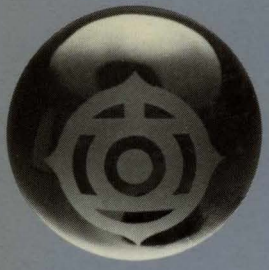
HITACHI

ISIS-II
6301 CROSS ASSEMBLER
USER'S MANUAL



#U29

ISIS-II
6301 CROSS ASSEMBLER
USER'S MANUAL

#U29

HITACHI

S31MDS1-EM

When using this manual, the reader should keep the following in mind:

1. This manual may, wholly or partially, be subject to change without notice.
2. All rights reserved: No one is permitted to reproduce or duplicate, in any form, the whole or part of this manual without Hitachi's permission.
3. Hitachi will not be responsible for any damage to the user that may result from accidents or any other reasons during operation of his unit according to this manual.
4. This manual neither ensures the enforcement of any industrial properties or other rights, nor sanctions the enforcement right thereof.

INTRODUCTION

The S31MDS1-F is a cross assembler which runs on Intel* development systems under the ISIS-II operating system, and processes assembly language programs for the HD6800 micro-processor and HD6801 and HD6301 single-chip microcomputers. The assembler accepts ISIS-II files containing 6800/6801/6301 source statements, and produces assembly listings and object files.

*"Intel" is a registered trademark of Intel Corporation, Santa Clara, CA.

The cross assembler is provided in the form of a single sided, double density floppy disk.

TABLE OF CONTENTS

Table of Contents -- Continued

1 DESCRIPTION OF ASSEMBLER

The ISIS-II 6301 Cross Assembler runs on Intel development systems under the ISIS-II operating system, and processes assembly language programs for the HD6800 microprocessor and HD6801 and HD6301 single-chip microcomputers. The assembler accepts ISIS-II files containing HD6800/6801/6301 source statements, and produces assembly listings and object files.

This manual assumes familiarity with the Intel development system, the ISIS-II operating system, and the HD6800 microprocessor and/or the HD6801 and/or HD6301 microcomputers. The purpose of the manual is to assist a user already familiar with the HD6800/6801/6301 instruction set and architecture in using the assembler. For further information on the development system and ISIS-II, consult the Intel publications: ISIS-II SYSTEM USER'S GUIDE, 9800306 and the publication describing your version of the MDS (development system). For information on the architecture and instruction set of the HD6800, HD6801, and/or HD6301, consult the appropriate Programming Reference Manual from the processor's manufacturer.

To use the assembler, you will require an Intel disk-based Microcomputer Development System with 32K or more of RAM memory and two or more disk drives, ISIS-II operating system software, and a diskette containing the program ASM31.*

*Program name is ASM31. HITACHI's Part No. is S31MDS1-F. Please note the difference.

Section 2 of this manual will describe the required assembly language statement format. Section 3 will describe the command lines used to invoke the assembler. Section 4 covers the files produced, including the source listing (with a list of possible error messages) and the object file.

The assembler normally assumes the opcodes to be for the HD6800 microprocessor. Use of the MOD01 or MOD31 control (see section 3.2) is necessary to enable use of the full HD6801 opcode set (of which the HD6800 opcodes are a subset). Use of the MOD31 control is necessary to enable use of the full HD6301 opcode set (of which the HD6800 and HD6801 opcodes are a subset).

2 FORMAT OF STATEMENTS

An assembly language program consists of a series of source lines, which fall into four categories:

1: Blank lines - contain no printing characters.
2: Comment lines - begin with an asterisk ("*").
3: Control lines - begin with a dollar sign ("S").
4: Assembly language statements, including instructions and directives.

Blank lines, which contain no characters other than spaces or tabs, and comment lines, which are identified by an asterisk as the first character, are ignored by the assembler except for purposes of listing, and can be included at any point before the END statement. Controls will be described in section 3, assembly language instructions in section 2.2, and assembler directives in section 2.3.

Each line consists of a series of ASCII characters ending with a carriage return and line feed. Tabs are converted to one or more spaces, as required to advance the listing to the next tab position on the line. (Tab positions occur at every eight columns starting at the first character of the line.)

```
$TITLE('     FIG. 2-1     ') ----------------------control line
INCHNP  EQU     $06A0
NEGDAT  EQU     255 ----------------------------assembly language statement
                            ---------------------blank line
        ORG     $06A5
* INPUT ONE HEXADECIMAL CHAR CONVERT TO BCD ----comment line
* NEG CONDITION CODE, RESET=GOOD HEX, SET=NOT
IN1H    BSR     INCHNP
CBCDHK  CMPA    #$30    < CONVERT BCD TO HEX >
        BMI     IN1HB   NOT HEX  --------------assembly language statement
        CMPA    #$39
        BLS     IN1HG   GOOD HEX
        CMPA    #$41
        BMI     IN1HB   NOT HEX
        CMPA    #$46
        BHI     IN1HB   NOT HEX
        SUBA    #7
IN1HG   ANDA    #$F     MASK TO BCD-RESET NEG
        RTS
IN1HB   TSTA    #$F     SET NEG CONDITION CODE
        RTS
        END
```

Fig. 2-1 Sample of source program

2.1 General Concepts

Before describing the formats of individual instruction and directive statements, this manual will explain the general form of assembly language statements.

2.1.1 Constants

Constants can be specified as numbers (in binary, octal, decimal, or hexadecimal notation) or characters.

Number constants consists of one or more digits valid in the particular base in which they are expressed, preceded (except in the case of decimal numbers) by a special character to indicate the base. The base-designation prefixes and valid characters are given for each base in the table below:

Base	Prefix Char	Valid Digits
2 (binary)	%	0,1
8 (octal)	@	0..7
10 (decimal)	(none)	0..9
16 (hexadecimal)	$	0..9, A..F

For example, the number 19 decimal could be expressed as "19", "$13", "@23", or "%10011".

The allowable range of values for constants is 0..65535 decimal ($0..$FFFF hexadecimal). If a larger constant is specified, the constant is converted to a value within the allowed range.

Character strings are identified by their first character, which is a single-quote character ("'"). The single-quote is followed by one or more ASCII characters and a closing single-quote. To represent the single-quote character itself as one of the characters in the string, the single-quote must occur twice in a row between the opening and closing single-quote marks.

For example, a space would be coded as "' '", the capital letter Z as "'Z'", and the word "CAN'T" as "'CAN''T'".

Note that non-printing ASCII characters cannot be coded in this fashion. They must be coded by their numerical equivalents; for instance, a carriage return can be coded as "$0D".

The interpreted value of the string is dependent on how the string is used. Each string is encoded as a series of bytes, one per character. The first byte corresponds to the first character of the string, and so on. If a one-character string is used in an expression, the value returned is the numeric equivalent of the ASCII character in that string, in the range $0020..$007F. If a two-character string is used, the value returned has the first character in the left-hand (more significant) byte and the second character in the right-hand (less significant) byte of a two-byte value. Each byte is again in the range $20..$7F.

For example, "'A'" returns $41, while "'AB'" returns $4142.

Strings of more than two characters are not meaningful in expressions; their use is limited to the FCC directive (see section 2.3.5) and the TITLE control (see section 3.3).

2.1.2 Location Counter

The location counter is an internal value maintained by the assembler. It indicates the address where the next instruction or data byte will be placed. The assembler assumes a starting location counter value of $0000. The location counter can be changed at any time using the ORG directive (see section 2.3.8).

Whenever a byte of code is stored (resulting from a 6800/6801/6301 instruction or a data definition directive), the location counter is incremented by one (1). For example, if the location counter value is $013A, and the instruction "LDAA #1" is encountered, the two-byte code for the instruction will be stored at locations $013A and $013B, and the location counter value will be changed to $013C.

The value of the location counter can be used in expressions via the asterisk ("*") in an expression-operand position. (In an expression-operator position, the asterisk denotes multiplication.) The location counter value is in the range $0000..$FFFF.

2.1.3 Symbols

Symbols are specified as a sequence of one to six alphanumeric characters, the first of which must be alphabetic. They can take on any values in the range $0000..$FFFF.

Some examples of valid symbol names are: "A", "LOCTN1", and "FFFF". Note that the latter is a symbol name and not a hexadecimal constant because it begins with a letter, not "$".

Symbols are defined when used as labels of instructions or directives. A symbol first defined as the label of a SET directive can be redefined later in the program via another SET directive. Prior to the first definition of that label, its value will be the last value to which it is SET in your program. After the first definition (including in the symbol table listing), the symbol always has the most recent value assigned to it by a SET directive. For example:

```
        ...
        reference 1 to symbol SYMB1
        ...
SYMB1   SET     4
        ...
        reference 2 to symbol SYMB1
        ...
SYMB1   SET     15
        ...
        reference 3 to symbol SYMB1
        ...
        END
```

In the above program, references 1 and 3 to symbol SYMB1 would return the value 15, while reference 2 would return the value 4. The symbol table would show the value 15 ($000F).

Any symbol first defined as a label of some statement other than SET may not be redefined later in the program. Attempting to redefine such a symbol will result in an error (see section 4.1.2). Attempting to redefine, by means other than SET, a symbol first defined via SET will result in a similar error.

2.1.4 Expressions

Expressions are combinations of symbols, constants, location counter references, and/or operators which produce a value. They are used in the operand field of a statement (see section 2.1.5 for more on statement fields). The minimum expression consists of a single symbol or constant or location counter reference. Using arithmetic and logical operators and additional symbols and/or constants, more complex expressions can be constructed.

The allowed operators (all single characters) are:

- `*` multiplication
- `/` division
- `+` addition
- `-` subtraction (or negation if used as unary operator)

In addition, left and right parentheses ("(", ")") are available for grouping and for overriding operator priorities, which are as follows (all operators on the same line have the same priority; operators on higher lines are executed first):

```
unary -
*   /
+   -
```

As mentioned above, any subexpression in parentheses is executed first, using the above priorities.

As an example of priorities of expression evaluation, the expression "5+(3+4)*2" would produce the value 19.

2.1.5 Fields

Each assembly statement (instruction or directive) is divided into up to four fields as follows:

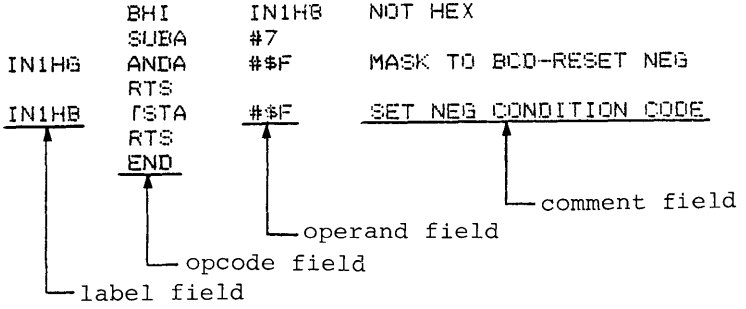


Fig. 2-2 Fields

(1) Field Usage

Only the opcode field is required in all assembly statements. (These fields do not apply to blank lines, comment lines, or control lines.) The opcode field contains the name of the instruction or directive. The requirements of the other fields are dependent on which instruction or directive is in the opcode field.

The label field is used for symbol definition in instructions, data definition directives, and EQU and SET directives, as described previously (see section 2.1.3).

If the instruction or directive in the opcode field uses an operand, the operand field is required. If no operand is used for that particular instruction or directive, the operand field is omitted. The contents of the operand field vary with the type of instruction or directive.

The comment field is ignored by the assembler except for purposes of listing. It allows the programmer to insert remarks into the text of the assembly language program.

Next come up to four pairs of hexadecimal digits, in one to four columns. For one-byte instructions, there is one column with two digits, representing the byte of code generated. For two-byte instructions, there are two columns with two digits each. For three-byte instructions, there are two columns -- the first with two characters, the second with four characters (representing two bytes). Data definition directives list up to four bytes per line, each as two hexadecimal characters.

After these columns is the line number and the source line.

(2) Field Formatting

The label field may optionally end with a colon (":") immediately following the symbol name. The label field must begin with the first character on the line (ie, it may not have any spaces or tabs preceding it), whether it is used with or without a colon.

If the label field is present, it is followed by one or more spaces and/or tabs followed by the opcode field. If the label field is omitted, the line must begin with one or more spaces and/or tabs followed by the opcode field.

If the opcode field contains a four-letter instruction name which references a register (A, B, D, or X) with the fourth letter (for example, LDAA), the fourth letter may optionally be separated from the first three with a space or tab. For example, LDAA can be written as "LDAA" or "LDA A".

The operand field, if present, is separated from the opcode field by one or more spaces and/or tabs.

The comment field, if present, is separated from the field preceding it (the opcode field if there is no operand, the operand field if an operand is present) by one or more spaces and/or tabs.

2.2 Instructions

Instructions are statements which generate actual code to be executed by an HD6800/HD6801/HD6301. They are classified here according to the addressing modes they use. The addressing mode, in turn, determines the use of the operand field of an instruction.

2.2.1 Inherent Addressing Mode

Instructions with inherent addressing have no operands; the operand field is thus omitted with these instructions.

The following instructions use inherent addressing (exclusively):

ABA	ASLA	ASLB	ASRA	ASRB	CBA	CLC	CLI
CLRA	CLRB	CLV	COMA	COMB	DAA	DECA	DECB
DES	DEX	INCA	INCB	INS	INX	LSRA	LSRB
NEGA	NEGB	NOP	PSHA	PSHB	PULA	PULB	ROLA
ROLB	RORA	RORB	RTI	RTS	SBA	SEC	SEI
SEV	SWI	TAB	TAP	TBA	TPA	TSTA	TSTB
TSX	TXA	WAI					
(for HD6801/HD6301 only:			ABX	ASLD	LSLA	LSLB	LSLD
			LSRD	MUL	PSHX	PULX)	
(for HD6301 only:			SLP	XGDX)			

These instructions generate one byte of code each.

2.2.2 Immediate Addressing Mode

Instructions with immediate addressing require an operand field consisting of the character "#" followed by an expression (for example, "LDAA #VALUE1-1".

All instructions which can use immediate addressing can also use direct, extended, and indexed addressing (discussed in sections 2.2.3 and 2.2.4 below).

The following instructions can use immediate addressing. The expression in the operand field should produce a value in the range 0..255 ($0..$FF) for the two-byte instructions and 0..65535 ($0..$FFFF) for the three-byte instructions. The two-byte immediate instructions are:

ADCA	ADCB	ADDA	ADDB	ANDA	ANDB	BITA	BITB
CMPA	CMPB	EORA	EORB	LDAA	LDAB	ORAA	ORAB
SBCA	SBCB	SUBA	SUBB				

The three-byte immediate instructions are:

CPX	LDS	LDX	
(for HD6801/HD6301 only:	ADDD	LDD	SUBD)

2.2.3 Direct and Extended Addressing Modes

Instructions with direct and extended addressing require an operand field consisting of an expression (which the HD6800/6801/6301 interprets as an address). If the instruction can use direct addressing, and the expression contains no forward references (see section 2.3.3 for an explanation of forward references) and evaluates to a value in the range 0..255 ($0..$FF), direct addressing will be automatically selected, and the instruction will produce two bytes of code. In all other cases, extended addressing will be used, and the instruction will produce three bytes of code.

The following instructions can use either direct or extended addressing:

ADCA	ADCB	ADDA	ADDB	ANDA	ANDB	BITA	BITB
CMPA	CMPB	CPX	EORA	EORB	LDAA	LDAB	LDS
LDX	ORAA	ORAB	SBCA	SBCB	SUBA	SUBB	
(for HD6801/HD6301 only:			ADDD	LDD	STD	SUBD)	

The following can use extended but not direct addressing:

ASL	ASR	CLR	COM	DEC	INC	JMP	JSR
LSR	NEG	ROL	ROR	TST			
(for HD6801/HD6301 only: LSL)							

For HD6801/HD6301 only: JSR can use direct as well as extended addressing.

2.2.4 Indexed Addressing Mode

Instructions with indexed addressing require an operand field consisting of an expression (interpreted by the 6800/6801/6301 as an address) followed immediately by the two-character suffix ",X". For example: "LDAA OFFSET,X". They generate three bytes of code.

The following instructions can use indexed addressing. Note that they are the same instructions as those which can use extended addressing:

ADCA	ADCB	ADDA	ADDB	ANDA	ANDB	ASL	ASR
BITA	BITB	CLR	CMPA	CMPB	COM	CPX	DEC
EORA	EORB	INC	JMP	JSR	LDAA	LDAB	LDS
LDX	LSR	NEG	ORAA	ORAB	ROL	ROR	SBCA
SBCB	SUBA	SUBB	TST				
(for HD6801/HD6301 only:			ADDD	LDD	LSL	STD	SUBD)

2.2.5 Relative Addressing Mode

Instructions with relative addressing require an operand field consisting of an expression producing as a value an address which is in the range *-126 to *+129 where "*" is the location counter value prior to processing the instruction (ie, "*" is the address at which the first byte of the instruction will be placed).

The following instructions use relative addressing (exclusively):

BCC	BCS	BEQ	BGE	BGT	BHI	BLE	BLS
BLT	BMI	BNE	BPL	BRA	BSR	BVC	BVS
(for HD6801/HD6301 only:			BHS	BLO	BRN)		

2.2.6 Immediate-Direct Addressing Mode (HD6301 only)

Instructions with immediate-direct addressing require two operands: the character "#" followed by an expression (in the range 0..255, or $0..$FF), and a direct address value (in the range 0..255, or $0..$FF), separated by commas. For example: "AIM #$7F,$D5".

The following HD6301 instructions use immediate-direct addressing (plus immediate-indexed addressing):

AIM EIM OIM TIM

2.2.7 Immediate-Indexed Addressing Mode (HD6301 only)

Instructions with immediate-indexed addressing require two operands: the character "#" followed by an expression (in the range 0..255, or $0..$FF), and an expression (with a value in the range 0..255, or $0..$FF) followed immediately by the two-character suffix ",X", separated by commas. For example: "OIM #$18,2,X".

See section 2.2.6 above for the HD6301 instructions which use immediate-indexed (and immediate-direct) addressing.

2.2.8 Bit-Direct Addressing Mode (HD6301 only)

Instructions with bit-direct addressing require two operands: the bit number (in the range 0..7) and a direct address value (in the range 0..255, or $0..$FF), separated by commas. For example: "BSET 4,$3B".

The following HD6301 instructions use bit-direct (and bit-indexed) addressing:

BCLR BSET BTGL BTST

2.2.9 Bit-Indexed Addressing Mode (HD6301 only)

These instructions (the same ones as listed in section 2.2.8 above) require two operands: the bit number (in the range 0..7), and an expression (with a value in the range 0..255, or $0..$FF) followed immediately by the two-character suffix ",X", separated by commas. For example: "BTGL 3,OFST,X".

2.3 Directives

Directives differ from instructions in that they do not produce code for instructions which can be executed by the 6800/6801/6301. Some place data constants into the code; others generate no code at all, but are used by the assembler for other purposes, such as changing the location counter or signifying the end of your program. The valid assembler directives are:

END ENDC EQU FCB FCC FDB IFC IFEQ
IFGE IFGT IFLE IFLT IFNC IFNE ORG RMB
SET SPC

2.3.1 END

The END directive is used to indicate the end of your program. It cannot be omitted, and must be the last line of the program. Any lines following the first END statement occurring in the file will be ignored (they will not even be listed). The END statement does not use the operand field.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.                    PAGE

                          1 $MOD31
                          2 *
                          3 *  END DIRECTIVE
                          4 *

  1000                    6          ORG      $1000
  1000 CC 0000            7 START    LDD      #0
  1003 FD 1006            8          STD      VAL

  1006                   10 VAL      RMB      2
  1008                   11          END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 2-3 END directive

2.3.2 ENDC

The ENDC directive is used to indicate the end of a conditional assembly section initiated by a corresponding IFC directive (see section 2.3.7). The maximum nesting depth allowed for conditional assembly directives (IFxx..ENDC blocks) is eight.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.          PAGE    1

                          1 *
                          2 *  CONDITIONAL ASSEMBLY
                          3 *

  0001                    5 FLAG1   SET     1
  0002                    6 FLAG2   SET     2

                          8         IFEQ    FLAG1
                          9         FCC     ' SKIP '
                         10         ENDC

                         12         IFNE    FLAG2
  0000 20 41 53 53       13         FCC     ' ASSEMBLE '
  0004 45 4D 42 4C       14
  0008 45 20             15
                         16         ENDC

                         18         IFEQ    FLAG1-1
  000A 20 41 53 53       19         FCC     ' ASSEMBLE '
  000E 45 4D 42 4C       20
  0012 45 20             21
                         22         ENDC

  0014                   24         END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 2-4 ENDC directive

2.3.3 EQU

The EQU (equate) directive is used to assign a value to a symbol. While a symbol may be defined by placing the symbol name in the label field of an instruction statement or a data constant directive, that definition can only give the symbol the value of the location counter at that point in the program. Symbol definition with the EQU directive allows the symbol to take on any value, which is stated as an expression in the [required] operand field. For example:

```
VAL     EQU     5+(3+4)*2
LOCA    EQU     *
LOCB    LDAA    #VAL
```

would assign the value 19 to VAL and use that value as the immediate operand of the LDAA instruction. LOCA and LOCB would both be assigned the same value, that of the location counter before the LDAA instruction is processed.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.          PAGE    1

                        1 *
                        2 *  EQU DIRECTIVE
                        3 *

 0000                   6 LAB1    RMB     100

 0005                   8 LAB2    EQU     LAB1+5

 0005                  10 LAB3    EQU     LAB2

 0180                  12 LAB4    EQU     75*512/100
 0180                  13 LAB5    EQU     $180
 9600                  14 LAB6    EQU     512*75
 0180                  15 LAB7    EQU     19200*2/100

 0064                  17         END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 2-5 EQU directive

One symbol may be used in defining another symbol. If the symbol used in an operand field expression is defined later in your program than its use in that expression, then it is said that the expression contains a "forward reference" to the symbol. Two-level forward referencing is when the definition of a forward-referenced symbol itself contains a forward reference. Such two-level forward referencing cannot be handled by the assembler, which will not be able to define the first symbol in the chain. Consider this section of a program:

```
VAL1    EQU     1
VAL2    EQU     VAL4
VAL3    EQU     VAL1
VAL4    EQU     VAL6
VAL5    EQU     VAL4
VAL6    EQU     6
VAL7    EQU     VAL3
```

In the above example, VAL1 is set to 1 and VAL6 is set to 6. VAL3 is also set to 1, then VAL7 is set to 1. Now consider VAL4; it contains a forward reference to VAL6 but the definition of VAL6 itself contains no forward references, so VAL4 will be correctly set to 6. VAL5 uses VAL4 after VAL4 has been defined, so again there is no problem. But VAL2 has a forward reference to VAL4, which in turn has a forward reference to VAL6. This is two levels of forward referencing, and cannot be processed correctly. VAL2 will be flagged as undefined.

2.3.4 FCB

The FCB (form constant byte) directive requires one or more operands separated by commas (","). The value of each of the operands should be in the range 0..255 ($0..$FF -- 8 bits). Each operand value will be stored as one byte in the generated code, in the order that the operands occur, left to right.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.                PAGE

                        1 *
                        2 *  FCB DIRECTIVE
                        3 *

  0000 FF               5           FCB     $FF
  0001 01 01            6           FCB     *,%01
  0003 0A               7 LAB       FCB     5*2
  0004 03 09 08         8           FCB     LAB,9,8
  0007                  9           END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 2-6 FCB directive

2.3.5 FCC

The FCC (form constant character string) directive requires in the operand field either: (1) a character string enclosed in single quotes; or (2) a number, followed by a comma, followed by a string in single quotes. Each character in the string will be stored as one byte in the generated code, in the order that the characters occur, left to right. If the second form of the operand is used, the number given will determine the number of bytes to be generated. If the number is greater than the count of characters in the string, space characters (" ", or $20) will be appended to the string until the specified number of bytes has been generated. Examples:

```
FORM1   FCC     'STRING OF TEXT'
FORM2   FCC     80,'STRING TO WHICH SPACES ARE APPENDED'
```

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.                PAGE    1

                        1 *
                        2 *  FCC DIRECTIVE
                        3 *

 0000 54 45 58 54       5 MSG1    FCC     'TEXT'
 0004 54 45 58 54       6 MSG2    FCC     9,'TEXT'
 0008 20 20 20 20       7
 000C 20                8
 000D                   9         END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 2-7 FCC directive

2.3.6 FDB

The FDB (form double-byte constant) directive requires one or more operands separated by commas. The value of each operand can be anywhere in the range 0..65535 ($0..$FFFF -- 16 bits). Each operand value will be stored as two consecutive bytes (more significant byte first) in the generated code.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.                    PAGE

                            1 *
                            2 *  FDB DIRECTIVE
                            3 *

  0000 00 02                5          FDB     2
  0002 00 0F 00 FF          6 LAB      FDB     $F,$FF,$FFF,$FFFF
  0006 0F FF FF FF          7
  000A 00 04                8          FDB     LAB+2
  000C                      9          END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 2-8 FDB directive

2.3.7 IFxx

These directives allow the assembler to conditionally assemble the section of your program between the IFxx directive and the corresponding ENDC directive. Two forms are available:

```
IFtype1 string1,string2
IFtype2 expression
```

With the first form, the two strings are compared. If they compare and the IFC directive is used, or if they do not compare and the IFNC directive is used, the conditional program section will be assembled; otherwise it will skipped.

With the second form, the expression is evaluated and compared to zero. The result of that comparison, along with the specific directive used, determines whether or not the conditional code will be assembled. The list of directives used with this form, along with the comparison results which will cause assembly of the conditional code to take place, is as follows:

```
IFEQ    expression = 0
IFGE    expression >= 0
IFGT    expression > 0
IFLE    expression <= 0
IFLT    expression < 0
IFNE    expression ≠ 0
```

The maximum nesting depth allowed for IFxx..ENDC blocks is eight.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.                PAGE    1

                                1 *
                                2 *  CONDITIONAL ASSEMBLY
                                3 *

  0001                          5 FLAG1   SET     1
  0002                          6 FLAG2   SET     2

                                8         IFEQ    FLAG1
                                9         FCC     ' SKIP '
                               10         ENDC

                               12         IFNE    FLAG2
  0000 20 41 53 53             13         FCC     ' ASSEMBLE '
  0004 45 4D 42 4C             14
  0008 45 20                   15
                               16         ENDC

                               18         IFEQ    FLAG1-1
  000A 20 41 53 53             19         FCC     ' ASSEMBLE '
  000E 45 4D 42 4C             20
  0012 45 20                   21
                               22         ENDC

  0014                         24         END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 2-9 IFxx directive

2.3.8 ORG

The ORG (origin) directive is used to change the value of the location counter to the value of the expression in the [required] operand field. The expression may not contain any forward references.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.          PAGE    1

                        1 *
                        2 *  ORG DIRECTIVE
                        3 *
0000                    4          RMB      10

0020                    6          ORG      $20
0020                    7          RMB      5

0100                    9          ORG      $100

0100                   11          END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 2-10 ORG directive

2.3.9 RMB

The RMB (reserve memory bytes) directive adds the value of the [required] operand field expression to the location counter, in order to reserve a block of memory, which is not set to any value. The expression may not contain any forward references.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.          PAGE    1

                          1
                          2 *
                          3 *  RMB DIRECTIVE
                          4 *

  0000                    6 CLAB1   RMB     1
  0001                    7 CLAB2   RMB     10
  000B                    8         END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 2-11 RMB directive

2.3.10 SET

The SET directive functions like the EQU directive (see section 2.3.3) except that it allows redefinition of the symbol in the label field. An operand is required. For a clarification of the difference between EQU and SET, see the explanation on the definition of symbols (section 2.1.3).

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.          PAGE    1

                          1 *
                          2 *  SET DIRECTIVE
                          3 *

  0000                    6 SLAB1   SET     *

  0000                    8 SLAB2   SET     SLAB1

  0005                   10 SLAB2   SET     5

  0004                   12 SLAB2   SET     SLAB2-1

  0000                   14         END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 2-12 SET directive

2.3.11 SPC

The SPC (space) directive is used to skip a number of blank lines on the listing. The line with the SPC directive is not printed, but in its place are inserted blank lines. The [required] operand value is the number of blank lines to be inserted (limited to the number of lines remaining on the listing page). "SPC 1" produces the same effect as a blank source line. "SPC 255" produces the same effect as a page eject control with normal length pages.

An assembly is initiated by one of the following commands to the ISIS-II operating system:

```
ASM31 sourcefile controls
:F1:ASM31 sourcefile controls
:Fn:ASM31 sourcefile controls
```

The first form is used if the assembler program, ASM31, is on the diskette inserted into drive 0 (typically your system diskette). The second form is used if the assembler is on the diskette inserted into drive 1. The third command is a general form of the second, where the drive number where the diskette with the assembler is located is substituted for the letter "n" in ":Fn:ASM31".

"sourcefile" is the name of a file containing the program to be assembled. A typical name might be ":F1:TEST2.M31" (if the file is on the diskette in drive 1).

"controls" is an optional list of assembly controls, in any order, as discussed below.

3.1 Primary vs General Controls

Controls are classified according to where they can be used. A primary control may appear in the console command, or as an embedded control line before the first assembly language statement, blank line, or comment line in the source file. A general control may appear in the console command, or as an embedded control line at any location in the source file prior to the END statement.

An embedded control line in the source file begins with a dollar sign ("$") as the first character and may contain one or more controls.

Some controls require a file name or a number enclosed in parentheses following the control name. There may optionally be one or more spaces between the control name and the open parenthesis character ("(").

3.2 Primary Controls

PRINT (name) Causes an assembly listing to be written to the specified file or device. Examples:

```
PRINT (:LP:)  -- list to the line printer
PRINT (:F1:TMPFIL.LST)  -- list to file
PRINT (:CO:)  -- list to the console
```

NOPRINT Suppresses the assembly listing.

OBJECT (name)	Causes an object module in Intel* hexadecimal paper tape format to be written to the specified file or device. Examples: OBJECT (:F1:OBJFIL.HEX) -- to disk file OBJECT (:HP:) -- output to tape punch
NOOBJECT	Suppresses the object module.
MOD01	Allows use of HD6801 opcodes (see section 1).
MOD31	Allows use of HD6301 opcodes (see section 1).
SYMBOLS	Causes a sorted listing of all user-defined symbols to be written on the list device at the end of the assembly.
NOSYMBOLS	Suppresses the symbol table listing.
XREF	Causes a sorted cross reference listing of all symbols, giving the line numbers where the symbol was used and defined, to be written on the list device at the end of the assembly.
NOXREF	Suppresses the cross reference listing.
PAGING	Causes the assembly listing to be segmented into pages, with a page number and header at the top of each page.
NOPAGING	Suppresses the paging option.

* "Intel" is a registered trademark of Intel Corporation.

FORMS — Causes the assembler to issue a form feed character for moving the listing to the top of a new page.

NOFORMS — Suppresses the forms option; causes the assembler to issue several line feed characters instead of the form feed character.

PAGEWIDTH (nn) — Causes the assembler listing pages to be "nn" columns wide. Examples:

```
PAGEWIDTH (132)
PAGEWIDTH (80)
```

PAGELENGTH (nn) — Causes the assembler listing pages to be "nn" lines long. Examples:

```
PAGELENGTH (84)  -- legal size paper
PAGELENGTH (66)  -- letter size paper
```

3.3 General Controls

LIST — Enables the listing of source statements until the next NOLIST control is encountered.

NOLIST — Disables the listing of statements until the next LIST control is encountered.

TITLE ('text') — Causes the specified text line (maximum 72 characters) to be printed as a title at the top of each page of the listing.

EJECT — Causes the next line of the listing to begin at the top of a new page.

INCLUDE (name) Following this control line, the assembler will input source lines from the file specified by "name"; at the end of the "name"d file, input of source lines will resume from the original source file, immediately following the INCLUDE control line. INCLUDEs can be nested to a depth of five.

3.4 Initial Settings of Controls

If not, or until, explicitly set in the command line or in embedded control line statements, the default settings of the controls are as follows:

```
PRINT (sourcefilename.LST)
OBJECT (sourcefilename.HEX)
SYMBOLS
NOXREF
PAGING
FORMS
PAGEWIDTH (120)
PAGELENGTH (66)
```

In explanation of the default PRINT and OBJECT controls: if not explicitly specified and not suppressed by the NOPRINT or NOOBJECT controls, the default listing and object files have the same name and disk drive number as the source file, and the extensions .LST for the listing file and .HEX for the object module file.

The MOD01 and MOD31 options are not assumed unless explicitly requested; without them ASM31 recognizes only HD6800 opcodes.

4 DESCRIPTION OF FILES PRODUCED

4.1 Listing File

The listing file (normally produced unless suppressed by the NOPRINT control) contains a listing of your source program with generated object code and any error messages, a symbol table (unless suppressed by the NOSYMBOLS control), and (if selected by the XREF control) a cross reference table. These sections of the listing are described in further detail in the following sections. (This description assumes that you are using the PAGING option.)

4.1.1 Source Listing Format

The first line on each page identifies the version of the assembler and gives the page number of the listing. The second line contains the title specified by the TITLE directive, if any.

The rest of the page consists of source file lines, one per listing line. There are several items on each listing line to the left of the source line. The first is the error character, normally blank unless the assembler detected an error in processing the source line. Next is a four-hexadecimal-digit value: on EQU and SET statements, it is the value assigned to the symbol in the label field. On ORG statements, it is the new location counter value. On instruction and data definition directive statements, it is the location at which the first byte is placed.

```
                        1 $MOD31

                        3 * REVISION 1
                        4 * CONVERT BINARY TO DECIMAL & STORE 5 CHAR
                        5 * (A,B) BINARY VALUE
                        6 * (X) POINTER TO STORE DACIMAL CHARS
                        7 * TEMPORARY STORAGE
0100                    8          ORG     $100

0100                   10 SAVEA    RMB     1       ACCUMULATOR A
0101                   11 SAVEX    RMB     2       STORE DATA POINTER
0103                   12 SAVEX1   RMB     2       POINTER TO CONSTANTS

0200                   14          ORG     $200
0200 FF 0101           15 CVBTD    STX     SAVEX   SAVE DATA POINTER
0203 CE 022F           16          LDX     #K10K   (X) POINTER TO CONSTANTS
0206 7F 0100           17 CVDEC1   CLR     SAVEA   INZ DEC CHAR
0209 A3 00             18 CVDEC2   SUBD    0,X
020B 25 05             19          BCS     CVDEC5  OVERFLOW
020D 7C 0100           20          INC     SAVEA   INC CHAR BEING BUILT
0210 20 F7             21          BRA     CVDEC2
0212 E3 00             22 CVDEC5   ADDD    0,X     RESTORE PARTICAL RESULT
0214 36                23          PSHA
0215 FF 0103           24          STX     SAVEX1
0218 FE 0101           25          LDX     SAVEX   X - STORE CHAR POINTER
021B 8B 30             26          ADDA    #$30    MAKE ASCII CHAR
021D A7 00             27          STAA    0,X
021F 32                28          PULA            RESTORE A
0220 08                29          INX
0221 FF 0101           30          STX     SAVEX
0224 FE 0103           31          LDX     SAVEX1  X=POINTER TO CONSTANTS
0227 08                32          INX
0228 08                33          INX
0229 8C 0239           34          CPX     #K10K+10
022C 26 D8             35          BNE     CVDEC1
022E 3E                36          WAI

                       38 * CONSTANTS FOR CONVERSION
022F 27 10             39 K10K     FDB     10000
0231 03 E8             40          FDB     1000
0233 00 64             41          FDB     100
0235 00 0A             42          FDB     10
0237 00 01             43          FDB     1
0239                   44          END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. 4-1 Source listing

4.1.2 Error Messages

The first error detected by the assembler on any source line processed is reported by a single letter placed in the first column of the listing line. The possible error codes are:

A - Invalid addressing mode.
B - Relative branch range error.
C - Command line error. An embedded control line in error, or a primary control occurring after an assembly statement.
D - Depth error. Complex expression overflows internal stack.
I - Invalid character in source.
L - Invalid label in label field.
M - Multiple symbol definition.
N - Nesting error. Parentheses or quotes do not match.
O - Opcode error. Unrecognizable opcode.
P - Phase error. Due to illegal forward references elsewhere.
Q - Questionable syntax.
R - Invalid or missing register specification.
S - Syntax error. Missing or extra operands.
U - Undefined symbol reference.
V - Value error. Operand exceeds allowed range.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.                PAGE    1

                                1           ORG     $1000   START AT 1000 HEX
                                2 *
                                3 IMMED     EQU     %01001110
                                4 LONGI     EQU     $1FF0
                                5 DIRECT    EQU     $6F
                                6 INDEX     EQU     $3C
                                7 *
S 1000 30 008B                  8           ADD A
  1003 48                       9           ASL A
B 1004 20 7D                   10           BRA     $2083
  1006 BD 006F                 11           JSR     DIRECT
                               12 *
  1009                         13           END

ASSEMBLY COMPLETE.    2 ERRORS.
```

(error code)

Fig. 4-2 Error Message

At the end of the source listing (after the END statement), a line is printed which says:

ASSEMBLY COMPLETE. n ERRORS.

where "n" is the number of lines in which errors were detected.

In addition to the error messages on the listing, some error messages may appear on the console. Discussion of these is deferred to the description of the console output (section 4.3).

4.1.3 Symbol Table Format

The symbol table starts on a new page. Each entry (there are several per line) consists of a symbol's name and its value (as a four-hexadecimal-digit number).

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.                PAGE   2

CVBTD  0200   CVDEC1 0206   CVDEC2 0209   CVDEC5 0212   K10K   022F
SAVEA  0100   SAVEX  0101   SAVEX1 0103

SYMBOL LISTING COMPLETE.
```

Fig. 4-3 Symbol table

4.1.4 Cross Reference Format

The cross reference table starts on a new page. Each line gives the symbol name followed by a list of lines where it is referenced. Any line where the symbol is defined has the character "#" following the line number.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.                PAGE   3

CV8TD       15#
CVDEC1      17#    35
CVDEC2      18#    21
CVDECS      19     22#
K10K        16     34     39#
SAVEA       10#    17     20
SAVEX       11#    15     25     30
SAVEX1      12#    24     31

CROSS REFERENCE COMPLETE.
```

Fig. 4-4 Cross reference table

4.2 Object File

The object file contains a series of records in the Intel* hexadecimal paper tape format. To convert the file to an ISIS-II compatible absolute object file, use the ISIS-II command HEXOBJ:

HEXOBJ filename.HEX to filename.OBJ

Remember to prefix the "filename" with the appropriate disk drive number in each case.

4.3 Console Output

Unless PRINT or OBJECT output is directed to the console (in which case see sections 4.1 and 4.2 for a description of that output), the console output is limited to the following:

1. A banner giving the version number of the assembler
2. "ASSEMBLY COMPLETE. n ERRORS." (as in listing file)
3. "COMMAND ERROR." if there is a problem with the controls in the command line invoking the ASM68 assembler.
4. "SYMBOL TABLE OVERFLOW." if the program being assembled has more labels than can be contained in available memory. Note: specifying the XREF option reduces the number of labels that can be handled.
5. "ERROR n USER PC xxxx": ISIS-II error messages which can occur at any time, due to disk errors, etc. A common cause of ERROR 35 (end of file): no END line.

* "Intel" is a registered trademark of Intel Corporation, Santa Clara CA.

APPENDIX A INSTRUCTION TABLE

OP : operation code
∿ : cycle count
: byte count

Mnemonic Code	6301 only	6801/6301 only	Addressing Mode																												
			Inherent			Immediate			Direct			Extened			Indexed			Relative			Immediate -Direct			Immediate -Indexed			Bit -Direct			Bit -Indexed		
			OP	∿	#	OP	∿	#	OP	∿	#	OP	∿	#	OP	∿	#	OP	∿	#	OP	∿	#	OP	∿	#	OP	∿	#	OP	∿	#
ABA			1B	2	1																										
ABX		●	3A	3	1																										
ADCA						89	2	2	99	3	2	B9	4	3	A9	4	2														
ADCB						C9	2	2	D9	3	2	F9	4	3	E9	4	2														
ADDA						8B	2	2	9B	3	2	BB	4	3	AB	4	2														
ADDB						CB	2	2	DB	3	2	FB	4	3	EB	4	2														
ADDD		●				C3	4	3	D3	5	2	F3	6	3	E3	6	2														
AIM	●																				71	6	3	61	7	3						
ANDA						84	2	2	94	3	2	B4	4	3	A4	4	2														
ANDB						C4	2	2	D4	3	2	F4	4	3	E4	4	2														
ASL												78	6	3	68	6	2														
ASLA			48	2	1																										
ASLB			58	2	1																										
ASLD		●	05	3	1																										
ASR												77	6	3	67	6	2														
ASRA			47	2	1																										
ASRB			57	2	1																										
BCC																		24	3	2											
BCLR	●																										71	6	3	61	7	3
BCS																		25	3	2											
BEQ																		27	3	2											
BGE																		2C	3	2											
BGT																		2E	3	2											
BHI																		22	3	2											
BHS		●																24	3	2											
BITA						85	2	2	95	3	2	B5	4	3	A5	4	2														
BITB						C5	2	2	D5	3	2	F5	4	3	E5	4	2														
BLE																		2F	3	2											
BLO																		25	3	2											
BLS																		23	3	2											
BLT																		2D	3	2											
BMI																		2B	3	2											
BNE																		26	3	2											
BPL																		2A	3	2											
BRA																		20	3	2											
BRN		●																21	3	2											
BSET	●																										72	6	3	62	7	3
BSR																		8D	6	2											
BTGL	●																										75	6	3	65	7	3
BTST	●																										7B	6	3	6B	7	3

Mnemonic Code	6301 only	6801/6301 only	Addressing Mode																												
			Inherent			Immediate			Direct			Extended			Indexed			Relative			Immediate -Direct			Immediate -Indexed			Bit -Direct			Bit -Indexed		
			OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#
BVC																		28	3	2												
BVS																		29	3	2												
CBA			11	2	1																										
CLC			0C	2	1																										
CLI			0E	2	1																										
CLR												7F	6	3	6F	6	2														
CLRA			4F	2	1																										
CLRB			5F	2	1																										
CLV			0A	2	1																										
CMPA						81	2	2	91	3	2	B1	4	3	A1	4	2														
CMPB						C1	2	2	D1	3	2	F1	4	3	E1	4	2														
COM												73	6	3	63	6	2														
COMA			43	2	1																										
COMB			53	2	1																										
CPX						8C	4	3	9C	5	2	BC	6	3	AC	6	2														
DAA			19	2	1																										
DEC												7A	6	3	6A	6	2														
DECA			4A	2	1																										
DECB			5A	2	1																										
DES			34	3	1																										
DEX			09	3	1																										
EIM	●																				75	6	3	65	7	3						
EORA						88	2	2	98	3	2	B8	4	3	A8	4	2														
EORB						C8	2	2	D8	3	2	F8	4	3	E8	4	2														
INC												7C	6	3	6C	6	2														
INCA			4C	2	1																										
INCB			5C	2	1																										
INS			31	3	1																										
INX			08	3	1																										
JMP												7E	3	3	6E	3	2														
JSR									9D	5	2	BD	6	3	AD	6	2														
LDAA						86	2	2	96	3	2	B6	4	3	A6	4	2														
LDAB						C6	2	2	D6	3	2	F6	4	3	E6	4	2														
LDD		●				CC	3	3	DC	4	2	FC	5	3	EC	5	2														
LDS						8E	3	3	9E	4	2	BE	5	3	AE	5	2														
LDX						CE	3	3	DE	4	2	FE	5	3	EE	5	2														
LSL		●										78	6	3	68	6	2														
LSLA		●	48	2	1																										
LSLB		●	58	2	1																										
LSLD		●	05	3	1																										
LSR												74	6	3	64	6	2														
LSRA			44	2	1																										
LSRB			54	2	1																										

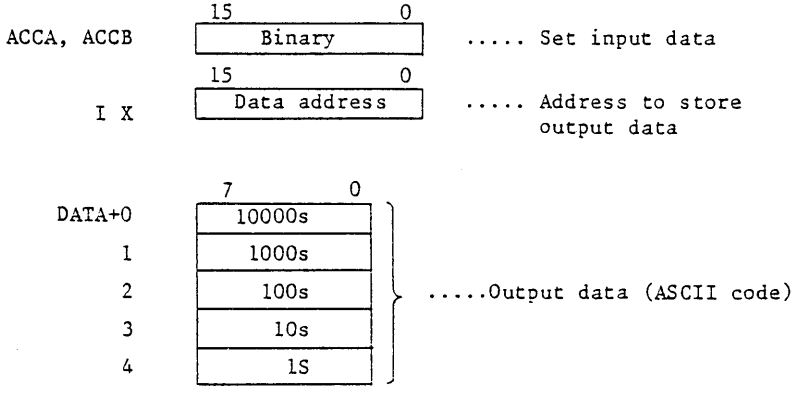
Mnemonic Code	6301 only	6801/6301 only	Addressing Mode																												
			Inherent			Immediate			Direct			Extened			Indexed			Relative			Immediate -Direct			Immediate -Indexed			Bit -Direct			Bit -Indexed		
			OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#
LSRD		●	04	3	1																										
MUL		●	3D	10	11																										
NEG									70	6	3	60	6	2																	
NEGA			40	2	1																										
NEGB			50	2	1																										
NOP			01	2	1																										
OIM	●																				72	6	3	62	7	3					
ORAA						8A	2	2	9A	3	2	BA	4	3	AA	4	2														
ORAB						CA	2	2	DA	3	2	FA	4	3	EA	4	2														
PSHA			36	3	1																										
PSHB			37	3	1																										
PSHX		●	3C	4	1																										
PULA			32	4	1																										
PULB			33	4	1																										
PULX		●	38	5	1																										
ROL												79	6	3	69	6	2														
ROLA			49	2	1																										
ROLB			59	2	1																										
RTI			3B	10	1																										
RTS			39	5	1																										
SBA			10	2	1																										
SBCA						82	2	2	92	3	2	B2	4	3	A2	4	2														
SBCB						C2	2	2	D2	3	2	F2	4	3	E2	4	2														
SEC			0D	2	1																										
SEI			0F	2	1																										
SEV			0B	2	1																										
SLP	●		1A	4	1																										
STAA									97	3	2	B7	4	3	A7	4	2														
STAB									D7	3	2	F7	4	3	E7	4	2														
STD		●							DD	4	2	FD	5	3	ED	5	2														
STS									9F	4	2	BF	5	3	AF	5	2														
STX									DF	4	2	FF	5	3	EF	5	2														
SUBA						80	2	2	90	3	2	B0	4	3	A0	4	2														
SUBB						C0	2	2	D0	3	2	F0	4	3	E0	4	2														
SUBD		●				83	4	3	93	5	2	B3	6	3	A3	6	2														
SWI			3F	12	1																										
TAB			16	2	1																										
TAP			06	2	1																										
TBA			17	2	1																										
TIM	●																				7B	6	3	6B	7	3					
TPA			07	2	1																										
TST												7D	6	3	6D	6	2														
TSTA			4D	2	1																										
TSTB			5D	2	1																										
TSX			30	3	1																										
TXS			35	3	1																										
WAI			3E	9	1																										
XGDX	●		18	2	1																										

APPENDIX B EXAMPLE OF PROGRAMMING

B.1 CVBTD Binary-to-Decimal Conversion Subroutine

CVBTD is a subroutine to convert a 16-bit binary number to 5-character decimal data.

Linkage: As shown below, sets the high-order 8 bis of input data in accumulator A and the low-order 8 bits in accumulator B. Then, sets the address to store output data in the index register.



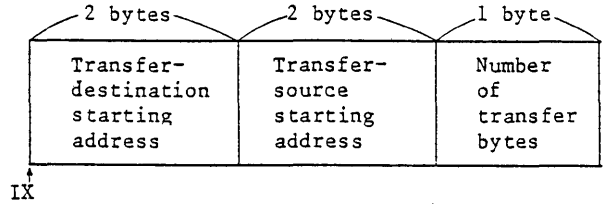
Result: 5-character ASCII codes are output to the addresses after Data address shown in the above.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.

                          1 $MOD31

                          3 * REVISION 1
                          4 * CONVERT BINARY TO DECIMAL & STORE 5 CHAR
                          5 * (A,B) BINARY VALUE
                          6 * (X) POINTER TO STORE DACIMAL CHARS
                          7 * TEMPORARY STORAGE
0100                      8          ORG      $100

0100                     10 SAVEA    RMB      1         ACCUMULATOR A
0101                     11 SAVEX    RMB      2         STORE DATA POINTER
0103                     12 SAVEX1   RMB      2         POINTER TO CONSTANTS

0200                     14          ORG      $200
0200 FF 0101             15 CVBTD    STX      SAVEX     SAVE DATA POINTER
0203 CE 022F             16          LDX      #K10K     (X) POINTER TO CONSTANTS
0206 7F 0100             17 CVDEC1   CLR      SAVEA     INZ DEC CHAR
0209 A3 00               18 CVDEC2   SUBD     0,X
020B 25 05               19          BCS      CVDEC5    OVERFLOW
020D 7C 0100             20          INC      SAVEA     INC CHAR BEING BUILT
0210 20 F7               21          BRA      CVDEC2
0212 E3 00               22 CVDEC5   ADDD     0,X       RESTORE PARTICAL RESULT
0214 36                  23          PSHA
0215 FF 0103             24          STX      SAVEX1
0218 FE 0101             25          LDX      SAVEX     X - STORE CHAR POINTER
021B 8B 30               26          ADDA     #$30      MAKE ASCII CHAR
021D A7 00               27          STAA     0,X
021F 32                  28          PULA               RESTORE A
0220 08                  29          INX
0221 FF 0101             30          STX      SAVEX
0224 FE 0103             31          LDX      SAVEX1    X-POINTER TO CONSTANTS
0227 08                  32          INX
0228 08                  33          INX
0229 8C 0239             34          CPX      #K10K+10
022C 26 D8               35          BNE      CVDEC1
022E 3E                  36          WAI

                         38 * CONSTANTS FOR CONVERSION
022F 27 10               39 K10K     FDB      10000
0231 03 E8               40          FDB      1000
0233 00 64               41          FDB      100
0235 00 0A               42          FDB      10
0237 00 01               43          FDB      1
0239                     44          END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. B-1 Binary-to-Decimal Conversion Subroutine

B.2 MOVE Memory-to-Memory Transfer Subroutine

Sets the starting address of transfer destination in the address which IX indicates, the starting address of transfer source in the IX+2 address, and the number of bytes to be transferred in the IX+4 address. Calls this subroutine after setting the address in which the starting address of transfer destination is stored.



```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.

                        1 $MOD31

                        3 ************************************************
                        4 *                                              *
                        5 *                                              *
                        6 * MOVE : THIS PROGRAM MOVE DATA                *
                        7 *                    BY SPECIFY LENGTH         *
                        8 *                                              *
                        9 *         IX = PARAMETER ADDRESS               *
                       10 *                                              *
                       11 *           IX+0 : BEGIN ADDRESS (2)           *
                       12 *           IX+2 : DESTINATION ADDRESS (2)     *
                       13 *           IX+4 : LENGTH (1)                  *
                       14 *                                              *
                       15 *                                              *
                       16 ************************************************

0400                   18          ORG     $400
0400                   19 MOVBEG   RMB     2
0402                   20 MOVDST   RMB     2

2000                   22          ORG     $2000
2000                   23 MOVE     EQU     *
2000 EC 00             24          LDD     0,X       ; SET PARAMETER
2002 FD 0400           25          STD     MOVBEG    ;
2005 EC 02             26          LDD     2,X       ;
2007 FD 0402           27          STD     MOVDST    ;
200A A6 04             28          LDAA    4,X
                       29 *
200C FE 0400           30          LDX     MOVBEG
200F E6 00             31 MOV010   LDAB    0,X
2011 3C                32          PSHX
2012 FE 0402           33          LDX     MOVDST
2015 E7 00             34          STAB    0,X
2017 08                35          INX
2018 FF 0402           36          STX     MOVDST
201B 38                37          PULX
201C 08                38          INX
201D 4A                39          DECA
201E 26 EF             40          BNE     MOV010
2020 3E                41          WAI
2021                   42          END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. B-2 Memory-to-Memory Transfer Subroutine

B.3 MULTI 16-Bit Multiplication Subroutine

MULTI is a subroutine to multiply 16-bit binary numbers and to output the product of 32-bit binary data.

Input parameter : ACCAB = Multiplicand
IX = Multiplier

Output parameter : Stored in 4-byte RESULT starting at the address $404, as shown below.

RESULT	+0	Highest-order byte
	+1	
	+2	
	+3	Lowest-order byte

```
1SIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.

                            1 $MOD51

                            3 ************************************************
                            4 *                                              *
                            5 *   MULTIPLY TWO 16-BIT POSITIVE VALUES        *
                            6 *   TO GENERATE A 32-BIT PRODUCT               *
                            7 *   AT TERMINATION , BOTH INPUT VALUES         *
                            8 *   AND THE RESULT WILL BE IN MEMORY           *
                            9 *                                              *
                           10 *   (A:B) * (C:D) = ACH:ACL                    *
                           11 *                 +     ADH:ADL               *
                           12 *                 +     BCH:BCL               *
                           13 *                 +           BDH:BDL         *
                           14 *                 ----- -----------            *
                           15 *                                              *
                           16 *         INPUT  : ACCAB = MULTIPLICAND        *
                           17 *                : IX = MULTIPLIER             *
                           18 *                                              *
                           19 *         OUTPUT : RESULT <4BYTE>              *
                           20 *                                              *
                           21 ************************************************

0400                       23          ORG     $400
0400                       24 MULAND   RMB     2
0402                       25 MULIER   RMB     2
0404                       26 RESULT   RMB     4

2000                       28          ORG     $2000
2000                       29 MULT     EQU     *
2000 FD 0400               30          STD     MULAND      ; SAVE MULTIPLICAND
2003 FF 0402               31          STX     MULIER      ; SAVE MULTIPLIER
2006 7F 0404               32          CLR     RESULT
2009 7F 0405               33          CLR     RESULT+1
200C 7F 0406               34          CLR     RESULT+2
200F B6 0401               35          LDAA    MULAND+1    ; #A LS BYTE
2012 F6 0403               36          LDAB    MULIER+1    ; #B LS BYTE
2015 3D                    37          MUL
2016 FD 0406               38          STD     RESULT+2
2019 B6 0400               39          LDAA    MULAND      ; #A MS BYTE
201C F6 0403               40          LDAB    MULIER+1    ; #B LS BYTE
201F 3D                    41          MUL
2020 F3 0405               42          ADDD    RESULT+1
2023 FD 0405               43          STD     RESULT+1
2026 24 03                 44          BCC     MULT10
2028 7C 0404               45          INC     RESULT
202B B6 0401               46 MULT10   LDAA    MULAND+1    ; #A LS BYTE
202E F6 0402               47          LDAB    MULIER      ; #B MS BYTE
2031 3D                    48          MUL
2032 F3 0405               49          ADDD    RESULT+1
2035 FD 0405               50          STD     RESULT+1
2038 24 03                 51          BCC     MULT20
203A 7C 0404               52          INC     RESULT
203D B6 0400               53 MULT20   LDAA    MULAND      ; #A MS BYTE
2040 F6 0402               54          LDAB    MULIER      ; #B MS BYTE
2043 3D                    55          MUL
2044 F3 0404               56          ADDD    RESULT
2047 FD 0404               57          STD     RESULT
204A 3E                    58          WAI
204B                       59          END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. B-3 16-Bit Multiplication Subroutine

APPENDIX C EXAMPLE OF PROGRAM DEVELOPMENT

The following shows an example of program development with the program shown in "Appendix B.2 MOVE Memory-to-Memory Transfer Subroutine". In this case, ISIS-II system diskette is set in unit 0 and the diskette containing 6301 cross assembler, in unit 1.

C.1 Producing Source Program

The following explains about producing source program with an example which uses ISIS-II text editor.

```
-EDIT :F1:MOVE.SRC (CR)  ————————————————————————  (a)
```

When inputting (a), the file of MOVE.SRC is assigned to the diskette in unit 1.

```
*I<source line>(CR)
  <source line>(CR)
        .
        .
        .
        .
        .
        .
        .
        .
        .
  <source line>(CR)

$$
↑↑————————————— displayed when keying in (ESC).
```

Key in I command first and source lines in succession. After completing the key-in of source lines, key in (ESC) twice.

```
* E$$
```

Key in E command, and then key in (ESC) twice. With this operation, the file of MOVE.SRC is produced in the diskette in unit 1. Contol returns to ISIS-II mode.

C.2 Assemble

```
-F1:ASM31 :F1:MOVE.SRC XREF PRINT(:LP:) CR ———————— (b)
```

When inputting (b), cross assembler ASM31 is loaded from the diskette in unit 1 and assembles. In this example, source file is input from the file of MOVE.SRC in the diskette in unit 1. Object file named MOVE.HEX is output to the diskette in unit 1. For lists, source object, symbol table and cross reference lists are output to the line printer. Fig. C-1, C-2 and C-3 show the output lists.

C.3 Conversion of Object File

```
-HEXOBJ :F1:MOVE.HEX TO :F1:MOVE.OBJ CR ———————— (c)
```

When inputting (c), the object file MOVE.HEX in Intel hexadecimal paper tape format which is output to the diskette in unit 1 by cross assembler is converted into the object file MOVE.OBJ in absolute format in the same diskette.

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.

                          1 $MOD31
                          3 ************************************************
                          4 *                                              *
                          5 *                                              *
                          6 * MOVE : THIS PROGRAM MOVE DATA                *
                          7 *                      BY SPECIFY LENGTH       *
                          8 *                                              *
                          9 *         IX = PARAMETER ADDRESS               *
                         10 *                                              *
                         11 *           IX+0 : BEGIN ADDRESS (2)           *
                         12 *           IX+2 : DESTINATION ADDRESS (2)     *
                         13 *           IX+4 : LENGTH (1)                  *
                         14 *                                              *
                         15 *                                              *
                         16 ************************************************

  0400                   18          ORG     $400
  0400                   19 MOVBEG   RMB     2
  0402                   20 MOVDST   RMB     2

  2000                   22          ORG     $2000
  2000                   23 MOVE     EQU     *
  2000 EC 00             24          LDD     0,X       ; SET PARAMETER
  2002 FD 0400           25          STD     MOVBEG    ;
  2005 EC 02             26          LDD     2,X       ;
  2007 FD 0402           27          STD     MOVDST    ;
  200A A6 04             28          LDAA    4,X
                         29 *
  200C FE 0400           30          LDX     MOVBEG
  200F E6 00             31 MOV010   LDAB    0,X
  2011 3C                32          PSHX
  2012 FE 0402           33          LDX     MOVDST
  2015 E7 00             34          STAB    0,X
  2017 08                35          INX
  2018 FF 0402           36          STX     MOVDST
  201B 38                37          PULX
  201C 08                38          INX
  201D 4A                39          DECA
  201E 26 EF             40          BNE     MOV010
  2020 3E                41          WAI
  2021                   42          END

ASSEMBLY COMPLETE. NO ERRORS.
```

Fig. C-1 Source Object List

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.

MOV010 200F   MOVBEG 0400   MOVDST 0402   MOVE   2000
SYMBOL LISTING COMPLETE.
```

Fig. C-2 Symbol Table List

```
ISIS-II 6301 CROSS ASSEMBLER, V1.0, HITACHI LTD.

MOV010      31#   40
MOVBEG      19#   25    30
MOVDST      20#   27    33    36
MOVE        23#

CROSS REFERENCE COMPLETE.
```

Fig. C-3 Cross Reference List

A World Leader in Technology

Hitachi America, Ltd.
Semiconductor and IC Sales and Service Division
1800 Bering Drive, San Jose, CA 95112
1-408-292-6404

HITACHI #U29

Printed in U.S.A.