 27127  5020     MOVE IV1,9,IV1
        217  01075  7 01075  FFFF   STOP  HALT/17H,377H,3,3
        218  01076                     PROC  PROC1
        219              003 6 1     IVV1  RIV  3,6,1
        220  01076  6 00004  5020     XMIT R4,AUX
        221              000002      X4  EQU 2
        222  01077  6 01002  5020     XMIT X4,R1
 U      223  01100  7 01100  5020     JMP LAB1
        224  01101  7 01131  5020     RTN
        225  01102  0 36102  5020     MOVE IVV1,R2
        226                          END PPOC1
        227  01103                   PROC  PROC2
 U      228  01103  7 01103  5020     JMP S1
D       229              000021      S1  SET 17
        230  01104  7 01131  5020     RTN
        231                          END PROC2
        232  01105                   PROC PROC3
 U      233  01105  6 00000  5020     XMIT S1,R1
        234  01106  0 37000  5020     MOVE MS1,AUX
        235  01107  7 01131  5020     RTN
        236  01110  0 00011  5020     MOVE AUX,R11
        237  01111  7 01131  5020     RTN
        238                          END PROC3
        239  01112                   PROC PROC5
        240  01112  6 02001  5020     XMIT 1,R2
        241  01113  6 11003  5020     CALL PROC1/
             01114  7 01076  5020
        242                          ENTRY ENTRY5
        243  01115  6 11004  5020     CALL PROC2
             01116  7 01103  5020
        244  01117  7 01131  5020     RTN/,,3
        245                          END PROC5
        246  01120                   PROC PROC8
        247  01120  0 01001  5020   P1  MOVE R1,R1
        248  01121  6 11005  5020     CALL ENTRY5
             01122  7 01115  5020
        249                          MAC2
        249  01123  7 01125  5020   *  JMP  *+2
```

Field 1 — Field 2 — Field 3 — Field 4 — Field 5 — Field 6

Figure 9-1. Interpretation of the Assembly Listing

9.2 THE CROSS REFERENCE TABLE

The accumulation of references may be started or stopped through use of the X parameter on the LIST and NLIST directives. If it is desired to list a complete cross reference table, the LIST X directive must be placed before the first symbol is used in the program. References during certain portions of a program will not be accumulated if the user specifies the NLIST X directive. Thus by use of the LIST and NLIST directives, the user may accumulate references wherever desired in the program. Typically a cross reference table will be generated for the entire program. References to internally defined Reserved Symbols are not accumulated. For a cross reference table to be generated at the end of the assembly listing, the LIST X directive must not have been turned off before the END directive.

The format of the cross reference table is shown in Example 9-1. A minus sign preceding a reference indicates that the symbol was defined on that line. A symbol may be defined on multiple lines by use of the SET directive. When 0 is given as a reference, it indicates that the symbol is a Reserved Symbol.

Example 9-1. Cross Reference Table Listing

LABEL	VALUE	REFERENCES			
AUX	000000	0			
IVL	000007	0			
MAIN	000000	-2	18		
TABLE	001057	105	-149	200	205

9.3 THE OBJECT MODULE

The object module is a machine readable output produced in either MCSIM, BNPF or ASCII-Hex format, as selected by the OBJ directive.

9.3.1 MCSIM Format

The MCSIM format is utilized by some 8X300/8X305 Development Systems, and can be loaded directly into it. This format is illustrated in Example 9-2.

Example 9-2. MCSIM Format

```
leader
program name (CR)(LF)
00000:  0 00000,6 01000,6 02000,6 07011,...,7 00400,
00010:  6 11000,7 00400,...
   .
   .
   .
(TAPE OFF)
END program name
```

9.3.2 BNPF Format

The BNPF format is used to produce paper tapes which can be used on most PROM programmers. This format is illustrated in Example 9-3.

Example 9-3. BNPF Format

```
leader
program name
leader (CR)(LF)
MODULE nn (STX)(CR)(LF)
0     BPNPNNNNNF      BNNNPNPNPF      BNPPNPNNNF
BNNPNPNNNF (CR)(LF)
4     BPNNPPPPPF      BNNNNNPNNF      BNPNPNPPPF
BNNNPNPNNF (CR)(LF)
  .
  .
  .
508     BPPNPNNNPF      BNNNNPNNPF      BPNNPNNNPF
BNNPNPNPNF (CR)(LF)(ETX)
leader (CR)(LF)
MODULE nn (STX)(CR)(LF)
  .
  .
  .
leader
END program name
```

The object module is divided into blocks according to the PROM statements in the MCCAP program. If a 512 by 8 PROM were defined, module 01 would represent 8X300/8X305 program storage locations 0 through 511, bits 0 through 7; and module 02 would represent locations 0 through 511, bits 8 through 15.

Note that since the program name may contain the letter B, the tape contains a leader following the program name so that the tape can be conveniently positioned in the programmer tape reader after the name.

9.3.3 ASCII-Hex (Quote) Format

The ASCII-Hex (Quote) format is one of three MCCAP formats which represent data as ASCII characters. The object module is divided into sections corresponding to the ROM size as specified in the PROM statement. A leader of blanks and the program name precede the module.

A STX character followed by a carriage return and a line feed indicates the start of each object module section. Each record in the object module consists of an address and eight words of data followed by a carriage return and a line feed. The address is a 4-digit hexadecimal number. Each word contains two hexadecimal characters written in pairs followed by a quote mark. An ETX character indicates the

end of each object module section. The trailer consisting of the characters END and the program name follow each object module. This format is illustrated in Example 9-4.

Example 9-4. ASCII-Hex (Quote) Format

```
leader
program name (CR)(LF)
MODULE nn'
(STX)(CR)(LF)
$A0000,  A0'15'68'28'9F'04'57'14' (CR)(LF)
$A0008,  05'F2'B3'21'00'81'DD'C2' (CR)(LF)
  .
  .
  .
$A01F8,  B1'26'79'36'D1'09'91'2A' (CR)(LF)
(ETX)

MODULE nn nn'
(STX)(CR)(LF)
  .
  .
  .
(ETX)
END program name
```

9.3.4 ASCII-Hex (Space) Format

The ASCII-Hex (Space) format also represents data as ASCII characters. This format is identical to the ASCII-Hex (Quote) format, with a space separating the data rather than a quote mark.

9.3.5 ASCII-HEX Word Format

The ASCII-Hex word format is a hexadecimal format widely used by development systems, ROM simulators, and PROM programmers. For word widths wider than eight bits, this format permits the entire word to be output as a single record.

The format generates a modified memory image, blocked into discrete records with length equal to one word. A word is defined to be one-user memory location and with MCCAP may be from 16 to 32 bits in width. Each record starts with a record mark and header consisting of length, type, and memory address (in user memory space) and is followed by a trailer consisting of two checksum characters. Data frames consist of ASCII-Hex characters where each character represents 4 bits. In cases where the micro word width is an odd number of nibbles, leading zeros are used to fill out the most significant byte of the data word to ensure that data records always contain a whole number of data bytes. A frame-by-frame description of the record is shown in Table 9-1.

Table 9-1. ASCII-Hex Word Format

Frame	Contents
0	**Record Mark.** Signals the start of a record. The ASCII character colon, 3AX, is used as the record mark.
1,2	**Record Length.** Two digits representing a hexadecimal number in the range 0 to FFX (0-255). This is the number of data bytes in the data frames. A record length of 0 indicates end of file.
3 to 6	**Load Address.** Four digits that represent the memory location where the data will begin loading.
7,8	**Record Type.** Two ASCII digits. Data records are type 00 and the end record is type 01 (length 0).
9 to 9+2*Length-1	**Data.** Each byte of memory is represented by two digits to represent 8 bits of binary data. These proceed from most significant nibble to least significant nibble. The number of data bytes is specified in Frames 1 and 2.
9+2*Length and 9+2*Length+1	**Checksum.** Two ASCII characters. The checksum is the two's complement of the 8-bit binary summation of all previous bytes in the record since the record mark (colon).

Example 9-5. ASCII-Hex Word Format

The 16-bit binary value 0101001111111000 is 53F8 in hexadecimal. To encode this, the first frame would contain the ASCII code for the character 5 (35X), the second frame would contain the ASCII code for the character 3 (33X), and so on.

If memory locations 1C40 through 1C42 contain 32-bit data of
53F8 EC40
1111 2222
3333 4444
the hex file produced (including control characters) would be:
:041C400053F8EC4029
:041C41001111222239
:041C420033334444B0
:00000001FF

9.4 ERROR CODES

If format or syntax errors are detected in the source code during the assembly process, an indication of the type of error is printed on the listing on the same line as the statement in error. Certain errors are considered to be catastrophic to the statement itself. Of these, some cause a "JMP *" to be assembled, while others enter a truncated or modulo value into a field if the specified value is too large. An error associated with a procedure definition will be reflected in references to the procedure. In all cases, however, object code is produced for every executable statement that is assembled, regardless of its validity.

Appendix D is a test program used to check for proper operation of the MCCAP assembler. Examples of error codes are presented in that test program.

Table 9-2. MCCAP Error Codes

Code	Error
A	**Argument Error:** 1. An operand (argument) is missing or contains an invalid character. 2. A PROG or PROC name is included in an expression.
B	**Bank Error:** In a MOVE, ADD, AND, or XOR, source and destination were data fields in the same bank but with different addresses.
C	**Context Error:** 1. A source or destination field contains a register or I/O data field variable used in an illegal context (that is, MOVE IV(2),A1; ADD R1,3,R3). 2. The name in a CALL statement is not a procedure name.
D	**Duplicate Definition:** 1. The symbol in the label field of a statement has been previously defined. 2. The procedure name has been previously defined.
F	**Format Error:** An instruction has a trailing comma or slash. (The instruction is assembled correctly.)
H	**Heading Error:** The program does not follow the correct format. That is, 1. no PROC statement after an END procedure statement; or 2. PROG is not the first statement in the program. (Some heading errors associated with the END statement will terminate the program.)

Code	Error
I	**I/O Data Field Error:** 1. I/O data fields whose precisions are not both eight and are in different banks are referenced in the same instruction. 2. I/O data fields within the same address but of different precisions are referenced in the same instruction.
L	**Label Error:** The symbol in the label field has 1. special characters, or 2. does not begin with an alphabetic character.
M	**Missing Symbol or RTN Statement:** 1. A statement requires a symbol. 2. A procedure does not have an RTN statement.
N	**Nesting Error:** An attempt was made to nest macros to more than three levels.
O	**Opcode Error:** 1. The code in the operation field has not been recognized as valid. 2. A RTN statement is used in the main program. 3. A macro definition is nested within another macro definition. 4. XML or XMR were used as opcodes without specifying 8X305 (See Sections 6.14 and 7.11).
P	**Paging Error:** An attempt was made to access a control storage address which is not in this page or segment (as applicable).
R	**Register Error:** 1. The register expression could not be evaluated. 2. The register expression is not in the proper range. 3. The register is not valid as used. (See Table 2-2.) 4. A rotate or a length field is out of range.
S	**Syntax Error:** A rule of syntax has been violated (for example, 4+*VAR).
T	**Table Overflow:** 1. The symbol table has overflowed. 2. More than 255 CALL statements were encountered by the Assembler. 3. The depth specified in a PROM directive is greater than the PROM buffer.

Code	Error
U	**Undefined Symbol:** There is a symbol in the operand field which 1. does not appear in any label field of this program segment or 2. has not been defined in a declaration statement.
V	**Value Error:** 1. An evaluated expression or constant is out of range for the field of the actual machine instruction in which it is to be contained. 2. For the LIV or RIV statements, the required length "bit+1" is not satisfied. 3. The PROM directive specifies more than 16 bits for instruction extension PROM's. 4. The number of bits in the PROM directive for standard 8X300/8X305 instructions does not total 16 bits. 5. More than 16 bits are defined in a DEF directive or a default value is too large for the field.
X	**Symbol Error:** A symbol is included in the label field of a statement for which it is not allowed.
"CROSS REFERENCE OVERFLOW AT LINE nnnn."	The cross reference table was filled at the line number specified.

APPENDIX A

STATEMENT/DEFINITION REFERENCE

ASSEMBLER DECLARATIONS

EQU	-	Define a statement
SET	-	Define or redefine a constant
LIV	-	Define a left bank data field variable
RIV	-	Define a right bank data field variable

ASSEMBLER DIRECTIVES

PROG	-	Program title statement
PROC	-	Procedure title statement
ENTRY	-	Secondary entry point into a procedure
END	-	End the program or a procedure
ORG	-	Set location counter
OBJ	-	Specify an objective format
IF, ENDIF	-	Conditional assembly
LIST	-	List the specified elements
NLIST	-	Suppress listing of elements
EJCT	-	Eject the listing page
SPAC	-	Line feed the listing
PROM	-	Specify PROM size
DEF	-	Define instruction extension fields

EXECUTABLE STATEMENTS

MOVE, ADD, AND, XOR	-	Data manipulation
XMIT	-	Load immediate
XEC	-	Execute
NZT	-	Non zero transfer
JMP	-	Unconditional jump
SEL		I/O data field selection
CALL	-	Procedure (subroutine) call
RTN	-	Procedure return
NOP	-	No operation
HALT	-	Stop processing
XML	-	8 bit load immediate to left bank (8X305 only)
XMR	-	8 bit load immediate to right bank (8X305 only)

APPENDIX B

UNASSEMBLED SAMPLE PROGRAM

```
**********************************    1
*  THIS PROGRAM SERVES ONLY AS A  *    2
*  DEMONSTRATION OF ALL MCCAP     *    3
*  STATEMENTS.                    *    4
**********************************    5
         LIST    A
         SPAC    1                      6
         PROG    SAMPLE                 7
         SPAC    2                      8
**********************************    9
*  DATA AND ADDRESS DECLARATIONS  *   10
**********************************   11
         SPAC    1                      12
FINAL    EQU     1                      13
PRELIM   EQU     0                      14
INC      EQU     1                      15
DEC      EQU     -1                     16
SINMSK   EQU     10000000B              17
OEMASK   EQU     1B                     18
LSMASK   EQU     7H                     19
SSMASK   EQU     LSMASK.L.3             20
MSMASK   EQU     LSMASK.R.1.L.6         21
ROT      EQU     3                      22
LEN      EQU     4                      23
VAL1     SET     0                      24
VAL2     SET     1                      25
VAL3     SET     2                      26
VAL4     SET     3                      27
DISCO    LIV     10H,7,8                28
DISCI    LIV     11H,7,8                29
DSTAT    LIV     DISCI,0                30
DSCLOK   LIV     DISCI,5                31
DRDWR    LIV     DISCI,6                32
DRDAT    LIV     DISCI                  33
DISP1    LIV     20H,7,8                34
DISP2    LIV     DISP1+1,7,8            35
DATA1    RIV     100H,7,8               36
D1SIGN   RIV     DATA1,0                37
D1ODEV   RIV     DATA1                  38
DATA2    RIV     DATA1+1,7,8            39
D2SIGN   RIV     DATA2,0                40
D2ODEV   RIV     DATA2                  41
TEMP1    RIV     200H,7,8               42
TEMP2    RIV     TEMP1+1,7,8            43
         SPAC    3                      44
         EJCT                           45
**********************************   46
*  CONDITIONALS AND SPECIAL       *   47
*  DIRECTIVES                     *   48
**********************************   49
         SPAC    1                      50
         IF      PRELIM                 51
         NLIST   S,O                    52
         ENDIF                          53
         SPAC    1                      54
         IF      FINAL                  55
         LIST    I,M,S,O                56
         OBJ     M                      57
         ENDIF                          58
         SPAC    1                      59
**********************************   60
*  MACRO DEFINITIONS              *   61
**********************************   62
```

```
LOOK     MACRO    REPL1,RX                  63
         ORG      4,256                     64
         SEL      DISC1                     65
         MOVE     REPL1,RX                  66
         NZT      RX,*-2                    67
         ENDM                               68
         SPAC     1                         69
LOOPCT   MACRO    RX                        70
         XMIT     -1,AUX                    71
         ADD      RX,RX                     72
         ENDM                               73
         SPAC     19                        74
         EJCT                               75
************************************       76
*  MAIN PROGRAM                      *       77
************************************       78
         SPAC     1                         79
         ORG      0                         80
START    NOP                                81
         XMIT     0,R1                      82
         XMIT     0,R2                      83
         LOOK     DSTAT,R1                  84
STC      CALL     ARITH                     85
         CALL     MOVMNT                    86
         CALL     TRNSMT                    87
         CALL     EXECT                     88
         LOOPCT   R6                        89
         NZT      OVF,START+3               90
LAST     HALT                               91
         SPAC     24                        92
         EJCT                               93
************************************       94
*  ARITH PROCEDURE                   *       95
************************************       96
         SPAC     1                         97
         PROC     ARITH                     98
         SPAC     1                         99
         ORG      256,256                  100
STAR     SEL      TEMP1                    101
         MOVE     R11,TEMP1                102
CANT     CALL     NONZXF                   103
         SEL      TEMP1                    104
         MOVE     TEMP1,R11                105
         XMIT     40H,AUX                  106
STAD     ADD      R1,R1                    107
         ADD      2,2                      108
         ADD      R3(ROT),R3               109
         SEL      DATA1                    110
         XMIT     LSMASK,AUX               111
STAND    AND      R1,DATA1                 112
         SEL      DATA2                    113
         XMIT     SSMASK,AUX               114
         AND      R2,LEN,DATA2             115
         XMIT     DATA2+1,IVR              116
         XMIT     MSMASK,AUX               117
         AND      R3,LEN,37H               118
         XMIT     DATA2+2,17H              119
         XMIT     263H,AUX                 120
         AND      4(4),AUX                 121
         SEL      DATA1                    122
         XMIT     -1,AUX                   123
STOR     XOR      DATA1,DATA1              124
```

```
          SEL     DATA2                    125
          XOR     DATA2,3,DATA2            126
          XMIT    DATA2+1,IVR              127
          XOR     37H,LEN,37H              128
          XMIT    DATA2+2,17H              129
          XOR     33H,LEN,37H              130
          SPAC    1                        131
          ENTRY   MOVMNT                   132
          SEL     DISCI                    133
STMV      MOVE    DSTAT,R1                 134
          MOVE    24H,LEN,R2               135
          MOVE    DRDAT,LEN,R3             136
          SPAC    4                        137
          EJCT                             138
**********************************         139
*  ARITH PROCEDURE (CONT'D)        *       140
**********************************         141
          SPAC    1                        142
          ENTRY   TRNSMT                   143
          SEL     DISP1                    144
STXT      XMIT    'G',R5                   145
          MOVE    R5,DISP1                 146
          SEL     DISP2                    147
          XMIT    'O',R5                   148
          MOVE    R5,DISP2                 149
          SEL     DISP1                    150
          XMIT    '!',R5                   151
          MOVE    R5,DISP1                 152
          XMIT    VAL1,DISP1,LEN           153
          XMIT    VAL2,23H,4               154
EAR       RTN                              155
          END     ARITH                    156
          SPAC    26                       157
          EJCT                             158
**********************************         159
*  EXECT PROCEDURE                 *       160
**********************************         161
          SPAC    1                        162
          PROC    EXECT                    163
          SPAC    1                        164
TABPTRS   RIV     240H,7,8                 165
T2PTR     RIV     TABPTRS                  166
T3PTR     RIV     TABPTRS,5                167
T4PTR     RIV     TABPTRS,3                168
T5PTR     RIV     TABPTRS,1                169
          ORG     7,256                    170
STXC      XEC     *+1(R6),5                171
          JMP     TAB1                     172
          JMP     TAB2                     173
          JMP     TAB3                     174
          JMP     TAB4                     175
          JMP     TAB5                     176
          ORG     6,256                    177
TAB1      XEC     *+1(R1)                  178
          JMP     DONE                     179
          JMP     DONE                     180
          JMP     DONE                     181
          JMP     DONE                     182
TAB2      SEL     TABPTRS                  183
          ORG     5,32                     184
          XEC     *+1(T2PTR)               185
          JMP     DONE                     186
```

```
         JMP      DONE                    187
TAB3     SEL      TABPTRS                 188
         ORG      7,32                    189
         XEC      *+1(T3PTR,2)            190
         JMP      DONE                    191
         JMP      DONE                    192
         JMP      DONE                    193
         JMP      DONE                    194
TAB4     SEL      TABPTRS                 195
         ORG      5,32                    196
         XEC      *+1(T4PTR),2            197
         JMP      DONE                    198
         JMP      DONE                    199
TAB5     SEL      TABPTRS                 200
         ORG      7,32                    201
         XEC      *+1(T5PTR,2),4          202
         JMP      DONE                    203
         JMP      DONE                    204
         JMP      DONE                    205
         JMP      DONE                    206
         ORG      32,32                   207
DONE     RTN                              208
         END      EXECT                   209
         SPAC     1                       210
**********************************       211
*  NONZXF PROCEDURE                 *     212
**********************************       213
         SPAC     1                       214
         PROC     NONZXF                  215
         SPAC     1                       216
VAL1     SET      VAL1+5                  217
VAL2     SET      VAL2+5                  218
VAL3     SET      VAL3+5                  219
VAL4     SET      VAL4+5                  220
         ORG      16,256                  221
         XMIT     VAL1,R5                 222
STNT     NZT      R5,*+8                  223
         SEL      DISP1                   224
         XMIT     VAL2,DISP1              225
         NZT      DISP1,*+7               226
         SEL      DISP2                   227
         XMIT     VAL3,DISP2              228
         NZT      23H,4,*+6               229
         RTN                              230
         LOOPCT   R5                      231
         SEL      DISP1                   232
         LOOPCT   DISP1                   233
         SEL      DISP2                   234
         LOOPCT   DISP2                   235
ENT      RTN                              236
         END      NONZXF                  237
         SPAC     12                      238
         END      SAMPLE                  239

R;
```

APPENDIX C

ASSEMBLED SAMPLE PROGRAM

```
1                                  ************************************    1
2                                  *  THIS PROGRAM SERVES ONLY AS A   *    2
3                                  *  DEMONSTRATION OF ALL MCCAP      *    3
4                                  *  STATEMENTS.                     *    4
5                                  ************************************    5
6                                          LIST    A

8                                          PROG    SAMPLE                   7


10                                 ************************************    9
11                                 *  DATA AND ADDRESS DECLARATIONS   *   10
12                                 ************************************   11

14          0001    FINAL   EQU    1                                       13
15          0000    PRELIM  EQU    0                                       14
16          0001    INC     EQU    1                                       15
17          FFFF    DEC     EQU    -1                                      16
18          0080    SINMSK  EQU    10000000B                               17
19          0001    OEMASK  EQU    1B                                      18
20          0007    LSMASK  EQU    7H                                      19
21          0038    SSMASK  EQU    LSMASK.L.3                              20
22          00C0    MSMASK  EQU    LSMASK.R.1.L.6                          21
23          0003    ROT     EQU    3                                       22
24          0004    LEN     EQU    4                                       23
25          0000    VAL1    SET    0                                       24
26          0001    VAL2    SET    1                                       25
27          0002    VAL3    SET    2                                       26
28          0003    VAL4    SET    3                                       27
29          0238    DISCO   LIV    10H,7,8                                 28
30          0278    DISCI   LIV    11H,7,8                                 29
31          0241    DSTAT   LIV    DISCI,0                                 30
32          0269    DSCLOK  LIV    DISCI,5                                 31
33          0271    DRDWR   LIV    DISCI,6                                 32
34          0279    DRDAT   LIV    DISCI                                   33
35          0438    DISP1   LIV    20H,7,8                                 34
36          0478    DISP2   LIV    DISP1+1,7,8                             35
37          1038    DATA1   RIV    100H,7,8                                36
38          1001    D1SIGN  RIV    DATA1,0                                 37
39          1039    D1ODEV  RIV    DATA1                                   38
40          1078    DATA2   RIV    DATA1+1,7,8                             39
41          1041    D2SIGN  RIV    DATA2,0                                 40
42          1079    D2ODEV  RIV    DATA2                                   41
43          2038    TEMP1   RIV    200H,7,8                                42
44          2078    TEMP2   RIV    TEMP1+1,7,8                             43
```

PROG SAMPLE MICROCONTROLLER CROSS ASSEMBLER VER 3.0

```
47                          ************************************    46
48                          *  CONDITIONALS AND SPECIAL        *    47
49                          *  DIRECTIVES                      *    48
50                          ************************************    49

52                                  IF      PRELIM                  51
53                                  ENDIF                           53

55                                  IF      FINAL                   55
56                                  LIST    I,M,S,O                 56
57                                  OBJ     M                       57
58                                  ENDIF                           58

60                          ************************************    60
61                          *  MACRO DEFINITIONS               *    61
62                          ************************************    62
63                          LOOK    MACRO   REPL1,RX                63
64                                  ORG     4,256                   64
65                                  SEL     DISCI                   65
66                                  MOVE    REPL1,RX                66
67                                  NZT     RX,*-2                  67
68                                  ENDM                            68

70                          LOOPCT  MACRO   RX                      70
71                                  XMIT    -1,AUX                  71
72                                  ADD     RX,RX                   72
73                                  ENDM                            73
```

```
76                          ************************************    76
77                          *  MAIN PROGRAM                    *    77
78                          ************************************    78

80                                  ORG     0                       80
81   0000   0000     START          NOP                             81
82   0001   C100                    XMIT    0,R1                    82
83   0002   C200                    XMIT    0,R2                    83
84                                  LOOK    DSTAT,R1                84
84                          +       ORG     4,256                   64
84   0003   C709            +       SEL     DISCI                   65
84   0004   1021            +       MOVE    DSTAT,R1                66
84   0005   A103            +       NZT     R1,*-2                  67
85   0006   C900             STC    CALL    ARITH                   85
     0007   E100
86   0008   C901                    CALL    MOVMNT                  86
     0009   E11F
87   000A   C902                    CALL    TRNSMT                  87
     000B   E123
88   000C   C903                    CALL    EXECT                   88
     000D   E12F
89                                  LOOPCT  R6                      89
89   000E   C0FF            +       XMIT    -1,AUX                  71
89   000F   2606            +       ADD     R6,R6                   72
90   0010   A803                    NZT     OVF,START+3             90
91   0011   E011             LAST   HALT                            91
```

```
 94                          ************************************   94
 95                          *  ARITH PROCEDURE                 *   95
 96                          ************************************   96

 98   0012                          PROC    ARITH                    98

100   0012   E100                   ORG     256,256                 100
101   0100   CF80    STAR           SEL     TEMP1                   101
102   0101   091F                   MOVE    R11,TEMP1               102
103   0102   C904    CANT           CALL    NONZXF                  103
      0103   E161
104   0104   CF80                   SEL     TEMP1                   104
105   0105   1F09                   MOVE    TEMP1,R11               105
106   0106   C020                   XMIT    40H,AUX                 106
107   0107   2101    STAD           ADD     R1,R1                   107
108   0108   2202                   ADD     2,2                     108
109   0109   2363                   ADD     R3(ROT),R3              109
110   010A   CF40                   SEL     DATA1                   110
111   010B   C007                   XMIT    LSMASK,AUX              111
112   010C   411F    STAND          AND     R1,DATA1                112
113   010D   CF41                   SEL     DATA2                   113
114   010E   C038                   XMIT    SSMASK,AUX              114
115   010F   429F                   AND     R2,LEN,DATA2            115
116   0110   CF42                   XMIT    DATA2+1,IVR             116
117   0111   C0C0                   XMIT    MSMASK,AUX              117
118   0112   439F                   AND     R3,LEN,37H              118
119   0113   CF43                   XMIT    DATA2+2,17H             119
120   0114   C0B3                   XMIT    263H,AUX                120
121   0115   4480                   AND     4(4),AUX                121
122   0116   CF40                   SEL     DATA1                   122
123   0117   C0FF                   XMIT    -1,AUX                  123
124   0118   7F1F    STOR           XOR     DATA1,DATA1             124
125   0119   CF41                   SEL     DATA2                   125
126   011A   7F7F                   XOR     DATA2,3,DATA2           126
127   011B   CF42                   XMIT    DATA2+1,IVR             127
128   011C   7F9F                   XOR     37H,LEN,37H             128
129   011D   CF43                   XMIT    DATA2+2,17H             129
130   011E   7B9F                   XOR     33H,LEN,37H             130

132                          ENTRY   MOVMNT                  132
133   011F   C709                   SEL     DISCI                   133
134   0120   1021    STMV           MOVE    DSTAT,R1                134
135   0121   1482                   MOVE    24H,LEN,R2              135
136   0122   1783                   MOVE    DRDAT,LEN,R3            136
```

```
                                  ************************************  139
                                  *  ARITH PROCEDURE (CONT'D)        *  140
                                  ************************************  141

                                          ENTRY   TRNSMT                  143
0123 C710                                 SEL     DISP1                   144
0124 C547                         STXT    XMIT    'G',R5                  145
0125 0517                                 MOVE    R5,DISP1                146
0126 C711                                 SEL     DISP2                   147
0127 C54F                                 XMIT    'O',R5                  148
0128 0517                                 MOVE    R5,DISP2                149
0129 C710                                 SEL     DISP1                   150
012A C521                                 XMIT    '!',R5                  151
012B 0517                                 MOVE    R5,DISP1                152
012C D780                                 XMIT    VAL1,DISP1,LEN          153
012D D381                                 XMIT    VAL2,23H,4              154
012E E173                         EAR     RTN                             155
                                          END     ARITH                   156
```

```
                                  ************************************
                                  *  EXECT PROCEDURE                 *
                                  ************************************

                                           PROC    EXECT

TABPTRS  RIV     240H,7,8
T2PTR    RIV     TABPTRS
T3PTR    RIV     TABPTRS,5
T4PTR    RIV     TABPTRS,3
T5PTR    RIV     TABPTRS,1
                 ORG     7,256
STXC     XEC     *+1(R6),5
         JMP     TAB1
         JMP     TAB2
         JMP     TAB3
         JMP     TAB4
         JMP     TAB5
                 ORG     6,256
TAB1     XEC     *+1(R1)
         JMP     DONE
         JMP     DONE
         JMP     DONE
         JMP     DONE
TAB2     SEL     TABPTRS
                 ORG     5,32
         XEC     *+1(T2PTR)
         JMP     DONE
         JMP     DONE
TAB3     SEL     TABPTRS
                 ORG     7,32
         XEC     *+1(T3PTR,2)
         JMP     DONE
         JMP     DONE
         JMP     DONE
         JMP     DONE
TAB4     SEL     TABPTRS
                 ORG     5,32
         XEC     *+1(T4PTR),2
         JMP     DONE
         JMP     DONE
TAB5     SEL     TABPTRS
                 ORG     7,32
         XEC     *+1(T5PTR,2),4
         JMP     DONE
         JMP     DONE
         JMP     DONE
         JMP     DONE
                 ORG     32,32
DONE     RTN
                 END     EXECT
```

```
211                          ************************************  211
212                          *  NONZXF PROCEDURE                *  212
213                          ************************************  213

215  0161                            PROC    NONZXF                 215

217            0005          VAL1    SET     VAL1+5                 217
218            0006          VAL2    SET     VAL2+5                 218
219            0007          VAL3    SET     VAL3+5                 219
220            0008          VAL4    SET     VAL4+5                 220
221                                  ORG     16,256                 221
222  0161      C505                  XMIT    VAL1,R5                222
223  0162      A56A          STNT    NZT     R5,*+8                 223
224  0163      C710                  SEL     DISP1                  224
225  0164      D706                  XMIT    VAL2,DISP1             225
226  0165      B70C                  NZT     DISP1,*+7              226
227  0166      C711                  SEL     DISP2                  227
228  0167      D707                  XMIT    VAL3,DISP2             228
229  0168      B38E                  NZT     23H,4,*+6              229
230  0169      E173                  RTN                            230
231                                  LOOPCT  R5                     231
231  016A      C0FF          +       XMIT    -1,AUX                  71
231  016B      2505          +       ADD     R5,R5                   72
232  016C      C710                  SEL     DISP1                  232
233                                  LOOPCT  DISP1                  233
233  016D      C0FF          +       XMIT    -1,AUX                  71
233  016E      3717          +       ADD     DISP1,DISP1                    72
234  016F      C711                  SEL     DISP2                  234
235                                  LOOPCT  DISP2                  235
235  0170      C0FF          +       XMIT    -1,AUX                  71
235  0171      3717          +       ADD     DISP2,DISP2                    72
236  0172      E173           ENT    RTN                            236
237                                  END     NONZXF                 237


239                                  END     SAMPLE                 239
```

```
PROG    SAMPLE                  MICROCONTROLLER CROSS ASSEMBLER VER 3.0

 0173     8974
 0174     E008
 0175     E00A
 0176     E00C
 0177     E00E
 0178     E104

  ASSEMBLER ERRORS =      0
```

SYMBOL TABLE

* 1

ARITH	0100	AUX	0000	D1ODEV	1039	D1SIGN	1001
D2ODEV	1079	D2SIGN	1041	DATA1	1038	DATA2	1078
DEC	FFFF	DISC1	0278	DISCO	0238	DISP1	0438
DISP2	0478	DRDAT	0279	DRDWR	0271	DSCLOK	0269
DSTAT	0241	EXECT	012F	FINAL	0001	INC	0001
IVL	0007	IVR	000F	LAST	0011	LEN	0004
LOOK	0001	LOOPCT	0006	LSMASK	0007	MOVMNT	011F
MSMASK	00C0	NONZXF	0161	OEMASK	0001	OVF	0008
PRELIM	0000	R0	0000	R1	0001	R11	0009
R12	000A	R13	000B	R14	000C	R15	000D
R16	000E	R17	000F	R2	0002	R3	0003
R4	0004	R5	0005	R6	0006	R7	0007
ROT	0003	SAMPLE	0000	SINMSK	0080	SSMASK	0038
START	0000	STC	0006	TEMP1	2038	TEMP2	2078
TRNSMT	0123	VAL1	0005	VAL2	0006	VAL3	0007
VAL4	0008						

* 2

* 3

* 4

CANT	0102	EAR	012E	STAD	0107	STAND	010C
STAR	0100	STMV	0120	STOR	0118	STXT	0124

* 5

DONE	0160	STXC	012F	T2PTR	2839	T3PTR	2829
T4PTR	2819	T5PTR	2809	TAB1	0135	TAB2	013A
TAB3	013E	TAB4	0145	TAB5	0149	TABPTR	2838

* 6

ENT	0172	STNT	0162

* 7

* 8

* 9

APPENDIX D

ASSEMBLER ERROR TEST PROGRAM

```
 1                                  *
 2                                  *
 3                                  * THIS PROGRAM IS USED TO TEST THE ASSEMBLER
 4                                  * FOR PROPER OPERATION.  IT SHOWS THE VARIOUS
 5                                  * INSTRUCTION FORMATS, ASSEMBLER DIRECTIVES,
 6                                  * AND ERROR CONDITIONS.
 7                                  *
 8                                      PROG  MAIN
 9                                    LIST    X       GENERATE CROSS REF TABLE
10                                    DEF     4(5),-8(2),2,2
11                                    PROM    128,8,8/256,4,8,4
12                                    OBJ     R,H/R
13                                  *
14                                   LIST M,I,A
15                                  MAC3  MACRO  A1,A2
16                                    MOVE  A1,A2
17                                    XEC   *(A1)
18                                    XMIT  1,A2
19                                    ENDM
20                                  MAC2  MACRO
21                                    JMP   *+2
22                                    MOVE  R1,IVL
23                                    SEL   WS1
24                                    MAC3  R1,R11
25                                    ENDM
26                                  MAC1  MACRO  P1,P2
27                                    MOVE P1,P2
28                                    MOVE P2,P1
29                                    ENDM
30                                  *
31              00B8                IV1   LIV 2,7,8
32              00FE                IV2    LIV  3,7,6
33              00F6                IV3    LIV  3,6,6
34              1938                WS1   RIV  100,7,8
35              1977                WS2   RIV  101,6,7
36              19AE                WS3   RIV  WS2+1,5,6
37    0000      E000    5020          HALT
38    0001      0000    0650          NOP/0,WS2
39    0002      0000    5020          NOP
40    0003      0000    7020          MOVE AUX,AUX/7
41    0004      0100    5020          MOVE R1,R0/
42    0005      0101    5020          MOVE R1,R1
43    0006      0109    1023          MOVE R1,R11/1,2,,3
44    0007      0102    2070          MOVE R1,R2/IV1,7
45    0008      0103    5A00          MOVE R1,R3/,LAB1
46    0009      0104    5020          MOVE R1,R4
47    000A      0105    FFF8          MOVE R1,R5/1111B,-1,2
48    000B      0106    0FFF          MOVE R1,R6/0,X1,3,3
49    000C      0107    5021          MOVE R1,IVL/,,,1
50    000D      010F    5020          MOVE R1,IVR
51    000E      0201    33F0          MOVE R2,R1/IV3,77H
52    000F      0802    20F5          MOVE OVF,R2/2,*,1,1
53    0010      0000    0000          MOVE AUX,AUX/0,0,0,0
54    0011      0001    5020          MOVE 0,1
```

```
      55   0012     0007   5020      MOVE 0,7
      56   0013     0009   5020      MOVE 0,9
      57   0014     000F   5020      MOVE 0,15
      58   0015     0010   5020      MOVE 0,16
      59   0016     0017   5020      MOVE 0,23
      60   0017     2121   5020      ADD R1(1),R1
      61   0018     2101   5020      ADD R1(8),R1
      62   0019     4017   5020      AND AUX,IV1
      63   001A     4117   5020      AND R1,IV1+1
      64   001B     4237   5020      AND R2,1,IV1
      65   001C     42F7   5020      AND R2,7,IV1
      66   001D     4217   5020      AND R2,8,IV1
      67   001E     7702   5020      XOR  IV1,R2
      68   001F     7702   5020      XOR IV1,0,R2
      69   0020     7722   5020      XOR IV1,1,R2
      70   0021     77E2   5020      XOR IV1,7,R2
      71   0022     C141   102F      XMIT 'A',R1/1,2,3,3
      72   0023     C25A   5020      XMIT 'Z',R2
      73   0024     C030   5020      XMIT '0',AUX
      74   0025     C927   5020      XMIT '''',R11
      75   0026     9707   5020      XEC  *+1(IV1)
      76   0027     9768   5160      XEC  *+1(IV1,3)/,LI2,0
      77   0028     C51F   5020      XMIT 1FX,R5
      78   0029     C7C4   5020      XMIT 0C4X,R7
      79   002A     C0E5   5020      XMIT -1BX,R0
      80   002B     C4FF   5020      XMIT -1,R4
V     81   002C     C900   5020      XMIT -256,R11
      82   002D     C700   5020      XMIT 0,IVL
      83   002E     CF01   5020      XMIT 1,IVR
      84   002F     D71E   5020      XMIT 30,IV1
      85   0030     FFFF   0000      JMP 8191/0,0,0,0
      86            05A1            LI2 LIV 22,4
      87   0031     C716   F00F      SEL LI2/17H,0,3,3
      88   0032     0234   5020      MOVE R2,LI2
      89            0699            LI4  LIV  LI2+4,3,1
      90   0033     C71A   5020      SEL LI4
      91   0034     3323   5020      ADD LI4,R3
      92                             * PAGING ERROR
      93                               ORG  128
      94                             NLIST A
P     95   00200  4 01200  5020      XEC  *(R1),256
P     96   00201  4 01201  5020      XEC  *(R1),253
      97   00202  4 01240  5020       XEC  LAB1(R1)
P     98   00203  4 27000  5020      XEC  LAB1(IV1)
      99   00204  4 27037  5020      XEC  LAB1-1(IV1)
P    100   00205  4 27037  5020      XEC  LAB1-1(IV1),32
     101   00206  4 27006  5020      XEC *(IV1),LAB1-*
P    102   00207  4 27010  5020      XEC  *+1(IV1),LAB1-*
     103   00210  4 27010  2640      XEC  *(IV1),LAB1-*/2,100
     104                             ORG  128+32
     105   00240  0 00000  5020     LAB1  MOVE AUX,AUX
     106   00241  5 01300  5020      NZT  R1,LAB2
P    107   00242  5 27000  5020      NZT  IV1,LAB2
     108   00243  5 27037  5020      NZT  IV1,LAB2-1
```

```
109  00244  5 27435  5020      NZT   IV2,4,LAB2-3
110  00245  5 27305  0024      NZT   IV1,3,*/0,IV1,1,0
111                            ORG   128+32+32
112  00300  0 00001  5020    LAB2  MOVE AUX,R1
113  00301  6 01000  5020     XMIT 0,R1
114  00302  4 27002  5020      XEC  *(IV1,8)
115  00303  4 01303  5020      XEC  *(R1),2
116                           * EXPAND MACRO
117                              MAC1  R1,AUX
117  00304  0 01000  5020    +  MOVE R1,AUX
117  00305  0 00001  5020    +  MOVE AUX,R1
118           000026          S1  SET 22
U    119  00306  7 00306  5020    JMP S1
120           000031          S1 SET S1+3
U    121  00307  7 00307  5020    JMP S1
122  00310  6 11000  5020      CALL PROC1
     00311  7 01121  5020
123  00312  0 00001  5020     MOVE AUX,R1
124  00313  6 11001  5020     CALL PROC2
     00314  7 01126  5020
125  00315  0 02000  5020     MOVE 2,AUX
126  00316  6 11002  0000     CALL PROC3/0,0
     00317  7 01130  0000
127                           *  SPACE 5 LINES
```

```
129                            ORG 512
130  01000  0 00000  5020      MOVE AUX,AUX
131  01001  6 01001  5020     XMIT 1,R1
132                           * IF DIRECTIVE
133                            IF 0
134                            MOVE  R1,1
135                            XMIT 2,R4
136                            ENDIF
137                            IF 1
138  01002  0 02003  5020     MOVE  2,R3
139                            ENDIF
140                           * VARIOUS EXPRESSIONS AND OPERATORS
141  01003  7 00100  5020     JMP 2.L.5
142  01004  7 00000  5020     JMP 2.R.5
143  01005  6 00004  5020     XMIT  31$4,AUX
144  01006  6 01020  5020     XMIT 2+3-6+17,R1
145  01007  6 00017  5020     XMIT 1001B+6,AUX
146  01010  6 00025  5020     XMIT R5+2.L.3,AUX
147  01011  6 02027  5020     XMIT  27H,R2
148  01012  6 00002  10A0    XMIT 2,R0/1,*,0
149                           *  EJECT TO NEXT PAGE
```

```
      151                                    *  ARGUMENT ERRORS
A     152  01013  6 00000  5020               XMIT ,IV2,8
A     153  01014  7 01014  5020               XEC ,R1
A     154  01015  6 00000  5020               XMIT -1,
A     155  01016  7 01016  5020               JMP MAIN
A     156  01017  7 01017  5020               JMP PROC1
A     157  01020  6 02377  0001               XMIT -1,R2/1,28H,1,0
      158           001237                   K1  EQU  1237H
A     159           000000                   K3  EQU  1238H
A     160           000000                   K4  EQU 1002B
      161                                    * SYNTAX ERRORS
S     162  01021  7 01021  5020               XEC  *,R1
S     163  01022  6 00000  5020                XMIT  1++2,R1
S     164  01023  6 00001  0005               XMIT 1,R0/,IV1+,
      165                                    * IV BYTE AND BYTE ERRORS
      166  01024  0 27027  5020               MOVE IV1,IV1
      167  01025  0 37027  5020               MOVE WS1,IV1
I     168  01026  0 36027  5020               MOVE WS2,IV1
      169  01027  0 27626  5020               MOVE IV2,IV3
I     170  01030  0 27627  5020               MOVE IV1,IV2
      171  01031  0 27036  5020               MOVE IV2,8,WS2
      172  01032  0 27036  5020               MOVE IV2,0,WS2
B     173  01033  0 27027  5020               MOVE IV1+1,IV1
      174                                    * VALUE ERRORS
      175                                      LIST A
V     176   021C    E21C   5020               XEC  *(IV1,9)
      177   021D    B71D   5020               NZT  IV1,8,*
V     178   021E    E21E   5020               NZT  IV1,9,*
V     179   021F    C300   5020               XMIT -257,R3
V     180   0220    D780   5020               XMIT 32,IV2,4
V     182   0221    D700   5020               XMIT -32,IV1
V     183   0222    D7C0   5020               XMIT -33,IV2
V     184   0223    E223   5020               JMP 8192
V     186   0225    C000   5020               XMIT 256,AUX
      187                                      LIST S
      188   0226    C105   F020               XMIT 5,R1/15
V     189   0227    C2FF   0020               XMIT -1,R2/16
      190                                      NLIST A
V     191  01050  1 01000  5021              ADD  R1,AUX/,,,5
      192           020000                   X5  EQU 8192
V     193  01051  7 01051  5020               JMP X5
      194          377 7 1                  RI2  RIV 255
V     195          000 0 0                  RI3  RIV 256
V     196          000 0 0                  RI5  RIV 2,8,0
      197                                    * CONTEXT ERRORS
C     198  01052  6 01002  5020               XMIT  2,R1,3
C     199  01053  7 01053  5020               NZT  R1,2,*
C     200  01054  6 07037  5020               XMIT 31,IVL,3
C     201  01055  7 01055  5020               XEC  *+1(R1,2)
      202           177777                   X1  EQU -1
C     203  01056  7 01056  5020               SEL X1
      204                                    * UNDEFINED LABELS, LABEL ERRORS,
      205                                    * AND DUPLICATE LABELS.
```

```
 U    206  01057  6 00000  5020    XMIT K7,R1
 L    207  01060  0 00000  5020  LABEL, EQU 2
      208          000002        ABCDEF EQU 2
 D    209          000003        ABCDEF EQU 3
      210          000005        ABCDEG EQU 5
      211  01061  0 02002  5020  J2 MOVE 2,2
 D    212  01062  0 03002  5020  J2 MOVE 3,2
 X    213                        LAB10 ORG *
 D    214  01063  6 00376  5020  S1  XMIT  -2,AUX
      215  01064  0 01007  5020    MOVE R1,R7
      216                        * REGISTER ERRORS
 R    217  01065  0 17001  5020    MOVE IVR,R1
 R    218  01066  0 01010  5020    MOVE R1,OVF
 R    219  01067  0 00012  5020    MOVE 0,10
      220  01070  0 00037  5020    MOVE 0,31
 R    221  01071  0 00000  5020    MOVE 0,32
 R    222  01072  6 10000  5020    XMIT 0,OVF
      223  01073  0 27027  5020    MOVE IV1,8,IV1
 R    224  01074  0 27127  5020    MOVE IV1,9,IV1
      225                          8X305
      226  01075  0 07007  5020    MOVE R7,IVL
      227  01076  0 15007  5020    MOVE R15,R7
      228  01077  0 12013  5020    MOVE R12,R13
      229  01100  0 13015  5020    MOVE R13,R15
      230  01101  0 14005  5020    MOVE R14,R5
      231  01102  0 00016  5020    MOVE R0,R16
      232  01103  0 17001  5020    MOVE R17,R1
      233                          8X300
      234  01104  0 01007  5020    MOVE R1,R7
 R    235  01105  0 07001  5020    MOVE R7,R1
 R    236  01106  0 12001  5020    MOVE R12,R1
 R    237  01107  0 07013  5020    MOVE IVL,R13
      238  01110  0 00017  5020    MOVE R0,IVR
 R    239  01111  0 02014  5020    MOVE R2,R14
 R    240  01112  0 15017  5020    MOVE R15,R17
 R    241  01113  0 16000  5020    MOVE R16,R0
      242                        * OP CODE ERRORS
      243                          8X305
      244  01114  6 12002  5020    XML 2
      245  01115  6 13017  5020    XMR 0FX
      246                          8X300
 O    247  01116  0 00000  5020    XML 7
 O    248  01117  0 00000  5020    XMR 11
      249  01120  7 01120  FFFF  STOP  HALT/17H,377H,3,3
      250  01121                    PROC  PROC1
      251          003 6 1       IVV1  RIV  3,6,1
      252  01121  6 00004  5020    XMIT R4,AUX
      253           000002       X4  EQU 2
      254  01122  6 01002  5020    XMIT X4,R1
 U    255  01123  7 01123  5020    JMP LAB1
      256  01124  7 01154  5020    RTN
      257  01125  0 36102  5020    MOVE IVV1,R2
      258                          END PROC1
      259  01126                    PROC  PROC2
```

```
 U     260  01126  7 01126  5020      JMP S1
D      261          000021          S1  SET 17
       262  01127  7 01154  5020      RTN
       263                           END PROC2
       264  01130                    PROC PROC3
       265                           LIST A
 U     266   0258    C000   5020      XMIT S1,R1
       267   0259    1F00   5020      MOVE WS1,AUX
       268   025A    E26C   5020      RTN
       269   025B    0009   5020      MOVE AUX,R11
       270   025C    E26C   5020      RTN
       271                           END PROC3
       272   025D                    PROC PROC5
       273   025D    C201   5020      XMIT 1,R2
       274   025E    C903   5020      CALL PROC1/
             025F    E251   5020
       275                           ENTRY ENTRY5
       276   0260    C904   5020      CALL PROC2
             0261    E256   5020
       277   0262    E26C   502C      RTN/,,3
       278                           END PROC5
       279   0263                    PROC PROC8
       280   0263    0101   5020    P1  MOVE R1,R1
       281   0264    C905   5020      CALL ENTRY5
             0265    E260   5020
       282                           MAC2
       282   0266    E268   5020  +   JMP  *+2
       282   0267    0107   5020  +   MOVE  R1,IVL
       282   0268    CF64   5020  +   SEL  WS1
       282                        +   MAC3  R1,R11
       282   0269    0109   5020  +   MOVE  R1,R11
       282   026A    816A   5020  +   XEC  *(R1)
       282   026B    C901   5020  +   XMIT  1,R11
 M     283                             END PROC8
       284                           END  MAIN/0,0,0,0
```

RETURN TABLE

```
       026C    896D   0000
       026D    E0CA   0000
       026E    E0CD   0000
       026F    E0D0   0000
       0270    E260   0000
       0271    E262   0000
       0272    E266   0000
```

ASSEMBLER ERRORS = 66

CROSS REFERENCE

LABEL	VALUE	REFERENCE
* 1		
ABCDEF	0002	-208 -209
ABCDEG	0005	-210
AUX	0000	0
ENTRY5	0260	-275 281
IV1	00B8	-31 44 62 63 64 65 66 67
		68 69 70 75 76 84 98 99
		100 101 102 103 107 108 110 110
		114 164 166 166 167 168 170 173
		173 176 177 178 182 223 223 224
		224
IV2	00FE	-32 109 169 170 171 172 180 183
IV3	00F6	-33 51 169
IVL	0007	0
IVR	000F	0
J2	0231	-211 -212
K1	029F	-158
K3	0000	-159
K4	0000	-160
LAB1	00A0	45 97 98 99 100 101 102 103
		-105
LAB2	00C0	106 107 108 109 -112
LI2	05A1	76 -86 87 88 89
LI4	0699	-89 90 91
MAC1	000A	0
MAC2	0005	0
MAC3	0001	0
MAIN	0000	155 284
OVF	0008	0
PROC1	0251	122 156 -250 258 274
PROC2	0256	124 -259 263 276
PROC3	0258	126 -264 271
PROC5	025D	-272 278
PROC8	0263	-279 283
R0	0000	0
R1	0001	0
R11	0009	0
R12	000A	0
R13	000B	0
R14	000C	0
R15	000D	0
R16	000E	0
R17	000F	0
R2	0002	0
R3	0003	0
R4	0004	0
R5	0005	0
1		
R6	0006	0
R7	0007	0
RI2	3FF9	-194
RI3	0000	-195
RI5	0000	-196
S1	0019	-118 119 -120 120 121 -214 260 -261
		266

STOP	0250	-249					
WS1	1938	-34	167	267	283		
WS2	1977	-35	36	38	168	171	172
WS3	19AE	-36					
X1	FFFF	48	-202	203			
X5	2000	-192	193				

* 2

* 3

IVV1	00F1	-251	257
X4	0002	-253	254

* 4

* 5

* 6

* 7

P1	0263	-280

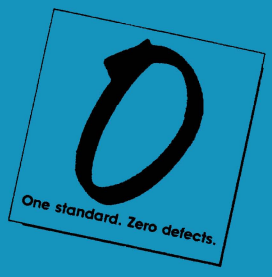
* 8

* 9

Signetics

a subsidiary of U.S. Philips Corporation

Signetics Corporation
811 E. Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088-3409
Telephone 408/991-2000



98-3001-830