Advanced
Micro
Computers
A subsidiary of
Advanced Micro Devices

MACRO8000
AmZ8000 Assembler

User's Manual

00680119

$10.00

	REVISION RECORD
REVISION	**DESCRIPTION**
01	Preliminary Issue
(4/27/79)	
A	Manual Released
(10/25/79)	
B	Manual updated to correct documentation errors.
(1/11/80)	
C	Manual updated to correct documentation errors.
(3/17/80)	
D	Manual updated to correct documentation errors and reprinted.
(4/25/80)	
E	Manual updated to support AmZ8001
(9/19/80)	
F	Manual updated to correct documentation errors and reprinted.
(12/31/80)	

Publication No.
00680119

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

Address comments concerning
this manual to:

ADVANCED MICRO COMPUTERS

Publications Department
3340 Scott Boulevard
Santa Clara, CA 95051

PREFACE

The AMC MACRO8000 macroassembler can assemble programs targeted for either the AmZ8001 or the AmZ8002, the two members of the AmZ8000 microprocessor family.

MACRO8000 supports many features usually associated with high-level languages. For example, an IF...THEN...ELSE constructs is available. Familiarity with a high-level language is therefore useful for readers of this manual.

Creation of AmZ8000 programs requires knowledge of the processor instruction set and the linker, as well as the macroassembler. The following documents contain information on the first two topics.

AmZ8001/2 Processor Instruction Set (AMD)
LINK8000 User's Manual (publication number 00680148)

The notations used in this manual are:

UPPERCASE	In syntax indicates a name that is specified as shown.
lowercase	In syntax indicates that a name or value must be supplied by the user.
...	In syntax indicates that an item can be repeated.
. . .	In examples indicates that some part of the program is not shown.

NOTE

The information in this publication is intended to be accurate in all respects. However, Advanced Micro Computers disclaims responsibility for any errors and any consequences resulting from errors. This product is intended for use as described in this manual.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

CHAPTER 1
OVERVIEW OF MACRO8000

AMC MACRO8000 uses AmZ8000 source files as input and produces output files containing either absolute or relocatable code. Relocatable output files must be further processed by the AMC linker, LINK8000. Code produced by MACRO8000 can be targeted to run on either the AmZ8002 or the AmZ8001.

1-1. The AmZ8000 PROGRAMMING ENVIRONMENT

The AmZ8000 programming environment is determined by the answer to one fundamental question:

> Does the source program or program module use any segmented addresses?

A segmented address is represented by a pair of numbers, a 7-bit segment number and a 16-bit offset; it is stored in a 32-bit register pair. A non-segmented address, on the other hand, is represented by a single, 16-bit number; it is stored in a word register. Segmented addresses can be used only by the AmZ8001, while non-segmented addresses can be used by either the AmZ8001 or the AmZ8002. (A bit in the FCW of the AmZ8001 controls the type of addresses it uses. See the AmZ8001/2 Instruction Set Manual for more details.)

When the assembler is invoked, the S option controls the programming environment. If the S option is not chosen, the output code will use exclusively non-segmented addresses. If the S option is chosen, the output code may use segmented addresses as well as non-segmented addresses, and the code must be run on an AmZ8001. If the S option is not chosen, the code will usually be run on an AmZ8002 (although with a user-supplied loader it is possible to run the code on an AmZ8001). Hereafter in this manual and in the LINK8000 User's Manual, we will use such phrases as "targeted for the AmZ8001" or "AmZ8001 code" to mean that the S option has been chosen, and conversely, we will use "targeted for the AmZ8002" or "AmZ8002 code" to mean that the S option has not been chosen. Chapter 5 contains a discussion of segmented an non-segmented addresses and how users can specify which kind is generated by the assembler.

If the S option is chosen, relocatable code is produced, but either relocatable or absolute code may be produced if the S option is not chosen. Relocatable code must be further processed by the linker, while absolute code may be immediately downloaded to the target processor.

An absolute output file is produced when a PROGRAM directive appears at the beginning of the source file (see section 3-1). Absolute files are produced in either binary (AMC Bin) format or Intel Hex format (see appendixes D and E), depending on whether the B or H option is chosen when the assembler is invoked. Hex and binary files can be used as input by PROM burning software. Binary files can also be downloaded to the Am95/4016 Evaluation Board or to user configured AmZ8002 systems, or used as input to RTE16, the AMC Real Time Emulator.

A relocatable output file is produced when a MODULE directive appears at the beginning of the source file and the O option is selected when the assembler is invoked. This directive usually indicates that the source file contains one module of a program having one or more separately-assembled modules. However, all source files targeted for an AmZ8001 must use the MODULE directive, even if they contain single, monolithic programs. Relocatable files must be processed by the linker, LINK8000, before they can be executed. The linker takes one or more relocatable files and combines them into a single absolute file (either binary or hex format) that can be used for PROM burning or downloading. See the LINK8000 User's Manual for more details. Figure 1-1 illustrates all the possible paths from source file to absolute file.

1-2. SOURCE PROGRAM

Input lines on the source file may be up to 96 characters in length. Lower case characters are treated as upper case characters.

The line numbers produced by the EDIT text editor are recognized but ignored for the purpose of user program assembly.

The source program can include the contents of other files if the INCLUDE directive is used (see chapter 3). At the place where each INCLUDE directive appears in the source program, the contents of the specified file are used as input the assembler.

1-3. INVOKING THE MACROASSEMBLER

The command to call the MACRO8000 assembler specifies the source file containing the program and specifies various assembly options. The call is:

```
MACZ source options overrides
```

The call is entered with a carriage return (new line key).

The specification of source is:

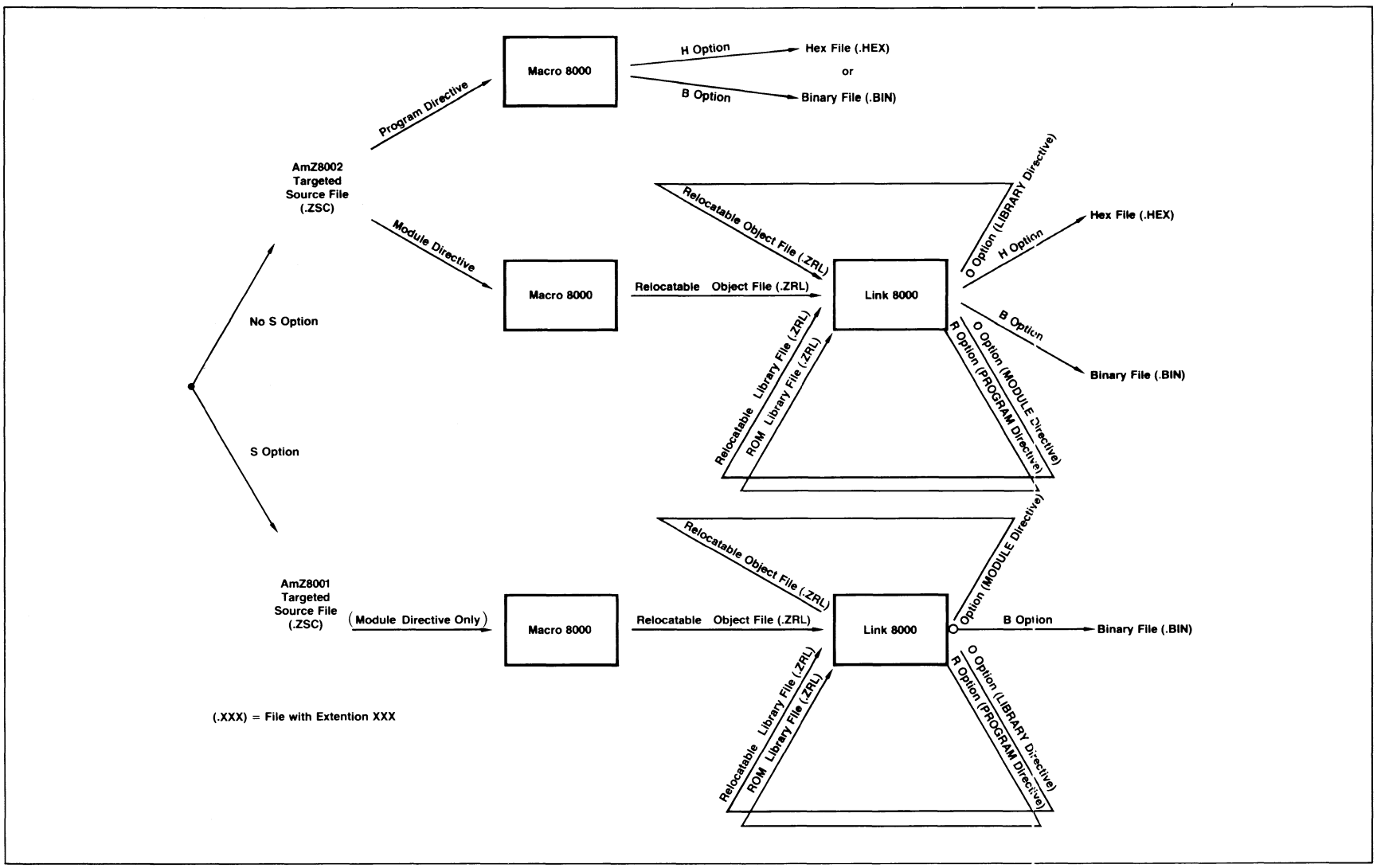


Figure 1-1. File Paths

Field	Meaning	Default
dev:	Is the optional drive name, such as A: or B:	Currently selected drive
filename	Is the source file name	--
.ext	Is the source file type	.ZSC

For example:

```
MACZ ONE
```

calls for assembly of file ONE.ZSC on the current drive, requests defaults for all the options, and omits any overrides.

The source specification can be followed by at least one space and then the selected options. The options can be specified in any order and are separated by commas or spaces. The options are:

Name	Default	Form	Meaning
Privilege	No privileged instructions can be used	P	Enable assembly of privileged AmZ8000 instructions.
Dots	No first pass listing	D	Produce a first pass listing that shows a dot for each statement assembled.
Error	Full program listing	E	Suppress full program listing and produce only a listing of errors.
Warning	No warnings	W	Enable listing of warning messages.
Trace	No trace	T	Trace macro expansion and any additional code generated by IF directives.
Map	No map	M	Produce a reference map for labels used in the program.
Listing	L=CON:	L	Send listing to dev:name.PRN on same drive as source, with same name as source.
		L=file	Send listing to the file dev:name.ext as specified.
		L=CON:	Send listing to console device (if printer is enabled with CONTROL P, listing also prints).

Name	Default	Form	Meaning
		L=LST:	Send listing to printer device.
Object	No object file for input to LINK8000	O	Create object file dev:name.ZRL on same drive as source, with same name as source.
		O=file	Create object file dev:name.ext as specified.
Hex	No hex file produced (AmZ8002 only)	H	Produce hex file dev:name.HEX on same drive as source, with same name as source. File can be used to program PROMs.
		H=file	Produce hex file dev:name.ext as specified. File can be used to program PROMs.
Bin	No binary file produced (AmZ8002 only)	B	Produce binary file dev:name.BIN on same drive as source, with same name as source. File can be downloaded to the 96/4016 AmZ8000 Evaluation Board.
		B=file	Produce binary file dev:name.ext as specified. File can be downloaded to the 96/4016 AmZ8000 Evaluation Board.
Segmentation	Targeted for the AmZ8002	S	Code targeted for the AmZ8001

For example:

```
MACZ ONE W,T,O
```

calls for assembly of file ONE.ZSC on the current drive, requests warning messages, requests trace messages, requests an object file named ONE.ZRL, uses defaults for the other options, and omits any overrides.

The product call line can optionally include overrides for one or more symbolic constant values in the program. The overrides follow all of the other options and are separated by commas or spaces.

The name of each symbolic constant must be at least 2 characters in length. The specified value is substituted for the value contained in the program, and the new value is in effect only for the current assembly. The overrides have the form:

			Meaning
Constant override	No override	con=val	Override symbolic constant with a different value for program assembly.

For example:

```
MACZ ONE E TEST5=TRUE,XY=128
```

calls for assembly of file ONE.ZSC on the current drive, requests an error listing only, requests defaults for all other options, overrides the symbolic constant TEST5 with the value TRUE, and overrides the symbolic constant XY with the value 128 for the current assembly.

The macroassembler can be interrupted by typing <CONTROL>C (or by typing any character). The macroassembler will then ask if you wish to abort the assembly process.

1-4. LISTING

The listing displays the source statements in the program, the code or data generated by the assembler, and the addresses of the code and data items. Each line of the listing has the following format:

```
Address        Assembled Code/Data        Source Statement
```

The first column of the listing, the address field, shows the current value of the location counter. The location counter contains a byte address for the code or data item. The location counter changes as each code or data item is assembled, or if an ORIGIN directive is encountered (see chapter 3), or if a SEGMENT directive is encountered (see chapter 5).

The next three columns, the assembled code/data field, contain 16-bit words (in hex) representing the value generated by the assembler for each source line. Space holders for unresolved segment numbers and relocatable addresses have the following formats:

*nnnn	For a relocatable, non-segmented address, where nnnn is a number standing in for the true location counter offset.
Sssss nnnn	For a segmented long-offset address, where ssss is a number standing in for the true segment number and nnnn is either the true location counter offset or a number standing in for the true offset.
Pssnn	For a segmented short offset address, where ss is a number standing in for the true segment number and nn is either the (maximum 256) location counter offset or a number standing in for the time offset.

Ixxxx	For a relocatable relative address reference, where xxxx is a combination of the opcode (CALR, DJNZ, DBJNZ, or JR) and a number standing in for the true displacement (7-bits, DJNZ and DBJNZ; 8-bits, JR; or 12-bits, CALR).
Rrrrr	For a relocatable relative address reference, where rrrr is a number standing in for the true 16-bit displacement.

The last item on the listing line is the original source line. Additional information in the listing includes header lines, error messages, and trace messages.

1-5. EXAMPLE

This sample program is targeted for an AmZ8002 and makes use of the monitor facilities provided by the Am96/4016 Evaluation Board for console input and output. (The System Call instructions in locations 501E and 506E access the monitor.)

The assembler listing is completed by the message:

```
NEITHER WARNING NOR ERROR MESSAGES
```

The lines following the assembler listing in the example illustrate the commands used to download the assembled program from a System 8/8 to the 4016 Evaluation Board, which serves as an execution vehicle, and then to run the program. For further details about the use of the 4016 Evaluation Board, consult the Am96/4016 Evaluation Board User's Manual.

```
0000
0000                        PROGRAM BUBBLE_SORT;
0000                        ORIGIN  #5000;
5000                        PAGE    41;
5000
5000                        CONST   SWAPS           = R0,
5000                                POINTER         = R1,
5000                                LAST            = R2,
5000                                THIS            = R3,
5000                                NEXT            = R4,
5000                                LAST_VAL        = RH5,
5000                                THIS_VAL        = RL5,
5000                                POS             = R6,
5000                                COUNT           = R7;
5000
5000                    BUBBLE_SORT:
5000                    READ:
5000                            (* SET UP CALL BLOCK *)
5000  4D05 5072 0100            LD      CALL_BLOCK(0), #0100;
5006  4D05 5074 0000            LD      CALL_BLOCK(2), 0;
500C  2101 507A                LD      POINTER, ^BUFFER;
5010  6F01 5076                LD      CALL_BLOCK(4), POINTER;
5014  4D05 5078 0050            LD      CALL_BLOCK(6), 80;
501A                            (* SET UP POINTER TO CALL BLOCK *)
501A  2101 5072                LD      POINTER, ^CALL_BLOCK;
501E  7F00                     SC      0;
5020                            (* GET CHARACTER COUNT *)
5020  6106 5078                LD      POS, CALL_BLOCK(6);
5024                            (* ADJUST FOR CR *)
5024  AB60                     DEC     POS, 1;
5026
5026                    SORT:
5026                            (* INITIALIZE FOR SORT *)
5026  2100 0000                LD      SWAPS, 0;
502A                            (* INITIALIZE POINTERS *)
502A  2103 507A                LD      THIS, ^BUFFER;
502E  A134                     LD      NEXT, THIS;
5030  A940                     INC     NEXT, 1;
5032                            (* ADJUST WORKING COUNT *)
5032  A167                     LD      COUNT, POS;
5034  AB70                     DEC     COUNT, 1;
5036                            (* CY LATER USED FOR OV *)
5036  8D83                     RESFLG  CY;
5038                    LOOK:
5038  BA46 073B                CPSIRB  THIS^, NEXT^, COUNT, LGT;
503C                            IF OV (* AT END OF LINE *)
503C                                THEN
503C  EC01 8D81                         SETFLG  CY;
5040                            IF ZR (* SWAP NEEDED *)
5040                                THEN
5040  EE07                              BEGIN
5042                                    (* GET POINTER TO LAST *)
5042  A132                              LD      LAST, THIS;
5044  AB20                              DEC     LAST, 1;
```

```
BUBBLE_SORT                          MACRO8000 AMZ8000 ASSEMBLER

5046                                         (* SWAP LAST AND THIS *)
5046   2025                                  LDB     LAST_VAL, LAST^;
5048   2031                                  LDB     THIS_VAL, THIS^;
504A   2E2D                                  LDB     LAST^, THIS_VAL;
504C   2E35                                  LDB     THIS^, LAST_VAL;
504E                                         (* INCREMENT SWAP COUNTER *)
504E                                         INC     SWAPS, 1
504E   A900                                  END;
5050                                 IF NC (* CY SAVED OV STATUS *)
5050                                     THEN
5050   E701 E8F2                             JR      LOOK;
5054                                 IF SWAPS NE 0 (* CHANGES WERE MADE *)
5054   8500                              THEN
5056   E601 E8E6                             JR      SORT;
505A
505A                         WRITE:
505A                                 (* CHANGE TYPE OF CALL *)
505A   4D05 5072 0200                LD      CALL_BLOCK(0), #0200;
5060                                 (* PLACE LINE FEED IN BUFFER *)
5060   A960                          INC     POS, 1;
5062   4C65 507A 0A0A                LDB     BUFFER(POS), #0A;
5068                                 (* PROVIDE ACTUAL COUNT *)
5068   A960                          INC     POS, 1;
506A   6F06 5078                     LD      CALL_BLOCK(6), POS;
506E   7F00                          SC      0;
5070
5070                         EXIT:
5070   7F01                          SC      1;
5072
5072                         CALL_BLOCK:
5072                                 WORD    (4);
507A
507A                         BUFFER:
507A                                 BYTE    (80);
50CA
50CA                                 END.
NEITHER WARNING NOR ERROR MESSAGES
A>HOST
SYSTEM 8/8 Z8000 HOST
(A)
LDPR A:BUBBLE.BIN
(A)
LOAD COMPLETED
*
G
(A)
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
(A)
        ABCDDEEEEFGHHIJKLMNOOOOPQRRSTTUUVWXYZ
PROGRAM EXIT 01
*
^LEND
HOST SHUTDOWN
A>
```

CHAPTER 2
STATEMENT ELEMENTS

The statements in a program are either MACRO8000 directives (described in chapter 3) or AmZ8000 instructions (described in chapter 6). The general rules for statements are described in this chapter.

2-1. STATEMENT FORM

MACRO8000 source statements are free-form, in that statements can begin in any column and additional blanks can be inserted at will. Blank characters and blank lines can be inserted anywhere to improve the readability of the program. Statements can be continued on several lines and several statements can be included in the same line.

Source lines of up to 96 characters in length are allowed. Lower case characters are treated as upper case characters, except that lower case letters in literal strings and comments are preserved. Tab characters are ignored for program assembly but appear correctly on the program listing. Any line numbers created by the EDIT text editor are recognized but ignored for the purpose of program assembly.

2-2. SINGLE STATEMENT

The single statement consists of a statement beginner, such as an op-code, followed by zero or more operands. A single statement is separated from the next statement by a semicolon. For example:

```
ADD     R4, R5;
INC     R4, 1;
```

2-3. COMPOUND STATEMENT

A compound statement consists of BEGIN, single statements, and END. The last single statement in the group has an optional semicolon. The END serves as the terminator for the last single statement. For example:

```
BEGIN
LDB     RL6, RL4;
SLLB    RL6, 8;
CALL    FIXVAL;
INC     R2, 1;
CALL    PLTEST
END;
```

2-4. COMMENTS

A comment statement can be placed at the end of a souce line by preceding it with a percent (%) symbol. A percent sign comment is terminated by the end of the line. For example:

```
INC  R6, 1;    %ADJUST COUNT
```

Comments can also be embedded anywhere in the source text by enclosing them between the symbols "(*" and "*)". For example,

```
INC  R6   (*PRESENT COUNT*), 1;
```

The only exception to this rule is that comments cannot appear within literal strings.

2-5. DELIMITERS

Within statements, the standard delimiters are blanks, commas, and parentheses. Blanks can be used freely in statements. Commas are used to separate operands. Parentheses are primarily used for instruction operands in certain addressing modes.

The keywords BEGIN and END are special delimiters used in compound statements. The keywords THEN and ELSE are special delimiters used in IF directives. The keywords IN and DO are special delimiters used in FOR directives.

2-6. IDENTIFIERS

The identifiers that can be used in MACRO8000 programs are labels, op-codes, macro names, symbolic constants, and object variables. Each type of identifier is discussed in this chapter.

Identifiers can be as long as 80 characters. The characters A through Z, 0 through 9, underline, and @ can be used in an identifier, but an identifier cannot start with a digit or an underline. For example, valid symbols are:

```
LOOP
LABEL5
@B14INC
TEST_FOR_NEGATIVE_VALUE
```

2-7. LABELS

A label is prefixed to a statement. A colon follows the label and precedes the statement. Labels can be used to identify code or data within the program.

For example:

```
NBT1:   LD      R10, 0;
```

associates label NBT1 with the LD instruction shown. Another statement such as:

```
        JR      NBT1;
```

would cause a jump to the statement labeled NBT1, so that the next instruction executed is the instruction associated with NBT1.

For example:

```
DAT4:   WORD:   12;
```

associates label DAT4 with the WORD directive shown, which reserves a single data word with the value 12. Another statement such as:

```
        LD      R7, DAT4;
```

references the current value of the memory location associated with DAT4.

Labels cannot be used directly in arithmetic exressions. See the description of effective address operands in section 2-30.

A label can have the same name as an opcode, a user-defined macro, or a redefinable assembler directive, but a warning message is issued. For instance, a warning message appears when the name TEST is used as a label (TEST is an AmZ8000 instruction).

The user can place multiple labels on a statement. For example:

```
X1: X4: CALL    SF_ROUTINE;
```

specifies that labels X1 and X4 are both associated with the statement.

2-8. STATEMENT BEGINNERS

The statement beginners are the identifiers that indicate the purpose of the statement. A statement beginner can be an opcode for an AmZ8000 instruction, the name of an assembler directive such as WORD, or the name of a macro defined by the user.

2-9. MNEMONICS

A mnemonic specifies one of the AmZ8000 instruction classes described in chapter 4. The mnemonics are 2 through 6 letters in length and are followed by operands for the instruction. When an mnemonic appears in the program, the assembler checks the operands and then generates the appropriate AmZ8000 code. For example:

```
LD      R0, 1;      % LD mnemonic is the statement beginner
```

Each AmZ8000 mnemonic is implemented as an intrinsic macro, that is, each mnemonic effectively references a macro supplied by the assembler and generates AmZ8000 code for the instruction. The user can therefore write a macro to redefine any AmZ8000 mnemonic. A warning message indicates that the user has redefined the name, and all subsequent references to the name reference the macro defined by the user, instead of the old AmZ8000 mnemonic.

2-10. DIRECTIVE NAMES

A directive is a special instruction to the assembler. For instance, directives are used to reserve memory space for data and to control the program listing. The MACRO8000 directives are described in chapters 3, 4, and 5. For example:

```
CONST   X = 4;      % CONST directive is the statement beginner
```

The directives are statement beginners, but some directives are considered reserved words and some can be redefined. The directives CONST, VAR, MACRO, BYTE, WORD, LONG, STRING, IF, and FOR are considered reserved words and cannot be redefined. The names of all the other directives can be redefined by the user.

2-11. MACRO NAMES

A macro is defined by the user with the MACRO directive described in chapter 4. The macro name is an identifier. A macro must be defined before being referenced, that is, the macro definition must precede any references to the macro. For example:

```
MACRO   HL7     PARM1;
        BEGIN
        .
        .
        .
        END;
.
.
.
HL7     5;          % HL7 macro name is the statement beginner
```

2-12. OPERANDS

In general, the operands in a statement always follow the statement beginner. For AmZ8000 instructions, the operands are instruction operands. For directives, the operands are values required for the directives. For macro references, the operands are the macro parameters. The rest of this chapter describes the types of operands that can be used in MACRO8000 statements.

2-13. IMMEDIATE OPERANDS

An immediate operand is a numeric constant or a numeric expression that is evaluated at assembly time. For example:

```
ADD     R4, 7;          % add the value 7 into register 4
ADD     R4, 8 * 17;     % add the value 8 * 17 into register 4
```

2-14. Constants

A numeric constant can take any of the forms:

Form	Base	Example
nnnn	Decimal	12
nnnnD	Decimal	39D
nnnnB	Binary	1101B
nnnnO	Octal	6710
nnnnQ	Octal	377Q
nnnnH	Hexadecimal	0E5H
#nnnn	Hexadecimal	#A5
r#nnnn	Variable base (base is chosen from 1 to 15)	4#3021
nnnnK	Multiple of 1024	4K

Numeric constants are represented internally as signed 32-bit constants. According to the situation, the least significant byte (8 bits), word (16 bits), or long word (all 32 bits) can be used.

Hexadecimal constants expressed in the form nnnnH rather than #nnnn cannot begin with a letter. Therefore, a 0 must be used as the first digit, as in:

```
0FF4H
```

which represents the same value as #FF4.

Numeric constants can contain any number of embedded underlines for improved readability. For example:

```
1_024
0111_1111_1010B
0F_55_5A_00H
```

2-15. Numeric Expressions

An immediate operand can also be an expression that is evaluated at assembly time to produce a 32-bit signed value. Numeric expressions can be written using the various arithmetic operators. Each arithmetic operation is defined like the corresponding AmZ8000 operation at run time. The operators for numeric expressions are:

Operator	Meaning	Example	Precedence
HIGH	Use the high byte of X	HIGH X	Next
LOW	Use the low byte of X	LOW X	
SWAP	Exchange high and low bytes	SWAP X	
*	Multiply	X * Y	Highest
/	Divide	X / Y	
MOD	Remainder after X / Y	X MOD Y	
SHR	Shift X right by Y bits	X SHR Y	
SHL	Shift X left by Y bits (SHR and SHL are like the Shift Dynamic Logical instruction)	X SHL Y	
-	Negation (0 - X)	- X	Next
+	Add	X + Y	Next
-	Subtract	X - Y	
NOT	Complement	NOT X	Next
AND	Logical AND	X AND Y	Lowest
OR	Inclusive OR	X OR Y	
XOR	Exclusive OR	X XOR Y	

A numeric expression can be written in any of the following ways:

1. A numeric constant, which is the simplest form of a numeric expression. For example:

   ```
   5
   #7F
   ```

2. A numeric expression consisting of a constant, one of the arithmetic operators, and another constant. For example:

   ```
   4K / 8
   ```

3. A numeric expression containing subexpressions, where subexpressions of the highest precedence are evaluated first. For example:

   ```
   5 * 4 + 1
   ```

   has the result 21. Parentheses can be used within an expression to force the order of evaluation. For example:

   ```
   5 * (4 + 1)
   ```

   has the result 25 because (4 + 1) is evaluated first.

4. A numeric expression consisting of an effective address operand. For example:

   ```
   ^LABEL1 - ^LABEL2
   ```

   Only subtraction is allowed in these expressions. See section 2-30 for more details.

Note that a string of 1 to 4 characters can be used as an operand in a numeric expression. A string of 1 to 3 characters is right-justified and zero-filled before being used in the numeric expression.

Note that AND, OR, and NOT may be used with boolean variables as well as numeric variables, and that NOT may be used with condition codes (see section 2-26).

2-16. REGISTER OPERANDS

A register operand specifies one of the AmZ8000 general purpose registers. The registers are:

8-bit Byte Register		16-bit Word Register	32-bit Register Pair	64-bit Register Quad
RH0	RL0	R0	RR0	RQ0
RH1	RL1	R1		
RH2	RL2	R2	RR2	
RH3	RL3	R3		
RH4	RL4	R4	RR4	RQ4
RH5	RL5	R5		
RH6	RL6	R6	RR6	
RH7	RL7	R7		
		R8	RR8	RQ8
		R9		
		R10	RR10	
		R11		
		R12	RR12	RQ12
		R13		
		R14	RR14	
		R15		

For example:

```
ADDB    RL4, RL6;       % add byte register RL6 into RL4
ADDB    RH4, RH6;       % add byte register RH6 into RH4
ADD     R4, R6;         % add word register R6 into R4
ADDL    RR4, RR6;       % add register pair RR6 into RR4
MULTL   RQ8, RR12;      % multiply register pair RR10 (lower half
                        % RQ8) by RR12 and put result in register
                        % quad RQ8
```

2-17. INDIRECT REGISTER OPERANDS

An indirect register operand is a register that contains the address of the operand. The register designator is followed by the pointer symbol ^. For example:

```
ADD     R4, R2^;        % add contents of word that R2 points to
                        % into R4
```

Any word register or register pair can be used as an indirect register operand (except for R0 and RR0). Word registers contain 16-bit non-segmented addresses, while register pairs contain segmented addresses.

2-18. DIRECT ADDRESS OPERANDS

A direct address operand is a label. The label is associated with an statement that is an instruction or that contains a data value to be used as the operand. For example:

```
CALL    JADX;           % call the routine beginning with the
                        % instruction labeled JADX
ADD     R4, LAB;        % add the data value associated with label
                        % LAB into R4
```

A direct address operand can be followed by one or more displacements, each enclosed in parentheses. Each displacement can be a positive or negative constant or numeric expression. For example:

```
ADD     R4, LAB(6);     % add into R4 the data value found at (LAB
                        % plus 6 bytes)
```

For AmZ8002 code, a direct address operand can also be an absolute address constant in the form constant^ or the form (expression)^. For example:

```
ADD     R4, #4344^;     % add the data value at address #4344
                        % into R4
```

2-19. RELATIVE ADDRESS OPERANDS

A relative address operand is the same as a direct address operand, except that relative addressing is used. For example:

```
CALR    JADX;           % call the routine beginning with the
                        % instruction labeled JADX
JR      NZ, POI5;       % jump relative on condition nonzero to
                        % the instruction labeled POI5
```

NOTE

The use of relative addressing for calls and jumps is highly recommended and is more efficient than absolute addressing in terms of memory usage.

2-20. INDEXED OPERANDS

An indexed operand is a direct address operand followed by a displacement that is a register. The displacement can be any word register except R0 and is enclosed in parentheses. The label and displacement together address the operand value. For example:

```
ADD     R4, LAB(R1);      % take the value in R1 as a displacement
                          % the data to be added into R4
                          % is found at (LAB plus displacement)
```

2-21. BASE ADDRESS OPERANDS

A base address operand is an indirect register operand followed by a displacement that is a constant or numeric expression. The indirect register can be any word register except R0 (non-segmented addresses), or any register pair except RR0 (segmented addresses). The displacement is enclosed in parentheses. The location addressed by the indirect register and the constant or expression displacement together address the operand value. For example:

```
LD      R4, R2^(20);      % Load into R4 the data value addressed by
                          the % contents of R2 plus 20
```

2-22. BASE INDEXED OPERANDS

A base indexed operand is an indirect register operand followed by a register operand enclosed in parentheses. The indirect register can be any word register except R0 (non-segmented addresses), or any register pair except RR0 (segmented addresses). The second register can be any word register except R0. The location addressed by the indirect register is combined with the displacement contained in the second register to produce the address of the operand. For example:

```
LD      R4, R2^(R1);      % take the value in R1 as a displacement
                          % Load the data value addressed by (R2
                          % plus displacement) into R4
```

2-23. PORT ADDRESS OPERANDS

A port address operand is an immediate operand that specifies a 16-bit port address for I/O operations. For example:

```
OUT     #0FC0, R4;        % output the contents of R4 to port #0FC0
```

2-24. PORT REGISTER OPERANDS

A port register operand is a register that contains a 16-bit port address for I/O operations. For example:

```
OUT     R2, R4;          % output the contents of R4 to the port
                         % address contained in R0
```

2-25. FLAGS

The flags are:

Name	Meaning
CY	Carry flag (C)
ZR	Zero flag (Z)
SGN	Sign flag (S)
PY	Parity flag (P)
OV	Overflow flag (V)

2-26. CONDITION CODES

The condition codes are:

Name	Meaning	If used, test for
TRUE	Always true	-
FALSE	Always false	-
NZ	Not zero	ZR= 0
ZR	Zero	ZR= 1
NC	No carry	CY= 0
CY	Carry	CY= 1
PO	Parity odd	PY= 0
PE	Parity even	PY= 1
PL	Plus	SGN= 0
MI	Minus	SGN= 1
NE	Not equal	ZR= 0
EQ	Equal	ZR= 1
NOV	Overflow is reset	OV= 0
OV	Overflow is set	OV= 1

Name	Meaning	If used, test for
GE	Greater than or equal	(SGN XOR OV)= 0
LT	Less than	(SGN XOR OV)= 1
GT	Greater than	(ZR OR (SGN XOR OV))= 0
LE	Less than or equal	(ZR OR (SGN XOR OV))= 1
LGE	Logical greater than or equal	CY= 0
LLT	Logical less than	CY= 1
LGT	Logical greater than	((CY= 0) AND (ZR= 0))= 1
LLE	Logical less than or equal	(CY or ZR)= 1

2-27. INTERRUPTS

The interrupts are:

Name	Meaning
VI	Vectored interrupt
NVI	Nonvectored interrupt
NMI	Nonmaskable interrupt

2-28. CONTROL REGISTERS

The control registers are:

Name	Meaning
FCW	Flag control word
FLAGS	Flag byte of the FCW
PSAPSEG	New program status area pointer (NPSAP) segment number
PSAPOFF	New program status area pointer (NPSAP) offset
NSPSEG	Normal stack pointer (R14) segment number
NSPOFF	Normal stack pointer (R15) offset
REFRESH	Refresh counter

2-29. LOCATION COUNTER

The current value of the location is represented as $, which is considered a label. For example:

```
ORIGIN  $(40);          % sets the new origin to the current value
                        % of the location counter plus the
                        % displacement of 40 bytes
```

2-30. EFFECTIVE ADDRESS OPERANDS

An effective address generated by one of the AmZ8000 addressing modes can be used as an operand in itself in the Load or Load Relative instructions. Certain effective address operands can also be used in the WORD directive (see chapter 3). An operand that is preceded by the symbol ^ is interpreted by the assembler as an effective address operand. The symbol ^ can be translated as the address of. For example:

```
LD  R4, ^LAB;              % load R4 with the address of LAB

LD  R11, ^(R2^ (20))       % load R11 with the address specified by
                           % the contents of R2 plus 20
```

Segmented addresses must be stored in 32-bit register pairs. For example:

```
LDL RR2, ^SEGLAB           % load RR2 with the segmented address
                           % of SEGLAB
```

The effective address notation allows programmers to use standard Load and Load Relative instructions instead of having to use the AmZ8000 LDA and LDAR mnemonics. Of course, when an effective address operand is used with a Load or Load Relative instruction, the assembler generates an LDA or LDAR opcode. See chapter 6 for examples.

An effective address operand can include simple arithmetic expressions (only + and - allowed as operators). The current value of the location counter (symbolized by $) can be used in these expressions. For example:

```
LD    R4, ^LAB + 32        % load R4 with the address of LAB
                           % displacement by 32 bytes

LD    R2, ^LAB1 - ^LAB2    % load R2 with the address of LAB1
                           % minus the address of LAB2

LD    R4, ^$ + 200         % load R4 with the current value of
                           % the location counter plus 200
```

2-31. STRINGS

Strings can be used in a number of MACRO8000 directives. In all cases, a string or string expression can be used. A string consists of zero or more characters delimited by apostrophes. The apostrophe itself is represented within a string by a double apostrophe. If there are zero characters between apostrophes, then the string is an empty string. For example, valid strings are:

```
'ABCDEF.ZSC'            % string with 10 characters
'00A0'                  % string with 4 characters
'IT''S'                 % string with 4 characters; value IT'S
''                      % empty string
```

The string operator is:

Operator	Meaning	Example
&	Concatenate	'ABC' & 'DEF'

The maximum length of a string is 254 characters. Two or more strings can be concatenated to form a string result. For example:

```
'B:' & 'ABCDEF' & '.ZSC' % string expression with the
                         % 12-character value 'B:ABCDEF.ZSC'
```

A string used as an operand in an arithmetic expression can contain no more than four ASCII characters. An empty string in an arithmetic expression has a value of zero. A string of 1 to 3 characters is right-justified and zero-filled in the 32-bit field.

2-32. LISTS

The primary use of lists is in passing lists of items as an argument in macro calls. When a macro parameter has a list value, the list value can be used in a FOR directive (described in chapter 3). A list is a composite object containing one or more items. The entire list is enclosed in parentheses. A list with one item has the form:

```
(item)
```

A list with two or more items has the form:

```
(item,item...)
```

2-33. SYMBOLIC CONSTANTS

A symbolic constant is an identifier that represents a fixed operand value during the assembly process. Symbolic constants are declared by the CONST declaration (described in chapter 3).

When a symbolic constant name appears as an operand, the value of the symbolic constant is used as the operand. For example:

```
CONST   REC_NUMBER = R8;
  .
  .
  .
INC     REC_NUMBER, 1;   % REC_NUMBER represents the value R8
```

Note that the value assigned to a symbolic constant can be any valid operand value, including an operand in any addressing mode, a flag, a condition code, an interrupt, a control register, an address constant, an arithmetic or string expression, or a list. A symbolic constant can be redefined or reassigned during assembly, although a warning message will be produced. The declared value can be overridden when assembler is invoked (see chapter 1).

2-34. OBJECT VARIABLES

An object variable is an identifier that represents a variable value during the assembly process. Object variables are declared with the VAR declaration (described in chapter 3).

When an object variable appears as an operand, the value of the object variable is used as the operand. Unlike a symbolic constant, an object variable is initially undefined and can be defined or redefined during program assembly. For example:

```
VAR     AR: OBJECT;       % AR is initially undefined
AR ::=  R4;               % AR is defined as R4
:
:
AR ::=  124;              % AR is redefined as 124
LD      R1, AR;           % loads R1 with 124, the current
                          % operand value of AR
```

The values that can be assigned to object variables are the same operand values that can be assigned to symbolic constants. The significant difference is that object variables can be redefined.

CHAPTER 3
DIRECTIVES

The MACRO8000 directives provide a framework within which the instructions are placed as appropriate. This chapter describes the basic directives. The MACRO directive and macro definitions are described in chapter 4. Directives associated with modular programs are described in chapter 5.

3-1. PROGRAM DIRECTIVE AND END

The PROGRAM directive is used to generate absolute code. This directive can be used only with AmZ8002 targeted programs. The PROGRAM directive has the form:

```
PROGRAM lab;
```

where lab is a label that specifies the entry point of the program.

The program begins with the PROGRAM directive, and END. is required at the end of the program. For example:

```
        PROGRAM START;
        .
        .
        .
START:                  % program entry point
        .
        .
        .
        END.
```

The entry point specified on the PROGRAM statement is used as the title at the top of each listing page. The title can be changed with the TITLE directive described later in this chapter.

Appearance of a PROGRAM directive indicates absolute code. The assembler can produce a binary file suitable for downloading. See the B option discussion in section 1-2, Invoking the Macroassembler. When a hex object file is produced, the address of the entry point is written at the end of the file as in Intel hex format.

Appearance of a MODULE directive would generate relocatable code for a module intended for input to LINK8000, as described in chapter 5.

3-2. ORIGIN DIRECTIVE

The ORIGIN directive can be used to set the location counter to a new value. The ORIGIN directive has the form:

```
ORIGIN  origin;
```

where origin is a constant, a numeric expression, the location counter, or a previously-defined label or address constant

Note that $ represents the current value of the location counter. For example:

```
ORIGIN  #4200;          % sets the origin at #4200

ORIGIN  #5000 + #300;   % sets the origin at #5300

ORIGIN  $(64);          % sets the origin at the present origin
                        % plus displacement 64 bytes

ORIGIN  ^MSG5;          % sets the origin at previously-defined
                        % label MSG5

ORIGIN  ^YZ + 64;       % sets the origin at the address of the
                        % previously-defined label YZ plus 64
                        % bytes
```

3-3. BYTE DIRECTIVE

The BYTE directive reserves or defines one or more bytes. The BYTE directive has one of two forms:

```
BYTE    (n);

BYTE:   value,... value;
```

where n is a constant or numeric expression for the number of bytes to be reserved

where each value is a constant, a numeric expression, or a string expression. One or more values can be specified, separated by commas.

The first form of the BYTE directive reserves successive memory locations beginning with the current location counter. The second form reserves memory locations that are defined with the specified values.

A numeric expression evaluates to a 32-bit signed value. A string expression evaluates to a sequence of bytes, one for each character. For example:

```
BYTE    (3);            % reserves 3 bytes
```

```
BYTE:   5;              % defines 1 byte

BYTE:   'STRING';       % defines 6 bytes

BYTE:   3,'AB',4;       % defines 4 bytes
```

A symbolic constant or an object variable can be used to define bytes. For example:

```
CONST   K = 'P';
BYTE:   K & 'AX';       % defines 3 bytes with value 'PAX'
```

3-4. WORD DIRECTIVE

The WORD directive reserves or defines one or more 16-bit words. The WORD directive has one of two forms:

```
WORD    (n);

WORD:   value,... value;
```

where n is a constant or numeric expression for the number of words to be reserved

where each value is a constant, a numeric expression, a string expression, a non-segmented label or effective address or the difference between two effective addresses. One or more values can be specified, separated by commas.

The WORD directive is similar to the BYTE directive, except that words rather than bytes are reserved or defined. For example:

```
WORD    (8);            % reserves 8 words

WORD:   'VAL';          % defines 2 words with values 'VA','L '

WORD:   ^MSG2;          % defines 1 word with a value that is the
                        % address of label MSG2

WORD:   ^RC - ^TH;      % defines 1 word that is the difference
                        % between the addresses of RC and TH
                        % (these addresses must be within the same
                        % segment)

CONST   CRLF = #0D0A;
WORD:   CRLF;           % defines 1 word with value #0D0A
```

NOTE

If the location counter is at an odd byte address, a WORD directive forces the location counter to the next even address.

3-5. LONG DIRECTIVE

The LONG directive reserves or defines one or more long words. A long word is a 32-bit word pair. The LONG directive has one of two forms:

```
LONG    (n);

LONG:   value,... value;
```

where n is a constant or numeric expression for the number of long words to be reserved

where each value is a constant, numeric expression, string expression, or segmented address. One or more values can be specified, separated by commas.

The LONG directive is similar to the BYTE directive, except that word pairs rather than bytes are reserved or defined. For example:

```
LONG    (6);              % reserves 6 long words

LONG:   'BR';             % defines 1 long word with value 'BR  '

LONG:   'ABCDEFGHI';      % defines 3 long words with values
                          % 'ABCD','EFGH','I---'

LONG:   ^MSG2             % where MSG2 is a segmented address

CONST   Y = #50;
LONG:   (Y SHL 16) OR Y; % defines 1 long word with value #00500050
```

NOTE

If the location counter is at an odd byte address, a LONG directive forces the location counter to the next even address.

3-6. STRING DIRECTIVE

The STRING directive defines a string and saves a 1-byte prefix containing the length of the string. The STRING directive has the form:

```
STRING: value,... value;
```

where each value is a string expression, a constant, or a numeric expression. One or more values can be specified, separated by commas.

The STRING directive saves the total length of the string as the first byte. The length of the string can be from 0 to 254 characters. For example:

```
STRING: 'ENTER CHOICE';  % saves length as 12, followed by the
                         % 12 characters in the string

STRING: 'DONE',#0D,#0A;  % saves the length as 6, followed by
                         % the 4 characters in the string and
                         % the carriage return/line feed

STRING: '';              % saves the length as 0, defining
                         % only 1 byte
```

3-7. CONST DIRECTIVE

The CONST directive declares a symbolic constant. A symbolic constant is an identifier that represents a constant value. The CONST directive has the form:

```
CONST   name = value,... name = value;
```

where each name = value specification declares the name of a symbolic constant and defines the value. One or more name = value assignments can be written, with commas separating the assignments.

The value assigned to a symbolic constant can be any valid operand value, including an operand in any addressing mode, a flag, a condition code, an interrupt, a control register, the location counter, an address constant, a string, or a list. The operands are described in chapter 2.

NOTE

A symbolic constant must be defined before being referenced.

When a symbolic constant is used, the value replaces the name. For example:

```
CONST   LINE_SIZE = 80,
        BUF_SIZE = 10 * LINE_SIZE,
        COUNT = R11;
:
:
LD      R13,LINE_SIZE;   % 80 is used for LINE_SIZE
LD      R14,BUF_SIZE;    % 800 is used for BUF_SIZE
INC     COUNT, 2;        % R11 is used as the register
```

The value of a symbolic constant is set in the declaration and cannot be changed later in the program. Note that for a single assembly, any symbolic constant can be overridden with a new value. The constant overrides can be placed on the MACZ product call, as described in chapter 1.

3-8. VAR DIRECTIVE

The VAR directive declares an object variable. The VAR directive has the form:

```
VAR     name,... name:  OBJECT;
```

where each name is an identifier. One of more names can be specified, separated by commas.

An object variable is an identifier that represents an operand value during the assembly process. The declaration of an object variable only declares the name and does not define the object variable. For example:

```
VAR     AR: OBJECT;
```

NOTE

An object variable must be declared before being referenced. At the time the object variable is referenced, it might be undefined or defined.

An object variable that has been declared can be defined with a statement of the form:

```
var ::=  value;
```

Whenever an object variable is used, the variable is replaced by the value that the variable represents. For example:

```
AR ::=  R4;             % defines value of AR
LD      AR, 3;          % identical to LD  R4, 3;
```

Unlike a symbolic constant, an object variable can be redefined with a different value. For example:

```
AR ::=  5;              % redefines object variable AR
LD      R0, AR;         % identical to LD  R0, 5;
```

Any variable declared with the VAR directive is initialized to the value NIL. NIL is a special identifier that indicates an undefined state. The operator NULL can be used to test for the special value NIL. See the IF directive later in this chapter.

All macro parameters are automatically treated as object variables within the macro. Macro parameters are discussed in chapter 4.

An object variable with the value NIL can be used as a label and causes definition of an assembler-generated label. Subsequent references to the object variable are replaced by the assembler-generated label, unless the object variable is redefined or reset to NIL. In order to use labels inside macros (see chapter 4), declare them as object variables.

3-9. IF DIRECTIVE

The IF directive can be used for conditional assembly. In conditional assembly, statements are assembled or not assembled depending on a particular condition.

The IF directive can also be effective at run time. If a particular condition cannot be evaluated at assembly time, the condition must be tested at run time. Additional code is generated by the assembler to support testing of the condition at run time.

The IF directive has one of two forms IF-THEN or IF-THEN-ELSE:

```
IF test
    THEN statement1;

IF test
    THEN statement1
    ELSE statement2;
```

where test is the condition test. The nature of the test determines whether the IF is for assembly time or run time.

where statement1 for the THEN part is a single or compound statement to be assembled or run if the test is true.

and where statement2 for the optional ELSE part is a single or compound statement to be assembled or run if the test is false.

An assembly time IF test can be:

1. The condition code TRUE or FALSE. For example:

```
CONST   SWITCH = TRUE;
.
.
.
IF SWITCH
    THEN
    .
    .
    .
```

2. An expression with the NULL operator and an object variable. The NULL test is true if the object variable has the special value NIL, which indicates an undefined state. For example:

```
VAR     URT: OBJECT;
.
.
.
IF NULL URT
    THEN
    .
    .
    .
```

3. An arithmetic comparison. The first argument is a numeric value or subexpression result. The operator is EQ, NE, a signed comparison operator (LT, LE, GT, GE), or an unsigned comparison operator (LLT, LLE, LGT, LGE). The second argument is a numeric value, a subexpression result, or a string of 4 characters or fewer. A string of 1 to 3 characters is right-justified and zero-filled before being used as a numeric value. For example:

```
CONST   BLOCK = 4;
:
:
IF BLOCK GT 2
    THEN
    :
    :
```

4. A string comparison. The first argument is a string or string expression. The operator is EQ, NE, or an unsigned comparison operator (LLT, LLE, LGT, LGE). The second argument is a string or string expression. The EQ and NE operators test for the same length and character value. The unsigned comparison operators test by ASCII collating sequence. For example:

```
CONST   DRIVE = 'B:';
:
:
IF DRIVE EQ 'A:'
    THEN
    :
    :
```

5. A logical operation with NOT. The operator NOT can be applied to the tests 1 through 4. For example:

```
IF NOT NULL URT
    THEN
    :
    :
```

6. A logical comparison with AND or OR. Any 2 of the tests 1 through 5 can be combined with the AND or OR operator. For example:

```
IF NOT NULL URT AND SWITCH
    THEN
    :
    :
```

A run time IF test generates additional AmZ8000 code that consists of CP (or OR) and JR instructions. The T (trace) option on the MACZ product call causes display of trace messages that identify the additional generated instructions. A run time IF test can be:

1. An AmZ8000 condition code. For example:

```
IF NOV
    THEN
    :
    :
```

2. An AmZ8000 condition code preceded by NOT. For example:

```
IF NOT CY
    THEN
    :
    :
```

3. A register comparison. The first argument is a register. The operator is EQ, NE, a signed comparison operator (LT, LE, GT, GE), or an unsigned comparison operator (LLT, LLE, LGT, LGE). The second argument is another register or a numeric expression. For example:

```
IF R3 GT 0
    THEN
    :
    :
```

The entire IF directive ends with a semicolon. No semicolons are used within an IF directive that contains single statements. For example:

```
IF NULL X
    THEN
        LD      R2, 0
    ELSE
        LD      R2, X;
```

Semicolons can occur within compound statements. Note that the last single statement has no semicolon. For example:

```
IF R12 LT 4
    THEN
        BEGIN
        CALL    XCALC;
        CALL    SHIFT
        END
    ELSE
        BEGIN
        LD      R10, R12;
        LD      R12, 14;
        CALL    SHIFT
        END;
```

IF directives can be nested to one or more levels. For example:

```
IF SWITCH
    THEN
        IF R12 LT 4
            THEN
            :
            :
```

3-10. FOR DIRECTIVE

The FOR directive is used for repetitive assembly, where statements are reassembled repeatedly in a specific way. The FOR directive has one of three forms:

```
FOR exp DO
        statement;

FOR var IN list DO
        statement;

FOR var IN string DO
        statement;
```

where exp is a constant or numeric expression for the number of times assembly is to be repeated.

where var is an object variable (or macro parameter).

where list is a list value.

where string is a string or string expression.

and where statement is a single or compound statement to be assembled repeatedly.

When the first form is used, assembly is repeated a specified number of times. For example:

```
FOR 12 DO
        BYTE:   0;
```

is equivalent to:

```
BYTE:   0,0,0,0,0,0,0,0,0,0,0,0;
```

When the second form is used, assembly is repeated as many times as var can be set to successive items in list. For example:

```
VAR     Y: OBJECT;
:
FOR Y IN (1,2,3,4,5,6) DO
        BYTE:   Y;
```

is equivalent to:

```
BYTE:   1,2,3,4,5,6;
```

When the third form is used, assembly is repeated as many times as var can be set to successive characters in a string expression. For example:

```
VAR     Z: OBJECT;
:
FOR Z IN 'ABCDEF' DO
        BYTE:   Z;
```

is equivalent to:

```
BYTE:   'A','B','C','D','E','F';
```

The statement used in the FOR directive can be a compound statement, as described in chapter 2. For example:

```
VAR     X: OBJECT;
X ::=   0;
FOR 256 DO
        BEGIN
        BYTE:   X;
        X ::=   X + 1
        END;
```

reserves a total of 256 bytes containing consecutive values from 0 through 255.

The FOR directive can be nested to one or more levels. For example:

```
FOR 10 DO
        BEGIN
        BYTE:   0;
        FOR 7 DO
                BYTE:  #40
        END;
```

defines a total of 80 bytes, with each group of 8 consisting of one byte with the value 0 and seven bytes with the value #40.

3-11. TITLE DIRECTIVE

The program name is automatically used as the title at the top of each listing page in the .PRN file. The TITLE directive is used to change the title and has the form:

```
TITLE   string;
```

where string is a string or string expression.

For example:

```
TITLE   'SIZE TEST';
```

3-12. PAGE DIRECTIVE

The PAGE directive sets the size of each listing page in the .PRN file and has the form:

```
PAGE    exp;
```

where exp is a constant or numeric expression for the number of source lines per listing page.

The default is 45 source lines per page, and the assembler inserts a form feed (ASCII 0C) into the listing after each group of 45 lines. The page size must be in the range 10 to 255. For example:

```
PAGE    48;
```

3-13. EJECT DIRECTIVE

The EJECT directive causes a page eject in the .PRN file and has the form:

```
EJECT;
```

The EJECT directive resets the line count for the PAGE directive to 0. EJECT is often used after each major part of a program.

3-14. INCLUDE DIRECTIVE

The INCLUDE directive specifies a file containing additional source text for the program. The contents of the file are included in the program, replacing the INCLUDE directive. The INCLUDE directive has the form:

```
INCLUDE filename;
```

where filename is any string or string expression that specifies the name of the file to be included.

The default drive designator is the same as the drive for the source file. The file name and file type (extension) must be specified.

For example:

```
INCLUDE 'ACE.ZSC';
```

includes the contents of the file ACE.ZSC in the program at the place where the INCLUDE directive appears.

The INCLUDE directive can be nested to one to three levels. A file that is being included can contain INCLUDE directives. The INCLUDE directives are processed in the order in which they are encountered.

3-15. NOLIST DIRECTIVE

The NOLIST directive suppresses listing of the program. Program listing is suppressed from the NOLIST directive to the end of the program, or from the NOLIST directive to the appearance of a LIST directive. The NOLIST directive has the form:

```
NOLIST;
```

3-16. LIST DIRECTIVE

The LIST directive reactivates listing of the program. Program listing is activated until the end of the program, or until another NOLIST directive is encountered. The LIST directive has the form:

```
LIST;
```

CHAPTER 4
MACRO USAGE

A macro is defined once and then called at various times during program assembly. Once a macro has been defined, a macro reference causes assembly of the code or data values defined by the macro. A macro is therefore like a subroutine, except that a macro is effective at assembly time rather than execution time.

Macro calls can cause assembly of the same code or data values at different places in the program. Like subroutines, macros can have parameters. If the macro has been defined with parameters, arguments can be passed to the macro when the macro is referenced. Therefore, macro calls can also cause assembly of different code or data values, depending on the arguments passed to the macro.

Macros in MACRO8000 are treated as syntax macros rather than lexical macros. When a macro definition is encountered during assembly, the assembler processes statements in the macro to produce an intermediate, internal representation. Later, when the macro is referenced, the assembler expands the macro from the already-processed intermediate form. This type of macro processing results in quick assembly of macros.

4-1. MACRO DIRECTIVE

The MACRO directive declares a macro and takes the form:

MACRO macro parameters;

where macro is the macro name

and where parameters is an optional list of macro parameter names. One or more parameters can appear, separated by commas.

The MACRO directive is the first statement, and the rest of the macro has the following structure:

Any CONST directives for symbolic constants defined in the macro

Any VAR directives for object variables defined in the macro

A compound statement (BEGIN through END) that is the body of the macro. The compound statement can include instructions, directives, and other compound statements, as well as references to other macros.

For example:

```
MACRO   FILL;            % defines 40 bytes that are spaces
        BEGIN
        FOR 40 DO
                BYTE:   #20
        END;
```

A macro reference for the FILL macro would be:

```
        FILL;
```

Note that the code or data values generated by the macro appear in the listing when the macro is referenced, not when the macro is defined.

NOTE

> Any macro referenced in the program must have been defined earlier in the program. In the same way, any macro referenced from within a macro must also have been defined earlier in the program.

4-2. MACRO PARAMETERS

All macro parameters listed on the MACRO directive are treated as object variables within the macro. VAR directives are not necessary and are not used for the macro parameters. Note that an object variable can have any valid operand value. A macro parameter can therefore have a value that is a constant, a numeric expression, an operand in any addressing mode, a flag, a condition code, and so forth.

Since the parameters are object variables, each parameter is defined as the corresponding argument passed to the macro. For example:

```
MACRO   PRT_HEX BYTE;
        CONST   FOUR = 4;
        BEGIN
        LDB     RL3, BYTE;
        CALL    HEXVERT;
        SRLB    RL3, FOUR;
        CALL    HEXVERT
        END;
```

can be followed by the macro reference:

```
        PRT_HEX RH6;
```

which passes the value RH6 as an argument to the macro. Within the macro, RH6 is used as the value of BYTE. The macro reference:

```
        PRT_HEX #41;
```

would pass a different value as an argument and result in the generation of different code, using #41 as the value of BYTE.

An argument can be supplied for each parameter, or some arguments can be omitted. Since parameters are treated as object variables, the NULL operator can be used to test for defined parameters. For example:

```
MACRO   XTR     A, B, C;
        BEGIN
        BYTE:   A;
        IF NOT NULL B THEN
        BYTE:   B;
        IF NOT NULL C THEN
        LONG:   C
        END;
```

is defined with 3 parameters. The macro reference:

```
        XTR     4;
```

defines one byte, while the macro reference:

```
        XTR     0, 1, 2;
```

defines two bytes and one long word by supplying values for parameters A, B, and C. If an argument that is supplied follows an omitted argument, then the position of the omitted argument must be preserved. For instance:

```
        XTR     #5050, ,#2020:
```

defines one byte and one long word by supplying values only for parameters A and C.

A particularly powerful feature is the ability to pass a list to a macro, so that the list can be used in a FOR directive. For example:

```
MACRO   ZERO    REG_LIST;
        VAR     X: OBJECT;
        BEGIN
        FOR X IN REG_LIST DO
        LD      X, 0
        END;
        .
        .
        .
        ZERO    (R5);     % sets one register to zero
        .
        .
        .
        ZERO    (R0, R1, R2, R3, R4, R5, R12, R13);
                          % sets all eight registers to zero
```

CHAPTER 5
MODULES AND SEGMENTS

5-1. PROGRAM SEGMENTS

An AmZ8000 source program can be constructed out of or more modules, each contained in a separate source file. The module is the smallest programming unit that can be assembled separately. Programmers can subdivide modules into program segments; for example, a module will be partitioned into a code segment and a data segment. Segments cannot be assembled separately; they are simply used to partition modules. However, once several modules have been assembled, each containing several segments, LINK8000 can be used to rearrange and combine the segments in an arbitrary manner. A segment is thus the smallest programming unit that can be manipulated by the linker. Consult the LINK8000 User's Manual for more details.

This chapter describes the directives used to construct modules and program segments. The concept of a program segment must be clearly distinguished from the concept of a segmented address. A program segment is a software concept; a segmented address is an architectural concept. Program segments can be defined for programs that will be executed on the AmZ8002 processor, but this processor cannot use segmented addresses.

5-2. SEGMENTED ADDRESSES

The AmZ8001 processor always generates two component memory address references. The first component, a seven-bit segment number, is generated on lines SN_0-SN_6 (see the AmZ8000 Family Data Book). The second component, a 15-bit offset, is generated on lines AD_0-AD_{15}. These two component segmented addresses are stored in two words of memory (32-bits) or in a register pair. The storage format of a segmented address is shown in Figure 5-1.

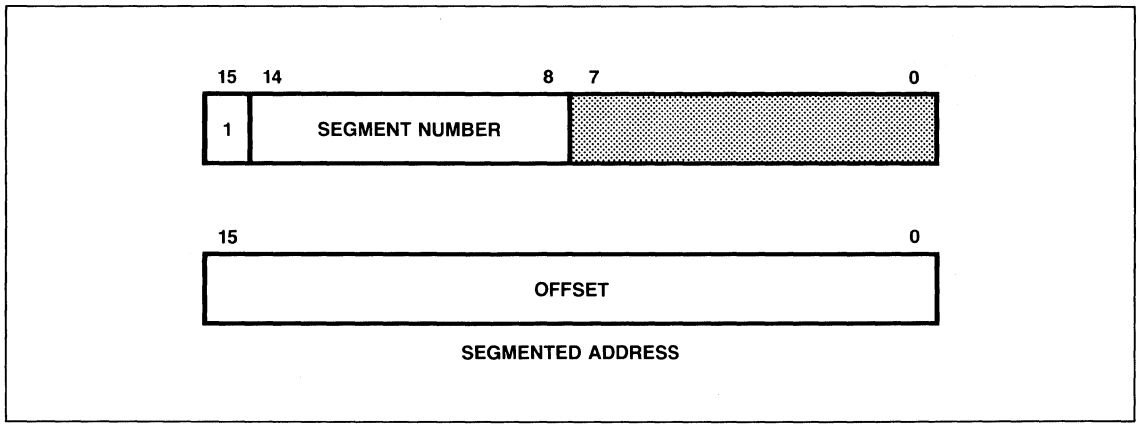


Figure 5-1. Segmented Address

The AmZ8002 processor, which lacks lines SN_0-SN_6. generates one component non-segmented addresses that are 15-bits long. These non-segmented addresses are stored in one word of memory or in a word register (see figure 5-2).

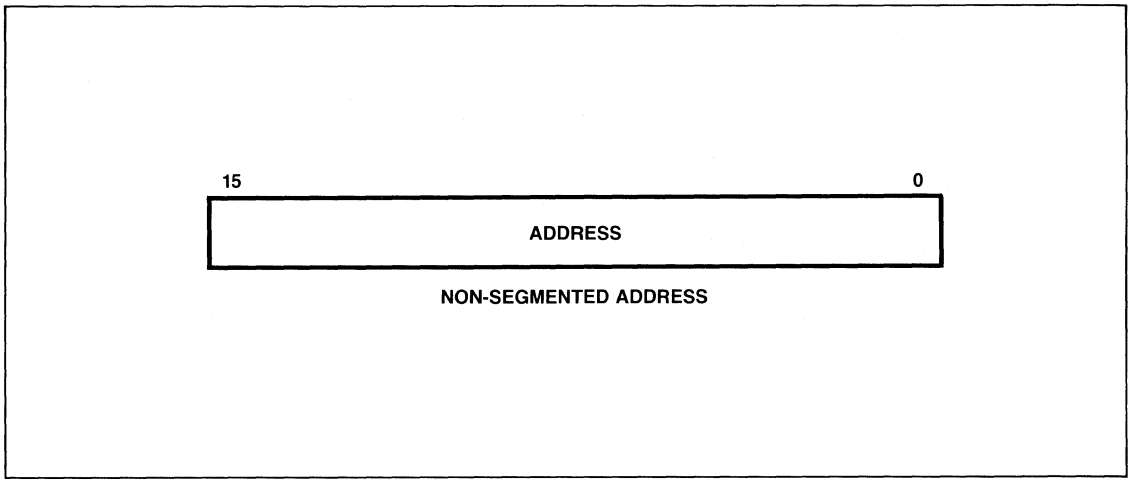


Figure 5-2. Non-Segmented Address

The address space of the AmZ8002 is thus a single, 64K linear space. The AmZ8001 address space, on the other hand, consists of 2^7=128 separate 64K linear address spaces, which we will call hardware segments (to distinguish them from program segments). The linker directive SETLSEG can be used to assign program segments to different AmZ8001 hardware segments. In the AmZ8002, all program segments are put into the same 64K address space. Read chapter 3 of the LINK8000 User's Manual for more information.

Two features of the AmZ8001, non-segmented mode and the short-offset format, give users the option of storing addresses in 15-bit words instead of 32-bit word pairs. Both these features are supported by MACRO8000.

The segmentation mode is controlled by a bit in the FCW (see the AmZ8000 Data Book). When this bit is set, the AmZ8001 expects all addresses to be in the two-word format shown in figure 5-1. When this bit is cleared, the AmZ8001 is said to be in non-segmented mode; the segment number bits in the PC are frozen and cannot be altered; and the processor expects addresses to be in the 15-bit format shown in figure 5-2. The AmZ8001 still generates segmented addresses when it is in non-segmented mode, but the SN_0-SN_6 lines cannot be changed by the programmer. MACRO8000 supports non-segmented mode through the directives MOD_NS and MOD_SEG, which are discussed in section 5-11.

The short offset format of segmented addresses is shown in figure 5-3. A short offset address contains a full 7-bit segment number, but only an 8-bit offset. Consequently, short offset addresses can access only the first 2^8=256 locations in each segment. These first 256 locations are called page zero. The processor distinguishes short format addresses from long format addresses by examining bit 15 in the first word of the address. If that bit is 1, the address has two

words; if the bit is 0, the address is a 15-bit short format address. The assembler directives PAGE0, EXT_PG0, and GLB_PG0, which are discussed in section 5-12, enable the user to specify the address format.

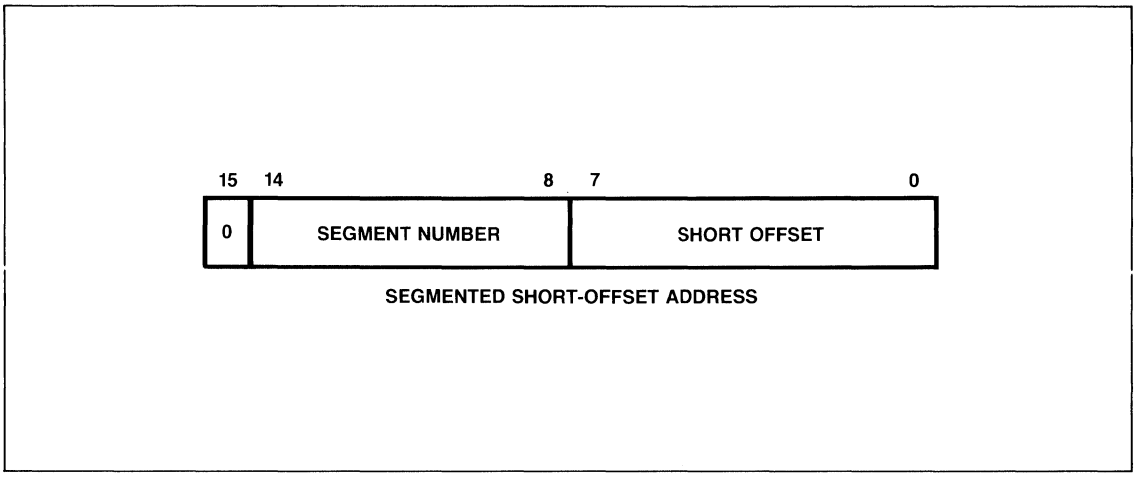


Figure 5-3. Segmented Short-Offset Address

5-3. MODULE DIRECTIVE AND END

The MODULE directive is required to establish the name of a module. A source file contains a single module, and a modular program consists of one or more separately assembled modules. The MODULE directive has the form:

```
MODULE  string;
```

where string is a string or string expression that specifies the module name.

Each module begins with a MODULE directive, and END. is required at the end of the module. For example:

```
MODULE  'THIS ONE';
:
:
END.
```

The module name is used as the title at the top of each listing page. The title can be changed with the TITLE directive (see chapter 3).

5-4. HEADER DIRECTIVE

The HEADER directive supplies one or more header lines at the beginning of the relocatable object file for each module. The HEADER directive has the form:

```
HEADER  string,... string;
```

where each string is a string or string expression that produces one identification line in the .ZRL file. One or more identification lines can be specified.

The user can supply date and time information, for instance, at the beginning of each .ZRL file. When a .ZRL file is listed at the console, the header lines provide for identification of the file. The other information in the relocatable file is specially coded for the linker LINK8000 and is not readable.

5-5. SEGMENT DIRECTIVE

The SEGMENT directive defines a program segment that can be manipulated by the linker LINK8000. The assigning program segments to AmZ8001 hardware segments is accomplished by linker (see the LINK8000 User's Manual). The SEGMENT directive has one of the forms:

```
SEGMENT string;

SEGMENT [attr], string;

SEGMENT @PRIOR;
```

where string is a string or string expression for the segment name.

where attr is an optional segment attribute @COM for common.

and where @PRIOR indicates a reset to the segment previously defined.

If the segment name is supplied but no segment attribute is specified, LINK8000 directives later determine where the segment is placed.

The attribute associated with the named segment cannot be changed within the program.

At the beginning of a new segment, the assembler automatically resets the location counter to 0. The location counter can be changed with the ORIGIN directive described in chapter 3.

Each segment is known by its name. Any attribute declared for a segment continues to be associated with the segment. Therefore, a @COM attribute need not be repeated.

A module that contains no segment definitions is treated as a single segment with no attributes and with the same name as the module.

5-6. COMMON SEGMENT

A segment with attribute @COM is a relocatable common segment that can be used for code and/or data. The location counter is set to 0 for the first SEGMENT directive that names a common segment. The location counter can be changed with an ORIGIN directive.

During assembly, common segments look quite similar to ordinary segments. Subsequent directives for the same common segment in different modules refer, at link time, to the same space. Common segments in different modules, to be linked, must be the same size.

Multiple SEGMENT directives can name the same common segment. The first directive defines the common segment, and subsequent directives refer to the same space. Subsequent SEGMENT directives can extend the common segment within the same module. For example:

```
MODULE  'M4';
.
.
.
SEGMENT [@COM], 'C1';     % starts common segment
.
.
.
SEGMENT 'OTHER';          % start or extend another segment
.
.
.
SEGMENT [@COM], 'C1';     % extends common segment
.
.
.
END.
```

The size of a common segment is defined to be the largest value of the location number at the end of the segment.

5-7. PRIOR SEGMENT

If @PRIOR is used in the SEGMENT directive, the directive is a reset to the prior segment. For example:

```
MODULE  'M6';
.
.
.
SEGMENT 'X1';             % defines segment X1 with no attribute
.
.
.
SEGMENT [@COM], 'COMMON';
.
.
.
SEGMENT @PRIOR;           % continues segment X1
```

If no prior segment exists, SEGMENT @PRIOR generates an informative error and the current segment is used.

5-8. GLOBAL DIRECTIVE

Communication between modules involves the use of external labels. The GLOBAL and EXTERNAL directives are involved. The GLOBAL directive has the form:

```
GLOBAL  label,... label;
```

where each label is a label declared as global. One or more labels can be specified, separated by commas.

Each label declared GLOBAL defines an address that is known in other modules when those modules declare the same name as external. That is, the GLOBAL directive declares a label as global in scope.

5-9. EXTERNAL DIRECTIVE

The EXTERNAL directive declares a label as external to the module. The label declared external is associated at link time with a label declared as global in some other module. The EXTERNAL directive takes the form:

```
EXTERNAL label,... label;
```

where label is a label declared as global in another module. One or more labels can be specified, separated by commas.

For example, one assembly might be for the following module:

```
        MODULE   'Y2';
        GLOBAL   D4, D5;
        .
        .
        .
D4:
        .
        .
        .
D5:
        .
        .
        .
        END.
```

and another assembly might be for the following module:

```
MODULE   'Y';
EXTERNAL D4, D5;
.
.
.
END.
```

Another symbol can be used to declare labels external to the module: the double pound-sign (##) used as a suffix. When applied to a label, this symbol converts the label into an external reference (assuming it has not already been so identified). For example:

```
CALL EXP##;
```

is equivalent to

```
EXTERNAL EXP;
CALL EXP;
```

5-10. SEGMENTED ADDRESS GENERATION

When the assembler is invoked with the S option, the code produced is targeted for the AmZ8001. When the assembler is invoked without the S option, the code is targeted for the AmZ8002. All code produced for the AmZ8002 uses non-segmented addresses, but code produced for the AmZ8001 can use either non-segmented or segmented addresses, or both types. If the assembler is invoked with the S option, it will produce segmented addresses until it encounters the directive

```
MOD_NS;                    % activate non-segmented code generation
```

whereupon the assembler will switch to producing non-segmented addresses. (Obviously, if this directive occurs early in the module, the entire module can be made non-segmented.)

A second directive can be used to switch the assembler back to producing segmented addresses:

```
MOD_SEG;                   % activate segmented code generation
```

Any combination of the two directives can appear in one program module that is targeted for an AmZ8001.

It is important to realize that the two directives change only the code produced by the assembler, they are not AmZ8001 instructions and have no effect on the segmentation mode of the processor. The AmZ8001 must have the segmentation bit in the FCW set before it can run segmented code, and this bit must be cleared before it can run non-segmented code. Consequently, the programmer must include instructions to enable or disable this bit whenever a change is made from segmented code to non-segmented code or vice versa. See the AmZ8001 Instruction Set Manual for more details on the segmentation bit and the instructions that manipulate it.

5-11. PAGE 0 DIRECTIVES

As discussed at the beginning of this chapter, the assembler can generate segmented addresses in two formats: long format and short format. By default, long format addresses are generated. The programmer can specify the short address format for labels by using the directives

```
PAGE0     EXT_PG0   GLB_PG0
```

Each of these directives can be followed by a list of labels, which are thus identified to the assembler as requiring short-form addresses. The directive must appear before the label is used in an instruction. When the assembler encounters the label in an instruction, it will automatically generate a short-form address. Here are some examples:

```
PAGE0  LAB, TAB;          % declare LAB and TAB to be (internal)
                          % short-address labels

EXT_PG0  START;           % declare START to be an external short-
                          % address label

GLB_PG0  SORT;            % declare SORT to be a global short-
                          % address label
   .
   .
   .
```

Short-format addresses are restricted to the first 256 bytes in an AmZ8001 hardware segment. It is the responsibility of the programmer to ensure that the labels do in fact refer to addresses in this 255-byte page. At link time, the linker will generate an error message if the label address exceeds 255.

The directive PAGE0 can also be used without a list of labels. Used this way, the directive instructs the assembler to generate short-form addresses for all subsequently defined (or used) labels, with one exception; external labels will remain long-format unless they are explicitly changed with an EXT_PG0 directive.

CHAPTER 6
INSTRUCTIONS

The instructions used in a MACRO8000 program are the AmZ8000 instructions. The AmZ8000 processor is described in the AmZ8000 Family Reference Manuals available from Advanced Micro Devices, Inc. In particular, the instruction set is described in the AmZ8001/2 Processor Instruction Set manual. The instruction set manual contains detailed information such as instruction execution times and instruction formats.

MACRO8000 generates codes for the instructions that are written in the program, and reserves or defines space for any data values specified in the program. Currently, the only additional code generated by MACRO8000 are the extra CP, OR, and JR instructions involved in the run time form of the IF directive.

As described in chapter 2, an AmZ8000 instruction has the general form:

```
mnemonic  operands;
```

Each instruction requires a specific number of operands. From 0 through 4 operands are required, depending on the instruction. Each instruction generates from 1 to 5 words of assembled code.

In the instruction descriptions in this chapter, certain notations are used to show the type of operand required for the instruction:

Notation	Meaning
dst	Destination operand supplied by the user in one of the addressing modes listed.
src	Source operand supplied by the user in one of the addressing modes listed (when used in program flow instructions i.e. jumps and calls; refers to the destination address).
im	Immediate operand that is a constant, or an expression evaluated at assembly time.
r	Register operand that is a byte register, word register, register pair, or quad register, depending on the instruction.
ir	Indirect register operand that is a word register or register pair, depending on whether code is assembled for the nonsegmented AmZ8002 or segmented AmZ8001 processor.
exp	A constant, or an expression evaluated at assembly time.

cc	AmZ8000 condition code
SP	Stack pointer
PC	Program counter
PS	Program status

The notations used in instruction descriptions and throughout this manual are defined more precisely in appendix C.

Two assemblies are shown in this chapter. One shows all the AmZ8002 instructions, while the other shows the AmZ8001 instructions (including short offset format). The two lists are presented in parallel, with the AmZ8002 version at the top of the right-hand pages and the AmZ8001 version at the bottom of the right-hand pages. The left hand pages contain the corresponding descriptions.

```
0000
0000                    PROGRAM INSTRUCTIONS;
0000
0000                    PAGE 35;
0000
0000                    % THIS PROGRAM ASSEMBLES THE FULL
0000                    % SET OF AMZ8000 INSTRUCTIONS,
0000                    % USING EACH POSSIBLE OPCODE AND
0000                    % ADDRESSING MODE COMBINATION.
0000                    % MORE THAN 400 COMBINATIONS EXIST.
0000                    %
0000                    % THE POSSIBLE ADDRESSING MODES
0000                    % FOR A GIVEN INSTRUCTION ARE
0000                    % SHOWN IN ORDER AND WITH
0000                    % CONSISTENT VALUES FOR THE
0000                    % PURPOSE OF THESE EXAMPLES.
0000                    % THE VALUES ARE:
0000                    %
0000                    % IMMEDIATE (IM)          5
0000                    %
0000                    % REGISTER (R)            RH4,
0000                    %                         R4,
0000                    %                         RR4
0000                    %
0000                    % INDIRECT REGISTER (IR)  R2^
```

```
0000
0000                    MODULE "INSTRUCTIONS";
0000
0000                    PAGE 35;
0000
0000                    % THIS PROGRAM ASSEMBLES THE FULL
0000                    % SET OF AMZ8001 INSTRUCTIONS,
0000                    % USING EACH POSSIBLE OPCODE AND
0000                    % ADDRESSING MODE COMBINATION.
0000                    % MORE THAN 500 COMBINATIONS EXIST.
0000                    %
0000                    % THE POSSIBLE ADDRESSING MODES
0000                    % FOR A GIVEN INSTRUCTION ARE
0000                    % SHOWN IN ORDER AND WITH
0000                    % CONSISTENT VALUES FOR THE
0000                    % PURPOSE OF THESE EXAMPLES.
0000                    % THE VALUES ARE:
0000                    %
0000                    % IMMEDIATE (IM)          5
0000                    %
0000                    % REGISTER (R)            RH4,
0000                    %                         R4,
0000                    %                         RR4
0000                    %
0000                    % INDIRECT REGISTER (IR)  RR2^
0000                    %
0000                    % DIRECT ADDRESS (DA)     LAB
0000                    %
0000                    % DIRECT ADDRESS (DA)
0000                    % SEGMENTED SHORT OFFSET  LAB_SSO
```

```
0000                    %
0000                    % DIRECT ADDRESS (DA)      LAB
0000                    %
0000                    % RELATIVE ADDRESS (RA)    LAB2
0000                    %
0000                    % INDEXED (X)              LAB(R1)
0000                    %
0000                    % BASE ADDRESS (BA)        R2^(20)
0000                    %
0000                    % BASE INDEXED (BX)        R2^(R1)
0000                    %
0000                    % PORT ADDRESS (PA)        #0FC0
0000                    %
0000                    % PORT REGISTER (PR)       R13
0000
0000
0000                    ORIGIN  #4200;
4200
4200                    LAB:
4200                    % DEFINED FOR DA AND X OPERANDS
4200
4200                    INSTRUCTIONS:
4200
4200                    EJECT;
```

```
0000                    %
0000                    % RELATIVE ADDRESS (RA)    LAB2
0000                    %
0000                    % INDEXED (X)              LAB(R1)
0000                    %
0000                    % INDEXED (X)
0000                    % SEGMENTED SHORT OFFSET   LAB_SSO(R1)
0000                    %
0000                    % BASE ADDRESS (BA)        RR2^(20)
0000                    %
0000                    % BASE INDEXED (BX)        RR2^(R1)
0000                    %
0000                    % PORT ADDRESS (PA)        #0FC0
0000                    %
0000                    % PORT REGISTER (PR)       R13
0000
0000                    PAGE0   LAB_SSO;
0000
0000                    LAB_SSO:
0000                    % DEFINED FOR SEGMENTED SHORT OFFSET DA AND X OPERANDS
0000
0000                    ORIGIN  $(#1000);
1000
1000                    LAB:
1000                    % DEFINED FOR DA AND X OPERANDS
1000
1000                    INSTRUCTIONS:
1000
1000                    EJECT;
```

6-1. CLEAR, EXCHANGE AND LOAD INSTRUCTIONS

Instruction		Addressing Modes for dst or src	Description
CLRB CLR	dst	R,IR,DA,X	Clear dst <== 0
EXB EX	r,src	R,IR,DA,X	Exchange r <===> src

```
4200                            TITLE   'CLEAR, EXCHANGE, AND LOAD';
4200
4200                            % CLEAR
4200    8C48                    CLRB    RH4;
4202    0C28                    CLRB    R2^;
4204    4C08 4200               CLRB    LAB;
4208    4C18 4200               CLRB    LAB(R1);
420C    8D48                    CLR     R4;
420E    0D28                    CLR     R2^;
4210    4D08 4200               CLR     LAB;
4214    4D18 4200               CLR     LAB(R1);
4218
4218                            % EXCHANGE
4218    AC46                    EXB     RH6,RH4;
421A    2C26                    EXB     RH6,R2^;
421C    6C06 4200               EXB     RH6,LAB;
4220    6C16 4200               EXB     RH6,LAB(R1);
4224    AD46                    EX      R6,R4;
4226    2D26                    EX      R6,R2^;
4228    6D06 4200               EX      R6,LAB;
422C    6D16 4200               EX      R6,LAB(R1);
```

```
1000                            TITLE   'CLEAR, EXCHANGE, AND LOAD';
1000
1000                            % CLEAR
1000    8C48                    CLRB    RH4;
1002    0C28                    CLRB    RR2^;
1004    4C08S0001 1000          CLRB    LAB;
100A    4C08P0100               CLRB    LAB_SSO;
100E    4C18S0001 1000          CLRB    LAB(R1);
1014    4C18P0100               CLRB    LAB_SSO(R1);
1018    8D48                    CLR     R4;
101A    0D28                    CLR     RR2^;
101C    4D08S0001 1000          CLR     LAB;
1022    4D08P0100               CLR     LAB_SSO;
1026    4D18S0001 1000          CLR     LAB(R1);
102C    4D18P0100               CLR     LAB_SSO(R1);
1030
1030                            % EXCHANGE
1030    AC46                    EXB     RH6,RH4;
1032    2C26                    EXB     RH6,RR2^;
1034    6C06S0001 1000          EXB     RH6,LAB;
103A    6C06P0100               EXB     RH6,LAB_SSO;
103E    6C16S0001 1000          EXB     RH6,LAB(R1);
1044    6C16P0100               EXB     RH6,LAB_SSO(R1);
1048    AD46                    EX      R6,R4;
104A    2D26                    EX      R6,RR2^;
104C    6D06S0001 1000          EX      R6,LAB;
1052    6D06P0100               EX      R6,LAB_SSO;
1056    6D16S0001 1000          EX      R6,LAB(R1);
105C    6D16P0100               EX      R6,LAB_SSO(R1);
```

LDB LD LDL	r,src	IM,R,IR,DA,X,BA,BX	Load to Register r <== src

```
4230
4230                            % LOAD TO REGISTER
4230    C605                    LDB     RH6,5;
4232    A046                    LDB     RH6,RH4;
4234    2026                    LDB     RH6,R2^;
4236    6006 4200               LDB     RH6,LAB;
423A    6016 4200               LDB     RH6,LAB(R1);
423E    3026 0014               LDB     RH6,R2^(20);
4242    7026 0100               LDB     RH6,R2^(R1);
4246    2106 0005               LD      R6,5;
424A    A146                    LD      R6,R4;
424C    2126                    LD      R6,R2^;
424E    6106 4200               LD      R6,LAB;
4252    6116 4200               LD      R6,LAB(R1);
4256    3126 0014               LD      R6,R2^(20);
425A    7126 0100               LD      R6,R2^(R1);
425E    1406 0000 0005          LDL     RR6,5;
4264    9446                    LDL     RR6,RR4;
4266    1426                    LDL     RR6,R2^;
4268    5406 4200               LDL     RR6,LAB;
426C    5416 4200               LDL     RR6,LAB(R1);
4270    3526 0014               LDL     RR6,R2^(20);
4274    7526 0100               LDL     RR6,R2^(R1);
4278
```

```
1060
1060                            % LOAD TO REGISTER
1060    C605                    LDB     RH6,5;
1062    A046                    LDB     RH6,RH4;
1064    2026                    LDB     RH6,RR2^;
1066    6006S0001 1000          LDB     RH6,LAB;
106C    6006P0100               LDB     RH6,LAB_SSO;
1070    6016S0001 1000          LDB     RH6,LAB(R1);
1076    6016P0100               LDB     RH6,LAB_SSO(R1);
107A    3026 0014               LDB     RH6,RR2^(20);
107E    7026 0100               LDB     RH6,RR2^(R1);
1082    2106 0005               LD      R6,5;
1086    A146                    LD      R6,R4;
1088    2126                    LD      R6,RR2^;
108A    6106S0001 1000          LD      R6,LAB;
1090    6106P0100               LD      R6,LAB_SSO;
1094    6116S0001 1000          LD      R6,LAB(R1);
109A    6116P0100               LD      R6,LAB_SSO(R1);
109E    3126 0014               LD      R6,RR2^(20);
10A2    7126 0100               LD      R6,RR2^(R1);
10A6    1406 0000 0005          LDL     RR6,5;
10AC    9446                    LDL     RR6,RR4;
10AE    1426                    LDL     RR6,RR2^;
10B0    5406S0001 1000          LDL     RR6,LAB;
10B6    5406P0100               LDL     RR6,LAB_SSO;
10BA    5416S0001 1000          LDL     RR6,LAB(R1);
10C0    5416P0100               LDL     RR6,LAB_SSO(R1);
10C4    3526 0014               LDL     RR6,RR2^(20);
10C8    7526 0100               LDL     RR6,RR2^(R1);
```

LDB LD LDL	dst,r	IR,DA,X,BA,BX	Load to Memory dst <== r

```
4278                        % LOAD TO MEMORY
4278   2E26                 LDB     R2^,RH6;
427A   6E06 4200            LDB     LAB,RH6;
427E   6E16 4200            LDB     LAB(R1),RH6;
4282   3226 0014            LDB     R2^(20),RH6;
4286   7226 0100            LDB     R2^(R1),RH6;
428A   2F26                 LD      R2^,R6;
428C   6F06 4200            LD      LAB,R6;
4290   6F16 4200            LD      LAB(R1),R6;
4294   3326 0014            LD      R2^(20),R6;
4298   7326 0100            LD      R2^(R1),R6;
429C   1D26                 LDL     R2^,RR6;
429E   5D06 4200            LDL     LAB,RR6;
42A2   5D16 4200            LDL     LAB(R1),RR6;
42A6   3726 0014            LDL     R2^(20),RR6;
42AA   7726 0100            LDL     R2^(R1),RR6;
```

```
10CC
10CC                        % LOAD TO MEMORY
10CC   2E26                 LDB     RR2^,RH6;
10CE   6E06S0001 1000       LDB     LAB,RH6;
10D4   6E06P0100            LDB     LAB_SSO,RH6;
10D8   6E16S0001 1000       LDB     LAB(R1),RH6;
10DE   6E16P0100            LDB     LAB_SSO(R1),RH6;
10E2   3226 0014            LDB     RR2^(20),RH6;
10E6   7226 0100            LDB     RR2^(R1),RH6;
10EA   2F26                 LD      RR2^,R6;
10EC   6F06S0001 1000       LD      LAB,R6;
10F2   6F06P0100            LD      LAB_SSO,R6;
10F6   6F16S0001 1000       LD      LAB(R1),R6;
10FC   6F16P0100            LD      LAB_SSO(R1),R6;
1100   3326 0014            LD      RR2^(20),R6;
1104   7326 0100            LD      RR2^(R1),R6;
1108   1D26                 LDL     RR2^,RR6;
110A   5D06S0001 1000       LDL     LAB,RR6;
1110   5D06P0100            LDL     LAB_SSO,RR6;
1114   5D16S0001 1000       LDL     LAB(R1),RR6;
111A   5D16P0100            LDL     LAB_SSO(R1),RR6;
111E   3726 0014            LDL     RR2^(20),RR6;
1122   7726 0100            LDL     RR2^(R1),RR6;
```

LDB LD	dst,im	IR,DA,X	Load Immediate to Memory dst <== im
LD	r,^src	DA,X,BA,BX	Load Address r <== ^src
LDK	dst,exp	R	Load Constant r <== exp (exp is 0 to 15)

```
42AE
42AE                            % LOAD IMMEDIATE TO MEMORY
42AE    0C25 0505               LDB     R2^,5;
42B2    4C05 4200 0505          LDB     LAB,5;
42B8    4C15 4200 0505          LDB     LAB(R1),5;
42BE    0D25 0005               LD      R2^,5;
42C2    4D05 4200 0005          LD      LAB,5;
42C8    4D15 4200 0005          LD      LAB(R1),5;
42CE
42CE                            % LOAD ADDRESS
42CE    210B 4200               LD      R11,^LAB;
42D2    761B 4200               LD      R11,^LAB(R1);
42D6    342B 0014               LD      R11,^(R2^(20));
42DA    742B 0100               LD      R11,^(R2^(R1));
42DF
42DE                            % LOAD CONSTANT
42DE    BD48                    LDK     R4,8;
```

```
1126
1126                            % LOAD IMMEDIATE TO MEMORY
1126    0C25 0505               LDB     RR2^,5;
112A    4C05S0001 1000          LDB     LAB,5;
1130    0505
1132    4C05P0100 0505          LDB     LAB_SSO,5;
1138    4C15S0001 1000          LDB     LAB(R1),5;
113E    0505
1140    4C15P0100 0505          LDB     LAB_SSO(R1),5;
1146    0D25 0005               LD      RR2^,5;
114A    4D05S0001 1000          LD      LAB,5;
1150    0005
1152    4D05P0100 0005          LD      LAB_SSO,5;
1158    4D15S0001 1000          LD      LAB(R1),5;
115E    0005
1160    4D15P0100 0005          LD      LAB_SSO(R1),5;
1166
1166                            % LOAD ADDRESS
1166    140AS0001 1000          LDL     RR10,^LAB;
116C    760AP0100               LDL     RR10,^LAB_SSO;
1170    761AS0001 1000          LDL     RR10,^LAB(R1);
1176    761AP0100               LDL     RR10,^LAB_SSO(R1);
117A    342A 0014               LDL     RR10,^(RR2^(20));
117E    742A 0100               LDL     RR10,^(RR2^(R1));
1182
1182                            % LOAD CONSTANT
1182    BD48                    LDK     R4,8;
```

LDRB LDR LDRL	r,src	RA	Load Relative to Register r <== src
LDRB LDR LDRL	dst,r	RA	Load Relative to Memory dst <== r

NOTE

For AmZ8001, the relative address range available is the address space of the current segment.

LDR	r,^src	RA	Load Address Relative r <== ^src

(Z8001--current segment)

LDM	r,src,exp	IR,DA,X	Load Multiple r <== src (starting at r and src, load exp consecutive registers; exp is 1 to 16)

```
42E0
42E0
42E0                            % LOAD RELATIVE TO REGISTER
42E0    3006 FF1C               LDRB    RH6,LAB;
42E4    3106 FF18               LDR     R6,LAB;
42E8    3506 FF14               LDRL    RR6,LAB;
42EC
42EC                            % LOAD RELATIVE TO MEMORY
42EC    3206 FF10               LDRB    LAB,RH6;
42F0    3306 FF0C               LDR     LAB,R6;
42F4    3706 FF08               LDRL    LAB,RR6;
42F8
42F8                            %LOAD ADDRESS RELATIVE
42F8    340B FF04               LDR     R11,^LAB;
42FC
42FC                            % LOAD MULTIPLE TO REGISTER
42FC    1C21 0805               LDM     R8,R2^,6;
4300    5C01 0805 4200          LDM     R8,LAB,6;
4306    5C11 0805 4200          LDM     R8,LAB(R1),6;
```

```
1184
1184                            % LOAD RELATIVE TO REGISTER
1184    3006 FE78               LDRB    RH6,LAB;
1188    3106 FE74               LDR     R6,LAB;
118C    3506 FE70               LDRL    RR6,LAB;
1190
1190                            % LOAD RELATIVE TO MEMORY
1190    3206 FE6C               LDRB    LAB,RH6;
1194    3306 FE68               LDR     LAB,R6;
1198    3706 FE64               LDRL    LAB,RR6;
119C
119C                            %LOAD ADDRESS RELATIVE
119C    340A FE60               LDRL    RR10,^LAB;
11A0
11A0                            % LOAD MULTIPLE TO REGISTER
11A0    1C21 0805               LDM     R8,RR2^,6;
11A4    5C01 0805S0001          LDM     R8,LAB,6;
11A8    1000
11AC    5C01 0805P0100          LDM     R8,LAB_SSO,6;
11B2    5C11 0805S0001          LDM     R8,LAB(R1),6;
11B6    1000
11BA    5C11 0805P0100          LDM     R8,LAB_SSO(R1),6;
11C0
```

LDM	dst,r,exp	IR,DA,X	Load Multiple to Memory dst <== r (starting at dst and r, load exp consecutive registers; exp is 1 to 16)
LDDB LDD	dst,src,r	IR	Load and Decrement dst <== src Autodecrement dst and src r <== r - 1
LDDRB LDDR	dst,src,r	IR	Load, Decrement and Repeat dst <== src Autodecrement dst and src r <== r - 1 (repeat until r = 0)
LDIB LDI	dst,src,r	IR	Load and Increment dst <== src Autoincrement dst and src r <== r - 1
LDIRB LDIR	dst,src,r	IR	Load, Increment and Repeat dst <== src Autoincrement dst and src r <== r - 1 (repeat until r = 0)

```
430C
430C
430C                            % LOAD MULTIPLE TO MEMORY
430C    1C29 0805               LDM     R2^,R8,6;
4310    5C09 0805 4200          LDM     LAB,R8,6;
4316    5C19 0805 4200          LDM     LAB(R1),R8,6;
431C
431C
431C                            % LOAD AND DECREMENT
431C    BA29 0988               LDDB    R8^,R2^,R9;
4320    BB29 0988               LDD     R8^,R2^,R9;
4324
4324
4324                            % LOAD, DECREMENT, AND REPEAT
4324    BA29 0980               LDDRB   R8^,R2^,R9;
4328    BB29 0980               LDDR    R8^,R2^,R9;
432C
432C
432C
432C                            % LOAD AND INCREMENT
432C    BA21 0988               LDIB    R8^,R2^,R9;
4330    BB21 0988               LDI     R8^,R2^,R9;
4334
4334
4334                            % LOAD, INCREMENT, AND REPEAT
4334    BA21 0980               LDIRB   R8^,R2^,R9;
4338    BB21 0980               LDIR    R8^,R2^,R9;
433C
```

```
11C0                            % LO D MULTIPLE TO MEMORY
11C0    1C29 0805               LDM     RR2^,R8,6;
11C4    5C09 0805S0001          LDM     LAB,R8,6;
11C8    1000
11CC    5C09 0805P0100          LDM     LAB_SSO,R8,6;
11D2    5C19 0805S0001          LDM     LAB(R1),R8,6;
11D6    1000
11DA    5C19 0805P0100          LDM     LAB_SSO(R1),R8,6;
11E0
11E0                            % LOAD AND DECREMENT
11E0    BA29 0988               LDDB    RR8^,RR2^,R9;
11E4    BB29 0988               LDD     RR8^,RR2^,R9;
11E8
11E8                            % LOAD, DECREMENT, AND REPEAT
11E8    BA29 0980               LDDRB   RR8^,RR2^,R9;
11EC    BB29 0980               LDDR    RR8^,RR2^,R9;
11F0
11F0
11F0
11F0                            % LOAD AND INCREMENT
11F0    BA21 0988               LDIB    RR8^,RR2^,R9;
11F4    BB21 0988               LDI     RR8^,RR2^,R9;
11F8
11F8                            % LOAD, INCREMENT, AND REPEAT
11F8    BA21 0980               LDIRB   RR8^,RR2^,R9;
11FC    BB21 0980               LDIR    RR8^,RR2^,R9;
```

6-2. STACK MANIPULATION INSTRUCTIONS

Instruction		Addressing Modes for dst or src	Description
POP POPL	dst,ir	R,IR,DA,X	Pop dst <== what ir points to Autoincrement ir after pop
PUSH	ir,src	IM,R,IR,DA,X	Push Autodecrement ir before push What ir points to <== src
PUSHL	ir,src	R,IR,DA,X	Push long Autodecrement ir before push What ir points to <== src

```
433C                            TITLE   'STACK MANIPULATION';
433C
433C                            % POP
433C    97C4                    POP     R4,R12^;
433E    17C2                    POP     R2^,R12^;
4340    57C0 4200               POP     LAB,R12^;
4344    57C1 4200               POP     LAB(R1),R12^;
4348    95C4                    POPL    RR4,R12^;
434A    15C2                    POPL    R2^,R12^;
434C    55C0 4200               POPL    LAB,R12^;
4350    55C1 4200               POPL    LAB(R1),R12^;
4354
4354                            % PUSH
4354    0DC9 0005               PUSH    R12^,5;
4358    93C4                    PUSH    R12^,R4;
435A    13C2                    PUSH    R12^,R2^;
435C    53C0 4200               PUSH    R12^,LAB;
4360    53C1 4200               PUSH    R12^,LAB(R1);
4364    91C4                    PUSHL   R12^,RR4;
4366    11C2                    PUSHL   R12^,R2^;
4368    51C0 4200               PUSHL   R12^,LAB;
436C    51C1 4200               PUSHL   R12^,LAB(R1);
```

```
1200
1200                            TITLE   'STACK MANIPULATION';
1200
1200                            % POP
1200    97C4                    POP     R4,RR12^;
1202    17C2                    POP     RR2^,RR12^;
1204    57C0S0001 1000          POP     LAB,RR12^;
120A    57C0P0100               POP     LAB_SSO,RR12^;
120E    57C1S0001 1000          POP     LAB(R1),RR12^;
1214    57C1P0100               POP     LAB_SSO(R1),RR12^;
1218    95C4                    POPL    RR4,RR12^;
121A    15C2                    POPL    RR2^,RR12^;
121C    55C0S0001 1000          POPL    LAB,RR12^;
1222    55C0P0100               POPL    LAB_SSO,RR12^;
1226    55C1S0001 1000          POPL    LAB(R1),RR12^;
122C    55C1P0100               POPL    LAB_SSO(R1),RR12^;
1230
1230                            % PUSH
1230    0DC9 0005               PUSH    RR12^,5;
1234    93C4                    PUSH    RR12^,R4;
1236    13C2                    PUSH    RR12^,RR2^;
1238    53C0S0001 1000          PUSH    RR12^,LAB;
123E    53C0P0100               PUSH    RR12^,LAB_SSO;
1242    53C1S0001 1000          PUSH    RR12^,LAB(R1);
1248    53C1P0100               PUSH    RR12^,LAB_SSO(R1);
124C    91C4                    PUSHL   RR12^,RR4;
124E    11C2                    PUSHL   RR12^,RR2^;
1250    51C0S0001 1000          PUSHL   RR12^,LAB;
1256    51C0P0100               PUSHL   RR12^,LAB_SSO;
125A    51C1S0001 1000          PUSHL   RR12^,LAB(R1);
1260    51C1P0100               PUSHL   RR12^,LAB_SSO(R1);
1264
```

6-3. ARITHMETIC INSTRUCTIONS

Instruction		Addressing Modes for dst or src	Description
ADCB ADC	r,src	R	Add with Carry r <== r + src + carry
ADDB ADD ADDL	r,src	IM,R,IR,DA,X	Add r <== r + src
DAB	dst	R	Decimal Adjust (decimal adjust of dst)

```
4370
4370                        TITLE   'ARITHMETIC';
4370
4370                        % ADD WITH CARRY
4370   B446                 ADCB    RH6,RH4;
4372   B546                 ADC     R6,R4;
4374
4374                        % ADD
4374   0006 0505            ADDB    RH6,5;
4378   8046                 ADDB    RH6,RH4;
437A   0026                 ADDB    RH6,R2^;
437C   4006 4200            ADDB    RH6,LAB;
4380   4016 4200            ADDB    RH6,LAB(R1);
4384   0106 0005            ADD     R6,5;
4388   8146                 ADD     R6,R4;
438A   0126                 ADD     R6,R2^;
438C   4106 4200            ADD     R6,LAB;
4390   4116 4200            ADD     R6,LAB(R1);
4394   1606 0000 0005       ADDL    RR6,5;
439A   9646                 ADDL    RR6,RR4;
439C   1626                 ADDL    RR6,R2^;
439E   5606 4200            ADDL    RR6,LAB;
43A2   5616 4200            ADDL    RR6,LAB(R1);
43A6
43A6                        % DECIMAL ADJUST BYTE
43A6   B040                 DAB     RH4;
```

```
1264                        TITLE   'ARITHMETIC';
1264
1264                        % ADD WITH CARRY
1264   B446                 ADCB    RH6,RH4;
1266   B546                 ADC     R6,R4;
1268
1268                        % ADD
1268   0006 0505            ADDB    RH6,5;
126C   8046                 ADDB    RH6,RH4;
126E   0026                 ADDB    RH6,RR2^;
1270   4006S0001 1000       ADDB    RH6,LAB;
1276   4006P0100            ADDB    RH6,LAB_SSO;
127A   4016S0001 1000       ADDB    RH6,LAB(R1);
1280   4016P0100            ADDB    RH6,LAB_SSO(R1);
1284   0106 0005            ADD     R6,5;
1288   8146                 ADD     R6,R4;
128A   0126                 ADD     R6,RR2^;
128C   4106S0001 1000       ADD     R6,LAB;
1292   4106P0100            ADD     R6,LAB_SSO;
1296   4116S0001 1000       ADD     R6,LAB(R1);
129C   4116P0100            ADD     R6,LAB_SSO(R1);
12A0   1606 0000 0005       ADDL    RR6,5;
12A6   9646                 ADDL    RR6,RR4;
12A8   1626                 ADDL    RR6,RR2^;
12AA   5606S0001 1000       ADDL    RR6,LAB;
12B0   5606P0100            ADDL    RR6,LAB_SSO;
12B4   5616S0001 1000       ADDL    RR6,LAB(R1);
12BA   5616P0100            ADDL    RR6,LAB_SSO(R1);
12BE
12BE                        % DECIMAL ADJUST BYTE
12BE   B040                 DAB     RH4;
```

DECB DEC	dst,exp	R,IR,DA,X	Decrement dst <== dst - exp (exp is 1 to 16)
DIV DIVL	r,src	IM,R,IR,DA,X	Signed Divide upper half dst <== dst / src lower half dst <== remainder

```
43A8
43A8                            % DECREMENT
43A8    AA4B                    DECB    RH4,12;
43AA    2A2B                    DECB    R2^,12;
43AC    6A0B 4200               DECB    LAB,12;
43B0    6A1B 4200               DECB    LAB(R1),12;
43B4    AB4B                    DEC     R4,12;
43B6    2B2B                    DEC     R2^,12;
43B8    6B0B 4200               DEC     LAB,12;
43BC    6B1B 4200               DEC     LAB(R1),12;
43C0
43C0                            % DIVIDE
43C0    1B08 0005               DIV     RR8,5;
43C4    9B48                    DIV     RR8,R4;
43C6    1B28                    DIV     RR8,R2^;
43C8    5B08 4200               DIV     RR8,LAB;
43CC    5B18 4200               DIV     RR8,LAB(R1);
43D0    1A08 0000 0005          DIVL    RQ8,5;
43D6    9A48                    DIVL    RQ8,RR4;
43D8    1A28                    DIVL    RQ8,R2^;
43DA    5A08 4200               DIVL    RQ8,LAB;
43DE    5A18 4200               DIVL    RQ8,LAB(R1);
43E2
```

```
12C0
12C0                            % DECREMENT
12C0    AA4B                    DECB    RH4,12;
12C2    2A2B                    DECB    RR2^,12;
12C4    6A0BS0001 1000          DECB    LAB,12;
12CA    6A0BP0100               DECB    LAB_SSO,12;
12CE    6A1BS0001 1000          DECB    LAB(R1),12;
12D4    6A1BP0100               DECB    LAB_SSO(R1),12;
12D8    AB4B                    DEC     R4,12;
12DA    2B2B                    DEC     RR2^,12;
12DC    6B0BS0001 1000          DEC     LAB,12;
12E2    6B0BP0100               DEC     LAB_SSO,12;
12E6    6B1BS0001 1000          DEC     LAB(R1),12;
12EC    6B1BP0100               DEC     LAB_SSO(R1),12;
12F0
12F0                            % DIVIDE
12F0    1B08 0005               DIV      R8,5;
12F4    9B48                    DIV     RR8,R4;
12F6    1B28                    DIV     RR8,RR2^;
12F8    5B08S0001 1000          DIV     RR8,LAB;
12FE    5B08P0100               DIV     RR8,LAB_SSO;
1302    5B18S0001 1000          DIV     RR8,LAB(R1);
1308    5B18P0100               DIV     RR8,LAB_SSO(R1);
130C    1A08 0000 0005          DIVL    RQ8,5;
1312    9A48                    DIVL    RQ8,RR4;
1314    1A28                    DIVL    RQ8,RR2^;
1316    5A08S0001 1000          DIVL    RQ8,LAB;
131C    5A08P0100               DIVL    RQ8,LAB_SSO;
1320    5A18S0001 1000          DIVL    RQ8,LAB(R1);
1326    5A18P0100               DIVL    RQ8,LAB_SSO(R1);
132A
132A                            EJECT;
```

EXTSB EXTS EXTSL	dst	R	Extend Sign (extend sign of lower half dst to upper half dst)
INCB INC	dst,exp	R,IR,DA,X	Increment dst <== dst + exp (exp is 1 to 16)
MULT MULTL	r,src	IM,R,IR,DA,X	Signed Multiply dst <== lower half dst * src

```
43E2                        % EXTEND SIGN
43E2   B180                 EXTSB   R8;
43E4   B18A                 EXTS    RR8;
43E6   B187                 EXTSL   RQ8;
43E8
43E8                        % INCREMENT
43E8   A843                 INCB    RH4,4;
43EA   2823                 INCB    R2^,4;
43EC   6803 4200            INCB    LAB,4;
43F0   6813 4200            INCB    LAB(R1),4;
43F4   A943                 INC     R4,4;
43F6   2923                 INC     R2^,4;
43F8   6903 4200            INC     LAB,4;
43FC   6913 4200            INC     LAB(R1),4;
4400
4400                        % MULTIPLY
4400   1908 0005            MULT    RR8,5;
4404   9948                 MULT    RR8,R4;
4406   1928                 MULT    RR8,R2^;
4408   5908 4200            MULT    RR8,LAB;
440C   5918 4200            MULT    RR8,LAB(R1);
4410   1808 0000 0005       MULTL   RQ8,5;
4416   9848                 MULTL   RQ8,RR4;
4418   1828                 MULTL   RQ8,R2^;
441A   5808 4200            MULTL   RQ8,LAB;
441E   5818 4200            MULTL   RQ8,LAB(R1);
```

```
132A                        % EXTEND SIGN
132A   B180                 EXTSB   R8;
132C   B18A                 EXTS    RR8;
132E   B187                 EXTSL   RQ8;
1330
1330                        % INCREMENT
1330   A843                 INCB    RH4,4;
1332   2823                 INCB    RR2^,4;
1334   6803S0001 1000       INCB    LAB,4;
133A   6803P0100            INCB    LAB_SSO,4;
133E   6813S0001 1000       INCB    LAB(R1),4;
1344   6813P0100            INCB    LAB_SSO(R1),4;
1348   A943                 INC     R4,4;
134A   2923                 INC     RR2^,4;
134C   6903S0001 1000       INC     LAB,4;
1352   6903P0100            INC     LAB_SSO,4;
1356   6913S0001 1000       INC     LAB(R1),4;
135C   6913P0100            INC     LAB_SSO(R1),4;
1360
1360                        % MULTIPLY
1360   1908 0005            MULT    RR8,5;
1364   9948                 MULT    RR8,R4;
1366   1928                 MULT    RR8,RR2^;
1368   5908S0001 1000       MULT    RR8,LAB;
136E   5908P0100            MULT    RR8,LAB_SSO;
1372   5918S0001 1000       MULT    RR8,LAB(R1);
1378   5918P0100            MULT    RR8,LAB_SSO(R1);
137C   1808 0000 0005       MULTL   RQ8,5;
1382   9848                 MULTL   RQ8,RR4;
1384   1828                 MULTL   RQ8,RR2^;
1386   5808S0001 1000       MULTL   RQ8,LAB;
138C   5808P0100            MULTL   RQ8,LAB_SSO;
1390   5818S0001 1000       MULTL   RQ8,LAB(R1);
1396   5818P0100            MULTL   RQ8,LAB_SSO(R1);
139A
```

NEGB NEG	dst	R,IR,DA,X	Negate dst <== 0 - dst
SBCB SBC	r,src	R	Subtract with Carry r <== r - src - carry

```
4422
4422                            % NEGATE
4422    8C42                    NEGB    RH4;
4424    0C22                    NEGB    R2^;
4426    4C02 4200               NEGB    LAB;
442A    4C12 4200               NEGB    LAB(R1);
442E    8D42                    NEG     R4;
4430    0D22                    NEG     R2^;
4432    4D02 4200               NEG     LAB;
4436    4D12 4200               NEG     LAB(R1);
443A
443A                            % SUBTRACT WITH CARRY
443A    B645                    SBCB    RH5,RH4;
443C    B745                    SBC     R5,R4;
443E
```

```
139A                            % NEGATE
139A    8C42                    NEGB    RH4;
139C    0C22                    NEGB    RR2^;
139E    4C02S0001 1000          NEGB    LAB;
13A4    4C02P0100               NEGB    LAB_SSO;
13A8    4C12S0001 1000          NEGB    LAB(R1);
13AE    4C12P0100               NEGB    LAB_SSO(R1);
13B2    8D42                    NEG     R4;
13B4    0D22                    NEG     RR2^;
13B6    4D02S0001 1000          NEG     LAB;
13BC    4D02P0100               NEG     LAB_SSO;
13C0    4D12S0001 1000          NEG     LAB(R1);
13C6    4D12P0100               NEG     LAB_SSO(R1);
13CA
13CA                            % SUBTRACT WITH CARRY
13CA    B645                    SBCB    RH5,RH4;
13CC    B745                    SBC     R5,R4;
```

SUBB SUB SUBL	r,src	IM,R,IR,DA,X	Subtract r <== r - src

```
443E                          % SUBTRACT
443E   0205 0505               SUBB    RH5,5;
4442   8245                    SUBB    RH5,RH4;
4444   0225                    SUBB    RH5,R2^;
4446   4205 4200               SUBB    RH5,LAB;
444A   4215 4200               SUBB    RH5,LAB(R1);
444E   0305 0005               SUB     R5,5;
4452   8345                    SUB     R5,R4;
4454   0325                    SUB     R5,R2^;
4456   4305 4200               SUB     R5,LAB;
445A   4315 4200               SUB     R5,LAB(R1);
445E   9224                    SUBL    RR4,RR2;
4460   1204 0000 0005          SUBL    RR4,5;
4466   1224                    SUBL    RR4,R2^;
4468   5204 4200               SUBL    RR4,LAB;
446C   5214 4200               SUBL    RR4,LAB(R1);
4470
```

```
13CE
13CE                          % SUBTRACT
13CE   0205 0505               SUBB    RH5,5;
13D2   8245                    SUBB    RH5,RH4;
13D4   0225                    SUBB    RH5,RR2^;
13D6   4205S0001 1000          SUBB    RH5,LAB;
13DC   4205P0100               SUBB    RH5,LAB_SSO;
13E0   4215S0001 1000          SUBB    RH5,LAB(R1);
13E6   4215P0100               SUBB    RH5,LAB_SSO(R1);
13EA   0305 0005               SUB     R5,5;
13EE   8345                    SUB     R5,R4;
13F0   0325                    SUB     R5,RR2^;
13F2   4305S0001 1000          SUB     R5,LAB;
13F8   4305P0100               SUB     R5,LAB_SSO;
13FC   4315S0001 1000          SUB     R5,LAB(R1);
1402   4315P0100               SUB     R5,LAB_SSO(R1);
1406   9224                    SUBL    RR4,RR2;
1408   1204 0000 0005          SUBL    RR4,5;
140E   1224                    SUBL    RR4,RR2^;
1410   5204P0100               SUBL    RR4,LAB_SSO;
1414   5204S0001 1000          SUBL    RR4,LAB;
141A   5214P0100               SUBL    RR4,LAB_SSO(R1);
141E   5214S0001 1000          SUBL    RR4,LAB(R1);
```

6-4. LOGICAL INSTRUCTIONS

Instruction		Addressing Modes for dst or src	Description
ANDB AND	r,src	IM,R,IR,DA,X	AND r <== r AND src
COMB COM	dst	R,IR,DA,X	Complement dst <== NOT dst

```
4470                        TITLE   'LOGICAL';
4470
4470                        % AND
4470    0607 0505           ANDB    RH7,5;
4474    8647                ANDB    RH7,RH4;
4476    0627                ANDB    RH7,R2^;
4478    4607 4200           ANDB    RH7,LAB;
447C    4617 4200           ANDB    RH7,LAB(R1);
4480    0707 0005           AND     R7,5;
4484    8747                AND     R7,R4;
4486    0727                AND     R7,R2^;
4488    4707 4200           AND     R7,LAB;
448C    4717 4200           AND     R7,LAB(R1);
4490
4490                        % COMPLEMENT
4490    8C40                COMB    RH4;
4492    0C20                COMB    R2^;
4494    4C00 4200           COMB    LAB;
4498    4C10 4200           COMB    LAB(R1);
449C    8D40                COM     R4;
449E    0D20                COM     R2^;
44A0    4D00 4200           COM     LAB;
44A4    4D10 4200           COM     LAB(R1);
```

```
1424
1424                        TITLE   'LOGICAL';
1424
1424                        % AND
1424    0607 0505           ANDB    RH7,5;
1428    8647                ANDB    RH7,RH4;
142A    0627                ANDB    RH7,RR2^;
142C    4607S0001 1000      ANDB    RH7,LAB;
1432    4607P0100           ANDB    RH7,LAB_SSO;
1436    4617S0001 1000      ANDB    RH7,LAB(R1);
143C    4617P0100           ANDB    RH7,LAB_SSO(R1);
1440    0707 0005           AND     R7,5;
1444    8747                AND     R7,R4;
1446    0727                AND     R7,RR2^;
1448    4707S0001 1000      AND     R7,LAB;
144E    4707P0100           AND     R7,LAB_SSO;
1452    4717S0001 1000      AND     R7,LAB(R1);
1458    4717P0100           AND     R7,LAB_SSO(R1);
145C
145C                        % COMPLEMENT
145C    8C40                COMB    RH4;
145E    0C20                COMB    RR2^;
1460    4C00S0001 1000      COMB    LAB;
1466    4C00P0100           COMB    LAB_SSO;
146A    4C10S0001 1000      COMB    LAB(R1);
1470    4C10P0100           COMB    LAB_SSO(R1);
1474    8D40                COM     R4;
1476    0D20                COM     RR2^;
1478    4D00S0001 1000      COM     LAB;
147E    4D00P0100           COM     LAB_SSO;
1482    4D10S0001 1000      COM     LAB(R1);
1488    4D10P0100           COM     LAB_SSO(R1);
```

ORB OR	r,src	IM,R,IR,DA,X	OR r <== r OR src

```
44A8
44A8                            % OR
44A8    0407 0505               ORB     RH7,5;
44AC    8447                    ORB     RH7,RH4;
44AE    0427                    ORB     RH7,R2^;
44B0    4407 4200               ORB     RH7,LAB;
44B4    4417 4200               ORB     RH7,LAB(R1);
44B8    0507 0005               OR      R7,5;
44BC    8547                    OR      R7,R4;
44BE    0527                    OR      R7,R2^;
44C0    4507 4200               OR      R7,LAB;
44C4    4517 4200               OR      R7,LAB(R1);
```

```
148C
148C                            % OR
148C    0407 0505               ORB     RH7,5;
1490    8447                    ORB     RH7,RH4;
1492    0427                    ORB     RH7,RR2^;
1494    4407S0001 1000          ORB     RH7,LAB;
149A    4407P0100               ORB     RH7,LAB_SSO;
149E    4417S0001 1000          ORB     RH7,LAB(R1);
14A4    4417P0100               ORB     RH7,LAB_SSO(R1);
14A8    0507 0005               OR      R7,5;
14AC    8547                    OR      R7,R4;
14AE    0527                    OR      R7,RR2^;
14B0    4507S0001 1000          OR      R7,LAB;
14B6    4507P0100               OR      R7,LAB_SSO;
14BA    4517S0001 1000          OR      R7,LAB(R1);
14C0    4517P0100               OR      R7,LAB_SSO(R1);
```

TESTB TEST TESTL	dst	R,IR,DA,X	Test dst OR 0 (sets flags)
TCCB TCC	cc,dst	R	Test Condition Code If cc is true: lsb of dst <== 1 otherwise: lsb of dst <== 0 (lsb is least significant bit)

```
44C8
44C8                        % TEST
44C8    8C44                TESTB   RH4;
44CA    0C24                TESTB   R2^;
44CC    4C04 4200           TESTB   LAB;
44D0    4C14 4200           TESTB   LAB(R1);
44D4    8D44                TEST    R4;
44D6    0D24                TEST    R2^;
44D8    4D04 4200           TEST    LAB;
44DC    4D14 4200           TEST    LAB(R1);
44E0    9C48                TESTL   RR4;
44E2    1C28                TESTL   R2^;
44E4    5C08 4200           TESTL   LAB;
44E8    5C18 4200           TESTL   LAB(R1);
44EC
44EC                        % TEST CONDITION CODE
44EC    AE46                TCCB    ZR,RH4;
44EE    AF46                TCC     ZR,R4;
```

```
14C4
14C4                        % TEST
14C4    8C44                TESTB   RH4;
14C6    0C24                TESTB   RR2^;
14C8    4C04S0001 1000      TESTB   LAB;
14CE    4C04P0100           TESTB   LAB_SSO;
14D2    4C14S0001 1000      TESTB   LAB(R1);
14D8    4C14P0100           TESTB   LAB_SSO(R1);
14DC    8D44                TEST    R4;
14DE    0D24                TEST    RR2^;
14E0    4D04S0001 1000      TEST    LAB;
14E6    4D04P0100           TEST    LAB_SSO;
14EA    4D14S0001 1000      TEST    LAB(R1);
14F0    4D14P0100           TEST    LAB_SSO(R1);
14F4    9C48                TESTL   RR4;
14F6    1C28                TESTL   RR2^;
14F8    5C08S0001 1000      TESTL   LAB;
14FE    5C08P0100           TESTL   LAB_SSO;
1502    5C18S0001 1000      TESTL   LAB(R1);
1508    5C18P0100           TESTL   LAB_SSO(R1);
150C
150C                        EJECT;
150C                        % TEST CONDITION CODE
150C    AE46                TCCB    ZR,RH4;
150E    AF46                TCC     ZR,R4;
1510
```

XORB XOR	r,src	IM,R,IR,DA,X	Exclusive OR r <== r XOR src

```
44F0
44F0                            % EXCLUSIVE OR
44F0    0807 0505               XORB    RH7,5;
44F4    8847                    XORB    RH7,RH4;
44F6    0827                    XORB    RH7,R2^;
44F8    4807 4200               XORB    RH7,LAB;
44FC    4817 4200               XORB    RH7,LAB(R1);
4500    0907 0005               XOR     R7,5;
4504    8947                    XOR     R7,R4;
4506    0927                    XOR     R7,R2^;
4508    4907 4200               XOR     R7,LAB;
450C    4917 4200               XOR     R7,LAB(R1);
```

```
1510
1510                            % EXCLUSIVE OR
1510    0807 0505               XORB    RH7,5;
1514    8847                    XORB    RH7,RH4;
1516    0827                    XORB    RH7,RR2^;
1518    4807S0001 1000          XORB    RH7,LAB;
151E    4807P0100               XORB    RH7,LAB_SSO;
1522    4817S0001 1000          XORB    RH7,LAB(R1);
1528    4817P0100               XORB    RH7,LAB_SSO(R1);
152C    0907 0005               XOR     R7,5;
1530    8947                    XOR     R7,R4;
1532    0927                    XOR     R7,RR2^;
1534    4907S0001 1000          XOR     R7,LAB;
153A    4907P0100               XOR     R7,LAB_SSO;
153E    4917S0001 1000          XOR     R7,LAB(R1);
1544    4917P0100               XOR     R7,LAB_SSO(R1);
1548
1548                            EJECT;
```

6-5. ROTATE AND SHIFT INSTRUCTIONS

Instruction		Addressing Modes for dst or src	Description
RLDB	r,src	R	Rotate Left Digit
RRDB	r,src	R	Rotate Right Digit
RLB RL	dst,exp	R	Rotate Left (rotate dst left; exp is 1 or 2, default 1)
RLCB RLC	dst,exp	R	Rotate Left through Carry (rotate dst left; exp is 1 or 2, default 1)
RRB RR	dst,exp	R	Rotate Right (rotate dst right; exp is 1 or 2, default 1)
RRCB RRC	dst,exp	R	Rotate Right through Carry (rotate dst right; exp is 1 or 2, default 1)
SDAB SDA SDAL	dst,r	R	Shift Dynamic Arithmetic (shift dst left or right by r bits; positive left, negative right)

```
4510
4510                        TITLE   'ROTATE AND SHIFT';
4510
4510                        % ROTATE LEFT DIGIT
4510   BE47                 RLDB    RH7,RH4;
4512
4512                        % ROTATE RIGHT DIGIT
4512   BC47                 RRDB    RH7,RH4;
4514
4514                        % ROTATE LEFT
4514   B240                 RLB     RH4,1;
4516   B340                 RL      R4,1;
4518
4518                        % ROTATE LEFT THROUGH CARRY
4518   B248                 RLCB    RH4,1;
451A   B348                 RLC     R4;
451C
451C                        % ROTATE RIGHT
451C   B244                 RRB     RH4,1;
451E   B344                 RR      R4,1;
4520
4520                        % ROTATE RIGHT THROUGH CARRY
4520   B24C                 RRCB    RH4,1;
4522   B34C                 RRC     R4,1;
4524
4524                        % SHIFT DYNAMIC ARITHMETIC
4524   B24B 0900            SDAB    RH4,R9;
4528   B34B 0900            SDA     R4,R9;
452C   B34F 0900            SDAL    RR4,R9;
```

```
1548                        TITLE   'ROTATE AND SHIFT';
1548
1548                        % ROTATE LEFT DIGIT
1548   BE47                 RLDB    RH7,RH4;
154A
154A                        % ROTATE RIGHT DIGIT
154A   BC47                 RRDB    RH7,RH4;
154C
154C                        % ROTATE LEFT
154C   B240                 RLB     RH4,1;
154E   B340                 RL      R4,1;
1550
1550                        % ROTATE LEFT THROUGH CARRY
1550   B248                 RLCB    RH4,1;
1552   B348                 RLC     R4;
1554
1554                        % ROTATE RIGHT
1554   B244                 RRB     RH4,1;
1556   B344                 RR      R4,1;
1558
1558                        % ROTATE RIGHT THROUGH CARRY
1558   B24C                 RRCB    RH4,1;
155A   B34C                 RRC     R4,1;
155C
155C                        % SHIFT DYNAMIC ARITHMETIC
155C   B24B 0900            SDAB    RH4,R9;
1560   B34B 0900            SDA     R4,R9;
1564   B34F 0900            SDAL    RR4,R9;
```

SDLB SDL SDLL	dst,r	R	Shift Dynamic Logical (shift dst left or right by r bits; positive left, negative right)
SLAB SLA SLAL	dst,exp	R	Shift Left Arithmetic (shift dst left by exp bits)
SLLB SLL SLLL	dst,exp	R	Shift Left Logical (shift dst left by exp bits)
SRAB SRA SRAL	dst,exp	R	Shift Right Arithmetic (shift dst right by exp bits)
SRLB SRL SRLL	dst,exp	R	Shift Right Logical (shift dst right by exp bits)

```
4530
4530                        % SHIFT DYNAMIC LOGICAL
4530   B243 0900            SDLB    RH4,R9;
4534   B343 0900            SDL     R4,R9;
4538   B347 0900            SDLL    RR4,R9;
453C
453C                        % SHIFT LEFT ARITHMETIC
453C   B249 0002            SLAB    RH4,2;
4540   B349 0002            SLA     R4,2;
4544   B34D 0002            SLAL    RR4,2;
4548
4548                        % SHIFT LEFT LOGICAL
4548   B241 0002            SLLB    RH4,2;
454C   B341 0002            SLL     R4,2;
4550   B345 0002            SLLL    RR4,2;
4554
4554                        % SHIFT RIGHT ARITHMETIC
4554   B249 FFFE            SRAB    RH4,2;
4558   B349 FFFE            SRA     R4,2;
455C   B34D FFFE            SRAL    RR4,2;
4560
4560                        % SHIFT RIGHT LOGICAL
4560   B241 FFFE            SRLB    RH4,2;
4564   B341 FFFE            SRL     R4,2;
4568   B345 FFFE            SRLL    RR4,2;
456C
```

```
1568
1568                        % SHIFT DYNAMIC LOGICAL
1568   B243 0900            SDLB    RH4,R9;
156C   B343 0900            SDL     R4,R9;
1570   B347 0900            SDLL    RR4,R9;
1574
1574                        % SHIFT LEFT ARITHMETIC
1574   B249 0002            SLAB    RH4,2;
1578   B349 0002            SLA     R4,2;
157C   B34D 0002            SLAL    RR4,2;
1580
1580                        % SHIFT LEFT LOGICAL
1580   B241 0002            SLLB    RH4,2;
1584   B341 0002            SLL     R4,2;
1588   B345 0002            SLLL    RR4,2;
158C
158C                        EJECT;
158C                        % SHIFT RIGHT ARITHMETIC
158C   B249 FFFE            SRAB    RH4,2;
1590   B349 FFFE            SRA     R4,2;
1594   B34D FFFE            SRAL    RR4,2;
1598
1598                        % SHIFT RIGHT LOGICAL
1598   B241 FFFE            SRLB    RH4,2;
159C   B341 FFFE            SRL     R4,2;
15A0   B345 FFFE            SRLL    RR4,2;
15A4
15A4                        EJECT;
```

6-6. BIT MANIPULATION INSTRUCTIONS

Instruction		Addressing Modes for dst or src	Description
BITB BIT	dst,exp	R,IR,DA,X	Test Bit Static Z flag <== NOT (bit exp of dst)
BITB BIT	dst,r	R	Test Bit Dynamic Z flag <== NOT (bit r of dst)
RESB RES	dst,exp	R,IR,DA,X	Reset Bit Static bit exp of dst <== 0

```
456C                        TITLE   'BIT MANIPULATION';
456C
456C                        % TEST BIT STATIC
456C   A640                 BITB    RH4,0;
456E   2620                 BITB    R2^,0;
4570   6600 4200            BITB    LAB,0;
4574   6610 4200            BITB    LAB(R1),0;
4578   A740                 BIT     R4,0;
457A   2720                 BIT     R2^,0;
457C   6700 4200            BIT     LAB,0;
4580   6710 4200            BIT     LAB(R1),0;
4584
4584                        % TEST BIT DYNAMIC
4584   2606 0400            BITB    RH4,R6;
4588   2706 0400            BIT     R4,R6;
458C
458C                        % RESET BIT STATIC
458C   A240                 RESB    RH4,0;
458E   2220                 RESB    R2^,0;
4590   6200 4200            RESB    LAB,0;
4594   6210 4200            RESB    LAB(R1),0;
4598   A340                 RES     R4,0;
459A   2320                 RES     R2^,0;
459C   6300 4200            RES     LAB,0;
45A0   6310 4200            RES     LAB(R1),0;
```

```
15A4                        TITLE   'BIT MANIPULATION';
15A4
15A4                        % TEST BIT STATIC
15A4   A640                 BITB    RH4,0;
15A6   2620                 BITB    RR2^,0;
15A8   6600S0001 1000       BITB    LAB,0;
15AE   6600P0100            BITB    LAB_SSO,0;
15B2   6610S0001 1000       BITB    LAB(R1),0;
15B8   6610P0100            BITB    LAB_SSO(R1),0;
15BC   A740                 BIT     R4,0;
15BE   2720                 BIT     RR2^,0;
15C0   6700S0001 1000       BIT     LAB,0;
15C6   6700P0100            BIT     LAB_SSO,0;
15CA   6710S0001 1000       BIT     LAB(R1),0;
15D0   6710P0100            BIT     LAB_SSO(R1),0;
15D4
15D4                        % TEST BIT DYNAMIC
15D4   2606 0400            BITB    RH4,R6;
15D8   2706 0400            BIT     R4,R6;
15DC
15DC                        % RESET BIT STATIC
15DC   A240                 RESB    RH4,0;
15DE   2220                 RESB    RR2^,0;
15E0   6200S0001 1000       RESB    LAB,0;
15E6   6200P0100            RESB    LAB_SSO,0;
15EA   6210S0001 1000       RESB    LAB(R1),0;
15F0   6210P0100            RESB    LAB_SSO(R1),0;
15F4   A340                 RES     R4,0;
15F6   2320                 RES     RR2^,0;
15F8   6300S0001 1000       RES     LAB,0;
15FE   6300P0100            RES     LAB_SSO,0;
1602   6310S0001 1000       RES     LAB(R1),0;
1608   6310P0100            RES     LAB_SSO(R1),0;
```

RESB RES	dst,r	R	Reset Bit Dynamic bit r of dst <== 0
SETB SET	dst,exp	R,IR,DA,X	Set Bit Static bit exp of dst <== 1
SETB SET	dst,r	R	Set Bit Dynamic bit r of dst <== 1

```
45A4
45A4                            % RESET BIT DYNAMIC
45A4    2206 0400               RESB    RH4,R6;
45A8    2306 0400               RES     R4,R6;
45AC
45AC                            % SET BIT STATIC
45AC    A440                    SETB    RH4,0;
45AE    2420                    SETB    R2^,0;
45B0    6400 4200               SETB    LAB,0;
45B4    6410 4200               SETB    LAB(R1),0;
45B8    A540                    SET     R4,0;
45BA    2520                    SET     R2^,0;
45BC    6500 4200               SET     LAB,0;
45C0    6510 4200               SET     LAB(R1),0;
45C4
45C4                            % SET BIT DYNAMIC
45C4    2406 0400               SETB    RH4,R6;
45C8    2506 0400               SET     R4,R6;
45CC
```

```
160C
160C                            % RESET BIT DYNAMIC
160C    2206 0400               RESB    RH4,R6;
1610    2306 0400               RES     R4,R6;
1614
1614                            % SET BIT STATIC
1614    A440                    SETB    RH4,0;
1616    2420                    SETB    RR2^,0;
1618    6400S0001 1000          SETB    LAB,0;
161E    6400P0100               SETB    LAB_SSO,0;
1622    6410S0001 1000          SETB    LAB(R1),0;
1628    6410P0100               SETB    LAB_SSO(R1),0;
162C    A540                    SET     R4,0;
162E    2520                    SET     RR2^,0;
1630    6500S0001 1000          SET     LAB,0;
1636    6500P0100               SET     LAB_SSO,0;
163A    6510S0001 1000          SET     LAB(R1),0;
1640    6510P0100               SET     LAB_SSO(R1),0;
1644
1644                            % SET BIT DYNAMIC
1644    2406 0400               SETB    RH4,R6;
1648    2506 0400               SET     R4,R6;
164C
164C                            EJECT;
```

TSETB TSET	dst	R,IR,DA,X	Test and Set S flag <== msb of dst (all bits in dst are set to 1; msb is most significant bit)

```
45CC                        % TEST AND SET
45CC    8C46                TSETB   RH4;
45CE    0C26                TSETB   R2^;
45D0    4C06 4200           TSETB   LAB;
45D4    4C16 4200           TSETB   LAB(R1);
45D8    8D46                TSET    R4;
45DA    0D26                TSET    R2^;
45DC    4D06 4200           TSET    LAB;
45E0    4D16 4200           TSET    LAB(R1);
45E4
```

```
164C                        % TEST AND SET
164C    8C46                TSETB   RH4;
164E    0C26                TSETB   RR2^;
1650    4C06S0001 1000      TSETB   LAB;
1656    4C06P0100           TSETB   LAB_SSO;
165A    4C16S0001 1000      TSETB   LAB(R1);
1660    4C16P0100           TSETB   LAB_SSO(R1);
1664    8D46                TSET    R4;
1666    0D26                TSET    RR2^;
1668    4D06S0001 1000      TSET    LAB;
166E    4D06P0100           TSET    LAB_SSO;
1672    4D16S0001 1000      TSET    LAB(R1);
1678    4D16P0100           TSET    LAB_SSO(R1);
167C
167C                        EJECT;
```

6-7. COMPARE INSTRUCTIONS

Instruction	Addressing Modes for dst or src	Description
CPB r,src CP	IM,R,IR,DA,X	Compare Register with Memory r - src (affects flags)

```
45E4                        TITLE   'COMPARE';
45E4
45E4                        % COMPARE REGISTER WITH MEMORY
45E4   0A06 0505            CPB     RH6,5;
45E8   8A46                 CPB     RH6,RH4;
45EA   0A26                 CPB     RH6,R2^;
45EC   4A06 4200            CPB     RH6,LAB;
45F0   4A16 4200            CPB     RH6,LAB(R1);
45F4   0B06 0005            CP      R6,5;
45F8   8B46                 CP      R6,R4;
45FA   0B26                 CP      R6,R2^;
45FC   4B06 4200            CP      R6,LAB;
4600   4B16 4200            CP      R6,LAB(R1);
4604   1006 0000 0005       CPL     RR6,5;
460A   9046                 CPL     RR6,RR4;
460C   1026                 CPL     RR6,R2^;
460E   5006 4200            CPL     RR6,LAB;
4612   5016 4200            CPL     RR6,LAB(R1);
```

```
167C                        TITLE   'COMPARE';
167C
167C                        % COMPARE REGISTER WITH MEMORY
167C   0A06 0505            CPB     RH6,5;
1680   8A46                 CPB     RH6,RH4;
1682   0A26                 CPB     RH6,RR2^;
1684   4A06S0001 1000       CPB     RH6,LAB;
168A   4A06P0100            CPB     RH6,LAB_SSO;
168E   4A16S0001 1000       CPB     RH6,LAB(R1);
1694   4A16P0100            CPB     RH6,LAB_SSO(R1);
1698   0B06 0005            CP      R6,5;
169C   8B46                 CP      R6,R4;
169E   0B26                 CP      R6,RR2^;
16A0   4B06S0001 1000       CP      R6,LAB;
16A6   4B06P0100            CP      R6,LAB_SSO;
16AA   4B16S0001 1000       CP      R6,LAB(R1);
16B0   4B16P0100            CP      R6,LAB_SSO(R1);
16B4   1006 0000 0005       CPL     RR6,5;
16BA   9046                 CPL     RR6,RR4;
16BC   1026                 CPL     RR6,RR2^;
16BE   5006S0001 1000       CPL     RR6,LAB;
16C4   5006P0100            CPL     RR6,LAB_SSO;
16C8   5016S0001 1000       CPL     RR6,LAB(R1);
16CE   5016P0100            CPL     RR6,LAB_SSO(R1);
```

CPB CP	dst,im	IR,DA,X	Compare Memory with Immediate dst - im (affects flags)
CPDB CPD	r1,src,r2,cc	IR	Compare and Decrement r1 - src Autodecrement src r2 <== r2 - 1
CPDRB CPDR	r1,src,r2,cc	IR	Compare, Decrement and Repeat r1 - src Autodecrement src r2 <== r2 - 1 (repeat until cc true or r2 = 0)
CPIB CPI	r1,src,r2,cc	IR	Compare and Increment r1 - src Autoincrement src r2 <== r2 - 1

```
4616
4616                            % COMPARE MEMORY WITH IMMEDIATE
4616    0C21 0505               CPB     R2^,5;
461A    4C01 4200 0505          CPB     LAB,5;
4620    4C11 4200 0505          CPB     LAB(R1),5;
4626    0D21 0005               CP      R2^,5;
462A    4D01 4200 0005          CP      LAB,5;
4630    4D11 4200 0005          CP      LAB(R1),5;
4636
4636                            % COMPARE AND DECREMENT
4636    BA28 0765               CPDB    RH6,R2^,R7,MI;
463A    BB28 0765               CPD     R6,R2^,R7,MI;
463E
463E
463E                            % COMPARE, DECREMENT, AND REPEAT
463E    BA2C 0765               CPDRB   RH6,R2^,R7,MI;
4642    BB2C 0765               CPDR    R6,R2^,R7,MI;
4646
4646
4646
4646                            % COMPARE AND INCREMENT
4646    BA20 0765               CPIB    RH6,R2^,R7,MI;
464A    BB20 0765               CPI     R6,R2^,R7,MI;
464E
464E                            EJECT;
```

```
16D2
16D2                            % COMPARE MEMORY WITH IMMEDIATE
16D2    0C21 0505               CPB     RR2^,5;
16D6    4C01S0001 1000          CPB     LAB,5;
16DC    0505
16DE    4C01P0100 0505          CPB     LAB_SSO,5;
16E4    4C11S0001 1000          CPB     LAB(R1),5;
16EA    0505
16EC    4C11P0100 0505          CPB     LAB_SSO(R1),5;
16F2    0D21 0005               CP      RR2^,5;
16F6    4D01S0001 1000          CP      LAB,5;
16FC    0005
16FE    4D01P0100 0005          CP      LAB_SSO,5;
1704    4D11S0001 1000          CP      LAB(R1),5;
170A    0005
170C    4D11P0100 0005          CP      LAB_SSO(R1),5;
1712
1712                            % COMPARE AND DECREMENT
1712    BA28 0765               CPDB    RH6,RR2^,R7,MI;
1716    BB28 0765               CPD     R6,RR2^,R7,MI;
171A
171A
171A                            % COMPARE, DECREMENT, AND REPEAT
171A    BA2C 0765               CPDRB   RH6,RR2^,R7,MI;
171E    BB2C 0765               CPDR    R6,RR2^,R7,MI;
1722
1722
1722
1722                            % COMPARE AND INCREMENT
1722    BA20 0765               CPIB    RH6,RR2^,R7,MI;
1726    BB20 0765               CPI     R6,RR2^,R7,MI;
172A
172A                            EJECT;
```

CPIRB CPIR	r1,src,r2,cc	IR	Compare, Increment and Repeat r1 - src Autoincrement src r2 <== r2 - 1 (repeat until cc true or r2 = 0)
CPSDB CPSD	dst,src,r,cc	IR	Compare String and Decrement dst - src Autodecrement dst and src r <== r - 1
CPSDRB CPSDR	dst,src,r,cc	IR	Compare String, Decrement and Repeat dst - src Autodecrement dst and src r <== r - 1 (repeat until cc true or r = 0)
CPSIB CPSI	dst,src,r,cc	IR	Compare String and Increment dst - src Autoincrement dst and src r <== r - 1
CPSIRB CPSIR	dst,src,r,cc	IR	Compare String, Increment and Repeat dst - src Autoincrement dst and src r <== r - 1 (repeat until cc true or r = 0)

```
464E                        % COMPARE, INCREMENT, AND REPEAT
464E    BA24 0765           CPIRB   RH6,R2^,R7,MI;
4652    BB24 0765           CPIR    R6,R2^,R7,MI;
4656
4656
4656
4656                        % COMPARE STRING AND DECREMENT
4656    BA2A 07BE           CPSDB   R11^,R2^,R7,NE;
465A    BB2A 07BE           CPSD    R11^,R2^,R7,NE;
465E
465E                        % COMPARE STRING, DEC. AND REPEAT
465E    BA2E 07BE           CPSDRB  R11^,R2^,R7,NE;
4662    BB2E 07BE           CPSDR   R11^,R2^,R7,NE;
4666
4666                        % COMPARE STRING AND INCREMENT
4666    BA22 07BE           CPSIB   R11^,R2^,R7,NE;
466A    BB22 07BE           CPSI    R11^,R2^,R7,NE;
466E
466E                        % COMPARE STRING, INC. AND REPEAT
466E    BA26 07BE           CPSIRB  R11^,R2^,R7,NE;
4672    BB26 07BE           CPSIR   R11^,R2^,R7,NE;
4676
```

```
172A                        % COMPARE, INCREMENT, AND REPEAT
172A    BA24 0765           CPIRB   RH6,RR2^,R7,MI;
172E    BB24 0765           CPIR    R6,RR2^,R7,MI;
1732
1732
1732
1732                        % COMPARE STRING AND DECREMENT
1732    BA2A 07AE           CPSDB   RR10^,RR2^,R7,NE;
1736    BB2A 07AE           CPSD    RR10^,RR2^,R7,NE;
173A
173A
173A                        % COMPARE STRING, DEC. AND REPEAT
173A    BA2E 07AE           CPSDRB  RR10^,RR2^,R7,NE;
173E    BB2E 07AE           CPSDR   RR10^,RR2^,R7,NE;
1742
1742
1742
1742
1742                        % COMPARE STRING AND INCREMENT
1742    BA22 07AE           CPSIB   RR10^,RR2^,R7,NE;
1746    BB22 07AE           CPSI    RR10^,RR2^,R7,NE;
174A
174A
174A                        % COMPARE STRING, INC. AND REPEAT
174A    BA26 07AE           CPSIRB  RR10^,RR2^,R7,NE;
174E    BB26 07AE           CPSIR   RR10^,RR2^,R7,NE;
1752
1752                        EJECT;
```

6-8. TRANSLATE INSTRUCTIONS

Instruction		Addressing Modes for dst or src	Description
TRDB	dst,src,r	IR	Translate and Decrement dst <== src(dst) Autoincrement dst r <== r - 1
TRDRB	dst,src,r	IR	Translate, Decrement and Repeat dst <== src(dst) Autodecrement dst r <== r - 1 (repeat until r = 0)
TRIB	dst,src,r	IR	Translate and Increment dst <== src(dst) Autoincrement dst r <== r - 1
TRIRB	dst,src,r	IR	Translate, Increment and Repeat dst <== src(dst) Autoincrement dst r <== r - 1 (repeat until r = 0)

```
4676                            TITLE   'TRANSLATE';
4676
4676                            % TRANSLATE AND DECREMENT
4676    B8B8 0620               TRDB    R11^,R2^,R6;
467A
467A                            % TRANSLATE, DECREMENT, AND REPEAT
467A    B8BC 0620               TRDRB   R11^,R2^,R6;
467E
467E
467E
467E
467E                            % TRANSLATE AND INCREMENT
467E    B8B0 0620               TRIB    R11^,R2^,R6;
4682
4682                            % TRANSLATE, INCREMENT AND REPEAT
4682    B8B4 0620               TRIRB   R11^,R2^,R6;
4686
```

```
1752                            TITLE   'TRANSLATE';
1752
1752                            % TRANSLATE AND DECREMENT
1752    B8A8 0620               TRDB    RR10^,RR2^,R6;
1756
1756
1756
1756                            % TRANSLATE, DECREMENT, AND REPEAT
1756    B8AC 0620               TRDRB   RR10^,RR2^,R6;
175A
175A
175A
175A
175A                            % TRANSLATE AND INCREMENT
175A    B8A0 0620               TRIB    RR10^,RR2^,R6;
175E
175E
175E
175E                            % TRANSLATE, INCREMENT AND REPEAT
175E    B8A4 0620               TRIRB   RR10^,RR2^,R6;
1762
1762
```

TRTDB	src1,src2,r	IR	Translate and Test, Decrement RH1 <== src2(src1) Autodecrement src1 r <== r - 1
TRTDRB	src1,src2,r	IR	Translate and Test, Decrement and Repeat RH1 <== src2(src1) Autodecrement src1 r <== r - 1 (repeat until r = 0 or RH1 = 0)
TRTIB	src1,src2,r	IR	Translate and Test, Increment RH1 <== src2(src1) Autoincrement src1 r <== r - 1
TRTIRB	src1,src2,r	IR	Translate and Test, Increment and Repeat RH1 <== src2(src1) Autoincrement src1 r <== r - 1 (repeat until r = 0 or RH1 = 0)

```
4686                            % TRANSLATE AND TEST, DECREMENT
4686    B8BA 0620               TRTDB   R11^,R2^,R6;
468A
468A                            % TRANSLATE AND TEST, DEC. AND REPEAT
468A    B8BE 062E               TRTDRB  R11^,R2^,R6;
468E
468E                            % TRANSLATE AND TEST, INCREMENT
468E    B8B2 0620               TRTIB   R11^,R2^,R6;
4692
4692                            % TRANSLATE AND TEST, INC. AND REPEAT
4692    B8B6 062E               TRTIRB  R11^,R2^,R6;
```

```
1762
1762
1762                            % TRANSLATE AND TEST, DECREMENT
1762    B8AA 0620               TRTDB   RR10^,RR2^,R6;
1766
1766
1766
1766                            % TRANSLATE AND TEST, DEC. AND REPEAT
1766    B8AE 062E               TRTDRB  RR10^,RR2^,R6;
176A
176A
176A
176A
176A
176A                            % TRANSLATE AND TEST, INCREMENT
176A    B8A2 0620               TRTIB   RR10^,RR2^,R6;
176E
176E
176E
176E                            % TRANSLATE AND TEST, INC. AND REPEAT
176E    B8A6 062E               TRTIRB  RR10^,RR2^,R6;
1772
1772                            EJECT;
```

6-9. INPUT/OUTPUT INSTRUCTIONS

Privileged instructions are marked with **. The input/output instructions are all privileged.

Instruction		Addressing Modes for dst or src		Description
INB IN	r,src	PA,PR	**	Input r <== src
INDB IND	dst,src,r	IR for dst PR for src	**	Input and Decrement dst <== src Autodecrement dst r <== r - 1
INDRB INDR	dst,src,r	IR for dst PR for src	**	Input, Decrement and Repeat dst <== src Autodecrement dst r <== r - 1 (repeat until r = 0)
INIB INI	dst,src,r	IR for dst PR for src	**	Input and Increment dst <== src Autoincrement dst r <== r - 1
INIRB INIR	dst,src,r	IR for dst PR for src	**	Input, Increment and Repeat dst <== src Autoincrement dst r <== r - 1 (repeat until r = 0)

```
4696
4696                            TITLE   'INPUT/OUTPUT';
4696
4696                            % INPUT
4696    3A44 0FC0               INB     RH4,#0FC0;
469A    3CD4                    INB     RH4,R13;
469C    3B44 0FC0               IN      R4,#0FC0;
46A0    3DD4                    IN      R4,R13;
46A2
46A2                            % INPUT AND DECREMENT
46A2    3AD8 0928               INDB    R2^,R13,R9;
46A6    3BD8 0928               IND     R2^,R13,R9;
46AA
46AA                            % INPUT, DECREMENT AND REPEAT
46AA    3AD8 0920               INDRB   R2^,R13,R9;
46AE    3BD8 0920               INDR    R2^,R13,R9;
46B2
46B2
46B2                            % INPUT AND INCREMENT
46B2    3AD0 0928               INIB    R2^,R13,R9;
46B6    3BD0 0928               INI     R2^,R13,R9;
46BA
46BA                            % INPUT, INCREMENT AND REPEAT
46BA    3AD0 0920               INIRB   R2^,R13,R9;
46BE    3BD0 0920               INIR    R2^,R13,R9;
```

```
1772                            TITLE   'INPUT/OUTPUT';
1772
1772                            % INPUT
1772    3A44 0FC0               INB     RH4,#0FC0;
1776    3CD4                    INB     RH4,R13;
1778    3B44 0FC0               IN      R4,#0FC0;
177C    3DD4                    IN      R4,R13;
177E
177E                            % INPUT AND DECREMENT
177E    3AD8 0928               INDB    RR2^,R13,R9;
1782    3BD8 0928               IND     RR2^,R13,R9;
1786
1786
1786                            % INPUT, DECREMENT AND REPEAT
1786    3AD8 0920               INDRB   RR2^,R13,R9;
178A    3BD8 0920               INDR    RR2^,R13,R9;
178E
178E
178E
178E                            % INPUT AND INCREMENT
178E    3AD0 0928               INIB    RR2^,R13,R9;
1792    3BD0 0928               INI     RR2^,R13,R9;
1796
1796
1796                            % INPUT, INCREMENT AND REPEAT
1796    3AD0 0920               INIRB   RR2^,R13,R9;
179A    3BD0 0920               INIR    RR2^,R13,R9;
179E
179E
```

```
OUTB    dst,r       PA,PR           ** Output
OUT                                    dst <== r


OUTDB   dst,src,r   PR for dst      ** Output and Decrement
OUTD                IR for src         dst <== src
                                       Autodecrement src
                                       r <== r - 1


OTDRB   dst,src,r   PR for dst      ** Output, Decrement and Repeat
OTDR                IR for src         dst <== src
                                       Autodecrement src
                                       r <== r - 1
                                       (repeat until r = 0)


OUTIB   dst,src,r   PR for dst      ** Output and Increment
OUTI                IR for src         dst <== src
                                       Autoincrement src
                                       r <== r - 1


OTIRB   dst,src,r   PR for dst      ** Output, Increment and Repeat
OTIR                IR for src         dst <== src
                                       Autoincrement src
                                       r <== r - 1
                                       (repeat until r = 0)
```

```
46C2
46C2                            % OUTPUT
46C2    3A46 0FC0               OUTB    #0FC0,RH4;
46C6    3ED4                    OUTB    R13,RH4;
46C8    3B46 0FC0               OUT     #0FC0,R4;
46CC    3FD4                    OUT     R13,R4;
46CE
46CE                            % OUTPUT AND DECREMENT
46CE    3A2A 09D8               OUTDB   R13,R2^,R9;
46D2    3B2A 09D8               OUTD    R13,R2^,R9;
46D6
46D6                            % OUTPUT, DECREMENT AND REPEAT
46D6    3A2A 09D0               OTDRB   R13,R2^,R9;
46DA    3B2A 09D0               OTDR    R13,R2^,R9;
46DE
46DE                            % OUTPUT AND INCREMENT
46DE    3A22 09D8               OUTIB   R13,R2^,R9;
46E2    3B22 09D8               OUTI    R13,R2^,R9;
46E6
46E6                            % OUTPUT, INCREMENT AND REPEAT
46E6    3A22 09D0               OTIRB   R13,R2^,R9;
46EA    3B22 09D0               OTIR    R13,R2^,R9;
46EE
46EE
```

```
179E
179E                            % OUTPUT
179E    3A46 0FC0               OUTB    #0FC0,RH4;
17A2    3ED4                    OUTB    R13,RH4;
17A4    3B46 0FC0               OUT     #0FC0,R4;
17A8    3FD4                    OUT     R13,R4;
17AA
17AA                            % OUTPUT AND DECREMENT
17AA    3A2A 09D8               OUTDB   R13,RR2^,R9;
17AE    3B2A 09D8               OUTD    R13,RR2^,R9;
17B2
17B2                            EJECT;
17B2                            % OUTPUT, DECREMENT AND REPEAT
17B2    3A2A 09D0               OTDRB   R13,RR2^,R9;
17B6    3B2A 09D0               OTDR    R13,RR2^,R9;
17BA
17BA
17BA
17BA                            % OUTPUT AND INCREMENT
17BA    3A22 09D8               OUTIB   R13,RR2^,R9;
17BE    3B22 09D8               OUTI    R13,RR2^,R9;
17C2
17C2
17C2                            % OUTPUT, INCREMENT AND REPEAT
17C2    3A22 09D0               OTIRB   R13,RR2^,R9;
17C6    3B22 09D0               OTIR    R13,RR2^,R9;
17CA
17CA
```

```
SINB     r,src      PA              ** Special Input
SIN                                    r <== src


SINDB    dst,src,r  IR for dst      ** Special Input and Decrement
SIND                PR for src         dst <== src
                                       Autodecrement dst
                                       r <== r - 1


SINDRB   dst,src,r  IR for dst      ** Special Input, Decrement
SINDR               PR for src              and Repeat
                                       dst <== src
                                       Autodecrement dst
                                       r <== r - 1
                                       (repeat until r = 0)


SINIB    dst,src,r  IR for dst      ** Special Input and Increment
SINI                PR for src         dst <== src
                                       Autoincrement dst
                                       r <== r - 1


SINIRB   dst,src,r  IR for dst      ** Special Input, Increment
SINIR               PR for src              and Repeat
                                       dst <== src
                                       Autoincrement dst
                                       r <== r - 1
                                       (repeat until r = 0)
```

```
46EE                        % SPECIAL INPUT
46EE   3A45 0FC0            SINB    RH4,#0FC0;
46F2   3B45 0FC0            SIN     R4,#0FC0;
46F6
46F6                        % SPECIAL INPUT AND DECREMENT
46F6   3AD9 0928            SINDB   R2^,R13,R9;
46FA   3BD9 0928            SIND    R2^,R13,R9;
46FE
46FE                        % SPECIAL INPUT, DECREMENT AND REPEAT
46FE   3AD9 0920            SINDRB  R2^,R13,R9;
4702   3BD9 0920            SINDR   R2^,R13,R9;
4706
4706                        % SPECIAL INPUT AND INCREMENT
4706   3AD1 0928            SINIB   R2^,R13,R9;
470A   3BD1 0928            SINI    R2^,R13,R9;
470E
470E                        % SPECIAL INPUT, INCREMENT AND REPEAT
470E   3AD1 0920            SINIRB  R2^,R13,R9;
4712   3BD1 0920            SINIR   R2^,R13,R9;
4716
```

```
17CA
17CA                        % SPECIAL INPUT
17CA   3A45 0FC0            SINB    RH4,#0FC0;
17CE   3B45 0FC0            SIN     R4,#0FC0;
17D2
17D2                        % SPECIAL INPUT AND DECREMENT
17D2   3AD9 0928            SINDB   RR2^,R13,R9;
17D6   3BD9 0928            SIND    RR2^,R13,R9;
17DA
17DA
17DA                        % SPECIAL INPUT, DECREMENT AND REPEAT
17DA   3AD9 0920            SINDRB  RR2^,R13,R9;
17DE   3BD9 0920            SINDR   RR2^,R13,R9;
17E2
17E2
17E2
17E2
17E2                        % SPECIAL INPUT AND INCREMENT
17E2   3AD1 0928            SINIB   RR2^,R13,R9;
17E6   3BD1 0928            SINI    RR2^,R13,R9;
17EA
17EA
17EA                        % SPECIAL INPUT, INCREMENT AND REPEAT
17EA   3AD1 0920            SINIRB  RR2^,R13,R9;
17EE   3BD1 0920            SINIR   RR2^,R13,R9;
17F2
17F2                        EJECT;
```

```
SOUTB   dst,r       PA              ** Special Output
SOUT                                   dst <== r


SOUTDB  dst,src,r   PR for dst      ** Special Output and Decrement
SOUTD               IR for src         dst <== src
                                       Autodecrement src
                                       r <== r - 1


SOTDRB  dst,src,r   PR for dst      ** Special Output, Decrement
SOTDR               IR for src              and Repeat
                                       dst <== src
                                       Autodecrement src
                                       r <== r - 1
                                       (repeat until r = 0)


SOUTIB  dst,src,r   PR for dst      ** Special Output and Increment
SOUTI               IR for src         dst <== src
                                       Autoincrement src
                                       r <== r - 1


SOTIRB  dst,src,r   PR for dst      ** Special Output, Increment
SOTIR               IR for src              and Repeat
                                       dst <== src
                                       Autoincrement src
                                       r <== r - 1
                                       (repeat until r = 0)
```

```
4716                            % SPECIAL OUTPUT
4716    3A47 0FC0               SOUTB   #0FC0,RH4;
471A    3B47 0FC0               SOUT    #0FC0,R4;
471E
471E                            % SPECIAL OUTPUT AND DECREMENT
471E    3A2B 09D8               SOUTDB  R13,R2^,R9;
4722    3B2B 09D8               SOUTD   R13,R2^,R9;
4726
4726                            % SPECIAL OUTPUT, DECREMENT AND REPEAT
4726    3A2B 09D0               SOTDRB  R13,R2^,R9;
472A    3B2B 09D0               SOTDR   R13,R2^,R9;
472E
472E                            % SPECIAL OUTPUT AND INCREMENT
472E    3A23 09D8               SOUTIB  R13,R2^,R9;
4732    3B23 09D8               SOUTI   R13,R2^,R9;
4736
4736                            % SPECIAL OUTPUT, INCREMENT AND REPEAT
4736    3A23 09D0               SOTIRB  R13,R2^,R9;
473A    3B23 09D0               SOTIR   R13,R2^,R9;
```

```
17F2                            % SPECIAL OUTPUT
17F2    3A47 0FC0               SOUTB   #0FC0,RH4;
17F6    3B47 0FC0               SOUT    #0FC0,R4;
17FA
17FA                            % SPECIAL OUTPUT AND DECREMENT
17FA    3A2B 09D8               SOUTDB  R13,RR2^,R9;
17FE    3B2B 09D8               SOUTD   R13,RR2^,R9;
1802
1802
1802                            % SPECIAL OUTPUT, DECREMENT AND REPEAT
1802    3A2B 09D0               SOTDRB  R13,RR2^,R9;
1806    3B2B 09D0               SOTDR   R13,RR2^,R9;
180A
180A
180A
180A
180A                            % SPECIAL OUTPUT AND INCREMENT
180A    3A23 09D8               SOUTIB  R13,RR2^,R9;
180E    3B23 09D8               SOUTI   R13,RR2^,R9;
1812
1812
1812                            % SPECIAL OUTPUT, INCREMENT AND REPEAT
1812    3A23 09D0               SOTIRB  R13,RR2^,R9;
1816    3B23 09D0               SOTIR   R13,RR2^,R9;
181A
181A                            EJECT;
```

6-10. PROGRAM CONTROL INSTRUCTIONS

Privileged instructions are marked with **.

Instruction		Addressing Modes for dst or src		Description
CALL	dst	IR,DA,X		Call Autodecrement SP SP^ <== PC PC <== dst
CALR	dst	RA		Call Relative Autodecrement SP SP^ <== PC PC <== PC + dst (dst is -4092 to +4098 bytes)
DBJNZ DJNZ	r,dst	RA		Decrement and Jump if Nonzero r <== r - 1 If r ≠ 0: PC <== PC + dst (dst is -252 to 2 bytes; flags are not affected)
IRET	-	-	**	Interrupt return PS <== SP^ Autoincrement SP

```
473E
473E                        TITLE   'PROGRAM CONTROL';
473E
473E                        LAB2:
473E                        % DEFINED FOR RA OPERANDS
473E
473E                        % CALL
473E   1F20                 CALL    R2^;
4740   5F00 4200            CALL    LAB;
4744   5F10 4200            CALL    LAB(R1);
4748
4748                        % CALL RELATIVE
4748   D006                 CALR    LAB2;
474A
474A                        % DECREMENT AND JUMP IF NONZERO
474A   FF07                 DBJNZ   RL7,LAB2;
474C   F788                 DJNZ    R7,LAB2;
474E
474E                        % INTERRUPT RETURN
474E   7B00                 IRET;
4750
```

```
181A                        TITLE   'PROGRAM CONTROL ;
181A
181A                        LAB2:
181A                        % DEFINED FOR RA OPERANDS
181A
181A                        % CALL
181A   1F20                 CALL    RR2^;
181C   5F00S0001 1000       CALL    LAB;
1822   5F00P0100            CALL    LAB_SSO;
1826   5F10S0001 1000       CALL    LAB(R1);
182C   5F10P0100            CALL    LAB_SSO(R1);
1830
1830                        % CALL RELATIVE
1830   D00C                 CALR    LAB2;
1832
1832
1832
1832
1832                        % DECREMENT AND JUMP IF NONZERO
1832   FF0D                 DBJNZ   RL7,LAB2;
1834   F78E                 DJNZ    R7,LAB2;
1836
1836
1836
1836                        % INTERRUPT RETURN
1836   7B00                 IRET;
1838
```

JP	cc,dst	IR,DA,X	Jump If cc true: PC <== dst (cc is optional)
JR	cc,dst	RA	Jump Relative If cc true: PC <== PC + 2 + dst (cc is optional; dst is -254 to +256 bytes)
RET	cc	-	Return If cc true: PC <== SP^ Autoincrement SP (cc is optional)
SC	exp	-	System Call Autodecrement SP SP^ <== old PS Push instruction PS <== syst

```
4750                        % JUMP
4750    1E2E                JP      NZ,R2^;
4752    1E28                JP      R2^;
4754    5E0E 4200           JP      NZ,LAB;
4758    5E08 4200           JP      LAB;
475C    5E1E 4200           JP      NZ,LAB(R1);
4760    5E18 4200           JP      LAB(R1);
4764
4764                        % JUMP RELATIVE
4764    EEEC                JR      NZ,LAB2;
4766    E8EB                JR      LAB2;
4768
4768
4768                        % RETURN
4768    9E0E                RET     NZ;
476A    9E08                RET;
476C
476C
476C                        % SYSTEM CALL
476C    7F2C                SC      44;
476E
```

```
1838
1838                        % JUMP
1838    1E2E                JP      NZ,RR2^;
183A    1E28                JP      RR2^;
183C    5E0ES0001 1000      JP      NZ,LAB;
1842    5E0EP0100           JP      NZ,LAB_SSO;
1846    5E08S0001 1000      JP      LAB;
184C    5E08P0100           JP      LAB_SSO;
1850    5E1ES0001 1000      JP      NZ,LAB(R1);
1856    5E1EP0100           JP      NZ,LAB_SSO(R1);
185A    5E18S0001 1000      JP      LAB(R1);
1860    5E18P0100           JP      LAB_SSO(R1);
1864
1864                        % JUMP RELATIVE
1864    EEDA                JR      NZ,LAB2;
1866    E8D9                JR      LAB2;
1868
1868
1868                        % RETURN
1868    9E0E                RET     NZ;
186A    9E08                RET;
186C
186C
186C                        % SYSTEM CALL
186C    7F2C                SC      44;
186E
186E                        EJECT;
```

6-11. CPU CONTROL INSTRUCTIONS

Privileged instructions are marked with **.

Instruction		Addressing Modes for dst or src	Description
COMFLG	flags	-	Complement Flags (flags is a list of CY, ZR, SGN, PY, or OV)
DI	ints	-	** Disable Interrupt (ints is a list of NVI or VI)
EI	ints	-	** Enable Interrupt (ints is a list of NVI or VI)
HALT	-	-	** Halt
LDCTL	ctlr,src	R	** Load Control Register ctlr <== src (ctlr is FCW, PSAPSEG, PSAPOFF, NSPSEG, NSPOFF, or REFRESH)
LDCTL	dst,ctlr	R	** Load from Control Register dst <== ctlr (ctlr is FCW, PSAPSEG, PSAPOFF, NSPSEG, NSPOFF, or REFRESH)

```
476E                    TITLE   'CPU CONTROL';
476E
476E                    % COMPLEMENT FLAGS
476E   8DC5               COMFLG  CY,ZR;
4770
4770
4770                    % DISABLE INTERRUPT
4770   7C01               DI      VI;
4772
4772                    % ENABLE INTERRUPT
4772   7C04               EI      NVI,VI;
4774
4774                    % HALT
4774   7A00               HALT;
4776
4776                    % LOAD CONTROL REGISTER
4776   7DCA               LDCTL   FCW,R12;
4778
4778                    % LOAD FROM CONTROL REGISTER
4778   7DC2               LDCTL   R12,FCW;
477A
```

```
186E                    TITLE   'CPU CONTROL';
186E
186E                    % COMPLEMENT FLAGS
186E   8DC5               COMFLG  CY,ZR;
1870
1870
1870                    % DISABLE INTERRUPT
1870   7C01               DI      VI;
1872
1872                    % ENABLE INTERRUPT
1872   7C04               EI      NVI,VI;
1874
1874                    % HALT
1874   7A00               HALT;
1876
1876                    % LOAD CONTROL REGISTER
1876   7DCA               LDCTL   FCW,R12;
1878
1878
1878
1878                    % LOAD FROM CONTROL REGISTER
1878   7DC2               LDCTL   R12,FCW;
187A
187A
```

LDCTLB	FLAGS,src	R	Load Flag Byte FLAGS <== src
LDCTLB	dst,FLAGS	R	Load from Flag Byte dst <== FLAGS
LDPS	src	IR,DA,X	** Load Program Status PS <== src
MBIT	-	-	** Multi-micro Test
MREQ	r	-	** Multi-micro Request
MRES	-	-	** Multi-micro Reset
MSET	-	-	** Multi-micro Set

```
477A                        % LOAD FLAG BYTE
477A    8C79                LDCTLB  FLAGS,RH7;
477C
477C                        % LOAD FROM FLAG BYTE
477C    8C71                LDCTLB  RH7,FLAGS;
477E
477E                        % LOAD PROGRAM STATUS
477E    3920                LDPS    R2^;
4780    7900 4200           LDPS    LAB;
4784    7910 4200           LDPS    LAB(R1);
4788
4788                        % MULTI-MICRO TEST
4788    7B0A                MBIT;
478A
478A                        % MULTI-MICRO REQUEST
478A    7BCD                MREQ    R12;
478C
478C                        % MULTI-MICRO RESET
478C    7B09                MRES;
478E
478E                        % MULTI-MICRO SET
478E    7B08                MSET;
4790
```

```
187A
187A                        % LOAD FLAG BYTE
187A    8C79                LDCTLB  FLAGS,RH7;
187C
187C                        % LOAD FROM FLAG BYTE
187C    8C71                LDCTLB  RH7,FLAGS;
187E
187E                        % LOAD PROGRAM STATUS
187E    3920                LDPS    RR2^;
1880    7900S0001 1000      LDPS    LAB;
1886    7900P0100           LDPS    LAB_SSO;
188A    7910S0001 1000      LDPS    LAB(R1);
1890    7910P0100           LDPS    LAB_SSO(R1);
1894
1894                        % MULTI-MICRO TEST
1894    7B0A                MBIT;
1896
1896                        % MULTI-MICRO REQUEST
1896    7BCD                MREQ    R12;
1898
1898                        % MULTI-MICRO RESET
1898    7B09                MRES;
189A
189A                        % MULTI-MICRO SET
189A    7B08                MSET;
189C
189C                        EJECT;
```

NOP	-	-	No Operation
RESFLG	flags	-	Reset Flags (flags is a list of CY, ZR, SGN, PY, or OV)
SETFLG	flags	-	Set Flags (flags is a list of CY, ZR, SGN, PY, or OV)

```
4790                    % NO OPERATION
4790    8D07            NOP;
4792
4792                    % RESET FLAGS
4792    8D43            RESFLG  ZR;
4794
4794
4794                    % SET FLAGS
4794    8D71            SETFLG  ZR,SGN,OV;
4796
4796
4796                    END.
```

```
189C                    % NO OPERATION
189C    8D07            NOP;
189E
189E                    % RESET FLAGS
189E    8D43            RESFLG  ZR;
18A0
18A0
18A0                    % SET FLAGS
18A0    8D71            SETFLG  ZR,SGN,OV;
18A2
18A2
18A2                    END.
```

APPENDIX A
ASCII CHARACTER SET

The ASCII character set is shown in table A-1.

TABLE A-1. ASCII

Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
00	0	NUL	20	32	SP	40	64	@	60	96	
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	EOT	24	36	$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	'	47	71	G	67	103	g
08	8	BS	28	40	(	48	72	H	68	104	h
09	9	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	DEL

APPENDIX B
ERROR MESSAGES

The MACRO8000 error messages are error descriptions rather than error numbers. The first part of the error message specifies:

(WARNING)

or indicates the functional part of the assembler that generated the message:

(ADDRESS RESOLVER)
(CODE GENERATOR)
(INPUT CONVERT)
(LEXICAL)
(FORM)
(DEFINE LABEL)
(STATEMENT)
(EVAL)
(SYNTAX PROCESSOR)
(FOR LOOP)
(EXECUTIVE)

The second part of the message describes the specific problem or provides additional information:

RELATIVE ADDRESS OUT OF RANGE: value
ODD ADDRESS BOUNDARY DETECTED: value
UNDEFINED LABEL: value
UNDEFINED MACRO: value
INVALID NUMBER OF OPERANDS: value
MISSING FLAG DESIGNATOR: value
MISSING CONDITION CODE FLAG: value
MISSING CONDITION CODE: value
INVALID MODE: value
INVALID REGISTER DESIGNATOR: value
STRING TOO LONG: value
MISSING OR INVALID IMMEDIATE (CONSTANT) OPERAND: value
MISSING VI OR NVI: value
IMMEDIATE OPERAND TOO LARGE
DIGIT EXCEEDS RADIX
RADIX EQ 0
MISSING (’): END OF LINE OR INVALID CHAR (LT #20)
RADIX TOO LARGE
INVALID NUMBER FORMAT
MISSING OR INVALID OPERAND: value
MISSING)
MISSING]
LABEL SAME AS MACRO OR OPCODE
INVALID LABEL IDENTIFIER: value
UNRECOGNIZED STATEMENT FORM: value
INVALID INITIALIZATION VALUE: value

```
INVALID LOCATION COUNTER RESET:  value
NOT YET IMPLEMENTED
MISSING END
MISSING OR INVALID STRING
MISSING =
INVALID IN MACRO BODY
MISSING OR INVALID INTEGER
INVALID MACRO STATEMENT
MISSING OR INVALID IDENTIFIER
REDEFINITION OF IDENTIFIER:  value
INVALID UNARY ^ OPERAND:  value
DIVISION BY 0
MISSING DELIMITER:  value
INVALID DEFINITION
UNDEFINED EXPRESION:  value
MISSING CONDITION CODE:  value
MISSING END. (OR EXTRA END)
INVALID STATEMENT BEGINNER:  value
LOCATION COUNTER ADJUSTED TO EVEN ADDRESS BOUNDARY
MISSING STATEMENT TERMINATOR:  value
MISSING OR INVALID MODULE STATEMENT
MISSING OR INVALID SEGMENT ATTRIBUTE SET:  value
MISSING OR INVALID SEGMENT NAME:  value
INVALID PORT REGISTER:  value
SEGMENT STACK UNDERFLOW
SEGMENT STACK OVERFLOW
INVALID STATEMENT - NON-RELOCATABLE MODE
FATAL ERROR - ASSEMBLY TERMINATED
MISSING OR INVALID CONST OBJECT
PRODUCT CALL OVERRIDE
ERROR COUNT OVERFLOW
ILLEGAL CHARACTER:
HEX CONSTANT TOO LARGE:
MACRO STACK OVERFLOW:
FATAL ERROR - ASSEMBLY TERMINATED
FILE STACK ERROR
ERROR IN EXTENDING FILE -
FILE SPACE OVERFLOW -
DIRECTORY OVERFLOW -
FILE CLOSE ERROR -
ATTEMPT TO READ UNWRITTEN DATA -
ATTEMPT TO READ BEYOND EOF -
UNABLE TO OPEN INPUT FILE -
OBJECT SPACE OVERFLOW
RELATIVE ADDRESS OUT OF RANGE:
ODD WORD BOUNDARY DETECTED:
UNDEFINED LABEL:
CONFLICTING OPTION - RELOCATABLE FILE CREATED
INVALID OPTION(S)
EMPTY INPUT FILE
ILLEGAL FILENAME
MISSING ; OR END
MISSING : OR END
UNRECOGNIZED STATEMENT FORM
```

APPENDIX C
NOTATIONS

This appendix is intended for quick look-up of terms and also as a BNF (Backus-Naur Form) description of the MACRO8000 assembler.

Notations are used throughout this manual, and a notation like exp (for numeric expression) or cc (for condition code) means the same thing whenever it appears.

Low-level items:

```
<identifier>            ::=  <first character>
                             | <first character><character sequence>

<first character>       ::=  <letter> | @

<letter>                ::=  ...letters A through Z and a through z

<character sequence>    ::=  <character>
                             | <character><character sequence>

<character>             ::=  <letter> | @ | <digit> | _

<digit>                 ::=  ...digits 0 through 9

<delimiter>             ::=  <space> | , | ( | )
                             | BEGIN | END | THEN | ELSE | IN | DO

<space>                 ::=  ...character blank

<comment>               ::=  %text | (*text*)

<text>                  ::=  ...any ASCII characters

<constant>              ::=  nnnn | nnnnD          ...character _ allowed
                             | nnnnB                  as noise character
                             | nnnnO | nnnnQ
                             | nnnnH | #nnnn
                             | nnnnK
                             | b#nnnn

<string>                ::=  'text' | <string> & <string>

<list>                  ::=  (<operand list>)
```

Items used in constructing valid statements:

```
<module>                ::=  <MODULE directive>;
                             <statement sequence>
                             END.
```

```
<statement sequence>    ::=  <statement>
                             | <statement> ; <statement sequence>

<statement>             ::=  <label name>: <statement>
                             | <single statement>
                             | <compound statement>

<label name>            ::=  <identifier>        ...any except reserved
                                                    statement beginner

<single statement>      ::=  <statement beginner> <operand list>
                             | <statement beginner>

<compound statement>    ::=  BEGIN
                             <statement sequence>
                             END

<statement beginner>    ::=  <redefinable beginner>
                             | <reserved beginner>
                             | <defined beginner>

<redefinable beginner>  ::=  <opcode>
                             | PROGRAM | ORIGIN
                             | TITLE   | PAGE     | EJECT   | INCLUDE
                             | NOLIST  | LIST
                             | MODULE  | HEADER
                             | GLOBAL  | EXTERNAL | SEGMENT
                             | PAGE0   | EXT_PG0  | GLB_PG0 | MOD_NS
                             | MOD_SEG |

<opcode>                ::=  <identifier>  ...AmZ8000 instruction
                              mnemonic

<reserved beginner>     ::=  BYTE | WORD | LONG | STRING
                             | CONST | VAR | MACRO
                             | IF | FOR

<defined beginner>      ::=  <macro name>

<macro name>            ::=  <identifier>  ...any except reserved
                                              statement beginner

<operand list>          ::=  <operand>
                             | <operand>, <operand list>

<operand>               ::=  <exp>               ...IM operand
                             | <reg>             ...R operand
                             | <rv>^             ...IR operand
                             | <lab>             ...DA operand
                             | <lab>(<rw>)       ...X operand
                             | <rv>^(<exp>)      ...BA operand
                             | <rv>^(<rw>)       ...BX operand
                             | <lab>             ...RA operand
                             | <address operand>
```

```
                                | <cc>
                                | <flag>
                                | <interrupt>
                                | <control word>
                                | <symbolic constant>
                                | <object variable>
                                | <reserved type>

<address operand>       ::=  ^<lab>                 ...IM operand
                             | ^<lab>(<rw>)         ...X operand
                             | ^(<rv>^(<exp>))      ...BA operand
                             | ^(<rv>^(<rw>))       ...BX operand
                             | ^<lab>               ...RA operand

<exp>                   ::=  <constant>
                             | <exp> * <exp>
                             | <exp> / <exp>
                             | <exp> MOD <exp>
                             | <exp> SHR <exp>
                             | <exp> SHL <exp>
                             | - <exp>
                             | <exp> + <exp>
                             | <exp> - <exp>
                             | NOT <exp>
                             | <exp> AND <exp>
                             | <exp> OR <exp>
                             | <exp> XOR <exp>

<reg>                   ::=  <rb> | <rw> | <rr> | <rq> | <rv>

<rb>                    ::=  RH0 | RL0              ...byte register
                             | RH1 | RL1
                             | RH2 | RL2
                             | RH3 | RL3
                             | RH4 | RL4
                             | RH5 | RL5
                             | RH6 | RL6
                             | RH7 | RL7

<rw>                    ::=  R0 | R1 | R2 | R3     ...word register
                             | R4 | R5 | R6 | R7
                             | R8 | R9 | R10 | R11
                             | R12 | R13 | R14 | R15

<rr>                    ::=  RR0 | RR2              ...register pair
                             | RR4 | RR6
                             | RR8 | RR10
                             | RR12 | RR14

<rq>                    ::=  RQ0 | RQ4              ...register quad
                             | RQ8 | RQ12

<rv>                    ::=  <rw> | <rr>            ...version register:
                                                       word for Z8002,
```

pair for Z8001

```
<lab>                ::=  <identifier>
                          | $                    ...location counter
                          | <lab>(<exp>)         ...label with offset
                          | <constant>^          ...abs. addr. constant
                          | (<exp>)^             ...abs. addr. constant

<addr constant>      ::=  ^<lab>
                          | <addr constant> + <exp>
                          | <addr constant> - <exp>

<cc>                 ::=  TRUE | FALSE
                          | NZ | ZR
                          | NC | CY
                          | PO | PE
                          | PL | MI
                          | NE | EQ
                          | NOV | OV
                          | GE | LT | GT | LE
                          | LGE | LLT | LGT | LLE

<logexp>             ::=  <cc>
                          | <comparison>
                          | NOT <logexp>
                          | <logexp> AND <logexp>   ...not with <cc>
                          | <logexp> OR <logexp>    ...not with <cc>
                          | NULL <object variable>

<comparison>         ::=  <numeric.comp>
                          | <string.comp>
                          | <register.comp>

<numeric.comp>       ::=  <exp> <equality.op> <exp>
                          | <exp> <signed.op> <exp>
                          | <exp> <unsigned.op> <exp>
                          | <exp> <equality.op> <string>  ...string
                          | <exp> <signed.op> <string>       4 chars
                          | <exp> <unsigned.op> <string>     or less

<string.comp>        ::=  <string> <equality.op> <string>
                          | <string> <unsigned.op> <string>

<register.comp>      ::=  <reg> <equality.op> <exp>
                          | <reg> <signed.op> <exp>
                          | <reg> <unsigned.op> <exp>
                          | <reg> <equality.op> <reg>
                          | <reg> <signed.op> <reg>
                          | <reg> <unsigned.op> <reg>

<equality.op>        ::=  EQ | NE

<signed.op>          ::=  GE | LT | GT | LE
```

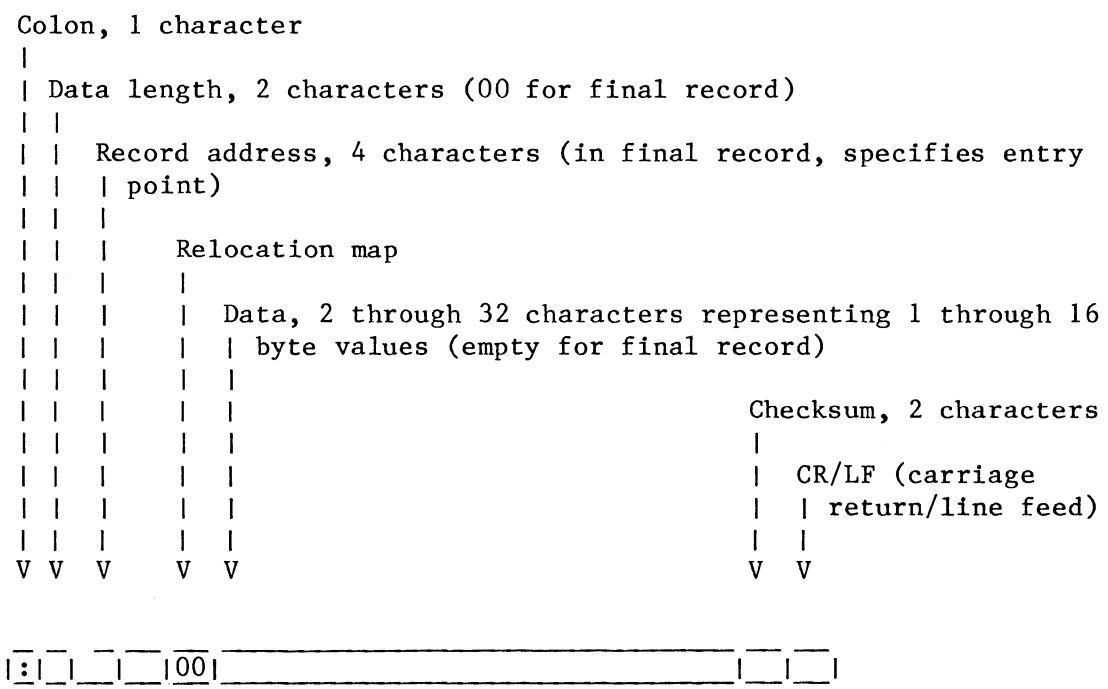
```
<unsigned.op>          ::=  LGE | LLT | LGT | LLE

<flag>                 ::=  CY | ZR | SGN | PY | OV

<interrupt>            ::=  VI | NVI

<control word>         ::=  FLAGS | FCW | PSAPSEG | PSAPOFF
                            | NSPSEG | NSPOFF | REFRESH

<symbolic constant>    ::=  <identifier>        ...any identifier
                                                with an operand value
                                                (except an operator,
                                                delimiter, condition
                                                code, flag, interrupt,
                                                control word, or
                                                reserved type)

<object variable>      ::=  <identifier>        ...any identifier
                                                with an operand value
                                                (except an operator,
                                                delimiter, condition
                                                code, flag, interrupt,
                                                control word, or
                                                reserved type)

<reserved type>        ::=  OBJECT

<operator>             ::=  <arithmetic.op>     ...numeric operations
                            | <equality.op>     ...comparison
                            | <signed.op>       ...comparison
                            | <unsigned.op>     ...comparison
                            | <logical.op>      ...mult. comparisons
                            | <external.op>     ...special EXTERNAL
                                                   Operator
                            | &                 ...concatenate
                            | ::=               ...object variable
                            | NULL              ...object variable

<arithmetic.op>        ::=  * | /
                            | MOD | SHR | SHL
                            | + | -
                            | HIGH | LOW | SWAP | NOT | AND | OR | XOR

<logical.op>           ::=  NOT | AND | OR

<external.op>          ::=  <label>##

Individual directive items:

<PROGRAM directive>    ::=  PROGRAM <lab>            ...entry point

<ORIGIN directive>     ::=  ORIGIN <exp>
                            | ORIGIN <lab>
                            | ORIGIN <addr constant>
```

```
<BYTE directive>        ::=  BYTE (<exp>)              ...reserves space
                             | BYTE: <byte list>      ...defines values

<byte list>             ::=  <byte value>
                             | <byte value>, <byte list>

<byte value>            ::=  <exp>
                             | <string>

<WORD directive>        ::=  WORD (<exp>)              ...reserves space
                             | WORD: <word list>      ...defines values

<word list>             ::=  <word value>
                             | <word value>, <word list>

<word value>            ::=  <exp>
                             | <string>
                             | <lab>
                             | <addr constant>
                             | ^<lab> - ^<lab>   ...if both defined

<LONG directive>        ::=  LONG (<exp>)              ...reserves space
                             | LONG: <long word list> ...defines values

<long word list>        ::=  <long word value>
                             | <long word value>, <long word list>

<long word value>       ::=  <exp>
                             | <string>

<STRING directive>      ::=  STRING: <byte list>       ...saves length

<CONST directive>       ::=  CONST <constant list>

<constant list>         ::=  <constant def>
                             | <constant def>, <constant list>

<constant def>          ::=  <symbolic constant> = <operand>

<VAR directive>         ::=  VAR <variable list>: OBJECT

<variable list>         ::=  <object variable>
                             | <object variable>, <variable list>

<MACRO directive>       ::=  MACRO <macro name>
                             | MACRO <macro name> <parameter list>

<parameter list>        ::=  <parameter>
                             | <parameter>, <parameter list>

<parameter>             ::=  <object variable>
```

```
<IF directive>          ::=  IF <logexp>
                                  THEN <statement>
                             | IF <logexp>
                                  THEN <statement>
                                  ELSE <statement>

<FOR directive>         ::=  FOR <exp> DO
                                  <statement>
                             | FOR <object variable> IN <list> DO
                                  <statement>
                             | FOR <object variable> IN <string> DO
                                  <statement>

<TITLE directive>       ::=  TITLE <string>          ...new listing title

<PAGE directive>        ::=  PAGE <exp>

<EJECT directive>       ::=  EJECT

<INCLUDE directive>     ::=  INCLUDE <string>        ...file name

<NOLIST directive>      ::=  NOLIST

<LIST directive>        ::=  LIST

<MODULE directive>      ::=  MODULE <string>         ...module name

<HEADER directive>      ::=  HEADER <string list>    ...header lines

<string list>           ::=  <string>
                             | <string>, <string list>

<GLOBAL directive>      ::=  GLOBAL <lab>

<EXTERNAL directive>    ::=  EXTERNAL <lab>

<SEGMENT directive>     ::=  SEGMENT <string>        ...segment name
                             | SEGMENT @PRIOR
                             | SEGMENT <seg.attr> <string>

<seg.attr>              ::=  [@CUM] | [@COM]

<EXT_PGO directive>     ::=  EXT_PGO <label list>

<GLB_PG) directive>     ::=  GLB_PGO <label list>

<PAGEO directive>       ::=  PAGEO <label list> ! PAGEO

<label list>            ::=  <label> | <label>, <label list>

<MOD_NS directive>      ::=  MOD_NS

<MOD_SEG directive>     ::=  MOD_SEG
```

APPENDIX D
HEX FILE FORMAT

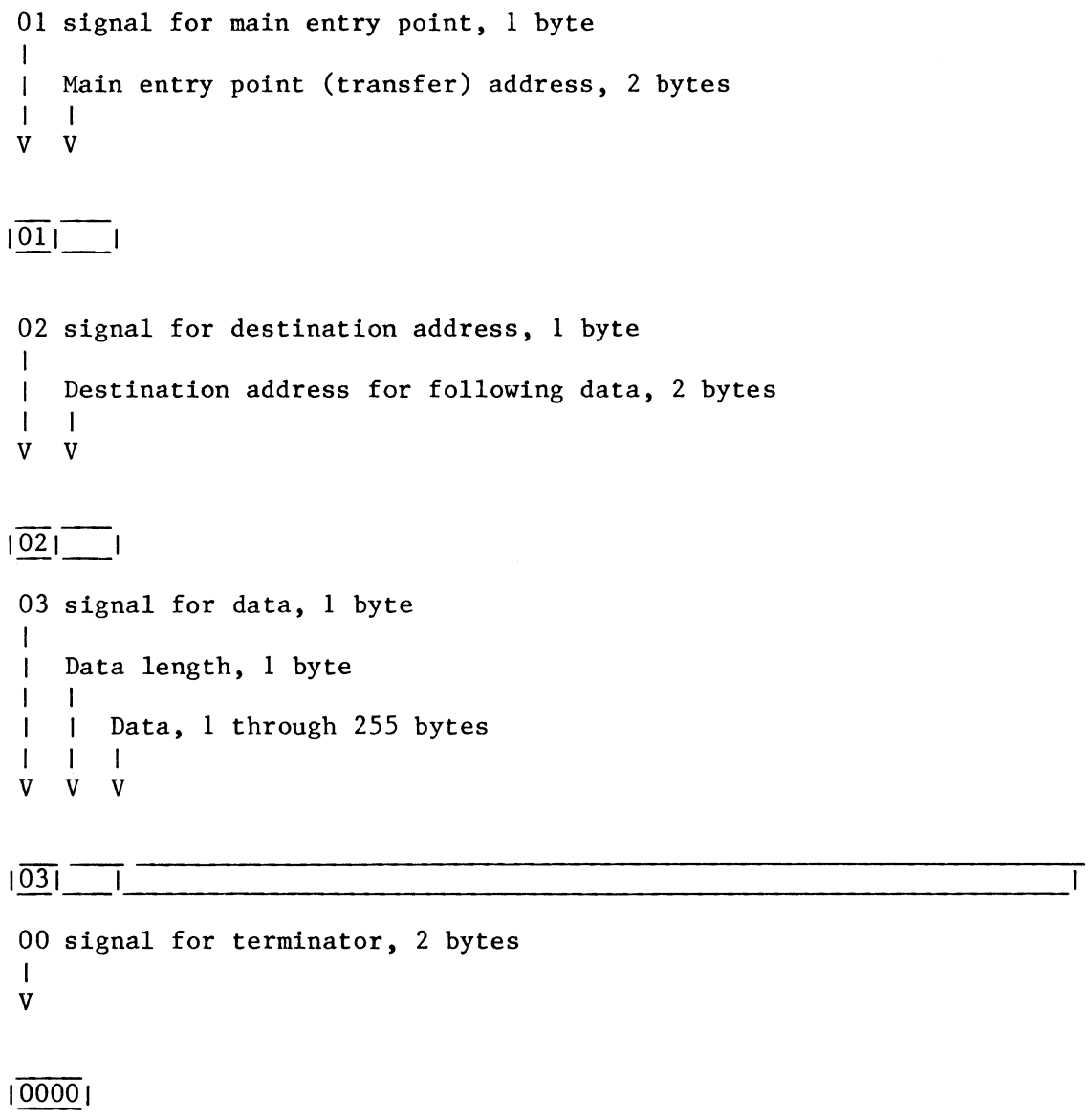
The assembler can produce hex files suitable for putting code into PROMs. Hex file creation is requested with the H option on the MACZ product call, as described in chapter 1. The format of a hex file is the INTEL hex file format:



APPENDIX E
BINARY FILE FORMAT

The assembler can produce binary files suitable for program downloading. Binary file creation is requested with the B option on the MACZ product call, as described in chapter 1. The AMC binary file contains the type of address information found in hex files, but the data is in the more efficient hexadecimal representation.

Each binary file contains an 01 group, an 02 group followed by one or more 03 groups, any additional 02 groups followed by one or more 03 groups, and a 00 terminator. The format of each group is:



INDEX

INDEX (Cont.)

INDEX (Cont.)

INDEX (Cont.)

COMMENT SHEET

Address comments to:

Advanced Micro Computers
Publications Department
3340 Scott Boulevard
Santa Clara, CA 95051

TITLE: MACRO8000 ASSEMBLER USER'S MANUAL
PUBLICATION NO. 00680119F

COMMENTS: (Describe errors, suggested additions or deletions, and include page numbers, etc.)

From: Name: _____ Position: _____

Company: _____

Address: _____

Advanced
Micro
Computers

A subsidiary of
Advanced Micro Devices
3340 Scott Boulevard
Santa Clara,
California 95051
(408) 988-7777
TELEX: 171 142

Printed in U.S.A. 12/80 AMC-567