Advanced
Micro
Computers
Distributed by
Advanced Micro Devices

Am96/4016-ASM
Evaluation Board Assembler

User's Manual

$5.00

REVISION	REVISION RECORD DESCRIPTION
01	Preliminary Issue
(9/21/79)	
A	Manual Released
(11/21/79)	
Publication No. 00680138	

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

This manual provides descriptions of the use, instructions, statement elements and directives used in assembling programs with the Am96/4016-ASM Assembler. The purpose of this Assembler is to produce absolute code which is entered directly into memory on the Am96/4016 Z8000 Evaluation Board for execution.

The user of this manual should be familiar with some type of assembler programming.

Creation of programs with this Assembler involves use of this manual and the following manuals.

- o AmZ8001/2 Processor Instruction Set Manual (Advanced Micro Devices, Inc.)
- o Am96/4016 Evaluation Board manual (Pub. No. 00680131).

The notations used in this manual are:

UPPERCASE	In syntax indicates keywords that are specified as shown
lowercase	In syntax indicates user-specified names or values. The description indicates any restrictions on the name or value supplied.
...	In syntax indicates that an item can be repeated as necessary
. . .	In examples indicates that other parts of the program are not shown.

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

CHAPTER 1
PRODUCT USAGE

1-1. PRODUCT ENVIRONMENT

The Assembler resides in four E-PROM circuits that must be inserted in the following sockets on the Am96/4016 Evaluation Board:

E-PROM	Socket
00250020	U61
00250018	U58
00250021	U62
00250019	U59

The Assembler requires service from the Evaluation Board E-PROM Monitor for execution. Optionally, the Monitor can handle I/O for the Assembler (see the Am96/4016 Evaluation Board Manual). The absolute code generated by the Assembler can be uploaded to an AmSYS 8/8 Development System for permanant Storage.

1-2. PRODUCT FUNCTIONS

The Assembler provides the mechanism for entering symbolic programs into the RAM memory of the Evaluation Board. The Assembler translates mnemonic operation codes, symbolic labels, and symbolic or absolute (hex) operands directly into machine code in a single pass and enter them directly into memory.

This is a limited one-pass assembler which does not generate object code files or save source code after entry (hence, no listing files are generated by the Assembler). Nor are macros, modules, arithmetic or logical expressions or high-level constructs supported. It is, however, upward source-compatible with the MACRO8000 Assembler, which runs on the AmSYS 8/8 Development System, and the AmZ8000 instruction set supported by the MACRO8000 Assembler. In this regard, it provides an excellent alternative to the conventional method of entering evaluation programs in hex code.

The Assembler reads user-supplied symbolic assembly statements from the user's console and assembles each statement as received. Forward symbolic references are satisfied when the forward reference is defined. If the statement is in error, a diagnostic is immediately available, requiring re-entry of the corrected statement. The user's program image is created in memory, as is the symbol (label) table. If the two components exceed available memory, a diagnostic is provided. At the conclusion of assembly (END statement) a diagnostic is provided for each unsatisfied reference (label). Unsatisfied references can be satisfied by entering more code through the assembler Evaluation Board Monitor.

Because of space constraints, a sub-set of the AmZ8000 instructions are implemented. The additional instructions not implemented may be inserted in hex format through the use of byte or word Directive statements.

The assembler produces only nonsegmented 16-bit addresses for the AmZ8002 nonsegmented processor.

1-3. PRODUCT CALL

The Assembler is invoked by one of two forms of the following Evaluation Board Monitor command:

```
ASM
ASM xxxx
```

where xxxx is a four-digit hex address indicating the beginning address for storing the absolute code generated by the Assembler. If the address is not specified, the default is hex 4280.

For example,

```
ASM 4500
```

Due to the display constraints on the optional Am96/4016-KBD Keyboard/ Display Console, a prompt is never displayed for inputs to the Assembler, even when a CRT is used as the system console.

1-4. UP/DOWN LOADING

Absolute code generated by the Assembler can be uploaded to an AmSYS 8/8 Development System for permanent storage on diskette. It can also be subsequently downloaded back to Evaluation Board memory. The Evaluation Board Monitor's SAVE and LOAD commands are used for this. See the Am96/4016 Evaluation Board Manual for details.

1-5. OUTPUT

While the Assembler does not generate lising files, it will display assembled code for each line of input immediately after you press the carriage return key. On a CRT or printer console, this assembled output line will appear just below your corresponding input statement. On an LED Keyboard/Display console, the line might be truncated on the right. The column format for the output is:

```
Columns 1-4:    Memory location of assembled code
Columns 7-20:   Assembled code (hex)
Columns 21 on:  Input statement
```

1-6. SAMPLE PROGRAM

The sample program shown in Figure 1-1 takes data bytes from the message buffer, transfers them to the line buffer and makes a system call to the AmZ8000 Evaluation Board Monitor to display the contents of the line buffer at the console. Program execution is initiated from the monitor by the G xxxx command: xxxx is the starting address of the assembled program.

```
(A)
B:HOST
SYSTEM 8/8 Z8000 HOST
(A)
ASM
(A)
EVAL BD ASM VER 1.0
  LD R8,↑MSG; ADDR OF MESSAGE IN REG 8
(A)
4280  21080000          LD R8,↑MSG; ADDR OF MESSAGE IN REG 8
  LD R9,↑LINE; ADDR OF LINE IN REG 9
(A)
4284  21090000          LD R9,↑LINE; ADDR OF LINE IN REG 9
  LD R7,64; NO. OF BYTES IN MSG BUFFER
(A)
4288  21070040          LD R7,64; NO. OF BYTES IN MSG BUFFER
  LDIRB R9↑,R8↑,R7; MOVE MSG DATA TO LINE BUFFER
(A)
428C  BA810790          LDIRB R9↑,R8↑,R7; MOVE MSG DATA TO LINE BUFFER
  LD R1,↑CBLK; ADDR OF CALL BLOCK
(A)
4290  21010000          LD R1,↑CBLK; ADDR OF CALL BLOCK
  LD R2,↑FILL; PUT ADDR OF FILL IN REG 2
(A)
4294  21020000          LD R2,↑FILL; PUT ADDR OF FILL IN REG 2
  LD R2↑,↑LINE; ADDR OF LINE IN FILL
(A)
4298  0D254286          LD R2↑,↑LINE; ADDR OF LINE IN FILL
  SC 0; PRINT LINE - CALL TO MONITOR
(A)
429C  7F00              SC 0; PRINT LINE - CALL TO MONITOR
  SC #D;
(A)
429E  7F0D              SC #D;
MSG:  WORD: #0D0A,#5448,#4953,#2049,#5320,#5448,#4520,#4E45;
(A)
42A0  0D0A5448495320MSG:  WORD: #0D0A,#5448,#4953,#2049,#5320,#5448,#4520,#4E45;
      WORD: #5720,#414D,#5A38,#3030,#3020,#0D0A,#4556;
(A)
42B0  5720414D5A3830      WORD: #5720,#414D,#5A38,#3030,#3020,#0D0A,#4556;
      WORD: #414C,#5541,#5449,#4F4E,#2042,#4F41,#5244;
(A)
42BE  414C554154494F      WORD: #414C,#5541,#5449,#4F4E,#2042,#4F41,#5244;
      WORD: #2041,#5353,#454D,#424C,#4552,#2021,#0D0A,#0D0A;
(A)
42CC  20415353454D42      WORD: #2041,#5353,#454D,#424C,#4552,#2021,#0D0A,#0D0A;
LINE: BYTE (64); INTERMEDIATE BUFFER
(A)
42DC                LINE: BYTE (64); INTERMEDIATE BUFFER
CBLK: WORD: #0200,0; OUTPUT DATA
(A)
431C  02000000      CBLK: WORD: #0200,0; OUTPUT DATA
FILL: WORD: 0,64; FOR SYSTEM CALL
(A)
4320  00000040      FILL: WORD: 0,64; FOR SYSTEM CALL
  END.
(A)
4320                  END.
PROGRAM EXIT 01
*


G 4280
(A)

THIS IS THE NEW AMZ8000
EVALUATION BOARD ASSEMBLER !


THPROGRAM EXIT 0D
*
```

FIGURE 1-1. Sample Program.

CHAPTER 2
STATEMENT ELEMENTS

2-1. INTRODUCTION

The statements in a program are either MACRO8000 directives (described in chapter 3) or AmZ8000 instructions (described in chapter 4). The general rules for statements are described in this chapter.

2-2. STATEMENTS

A statement must be a simple statement; compound statements are not allowed. A simple statement has one opcode per line and is terminated with a semicolon. For example:

```
ADD  R4,1;
```

You must space over two (2) spaces before entering non-labeled statements.

2-3. SPECIAL CHARACTERS

Certain special characters are used within statements to further describe the instruction or operand being entered. These characters are as follows:

Name	Character	Meaning
Number symbol	#	Denotes a hex constant
Parentheses	()	Enclose a subscript index register of the form Rn, where n must be in the range 1-15.
Circumflex	^	Denotes an address constant if it precedes a label. If the circumflex follows a word register, it denotes an indirect address. If the circumflex follows a constant and is followed by a left parentheses, it denotes the base for an indexed address. If a circumflex alone follows a constant, it denotes a direct address (the contents at the address).
Single Quote	'	Denotes an ASCII character string. The string must be enclosed by single

		quotes. The quote may occur within the string by its appearance twice in succession.
Comma	,	Separates multiple operands
Space		Element separator between label, operation code and operands
Plus or minus	+ -	Unary operators that may optionally precede immediate addresses used as operand values. Counts such as repeat and shift do not permit unary operators.
Colon	:	Denotes end of label
Semi-colon	;	Denotes end of statement

2-4. DELIMITERS

Within statements, the possible delimiters are blanks, commas, and parentheses.

2-5. SYMBOLS

The basic classes of symbols are opcodes, labels, and symbolic constants. Symbols can be as long as 6 characters. The upper-case characters A through Z and Ø through 9 are legal in a symbol; lower-case alpha characters are not allowed. A symbol cannot start with a digit and it cannot have embedded spaces. For example, valid symbols are:

```
LOOP
LABEL5
```

2-6. NUMERIC CONSTANTS

Numeric constants are represented internally as signed 32-bit constants. The notation for different types of numeric constants is as follows:

Forum	Base	Example
nnnn	Decimal	12
#nnnn	Hexadecimal	#A5

2-7. OPCODES

An opcode is one of the AmZ8000 instruction mnemonics listed in chapter 4. It can follow a label after one or more spaces, or start a statement if prededed by two or more spaces.

2-8. LABELS

A label is a symbol that is prefixed to a statement and followed by a colon. It must begin in columns 1 or 2, and be no longer than 6 characters in length with no embedded space. A label is not declared explicitly as a label; usage of the label serves to declare the label. For example, the label NBT1 is defined in the following statement:

```
NBT1:  LD  R1Ø,Ø;
```

Therefore, a statement such as:

```
JR  ZR,NBT1;
```

will cause a jump to the statement labeled NBT1.

2-9. ADDRESS CONSTANTS

A label can be used as an address constant when preceded by a circumflex (^). The preceding circumflex is interpreted as meaning address of or pointer to. For example:

```
LD   R2,^L1;
     (Load address of L1 into R2)
```

2-10. ABSOLUTE ADDRESS CONSTANTS

Absolute addresses can be used as operands when expressed in the form:

```
#4000
```

as in:

```
 LD R1,#4000^
```

which specifies the item at address 4000 hexadecimal.

2-11. SYMBOLIC CONSTANTS

A symbolic constant is a symbol that represents a constant. Symbolic constants are declared by the CONST direction described in chapter 3.

A symbolic constant must be defined before being referenced. For example:

```
CONST   LF = #ØA;
```

substitutes the value #ØA for all subsequent occurrences of the name LF.

2-12. STRINGS

A string is defined to be zero or more characters delimited by apostrophes. Each character is represented in memory by its 8-bit ASCII code. The maximum number of characters in a string is 255. The apostrophe itself is represented within a string by a double apostrophe. If there are zero characters between apostrophes, then the string is empty. For example, valid strings are:

```
'ABC1234'
'12AØ'
'IT''S'
''
```

2-13. EXPRESSIONS

Expressions can be used in directives and instructions. The simplest form of an expression is a numeric constant, such as 5. Expressions can be numeric constants or symbolic constants. Opcodes and arithmetic or logical operators cannot be used in expressions.

CHAPTER 3
DIRECTIVES

3-1. INTRODUCTION

Directives are all the statements in a program that do not create executable machine instructions; they control the operation of the Assembler, allocate storage, or associate symbolic names with constant values.

3-2. END DIRECTIVE

The END directive is required at the end of each program. The END directive has the form:

```
END.
```

END is followed by a period. Note that the END directive is only used at the end of a module.

3-3. BYTE DIRECTIVE

The BYTE directive reserves or defines one or more bytes; it can be preceded by a label. Without a label, The byte directive has one of two forms:

```
BYTE (n);

BYTE: exp ,... exp;
```

where:

n	Is an expression for the number of bytes to be reserved.
exp	Is a numeric expression or 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 to contain the specified expression values.

A numeric expression evaluates to a single byte. A string expression evaluates to a sequence of bytes, one for each character. For example:

```
BYTE   (3);          Reserves 3 bytes
BYTE:  5,A;          Defines  2 bytes with values of 5 and A
BYTE:  'STRING';     Defines  6 bytes with ASCII values of STRING
BYTE:  3,'AB',4;     Defines  4 bytes with values of 03, 41, 42
                     and 04
```

A symbolic constant can be used to define byte values. For example:

```
CONST  P = 5;
BYTE: P;                    Defines 1 byte with value 5
```

3-4. WORD DIRECTIVE

The WORD directive reserves or defines one or more 16-bit words. It can be preceded by a label. Without a label it has one of two forms:

```
WORD  (n);

WORD: exp ,... exp;
```

The WORD directive is similar to the BYTE directive, except that words, rather than bytes, are reserved or defined and only one word per expression (exp) can be reserved. For example:

```
CONST CRLF = #ØDØA;          Declares a symbolic constant
                             named CRLF
WORD: 'VA';                  Defines  1 Word with
                             the values 'VA'
WORD  (3);                   Reserves 3 Words
WORD: CRLF;                  Defines  1 Word with
                             the value #ØDØA
```

String expressions are left-justified and right-filled with blanks (ASCII 20). Constant expressions are right-justified and left-filled with zeros.

3-5. LONG DIRECTIVE

The LONG directive reserves or defines one or more long words (or 32-bit word pairs). It can be preceded by a label. Without a label it has one of two forms:

```
LONG  (n);

LONG: exp ,... exp;
```

The LONG directive is similar to the BYTE directive, except that word pairs, rather than bytes, are reserved or defined and only one word pair per expression (exp) can be reserved. For example:

```
CONST CRLF = #ØDØA;      Declares a symbolic constant
LONG: 'BR';              Defines 1 word pair
                         with the value 'BR'
                         named CRLF
LONG  (3);               Reserves 3 word pairs
LONG: CRLF, CRLF;        Defines 2 word pairs
                         with the value
                         'ØDØA' and 'ØDØA'
```

3-6. CONST DIRECTIVE

The CONST directive declares a name which is to be associated with a constant value. The CONST directive has the form:

```
CONST name = exp;
```

where:

name Is the symbolic name. One or more names can only be declared as symbolic constants in separate declarations.

exp Is a numeric or ACSII-string expression which evaluates to no more than one word in length.

When a symbolic constant is used, the expression is evaluated and the value of the expression is associated with the name. For example:

```
CONST  LINESZ = 8Ø;
  .
  .
  .
LD R14,LINESZ;        Value of LINESZ is 80
```

Once defined, the value of a symbolic constant cannot be changed later in the program.

3-7. QUIT DIRECTIVE

The QUIT directive unconditionally terminates assembly. It has the form:

```
QUIT;
```

This directive allows you to exit the Assembler without completing the Assembly process. A semicolon is required after the command.

CHAPTER 4
INSTRUCTIONS

4-1. INTRODUCTION

The AmZ8000 instructions available in the Assembler constitute a true source-compatible subset of the full instruction set available in the MACRO8000 Assembler, which runs on AMC's AmSYS 8/8 Development System. Instead of an object file, the Evaluation Board Assembler produces absolute code which is written directly into the memory. This means that there are no listings available from the assembler, since the source code is discarded immediately.

The full instruction set is described in detail in the AmZ8001/2 Processor Instruction Set book published by Advanced Micro Devices.

4-2. OPCODES

An opcode is an instruction mnemonic. Each instruction requires a specific number of operands. Zero, one, two, or three operands are required, depending on the instruction. Each instruction has the general form:

```
opcode  operands;
```

There is not a one-to-one correspondence between opcode mnemonics and hex equivalents. The operands themselves define the exact operation indicated by the generic opcode.

4-3. OPERANDS

An operand shown as src or dst is a source or destination value. An src or dst operand in an instruction must utilize one of the addressing modes listed for the instruction. The addressing modes are listed in table 4-1.

An operand shown as r is a word register. An rr operand is a register pair, and an rq operand is a register quadruple. See the R addressing mode in table 4-1.

An operand shown as im is an immediate operand. See the IM addressing mode in table 4-1.

An operand shown as ir is an indirect operand. See the IR addressing mode in table 4-1.

An operand shown as exp is a numeric expression. The simplest form of an expression is a constant.

An operand shown as cc is an AmZ8000 condition code. The condition codes are listed in table 4-2. The condition bits C, Z, S, and P/V are in the flag and control word of the program status registers.

4-4. INSTRUCTION SUMMARY

The relevant AmZ8000 instructions are listed in alphabetic order within the following groups:

Load and exchange instructions are in table 4-3.

Arithmetic instructions are in table 4-4.

Logical instructions are in table 4-5.

Program control instructions are in table 4-6.

Bit manipulation instructions are in table 4-7.

Rotate and shift instructions are in table 4-8.

Block transfer and string manipulation instructions are in table 4-9.

Input/output instructions are in table 4-10.

CPU control instructions are in table 4-11.

NOTE

Certain instructions are noted as privileged. A user running in normal mode on the AmZ8000 is prevented from executing privileged instructions.

TABLE 4-1. ADDRESSING MODES FOR SRC AND DST OPERANDS.

Mode	Assembly Notation	Examples
IM Immediate	exp (Numeric expression)	ADD RØ,5; (Add 5 into register Ø)
R Register	RLn (Lower byte reg., n=Ø to 7)	ADDB RLØ,RL4; (Add RL4 into RLØ)
	RHn (Upper byte reg., n=Ø to 7)	ADDB RHØ,RL4; (Add RL4 into RHØ)
	Rn (Word reg., n=Ø to 15)	ADD RØ,R4; (Add R4 into RØ)
	RRn (Reg. pair, n=Ø by 2 to 14)	ADDL RRØ,RR4; (Add RR4 into RRØ)
	RQn (Reg. quad, n=Ø, 4, 8, 12)	MULTL RQØ,RR4; (Multiply RR2 by RR4, result in RQØ)
IR Indirect Register	Rn (n = 1 to 15)	ADD RØ,R4^; (Add contents of word that R4 points to into RØ)
DA Direct Address	label (Program label)	ADD RØ,FLAG; (Add contents of FLAG into RØ. Equivalent to loading address of FLAG into Rx, then adding what Rx points to into RØ) LD R2, #4320^; (Load contents at address 4320 into R2.)
X Indexed	label(Rn) (Program label, with offset contained in a register)	ADD RØ,FLAG(R4); (Add contents of FLAG, offset by the contents contents of R4, into RØ) LD R2,#4320^(R4); (LD address 4320 offset by contents of R4, into R2.)
RA Relative Address	label (Program label)	JR L1; (Jump relative to L1)

TABLE 4-1. ADDRESSING MODES FOR SRC AND DST OPERANDS. (Cont.)

Mode	Assembly Notation	Examples
PA Port Address	exp (Numeric expression that specifies a 16-bit port adress)	OUT #FFCØ, R4; (Output contents of R4 to port #FFCØ)
PR Port Register	Rn (Word register that contains a 16-bit port address)	OUT R2,R4; (Output contents of R4 to port specified by R2)

TABLE 4-2. CONDITION CODES.

Code	Meaning	If used, test for
NZ	Not Zero	Z = Ø
ZR	Zero	Z = 1
NC	No Carry	C = Ø
CY	Carry	C = 1
PO	Parity odd	P/V = Ø
PE	Parity even	P/V = 1
PL	Plus	S = Ø
MI	Minus	S = 1
NE	Not equal	Z = Ø
EQ	Equal	Z = 1
NOV	Overflow is reset	P/V = Ø
OV	Overflow is set	P/V = 1
GE	Greater than or equal	(S XOR P/V) = Ø
LT	Less than	(S XOR P/V) = 1
GT	Greater than	(Z OR (S XOR P/V)) = Ø
LE	Less than or equal	(Z OR (S XOR P/V)) = 1
LGE	Logical greater than or equal	C = Ø
LLT	Logical less than	C = 1
LGT	Logical greater than	((C = Ø) AND (Z = Ø)) = 1
LLE	Logical less than or equal	(C or Z) = 1

TABLE 4-3. LOAD AND EXCHANGE INSTRUCTIONS.

Opcode	Operands	Addressing Modes for src or dst	Description
CLRB CLR	dst	R,IR,DA,X	Clear dst <== Ø
EXB EX	r,src	R,IR,DA,X	Exchange r <====> src
LDB LD LDL	r,src	IM,R,IR,DA,X	Load r <== src
LDB LD LDL	dst,r	IR,DA,X	Load to Memory dst <== r
LDB LD	dst,im	IR,DA,X	Load to Memory Immediate dst <== im
LD	r,^src	DA,X	Load Address r <== src (src means address of source)
LDM	r,src,exp	IR,DA,X	Load Multiple r <== src (starting at r and src, load exp consecutive words; exp is 1 to 16)
LDM	dst,r,exp	IR,DA,X	Load Multiple to Memory dst <== r (starting at dst and r, load exp consecutive words; exp is 1 to 16)
LDRB LDR	r,src	RA	Load Relative r <== src
LDRB LDR	dst,r	RA	Load Relative to Memory dst <== r

TABLE 4-3. LOAD AND EXCHANGE INSTRUCTIONS. (Cont.)

Opcode	Operands	Addressing Modes for src or dst	Description
POP POPL	dst,ir	R,IR,DA,X	Pop dst <== ir (autoincrement contents of register after pop)
PUSH PUSHL	ir,src	IM,R,IR,DA,X	Push ir <== src (autodecrement contents of register before push)

TABLE 4-4. ARITHMETIC INSTRUCTIONS.

Opcode	Operands	Addressing Modes for src or dst	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, set carry
CPB CP CPL	r,src	IM,R,IR,DA,X	Compare r - src (affects flags)
CPB CP	dst,im	IR,DA,X	Compare Memory with Immediate dst - im (affects flags)
DAB	r	-	Decimal Adjust (decimal adjust of r)
DECB DEC	dst,exp	R,IR,DA,X	Decrement dst <== dst - exp (exp is 1 to 16)

TABLE 4-4. ARITHMETIC INSTRUCTIONS. (Cont.)

Opcode	Operands	Addressing Modes for src or dst	Description
DIV DIVL	rr,src rq,src	IM,R,IR,DA,X	Signed Divide rr_{n+1} <== $rr_{n,n+1}$ / src rr_n <== remainder (register pair) $rq_{n+2,n+3}$ <== $rq_{n,n+1,n+2,n+3}$ / src $rq_{n,n+1}$ <== remainder (register quad)
EXTSB EXTS EXTSL	dst	R	Extend Sign (extend sign of dst_{low} to dst_{high})
INCB INC	dst,exp	R,IR,DA,X	Increment dst <== dst + exp (exp is 1 to 16)
MULT MULTL	rr,src rq,src	IM,R,IR,DA,X	Signed Multiply $rr_{n,n+1}$ <== rr_{n+1} * src (register pair) $rq_{n,n+1,n+2,n+3}$ <== $rq_{n+2,n+3}$ * src (register quad)
NEGB NEG	dst	R,IR,DA,X	Negate (two's complement) dst <== Ø - dst
SBCB SBC	r,src	R	Subtract with Carry r <== r - src - carry
SUBB SUB SUBL	r,src	IM,R,IR,DA,X	Subtract r <== r - src

TABLE 4-5. LOGICAL INSTRUCTIONS.

Opcode	Operands	Addressing Modes for src or dst	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
ORB OR	r,src	IM,R,IR,DA,X	OR r <== r OR src
TESTB TEST TESTL	dst	R,IR,DA,X	Test dst OR Ø
TCCB TCC	cc,r	R	Test Condition Code If cc is true: r_{lsb} <== 1 otherwise: r_{lsb} <== Ø (lsb is least significant bit)
XORB XOR	r,src	IM,R,IR,DA,X	Exclusive OR R <== R XOR src

TABLE 4-6. PROGRAM CONTROL INSTRUCTIONS.

Opcode	Operands	Addressing Modes for src or dst	Description
CALL	dst	IR,DA,X	Call Subroutine Autodecrement SP SP <== PC PC <== dst
CALR	dst	RA	Call Relative Autodecrement SP SP <== PC PC <== PC + dst (dst is -4092 to +4098)
DBJNZ	r,dst	RA	Decrement and Jump if Nonzero R <== R - 1 If R ≠ Ø: PC <== PC + dst (dst is -252 to +2; flags are not affected)
IRET	-	--	*Interrupt return PS <== SP^ Autoincrement SP
JP	cc,dst	IR,DA,X	Jump If cc is true: PC <== dst (cc is optional)
JR	cc,dst	RA	Jump Relative If cc is true: PC <== PC + dst (cc is optional; dst is -254 to +256)
RET	cc	--	Return Conditional If cc is true: PC <== SP Autoincrement SP
SC	exp	--	System Call Autodecrement SP SP <== old PS Push instruction PS <== system call PS (exp is Ø to 255; if exp is Ø, the Evaluation Board Monitor will perform an I/O operation)

*Privileged instruction

TABLE 4-7. BIT MANIPULATION INSTRUCTIONS.

Opcode	Operands	Addressing Modes for src or dst	Description
BITB BIT	dst,exp	R,IR,DA,X	Test Bit Static Z flag <== NOT dst_{exp}
BITB BIT	dst,r	R	Test Bit Dynamic Z flag <== NOT dst_r
RESB RES	dst,exp	R,IR,DA,X	Reset Bit Static dst_{exp} <== Ø
RESB RES	dst,r	R	Reset Bit Dynamic dst_r <== Ø
SETB SET	dst,exp	R,IR,DA,X	Set Bit Static dst_{exp} <== 1
SETB SET	dst,r	R	Set Bit Dynamic dst_r <== 1
TSETB TSET	dst	R,IR,DA,X	Test and Set S flag <== dst_{msb} (all bits in dst are set to 1; msb is most significant bit)

TABLE 4-8. ROTATE AND SHIFT INSTRUCTIONS.

Opcode	Operands	Addressing Modes for src or dst	Description
RLDB	r,src	R	Rotate Digit Left (4-bit digit)
RRDB	r,src	R	Rotate Digit Right (4-bit 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, defaualt 1)
SDAB SDA SDAL	dst,r	R	Shift Dynamic Arithmetic (shift dst left or right by r bits; positive left, negative right)
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)

TABLE 4-9. BLOCK TRANSFER AND STRING MANIPULATION INSTRUCTIONS.

Opcode	Operands	Addressing Modes for src or dst	Description
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 is true or r2 = Ø)
CPIB CPI	r1,src,r2,cc	IR	Compare and Increment r1 - src Autoincrement src r2 <== r2 - 1
CPIRB CPIR	r1,src,r2,cc	IR	Compare, Increment and Repeat r1 - src Autoincrement src r2 <== r2 - 1 (repeat until cc is true or r2 = Ø)
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 is true or r = Ø)
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 is true or r = Ø)

TABLE 4-9. BLOCK TRANSFER AND STRING MANIPULATION INSTRUCTIONS. (Cont.)

Opcode	Operands	Addressing Modes for src or dst	Description
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 = Ø)
LDIB LDI	dst,scr,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 = Ø)
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 = Ø)
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 = Ø)

TABLE 4-9. BLOCK TRANSFER AND STRING MANIPULATION INSTRUCTIONS. (Cont.)

Opcode	Operands	Addressing Modes for src or dst	Description
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 = Ø or RH1 = Ø)
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 = Ø or RH1 = Ø)

TABLE 4-10. INPUT/OUTPUT INSTRUCTIONS.

Opcode	Operands	Addressing Modes for src or dst	Description
INB IN	r,src	PA,PR	*Input r <== src (DA src indicates 16-bit port address)
INDB IND	dst,pr,r	IR	*Input and Decrement dst <== pr Autodecrement dst r <== r - 1
INDRB INDR	dst,pr,r	IR	*Input, Decrement and Repeat dst <== pr Autodecrement dst r <== r - 1 (repeat until r = Ø)
INIB INI	dst,pr,r	IR	*Input and Increment dst <== pr Autoincrement dst r <== r - 1
INIRB INIR	dst,pr,r	IR	*Input, Increment and Repeat dst <== pr Autoincrement dst r <== r - 1 (repeat until r = Ø)
OUTB OUT	dst,r	PA,PR	*Output dst <== r (DA dst indicates 16-bit port address)
OUTDB OUTD	pr,src,r	IR	*Output and Decrement pr <== src Autodecrement src r <== r - 1
OTDRB OTDR	pr,src,r	IR	*Output, Decrement and Repeat pr <== src Autodecrement src r <== r - 1 (repeat until r = Ø)
OUTIB OUTI	pr,src,r	IR	*Output and Increment pr <== src Autoincrement src r <== r - 1
*Privileged instruction			

TABLE 4-10. INPUT/OUTPUT INSTRUCTIONS. (Cont.)

Opcode	Operands	Addressing Modes for src or dst	Description
OTIRB OTIR	pr,src,r	IR	*Output, Increment and Repeat pr <== src Autoincrement src r <== r - 1 (repeat until r = Ø)
SINDB SIND	dst,pr,r	IR	*Special Input and Decrement dst <== pr Autodecrement dst r <== r - 1
SINDRB SINDR	dst,pr,r	IR	*Special Input, Decrement and Repeat dst <== pr Autodecrement dst r <== r - 1 (repeat until r = Ø)
SINIB SINI	dst,pr,r	IR	*Special Input and Increment dst <== pr Autoincrement dst r <== r - 1
SINIRB SINIR	dst,pr,r	IR	*Special Input, Increment and Repeat dst <== pr Autoincrement dst r <== r - 1 (repeat until r = Ø)
SOUTDB SOUTD	pr,src,r	IR	*Special Output and Decrement pr <== src Autodecrement src r <== r - 1
SOTDRB SOTDR	pr,src,r	IR	*Special Output, Decrement and Repeat pr <== src Autodecrement src r <== r - 1 (repeat until r = Ø)
SOUTIB SOUTI	pr,src,r	IR	*Special Output and Increment pr <== src Autoincrement src r <== r - 1

TABLE 4-10. INPUT/OUTPUT INSTRUCTIONS. (Cont.)

Opcode	Operands	Addressing Modes for src or dst	Description
SOTIRB SOTIR	pr,src,r	IR	*Special Output, Increment and Repeat pr <== src Autoincrement src r <== r - 1 (repeat until r = Ø)
*Privileged instruction			

TABLE 4-11. CPU CONTROL INSTRUCTIONS.

Opcode	Operands	Addressing Modes for src or dst	Description
COMFLG	flags	-	Complement Flags (flags are CY, ZR, SGN, PY, OV)
DI	ints	-	*Disable Interrupt (interrupts are NVI and VI)
EI	ints	-	*Enable Interrupt (interrupts are NVI and VI)
HALT	-	-	*Halt
LDCTLB	FLAGS,src	R	*Load Flag Byte FLAGS <== src
LDCTLB	dst,FLAGS	R	*Load from Flag Byte dst <== FLAGS
NOP	-	-	No Operation
RESFLG	flags	-	Reset Flags (flags are CY, ZR, SGN, PY, OV)
SETFLG	flags	-	Set Flags (flags are CY, ZR, SGN, PY, OV)
*Privileged instruction			

APPENDIX A
ALPHABETICAL LIST OF INSTRUCTIONS

A-1. INTRODUCTION

The following table lists permissible AmZ8000 instructions in alphabetical order. Under addressing modes, value indicates that the operand is immediate. Values in parentheses are hexadecimal equivalents for operation codes.

Mnemonic	Address Modes	Description
ADC	R(B5)	Add word with carry
ADCB	R(B4)	Add byte with carry
ADD	R(81), IM(01), IR(01), DA(41), X(41)	Add Word
ADDB	R(80), IM(00), IR(00), DA(40), X(40)	Add byte
ADDL	R(96), IM(16), IR(16), DA(56), X(56)	Add long word
AND	R(87), IM(07), IR(07), DA(47), X(47)	Logical AND word
ANDB	R(86), IM(06), IR(06), DA(46), X(46)	Logical AND byte
BIT	DY(27), R(A7), IR(27), DA(67), X(67)	Test word bit
BITB	DY(26), R(A6), IR(26), DA(66), X(66)	Test byte bit
BYTE		(See Directives)
CALL	IR(1F), DA(5F), X(5F)	Call subroutine
CALR	RA(D)	Call subroutine relative
CLR	R(8D), IR(0D), DA(4D), X(4D)	Clear word

---

*Privileged (system) instructions

Mnemonic	Address Modes	Description
CLRB	R(8C), IR(0C), DA(4C), X(4C)	Clear byte
COM	R(8D), IR(0D), DA(4D), X(4D)	Complement word
COMB	R(8C), IR(0C), DA(4C), X(4C)	Complement byte
COMFLG	C, Z, S, P, V(8D)	Complement flags
CONST		(SeeDirective)
CP	R(8B), IM(0B), IR(0B), DA(4B), X(4B), IM-IR(0D), IM-DA(4D), IM-X(4D)	Compare word
CPB	R(8A), IM(0A), IR(0A), DA(4A), X(4A), IM-IR(0C), IM-DA(4C), IM-X(4C)	Compare byte
CPD	IR(BB)	Compare word and decrement
CPDB	IR(BA)	Compare byte and decrement
CPDR	IR(BB)	Compare word decrement and repeat
CPDRB	IR(BA)	Compare byte decrement and repeat
CPI	IR(BB)	Compare word and increment
CPIB	IR(BA)	Compare byte and increment
CPIR	IR(BB)	Compare word increment repeat
CPIRB	IR(BA)	Compare byte increment repeat
CPL	R(90), IM(10), IR(10), DA(50), X(50)	Compare long word
CPSD	IR(BB)	Compare word string and decrement

---

*Privileged (system) instructions

Mnemonic	Address Modes	Description
CPSDB	IR(BA)	Compare byte string and decrement
CPSDR	IR(BB)	Compare word string decrement and repeat
CPSDRB	IR(BA)	Compare byte string decrement and repeat
CPSI	IR(BB)	Compare word string and increment
CPSIB	IR(BA)	Compare byte string and increment
CPSIR	IR(BB)	Compare word string increment and repeat
CPSIRB	IR(BA)	Compare byte string increment and repeat
DAB	R(BO)	Decimal adjust
DBJNZ	RA(F)	Decrement byte and jump non-zero
DEC	R(AB), IR(2B), DA(6B), X(6B)	Decrement word
DECB	R(AA), IR(2A), DA(6A), X(6A)	Decrement byte
*DI	VI, NVI (7C)	Disable interrupts
DIV	R(9B), IM(1B), IR(1B), DA(5B), X(5B)	Divide word
DIVL	R(9A), IM(1A), IR(1A), DA(5A), X(5A)	Divide long word
*EI	VI,NVI(7C)	Enable interrupts
END		(See Directive)
EX	R(AD), IR(2D), DA(6D), X(6D)	Exchange words
EXB	R(AC), IR(2C), DA(6C), X(6C)	Exchanges bytes
EXTS	R(B1)	Extend word sign

---

*Privileged (system) instructions

Mnemonic	Address Modes	Description
EXTSB	R(B1)	Extend byte sign
EXTSL	R(B1)	Extent long word sign
*HALT	(7A)	Halt
*IN	PR(3D), PA(3B)	Input word
*INB	PR(3C), PA(3C)	Input byte
INC	R(A9), IR(29), DA(69), X(69)	Increment word
INCB	R(A8), IR(28), DA(68), X(68)	Increment byte
*IND	IR(3B)	Input word and decrement
*INDB	IR(3A)	Input byte and decrement
*INDR	IR(3B)	Input word decrement and repeat
*INDRB	IR(3A)	Input byte decrement and repeat
*INI	IR(3B)	Input word and increment
*INIB	IR(3A)	Input byte and increment
*INIR	IR(3B)	Input word increment and repeat
*INIRB	IR(3A)	Input byte increment and repeat
*IRET	(7B)	System call return
JP	IR(1E), DA(5E), X(5E)	Conditional jump
JR	RA(E)	Jump relative condition
LD	R(A1), IM(21), IR(21), DA(61), X(61)	Load word register
LD	IR(2F), DA(6F), X(6F), IM-IR(OD), IM-DA(4D), IM-X(4D)	Load word memory

*Privileged (system) instructions

Mnemonic	Address Modes	Description
LDB	R(A0), IM(C), IR(20), DA(60), X(60)	Load byte register
LDB	IR(2E), DA(6E), X(6E), IM-IR(0C), IM-DA(4C), IM-X(4C)	Load byte memory
LDCTLB	R,FLAGS(8C), FLAGS,R(8C)	Load flag register
LDD	IR(BB)	Load word and decrement
LDDB	IR(BA)	Load byte and decrement
LDDR	IR(BB)	Load word decrement and repeat
LDDRB	IR(BA)	Load byte decrement and repeat
LDI	IR(BB)	Load word and increment
LDIB	IR(BA)	Load byte and increment
LDIR	IR(BB)	Load word increment and repeat
LDIRB	IR(BA)	Load byte increment and repeat
LDL	R(94), IM(14), IR(14), DA(54), X(54)	Load long word register
LDL	IR(1D), DA(5D), X(5D)	Load long word memory
LDM	IR(1C), DA(5C), X(5C)	Load multiple registers
LDM	IR(1C), DA(5C), X(5C)	Load multiple memory
LDR	RA-R(31), R-RA(33)	Load relative
LDRB	RA-R(30), R-RA(32)	Load relative byte
LONG		(See Directive)
MULT	R(99), IM(19), IR(19), DA(59), X(59)	Multiply word

---

*Privileged (system) instructions

Mnemonic	Address Modes	Description
MULTL	R(98), IM(18), IR(18), DA(58), X(58)	Multiply long word
NEG	R(8D), IR(0D), DA(4D), X(4D)	Negate word
NEGB	R(8C), IR(0C), DA(4C), X(4C)	Negate byte
NOP	(8D07)	No operation
OR	R(85), IM(05), IR(05), DA(45), X(45)	Logical OR word
ORB	R(84), IM(04), IR(04), DA(44), X(44)	Logical OR byte
OTDR	IR(3B)	Output word decrement and repeat
*OTDRB	IR(3A)	Output byte decrement and repeat
*OTIR	IR(3B)	Output word increment and repeat
*OTIRB	IR(3A)	Output byte increment and repeat
*OUT	PR(3F), PA(3B),	Output word
*OUTB	PR(3E), PA(3A),	Output byte
*OUTD	IR(3B)	Output word and decrement
*OUTDB	IR(3A)	Output byte and decrement
*OUTI	IR(3B)	Output word and increment
*OUTIB	IR(3A)	Output byte and increment
POP	R(97), IR(17), DA(57), X(57)	Pop stack word
POPL	R(95), IR(15), DA(55), X(55)	Pop stack long word

---

*Privileged (system) instructions

Mnemonic	Address Modes	Description
PUSH	R(93), IM(OD), IR(13), DA(53), X(53)	Push stack word
PUSHL	R(91), IR(11), DA(51), X(51)	Push stack long word
QUIT		(See Directives)
RES	DY(23), R(A3), IR(23), DA(63), X(63)	Reset word bit
RESFLG	C, Z, S, P, V(8D)	Reset flags
RET	(9E)	Conditional return
RL	Value (B3)	Rotate word left
RESB	DY(22), R(A2), IR(22), DA(62), X(62)	Reset byte bit
RLB	Value (B2)	Rotate byte left
RLC	Value (B3)	Rotate word left through carry
RLCB	Value (B2)	Rotate byte left through carry
RLDB	R(BE)	Rotate digit left
RR	Value (B3)	Rotate word right
RRB	Value (B2)	Rotate byte right
RRC	Value (B3)	Rotate word right through carry
RRCB	Value (B2)	Rotate byte right through carry
RRDB	R(BC)	Rotate digit right
SBC	R(B7)	Subtract word with carry
SBCB	R(B6)	Subtract byte with carry
SC	Value (7F)	System call

---

*Privileged (system) instructions

Mnemonic	Address Modes	Description
SDA	R(B3)	Shift dynamic arithmetic word
SDAB	R(B2)	Shift dynamic arithmetic byte
SDAL	R(B3)	Shift dynamic arithmetic long word
SDL	R(B3)	Shift dynamic long word
SDLB	R(B2)	Shift dynamic long byte
SDLL	R(B3)	Shift dynamic logical long word
SET	DY(25), R(A5), IR(25), DA(65), X(65)	Set word bit
SETB	DY(24), R(A4), IR(24), DA(64), X(64)	Set byte bit
SETFLG	C, Z, S, P, V(8D)	Set flags
*SIND	IR(3B)	Special input
*SINDB	IR(3A)	Special input and decrement
*SINDR	IR(3B)	Special input, decrement and repeat
*SINDRB	IR(3A)	Special input, decrement and repeat
*SINI	IR(3B)	Special input and increment
*SINIB	IR(3A)	Special input and increment
*SINIR	IR(3B)	Special input, increment and repeat
*SINIRB	IR(3A)	Special input, increment and repeat

---

*Privileged (system) instructions

Mnemonic	Address Modes	Description
SLA	Value (B3)	Shift left arithmetic word
SLAB	Value (B2)	Shift left arithmetic byte
SLAL	Value (B3)	Shift left arithmetic long word
SLL	Value (B3)	Shift left logical word
SLLB	Value (B2)	Shift left logical byte
SLLL	Value (B3)	Shift left logical long word
*SOTDR	IR(3B)	Special output, decrement and repeat
*SOTDRB	IR(3A)	Special output, decrement and repeat
*SOTIR	IR(3B)	Special output, increment and repeat
*SOTIRB	IR(3A)	Special output, increment and repeat
*SOUTD	IR(3B)	Special output and decrement
*SOUTDB	IR(3A)	Special output and decrement
*SOUTI	IR(3B)	Special output and increment
*SOUTIB	IR(3A)	Special output and increment
SRA	Value (B3)	Shift right arithmetic word
SRAB	Value (B2)	Shift right arithmetic byte

---

*Privileged (system) instructions

Mnemonic	Address Modes	Description
SRAL	Value (B3)	Shift right arithmetic long word
SRL	Value (B3)	Shift right logical word
SRLB	Value (B2)	Shift right logical byte
SRLL	Value (B3)	Shift right logical long word
SUB	R(83), IM(03), IR(03), DA(43), X(43)	Subtract word
SUBB	R(82), IM(02), IR(02), DA(42), X(42)	Subtract byte
SUBL	R(92), IM(12), IR(12), DA(52), X(52)	Subtract long word
TCC	R(AF)	Test condition and set word
TCCB	R(AE)	Test condition and set byte
TEST	R(8D), IR(0D), DA(4D), X(4D)	Test word
TESTB	R(8C), IR(0C), DA(4C), X(4C)	Test byte
TESTL	R(9C), IR(1C), DA(5C), X(5C)	Test long word
TRDB	IR(B8)	Translate byte and decrement
TRDRB	IR(B8)	Translate byte decrement and repeat
TRIB	IR(B8)	Translate byte and increment
TRIRB	IR(B8)	Translate byte increment and repeat
TRTDB	IR(B8)	Translate test byte and decrement

---

*Privileged (system) instructions

Mnemonic	Address Modes	Description
TRTDRB	IR(B8)	Translate test byte decrement and repeat
TRTIB	IR(B8)	Translate test byte and increment
TRTIRB	IR(B8)	Translate test byte increment and repeat
TSET	R(8D), IR(0D), DA(4D), X(4D)	Test word and set
TSETB	R(8C), IR(0C), DA(4C), X(4C)	Test byte and set
WORD		(See Directives)
XOR	R(89), IM(09), IR(09), DA(49), X(49)	Exclusive OR word
XORB	R(88), IM(08), IR(08), DA(48), X(48)	Exclusive OR byte

---

*Privileged (system) instructions

APPENDIX B
ERROR MESSAGES

B-1. INTRODUCTION

During the line by line assembly, statements are completely evaluated before any assembly occurs. If any error exists, a diagnostic is displayed and the assembler awaits re-entry of the instruction. The following codes are displayed:

L - A syntax error occurred during label processing, a label is required, or one is present and is not permitted.

D - A duplicate label has been encountered.

O - A syntax error occurred during opcode processing, or an undefined operation code was encounterd.

X - A system error occurred. This is the result of a software or hardware malfunction.

S - A syntax error occurred in statement processing.

U - An equivalence operand was not previously defined.

V - A memory overflow occurred. The program plus the number of labels exceeds machine capacity.

R - Occurs only after an END pseudo-operation and specifies an undefined label reference.

APPENDIX C
ASCII CHARACTER SET

C-1. INTRODUCTION

The character set is in the ANSI X3.4 1968 version shown in table C-1.

TABLE C-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	1
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(X-ON)	31	49	1	51	81	Q	71	113	q
12	18	DC2(TAPE)	32	50	2	52	82	R	72	114	r
13	19	DC3(X-OFF)	33	51	3	53	83	S	73	115	s
14	20	DC4(TAPE)	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

INDEX

INDEX (Cont.)

INDEX (Cont.)

COMMENT SHEET

Address comments to:

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

TITLE: Am96/4016-ASM
PUBLICATION NUMBER: 00680138A

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

From: Name: _____ Position: _____

Company: _____

Address: _____

ADVANCED MICRO COMPUTERS
3340 Scott Boulevard
Santa Clara, California 95051
Distributed by
Advanced Micro Devices