HITACHI

6301/6801 ASSEMBLER TEXT EDITOR USER'S MANUAL

#U24

6301/6801 ASSEMBLER TEXT EDITOR USER'S MANUAL

HITACHI

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.

PREFACE

The 6301/6801 Assembler-Text Editor User's Manual is a detailed explanation of operational methods for the S31MIX1-R/S61MIX2-R Assembler-Text Editor. The S31MIX1-R/S61MIX2-R is used with either the 6301 Evaluation Kit (H31EVT1) or the 6801 Evaluation Kit (H61EVT2).

The 6301/6801 Assembler-Text Editor programmed in HN462732 EPROMs will be mounted on the evaluation kit (H31EVT1 or H61EVT2). It has functions to edit and revise text on paper tape medium and to assemble source programs.

When using the present manual, there are two others you should refer to for details on the 6301 and/or 6801 assembly languages:

°6801 Assembly Language Manual (S61ASL1-EM), and

°HD6301 User's Manual

For details on the H31EVT1 evaluation kit monitor, see:

°6301 Evaluation Kit User's Manual (H31EVT1-EM);

and on the H61EVT2 monitor, see:

°6801 Evaluation Kit User's Manual (H61EVT2-EM).

Table of Contents

1. THE SYSTEM

1.1 System Overview

The 6301/6801 Assembler-Text Editor is operated on either the 6301 (H31EVT1) or the 6801 (H61EVT2) evaluation kit. These two evaluation kits are highly effective tools for users designing and developing systems which use the HD6301 or HD6801. By using a console connected to either kit, the user can develop hardware or software systems with extreme efficiency. Figures 1-1 and 1-2 show the program diagrams.

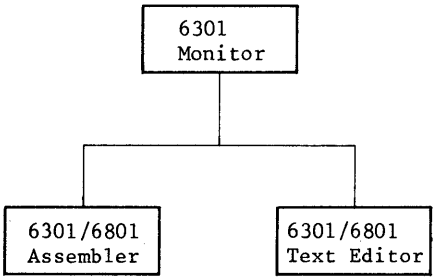


Figure 1-1 Programs in H31EVT1

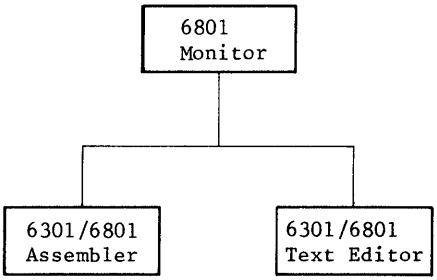


Figure 1-2 Programs in H61EVT2

The monitor program is indispensable to the evaluation kit. The user uses the program to debug hardware and/or software systems.

The 6301/6801 assembler program converts source programs, written in 6301/6801 assembly language, into object programs. Source and object programs are compatible with H68/SD series cross assemblers.

The user selects the 6301 or 6801 assembler by executing the assembler directive "OPT".

The 6301/6801 Text Editor is a program for editing and revising text. It is a particularly effective tool for revising and editing source programs input into the 6301 or the 6801 assembler.

1.2 System Equipment Configuration

Figure 1-3 shows the equipment configuration of the 6301 evaluation kit. The 6801 configuration is exactly the same.

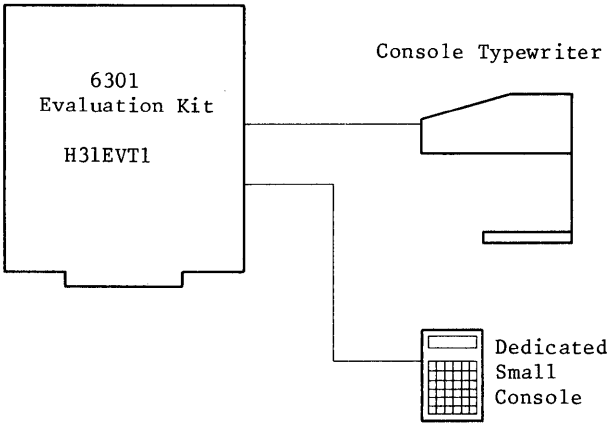


Figure 1-3 6301 Evaluation Kit Configuration

Execute the 6301/6801 Assembler-Text Editor with the console typewriter. You cannot use the small, dedicated console.

The EPROMs (HN462732) which are internal to the 6301/6801 Assembler-Text Editor will be mounted on the evaluation kit board.

1.3 Memory Map

Figure 1-4 is the software system memory map of the 6301 Evaluation Kit. Table 1-1 shows how each area is used. The chart and table also apply to the 6801 Evaluation Kit (H61EVT2).

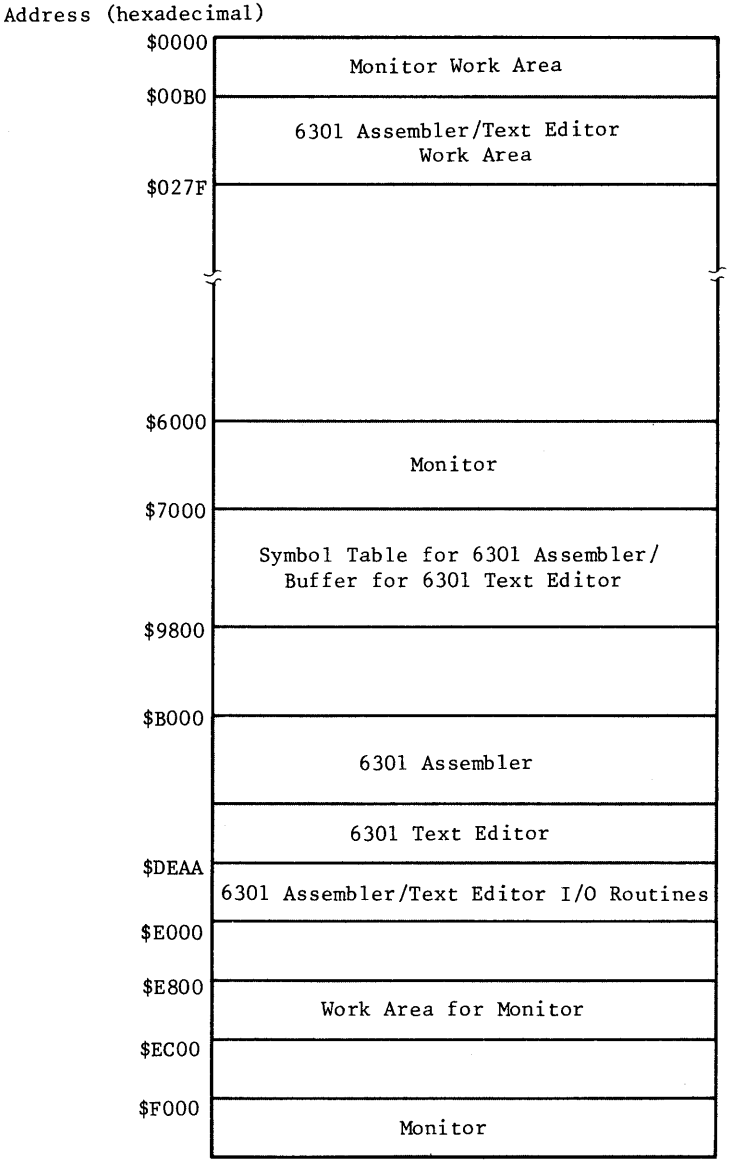


Figure 1-4 System Software Memory Map

Table 1-1 Use of Memory Areas

No.	Address (hexadecimal)	Use
1	$00B0 ∫ $027F	Work area and stack for 6301 assembler and 6301 text editor.
2	$7000 ∫ $9800	Area for 6301 text editor buffer or 6301 assembler symbol table.
3	$B000 ∫ $CCFF	Storage area for 6301 assembler
4	$D6DF ∫ $DE8F	Storage area for 6301 text editor
5	$DEAA ∫ $DFFF	Storage area for I/O routines shared by the 6301 assembler and 6301 text editor
6	$E800 ∫ $EBFF	Work area and monitor stack
7	$0000 ∫ $00AF $6000 ∫ $7000 $F000 ∫ $FFFF	Monitor areas

1.4 Using the System

The flow chart in Figure 1-5 shows the procedures for using this system to develop programs.

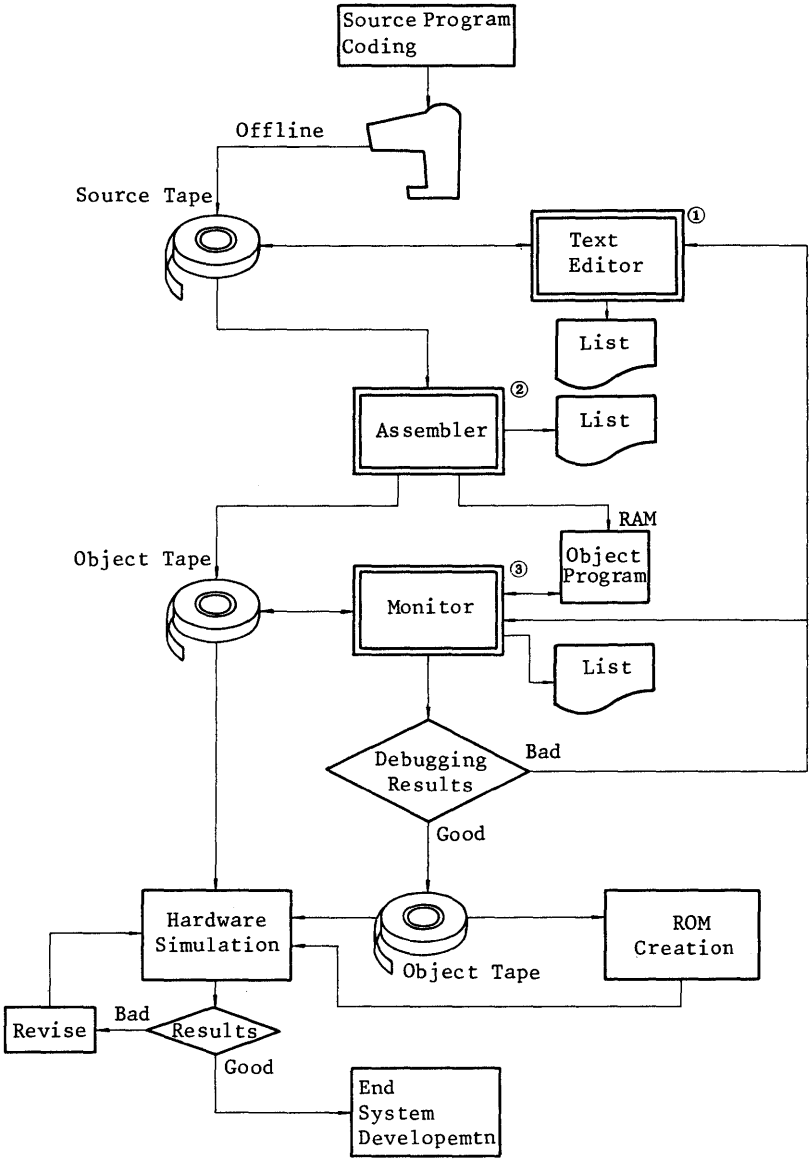


Figure 1-5 Program Development Procedure

Procedures in Figure 1-5:

① Input text from source tape, edit and revise. Output results to paper tape.

② Input source program from paper tape. Assemble and output object program to paper tape or directly to memory.

③ Use all monitor functions and debug object program stored in memory. With these results, revise the source program again and perform hardware simulation.

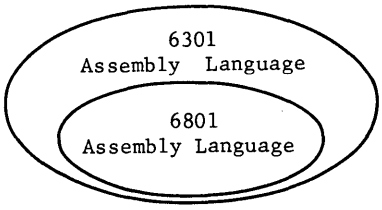
2. ASSEMBLER

2.1 Assembler Overview

The 6301/6801 Assembler (from now on, simply, "the assembler") programming system enters user source programs, written in 6301 or 6801 assembly language, onto paper tape. The assembler then converts the source programs into object programs.

You can use assembler source programs as input for 6301 (S31XAS2-F) or 6801 (S61XAS1-F) cross macro assemblers incorporated in the H68/SD series. The object program can be used for monitor input. It can also be used on paper tape for the creation of a mask read-only memory (ROM).

Figure 2-1 shows the relationship between the 6301 and 6801 assembly languages. For further details, see Tables A-1 and A-2 in appendix A "HD6301 and HD6801 Executive Instructions."



Note: 6301 assembly language has 10 more execution instructions than 6801 assembly language. (Table 2-5 lists the instructions which are exclusive to the 6301).

Table 2-1 6301 and 6801 Assembly Languages

2.2 Assembler Features

The assembler has the following features:

(1) Source programs compatible with source programs input to 6301 and 6801 cross macro assemblers.

(2) Direct output of object programs to memory.

(3) Adequate directives incorporated for easy observation of the assemble list.

(4) Displayable cross reference list (see (1) in 2.3.2) for easy observation of the assemble list.

(5) Reference to symbols used in other programs.

(6) Assembler directive "OPT" for switching to the 6801 assembler.

2.3 Assembler Input/Output

The assembler inputs source programs from paper tape and commands entered from the keyboard. It also outputs the assemble list and object program. Figure 2-2 diagrams assembler input/output.

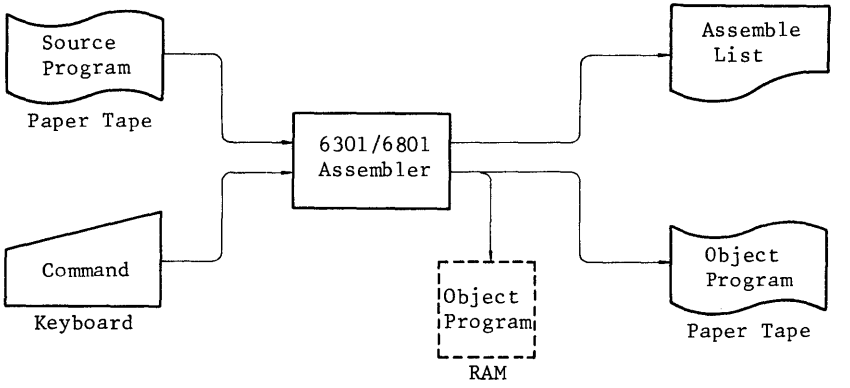


Figure 2-2 Assembler Input/Output

2.3.1 Input to the Assembler

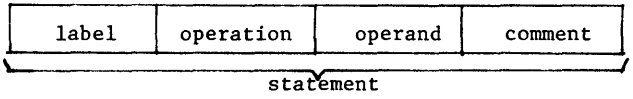
The assembler inputs paper tape on which source programs written in 6301 assembly language are punched. It inputs commands keyed in at the keyboard. For details on the commands, see 2.6, "Assembler Commands."

(1) Source statement format

A source program is a logical sequence of source statements written in assembly language. Each source statement is a sequence of ASCII characters ending with a carriage return (CR). See appendix B "ASCII Codes" to find out which characters in the ASCII set can be used.

Each statement line consists of the four fields shown below. If a label field begins with an asterisk ("*"), all remaining columns constitute the comment field.

A statement can have a maximum of 72 columns.



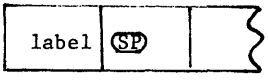
(Example)

```
*COMMENT LINE
LABEL (SP) CMPA (SP)   #$20 (SP) SPACE CODE? (CR)
label      operation   operand   comment
statement
```

(SP): at least one blank space

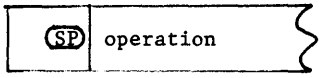
(a) Label field

The label (the symbol in the label field) is a name given to each statement so it can be easily referred to by an instruction. The label starts at the head of the statement.



When "*" is in the field header, the statement is a comment. Comments are output to the assemble list but are not converted into object codes.

If the header is blank, the statement has no label field, and consequently no label.



(i) Label rules

(a) A label symbol consists of 1 to 6 characters.

(b) The allowable symbol characters (in the label field) are:

°Alphabetics from A to Z,

°Numerics from 0 to 9.

(c) The first character in the symbol must be alphabetic.

(d) A, B and X are words reserved by the assembler and cannot be used independently as labels.

(e) Defining labels twice will cause an error.

(f) The location counter's value of the first byte in the instruction or data storage area is assigned to the label.

(g) You can assign any label to statements that have executive instructions. But, as Table 2-1 shows, certain assembler directives require a label and others must not be labeled.

Table 2-1 Assembler Directives for Label Addition

Type	Assembler Directive
Requires label	EQU
No label	END, NAM, OPT, ORG, PAGE, SPC

(h) The location counter's value cannot be assigned to the EQU directive. For further details, see the explanations for each directive.

(b) Operation Field

The operation field comes directly after the label field, and consists of an operation code of less than 5 characters. The two types of operation codes entered in the operation field are:

(i) HD6301 or HD6801 executive instructions (from now on, simply 6301 executive instructions or 6801 executive instructions)

These are operation codes shown in Appendix A's Table A-1 "6301 Executive Instructions" and Table A-2 "6801 Executive Instructions."

(ii) Assembler control instructions

The special operation codes recognized by the assembler. These codes will not be converted into machine language; they control assembler processing.

(c) Operand field

The 6301 and 6801 executive instructions determine the addressing mode in the operand field. Table 2-2 shows the operand field format and applicable addressing mode.

Table 2-2 Operand Formats

6301/6801	Operand Format	Addressing Mode
6801 or 6301	No operand	Implied
6801 or 6301	<expression>	Direct(1), extended or relative In selecting between direct (1) or extended mode, the assembler, if possible, automatically selects the direct address format.
6801 or 6301	#<expression>	Immediate
6801 or 6301	<expression>,X	Index(1)
6801 or 6301	"A" or "B"	Accumulator A space may or may not be entered between the operator and the accumulator designation. For example, "RORA" is the same as "RORΔA".
6301 only	<value>, <expression>*	Direct(2) Depending on the executive instruction, <value> takes the numeric value from 0 to 7 with BCLR, BSET, BTGL or BTST and the #<expression>* with AIM, EIM, OIM, or TIM. For example, "AIM #$3F,LABEL+3".
6301 only	<value>, <expression>*, X	Index(2) Just as with the Direct (2) format, <value> takes the numeric value from 0 to 7 or #<expression>* depending on what the executive instruction is. For example, "BCLR 3,DISP,X".

*The values in an expression are 1 byte (0 to 255).

(d) Comment field

The comment field is the final field in a source statement. When a program is assembled, the comment is not converted into machine language. It appears only in the source list. The

comment explains program processing, it helps you to understand the program. Comments are also important for their role in helping to simplify debugging and maintenance.

Rules on the comment field

(i) It is an optional field.

(ii) Leave at least one space after the operand field, then write the comment. If there is no operand field, leave at least one space after the operation field and then write the comment.

(iii) You can use any character in the ASCII code from $20(SP) to $5F(_).

(2) Expressions

An expression is composed of symbols, numerics and arithmetic operators and it specifies the operand value of the operation code. The arithmetic operators are:

(a) + addition

(b) - subtraction

(c) * multiplication

(d) / division

Expressions are computed serially from left to right no matter what the arithmetic operator is.

If an expression's operational result exceeds 2-byte lengths (65535), the value becomes undefined. If neither symbol nor numeric comes directly before an arithmetic operator, the computation is made as if a 0 were there.

(example: /SYM = 0/SYM)

Figure 2-3 is an example of expression use.

```
00001                                   NAM       EXPRES
00002           00AA        DATA1       EQU       $AA
00003           0055        DATA2       EQU       $55

00004                       * EXAMPLE OF EXPRESSION

00007   0000    0011                    FDB       8+6+3
00008   0002    0006                    FDB       4/2+4
00009   0004    000D                    FDB       5*4-7
00010   0006    08                      FCB       2*2*2

00012   0007    86  55                  LDA A     #DATA1-DATA2
00013   0009    B7  0011                STA A     WORK1
00014   000C    86  FF                  LDA A     #DATA2*3
00015   000E    B7  0012                STA A     WORK2

00017   0011    0001        WORK1       RMB       1
00018   0012    0001        WORK2       RMB       1
00019                                   END

TOTAL ERRORS 00000
```

Figure 2-3 Example of Expression Use

(a) Numerics

Table 2-3 shows the methods for expressing numerics.

Table 2-3 Expressing Numerics

Numeric Expression	Display Format	Example
Decimal	<numeric>	255
Hexadecimal	$<numeric> or <numeric>H (with the latter, the first digit must be 0 to 9)	$FF , 00FFH
Octal	@<numeric> or <numeric>O or <numeric>Q (only 0 to 7 can be used)	@377, 377O, 377Q
Binary	%<numeric> or <numeric>B (only 1 and 0 can be used)	%11111111, 11111111B

(b) Character constants

These constants are formed from character strings. Table 2-4 shows the method of expressing character constants.

Table 2-4 Expressing Character Constants

No.	Format	Explanation
1	'C	The character following "'" will be converted to 7-bit ASCII code.
2	n, character string	n characters following ",", will be converted to ASCII code.
3	d character string d	Character strings enclosed by d will be converted to ASCII code.

(c) Symbols

Symbols are strings of 1 to 6 alphanumerics beginning with an alphabetic. The rules for symbols are:

(i) The characters A, B and X are words reserved for the assembler and cannot be used by themselves.

(ii) "*" is the symbol for location counter. It also indicates the address of the first byte in an instruction word which has "*" in the operand.

(3) Addressing Modes

(a) Implied and accumulator addressing modes

In the HD6301 and HD6801, several operation codes have instructions with only one byte. These instructions are either in implied or accumulator addressing mode, and when coding in assembly language, there is no need to write in their operand fields.

(b) Immediate addressing mode

In the immediate addressing mode, 1 or 2 byte values can be immediately used for operands. Specify the immediate addressing mode by placing the character "#" at the beginning of the source statement's operand field. The expression after "#" takes a 1 or 2 byte value depending on the instruction.

(c) Relative addressing mode

The relative addressing mode is used with branching instructions. Branching is performed only when relative values from the branching instruction's first byte are -126 to +129.

$(PC + 2) - 128 < D < (PC + 2) + 127$

PC = address of first byte of branching instruction

D = address of destination to which branch is made.

The branch offset is actually entered into the 2nd byte of the branch machine instruction. It assigns, in 2's complementary, the difference between the branch destination address and the address directly after the branch instruction.

(d) Index addressing mode

The index address is related to the index register of the HD6301 or HD6801. When an instruction is executed, the effective address is calculated by adding the displacement in the machine instruction's 2nd byte to the 16 bit index register's present content. Since signs are not computed, negative values cannot be used in the offset.

The index addressing mode is specified by the characters ",X" usually after the operand field expression. ",X" or "X" can be expressed only when the character is "0,X".

(e) Direct and extended addressing modes

With direct or extended addressing modes, use 1 byte (direct) or 2 bytes (extended) in the operand's address. In the direct addressing mode, operand addresses are limited to a range of 0 to 255 in memory. Direct and extended addressing modes are differentiated according to the value of the expression in the source statement's operand field.

(4) 6301 and 6801 Executive Instructions

Table 2-5 lists the executive instructions for the 6301 and 6801. 6301 dedicated instructions are marked with an asterisk (*).

Table 2-5 Executive Instructions (addressing mode and machine cycle)

	2 Operand	ACCX	Immediate	Direct	Extended	Indexed	Implied	Relative
ABA		•	•	•	•	•	1-2	•
ABX		•	•	•	•	•	1-3	•
ADC		•	2	3	4	4	•	•
ADD		•	2	3	4	4	•	•
ADDD		•	3-4	4-5	5-6	5-6	•	•
*AIM	o	•	•	6	•	7	•	•
AND		•	2	3	4	4	•	•
ASL		1-2	•	•	6	6	•	•
ASLD		•	•	•	•	•	1-3	•
ASR		1-2	•	•	6	6	•	•
BCC		•	•	•	•	•	•	3
*BCLR	o	•	•	6	•	7	•	•
BCS		•	•	•	•	•	•	3
BEQ		•	•	•	•	•	•	3
BGE		•	•	•	•	•	•	3
BGT		•	•	•	•	•	•	3
BHI		•	•	•	•	•	•	3
BIT		•	2	3	4	4	•	•
BLE		•	•	•	•	•	•	3
BLS		•	•	•	•	•	•	3
BLT		•	•	•	•	•	•	3
BMI		•	•	•	•	•	•	3
BNE		•	•	•	•	•	•	3
BPL		•	•	•	•	•	•	3
BRA		•	•	•	•	•	•	3
BRN		•	•	•	•	•	•	3
*BSET	o	•	•	6	•	7	•	•
BSR		•	•	•	•	•	•	5-6
*BTGL	o	•	•	6	•	7	•	•
*BTST	o	•	•	4	•	5	•	•
BVC		•	•	•	•	•	•	3
BVS		•	•	•	•	•	•	3
CBA		•	•	•	•	•	1-2	•
CLC		•	•	•	•	•	1-2	•
CLI		•	•	•	•	•	1-2	•
CLR		1-2	•	•	5-6	5-6	•	•
CLV		•	•	•	•	•	1-2	•
CMP		•	2	3	4	4	•	•
COM		1-2	•	•	6	6	•	•
CPX		•	3-4	4-5	5-6	5-6	•	•
DAA		•	•	•	•	•	2	•
DEC		1-2	•	•	6	6	•	•
DES		•	•	•	•	•	1-3	•
DEX		•	•	•	•	•	1-3	•
*EIM	o	•	•	6	•	7	•	•
EOR		•	2	3	4	4	•	•
INC		1-2	•	•	6	6	•	•

	2 Operand	ACCX	Immediate	Direct	Extended	Indexed	Implied	Relative
INS		•	•	•	•	•	1-3	•
INX		•	•	•	•	•	1-3	•
JMP		•	•	•	3	3	•	•
JSR		•	•	5	6	5-6	•	•
LDA		•	2	3	4	4	•	•
LDD		•	3	4	5	5	•	•
LDS		•	3	4	5	5	•	•
LDX		•	3	4	5	5	•	•
LSR		1-2	•	•	6	6	•	•
LSRD		•	•	•	•	•	1-3	•
MUL		•	•	•	•	•	6-10	•
NEG		1-2	•	•	6	6	•	•
NOP		•	•	•	•	•	1-2	•
*OIM	o	•	•	6	•	7	•	•
ORA		•	2	3	4	4	•	•
PSH		4-3	•	•	•	•	•	•
PSHX		•	•	•	•	•	5-4	•
PUL		3-4	•	•	•	•	•	•
PULX		•	•	•	•	•	4-5	•
ROL		1-2	•	•	6	6	•	•
ROR		1-2	•	•	6	6	•	•
RTI		•	•	•	•	•	10	•
RTS		•	•	•	•	•	5	•
SBA		•	•	•	•	•	1-2	•
SBC		•	2	3	4	4	•	•
SEC		•	•	•	•	•	1-2	•
SEI		•	•	•	•	•	1-2	•
SEV		•	•	•	•	•	1-2	•
*SLP		•	•	•	•	•	4	•
STA		•	•	3	4	4	•	•
STD		•	•	4	5	5	•	•
STS		•	•	4	5	5	•	•
STX		•	•	4	5	5	•	•
SUB		•	2	3	4	4	•	•
SUBD		•	3-4	4-5	5-6	5-6	•	•
SWI		•	•	•	•	•	12	•
TAB		•	•	•	•	•	1-2	•
TAP		•	•	•	•	•	1-2	•
TBA		•	•	•	•	•	1-2	•
*TIM	o	•	•	4	•	5	•	•
TPA		•	•	•	•	•	1-2	•
TST		1-2	•	•	4-6	4-6	•	•
TSX		•	•	•	•	•	1-3	•
TXS		•	•	•	•	•	1-3	•
*XGDX		•	•	•	•	•	2	•
WAI		•	•	•	•	•	9	•

Note: Figures before "-" are the number of 6301 machine cycles, those after are the number of 6801 machine cycles.

2.3.2 Output from the Assembler

The assembler outputs the assembly listing and object program.

(1) Assembly Listing

There are 4 types:

- °Source statement listing
- °Error listing
- °Symbol table listing
- °Cross reference table listing

We will now explain these listings by first looking at Figure 2-4 which shows an entire assembly listing.

```
00001                          NAM     MOVE
00002                          OPT     SYMBOL,XREF
00003                          OPT     NOP
00004 0400                     ORG     $400
00005 0400 0002     MOVBEG     RMB     2
00006 0402 0002     MOVDST     RMB     2
00007 0404 0002     SAVEX      RMB     2


00009       0406    MOVE       EQU     *
00010 0406 EC 00               LDD     0,X       * SET PARAMETER
00011 0408 FD 0400             STD     MOVBEG    *
00012 040B EC 02               LDD     2,X       *
00013 040D FD 0402             STD     MOVDST    *
00014 0410 A6 04               LDA A   4,X
00015                  *
00016 0412 FE 0400             LDX     MOVBEG
00017 0415 E6 00    MOV010     LDA B   0,X
00018 0417 FF 0404             STX     SAVEX
00019 041A FE 0402             LDX     MOVDST
00020 041D E7 00               STA B   0,X
00021 041F 08                  INX
00022 0420 FF 0404             STX     SAVEX
00023 0423 FE 0404             LDX     SAVEX
00024 0426 08                  INX
00025 0427 4A                  DEC A
00026 0428 26 EB               BNE     MOV010
00027 042A 39                  RTS
00028                          END
MOVBEG 0400  MOVDST 0402  MOVE   0406  MOV010 0415  SAVEX  0404


MOVBEG 0400  00005* 00011  00016
MOVDST 0402  00006* 00013  00019
MOVE   0406  00009*
MOV010 0415  00017* 00026
SAVEX  0404  00007* 00018  00022  00023

TOTAL ERRORS 00000
```

Figure 2-4 Assembly Listing Example (entire)

(a) Source statement listing

The source statement listing includes the source statements, formatted for easier reading, as well as additional information generated by the assembler. Most lines in the listing correspond directly to a source statement. The listing is output in pass 2. Figure 2-5 is an example of a source statement listing.

```
PAGE  001   MOVE,   6801    ASSEMBLER  1.0
              ①

00001                        NAM    MOVE
00002                        OPT    SYMBOL
00003                        OPT    XREF
00004 0400                   ORG    $400
00005 0400 0002    MOVBEG    RMB    2
00006 0402 0002    MOVDST    RMB    2
00007 0404 0002    SAVEX     RMB    2

00009      0406    MOVE      EQU    *
00010 0406 EC 00             LDD    0,X        * SET PARAMETER
00011 0408 FD 0400           STD    MOVBEG     *
00012 040B EC 02             LDD    2,X        *
00013 040D FD 0402           STD    MOVDST     *
00014 0410 A6 04             LDA A  4,X
00015                   *
00016 0412 FE 0400           LDX    MOVBEG
00017 0415 E6 00    MOV010   LDA B  0,X
00018 0417 FF 0404           STX    SAVEX
00019 041A FE 0402           LDX    MOVDST
00020 041D E7 00             STA B  0,X
00021 041F 08                INX
00022 0420 FF 0402           STX    MOVDST
00023 0423 FE 0404           LDX    SAVEX
00024 0426 08                INX
00025 0427 4A                DEC A
00026 0428 26 EB             BNE    MOV010
00027 042A 39                RTS
00028                        END
  ②     ③   ④   ⑤                    ⑥
```

Figure 2-5 Source Statement Listing (example)

The circled numbers in Figure 2-5 are used to explain the source statement listing.

①: The program name for the assembly listing. Every time you go to a

new page in a list, the top of that page is displayed. The characters in the operand of the NAM directive, the first statement in the program, are used in the program name.

②: Statement numbers. The assembler assigns these numbers automatically.

③: Addresses in memory where the object program is stored (hexadecimal display).

④: Machine operation codes for the instructions (hexadecimal display).

⑤: Operand values of the instructions (hexadecimal display).

⑥: Formats and outputs the source statement.

(b) Error listing

A list is output of total errors occurring in pass 1 and pass 2 and of error messages for source statements which have produced errors. Figure 2-6 is an example of error listing output.

```
****ERROR  201
00001 0010            ORG $10
****ERROR  201
00002                 NAM PGM9
00003                *  ERROR PROGRAM
****ERROR  209
00004 0010 00 0000    LDA #$55
****ERROR  207
00005 0013 00 0000    BSS *
00006 0016 3D                 MUL
****ERROR  210
00007 0017 C6 2C      LDAB #300
00008                         END
                 Error Number

TOTAL ERRORS 00010
             Listing of Total Errors
```

Note: Error message and source statement where the error is located are output in pass 1.

Figure 2-6 Example of Error Listing Output

(c) Symbol table listing

This listing contains all symbols (labels) appearing in the source program, as well as the addresses or values by which those symbols were defined. The symbol values are displayed in hexadecimal.

The list will be output only if output is specified in the source program. Write the output specification as "OPT S" in the OPT directive.

Figure 2-7 is an example of symbol table output.

```
MOVBEG 0400 MOVDST 0402 MOVE 0406 MOV010 0415  SAVEX 0404
  ↑      ↑
  |      └address or value
  └symbol
```

Figure 2-7 Symbol Table Listing

(d) Cross reference table listing

This listing contains all symbols (labels) which appear in the source program as well as their corresponding defined addresses or values. It also contains the line numbers of defined or referenced statements. For the defined statement, the symbol displays "*" after the line number.

The listing will be output only if specified in the source program. If output is specified during execution, line numbers of label definitions and references prior to that point will not be output. Specify output by writing "OPT X" in the OPT directive.

If list output is specified and there is no space in the symbol table area for loading the cross reference table, the list won't be output (2 bytes are required to reference one symbol).

Figure 2-8 shows an example of cross reference table output.

```
MOVBEG 0400   00005* 00011   00016
MOVDST 0402   00006* 00013   00019   00022
MOVE   0406   00009*
MOV010 0415   00017* 00026
SAVEX  0404   00007* 00018   00023
  ①      ②              ③
```

1 : Symbols

2 : Addresses

3 : Line numbers being defined and referred to

("*" is a line number being defined.)

Figure 2-8 Cross Reference Table Listing

Note: When assembling two or more programs, an excess of line numbers will be output to the cross reference table listing.

(2) Object Program

Object programs are normally output to paper tape, but by executing an "OPT M" directive in the source program, you can have an object program output directly to memory.

(a) An object program output to paper tape will be in the S type format shown in Figure 2-9.

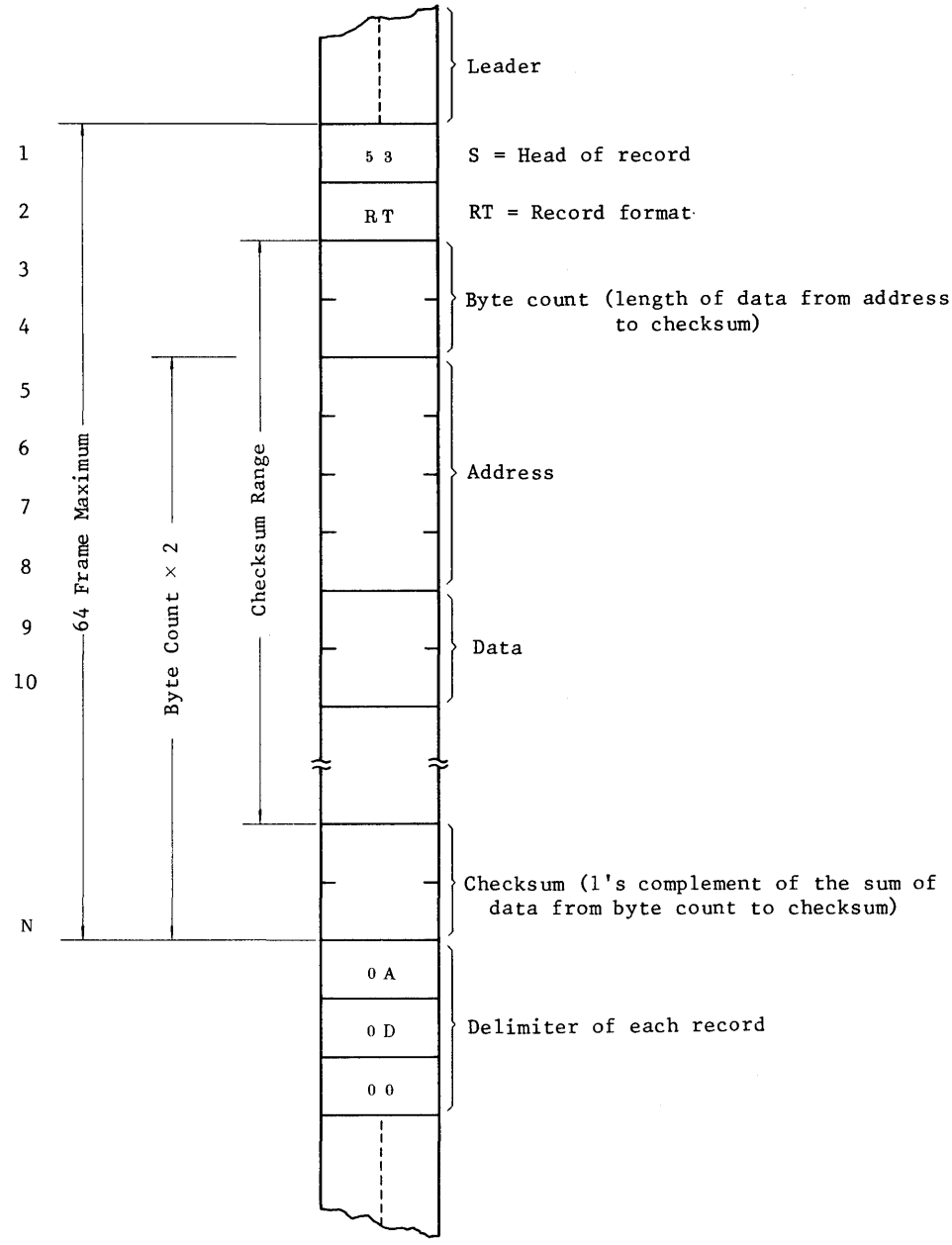


Figure 2-9 S Type Object Tape Format

Figure 2-10 is a diagram of all record formats.

① S00B0000484452202020202076
② S1161100864 4B701004142434445464748494A4B4C4D4E6D
③ S9030000FC

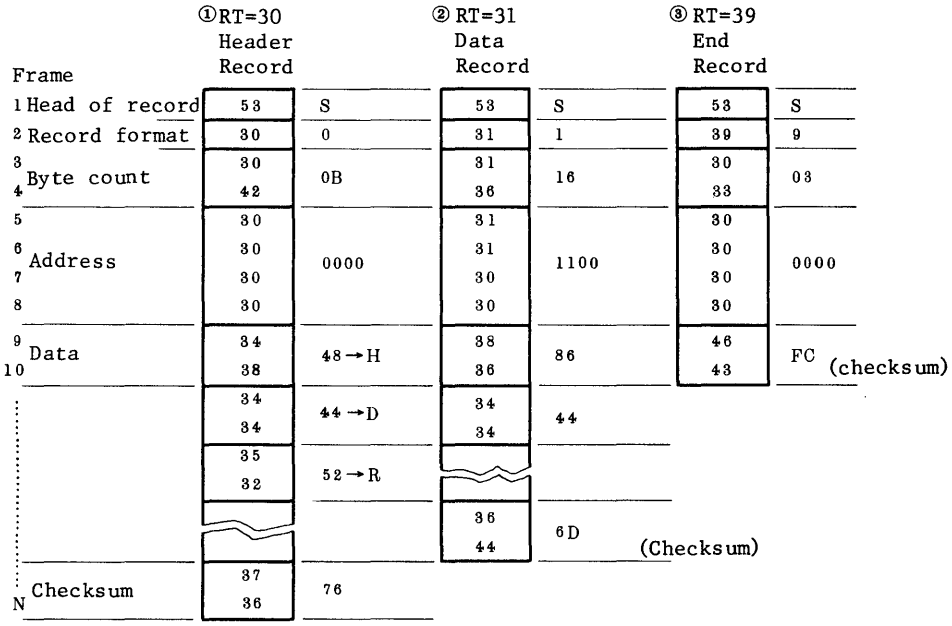


Figure 2-10 Record Formats for S Type Objects

In Figure 2-10:

①: The first record in the object program. Content written into the program's "NAM" directive operand is set into this record's data section.

②: This record includes machine codes.

③: The final record in the object program. The address is normally 0, but when an operand is written in the "END" directive, the operand value will be set as the start address.

(b) When "OPT M" is specified, the assembler outputs the object directly to the user memory area during assembler execution (pass 2).

By using this function, you don't have to load the object program into memory again after assembly terminates.

If you try to output the object to a non-packaged RAM area while using this function, the assembler will display error number 218. The object program will not then be output to memory, but to paper tape. Figure 2-11 shows the general concept of object output.

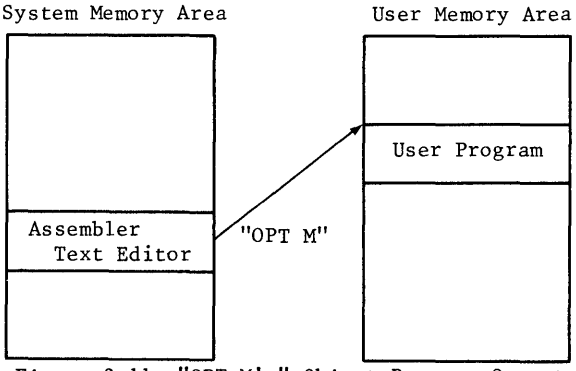


Figure 2-11 "OPT M's" Object Program Output

2.4 Executing the Assembler

Since the assembler is stored in the evaluation kit EPROM, turn on the power supply and key in as shown in Figure 2-12. The assembler then enters command request status.

```
/A(CR)
6301 ASSEMBLER
!
```

The user keys in the underlined section (as with all subsequent procedures)

Figure 2-12 Assembler Execution

Key to Figure 2-12:

① : Key in monitor command for assembler execution.

② : Display assembler title. Version and revision numbers are entered in the spaces marked v and r.

③ : Assembler enters command request status.

2.5 Operating the Assembler

2.5.1 Operational Overview

Figure 2-13 outlines the operations of program assembly. Code the source program first. Then, punch it onto paper tape at the console. When performing punch operations, the console must be off-line (LOCAL).

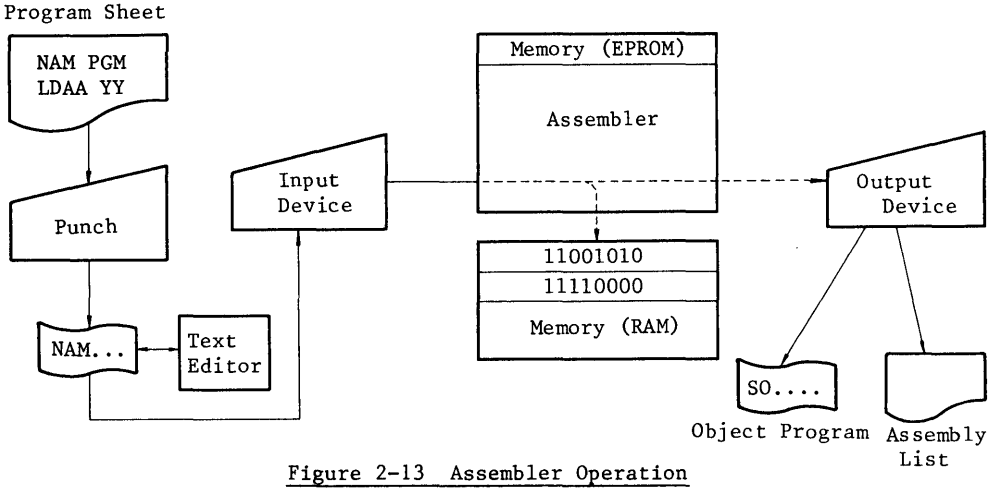


Figure 2-13 Assembler Operation

Use an assembler command to input paper tape created during offline operations or paper tape edited and corrected by text editor. Then, assemble and output the assembly listing or object tape to an output device.

2.5.2 Assembler Processing

The assembler inputs the source program twice. Figure 2-14 shows the flow for each pass.

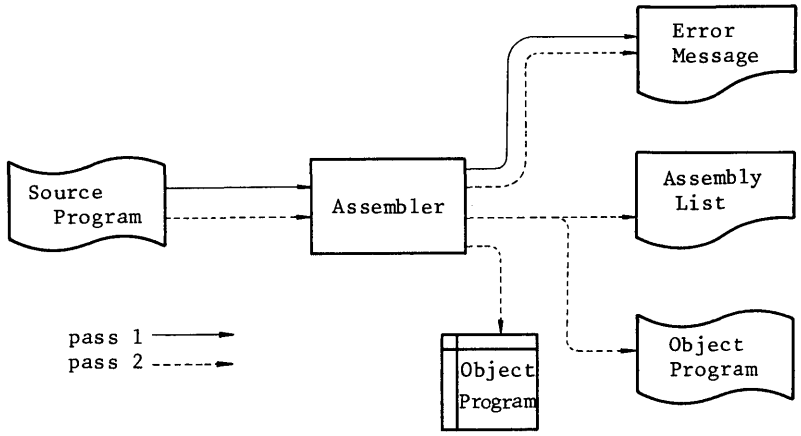


Figure 2-14 Flow of Assembly

The source program is checked for errors at each statement in pass 1. Every time an error occurs, an error message applicable to that statement is displayed. The symbol table is also created during pass 1.

In pass 2, the source program is read in and the assembly listing or object program is output in line with the specification. Pass 1 execution must end before pass 2 execution starts. Always process pass 1 first.

Each pass is specified by command. When a pass ends, the assembler again enters command request status. But, a word of caution: An END directive must be assigned to end a source program, otherwise the pass will not be considered ended and the assembler will not enter command request status.

2.6 Assembler Commands

This section explains the specifics of assembler commands using the table format shown. Figure 2-15 is the format for explaining commands.

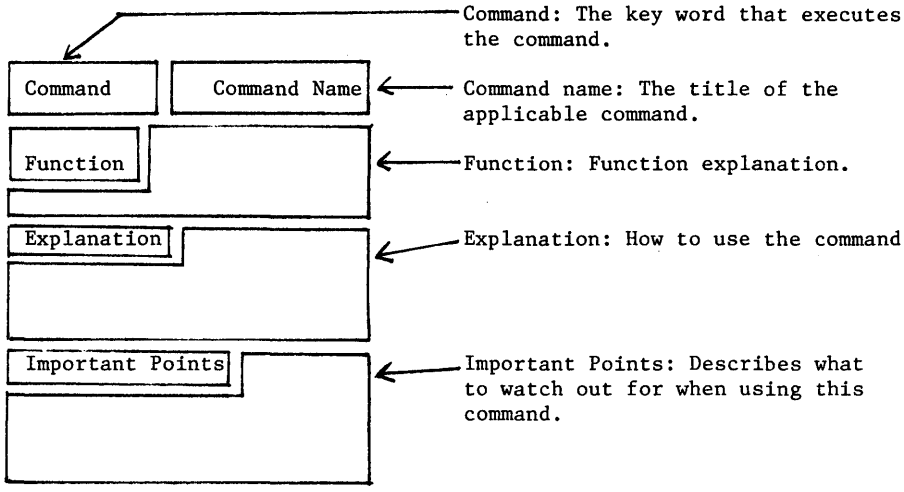


Figure 2-15 Understanding the Assembler Command Table

2.6.1 1S	Executing Pass 1 (not clearing symbol table)
Function	(1) Executes pass 1 without clearing symbol table
Explanation	(1) Check syntax. If there is an error, the error message and statement producing the error will be displayed and the symbol table will be cataloged. (2) Because pass 1 is executed with the symbol table on hold, several programs can simultaneously cross reference the same symbols. (3) After setting the source tape in the input unit, key in "1S".

2.6.2 1P	Executing Pass 1 (clearing symbol table)
Function	(1) Clears symbol table and executes pass 1.
Explanation	(1) Check syntax. If there is an error, the error message and statement causing the error will be displayed and the symbol table will be cataloged. (2) Clear symbol table. (3) Set the source tape in the input unit and key in "1P".

2.6.3 2L	Executing Pass 2 (outputting assembly listing)
Function	(1) Executes pass 2 and outputs assembly listing
Explanation	(1) After pass 1 ends, set source program in input unit again and key in "2L". (2) Assembly listing is output while the source program is being read. (3) If there is an error, an error message will be displayed at the applicable position in the list. (4) The total number of errors in pass 1 and pass 2 will be displayed at the end of the assembly listing.
Important Points	(1) The object program will be output only when option M is specified.

2.6.4 2T	Executing Pass 2 (outputting object program)
Function	(1) Executes pass 2 and outputs object program.
Explanation	(1) After pass 1 ends, set the source program in the input unit again and key in "2T". (2) The object program is output while the source program is being read. (3) Since errors are checked, error messages will be punched on paper tape, but this presents no problem since these messages will be ignored when the monitor inputs the object program. (4) When the object is output to paper tape, a leader will be output at the beginning of the object tape and a trailer at the end. After the trailer is output, the total number of errors in pass 1 and pass 2 will be output.
Important Points	(1) No output of assembly listing.

2.6.5 2P	Executing Pass 2 (simultaneous output of assembly listing and object program)
Function	(1) Executes pass 2, simultaneously outputs assembly listing and object program.
Explanation	(1) After pass 1 ends, set the source program in the input unit again and key in "2P". (2) The assembly listing and object program will be output at the same time as the source program is being read. (3) The assembly listing will be punched on to object tape depending on what the output device type is, but this will present no problem since the monitor ignores the assembly listing when the object program is input.

2.6.6 X	Return to Monitor
Function	(1) Returns control from the assembler to the monitor and places the monitor in command request status.
Explanation	(1) After the end of assembly, key in the X command and return control to the monitor.

2.7 Assembler Directives

This section explains the directives recognized by the assembler. Except for those which define data, assembler directives control processing, they do not convert directly into object codes. Table 2-6 lists assembler directives by function. The symbols used in this section and their meanings are:

{ } Select one

[ ] Optional

[ ]... Repeats the number of times chosen

< > Character string such as label or operand (<label> <operand>)

A box in each table shows whether a label or operand is required.

label	operand
*	*

In the descriptions, O, X or Δ will be where the asterisks are. The symbols mean:

O.....required

X.....cannot be used

Δ.....either may be used

Table 2-6 Assembler Directives

No.	Type	Control Instruction	Function	Section Number
1	Assembly Control	NAM	Specifies program name	2.7.6
		OPT T	Selects 6301 or 6801 Assembler	2.7.8
		OPT M	Outputs object code directly to memory	2.7.9
		OPT O	Selects object output	2.7.10
		ORG	Specifies origin	2.7.11
		END	Specifies end of program	2.7.1
2	Define Symbol	EQU	Assigns non-variable value	2.7.2
3	Define Data and Reserve Area	FCC	Obtains character constant data	2.7.4
		FCB	Obtains 1 byte constant data	2.7.3
		FDB	Obtains 2 byte constant data	2.7.5
		RMB	Obtains memory area	2.7.13
4	Control of Listing	PAGE	Changes page	2.7.12
		SPC	Outputs blank line	2.7.14
		OPT X	Outputs cross reference table	2.7.7
		OPT S	Outputs symbol table	2.7.7
		OPT G	Outputs Expanded line of FCB, FCC and FDB	2.7.7
		OPT P	Lists in page format	2.7.7
		OPT L	Outputs list	2.7.7

Assembly control
END

2.7.1 END [End of Program]

Format	END [<expression>] [<comment>]	Label	Operand
		X	Δ
Function	Specifies the end of a program		
Explanation	The END directive tells the assembler the source program has ended. The assembler thus ignores any source statement after an END statement. During the execution of each pass, the assembler reads the END statement and ends the pass. Expressions can be written in the END directive's operand field. The expression's value indicates the start address of the program and is entered into the end record of the object tape. For object formats see Figures 2-9 and 2-10. Figure 2-16 shows an example of how this command is used. The statement of the END directive is underlined. You must write an END directive at the end of a source program.		

```
00001                          NAM     END

00003 1000                     ORG     $1000
00004 1000 96 00    START      LDA A   0
00005 1002 97 01               STA A   1
00006 1004 7E F000             JMP     $F000

00008                          END     START     PROGRAM BEGINS AT START

TOTAL ERRORS 00000
```

(END statement's operand field includes an expression)

```
00001                          NAM     END2

00003 0000 96 00    START      LDA A   0
00004 0002 97 01               STA A   1
00005 0004 7E F000             JMP     $F000

00007                          END

TOTAL ERRORS 00000
```

(END statement's operand field does not include an expression)

Figure 2-16 Example of END Directive

2.7.2 EQU [Equate Symbol Value]			
Format	<label> EQU <expression> [<comment>]	Label	Operand
		0	0
Function	Defines the value of a symbol.		
Explanation	The EQU directive assigns the value of an operand field expression to a symbol in the label field. The label and expression follow the rules in 2.3.1 "Input to Assembler." It should be noted that the EQU directive does not assign a program location counter to a label, it assigns only the counter's value. Neither label nor operand field can be omitted. Labels defined by EQU directive cannot be redefined elsewhere in the program. An EQU directive's operand field cannot contain undefined symbols. Figure 2-17 gives an example of how the directive is used. The underlined sections are the statements of the EQU directive.		

```
00001                       NAM     EQU

00003 0000 0064     LABEL1 RMB     100

00005       0032    LABEL2 EQU     LABEL1+50

00007       0032    LABEL3 EQU     LABEL2

****ERROR  206
00009       03E8    LABEL4 EQU LABEL5  ERROR -- FORWARD REFERENCE

00011       03E8    LABEL5 EQU     1000


00013                       END

TOTAL ERRORS 00001
```

Figure 2-17 EQU Directive Example

Data Definition
FCB

2.7.3 FCB [Form Constant Byte]

Format

Label	Operand
Δ	O

[<label>] FCB {<expression> / <null>} {<expression> / <null>}•••,{<expression> / <null>} [<comment>]

Function	Forms a constant of one byte
Explanation	The FCB directive stores an 8-bit unsigned binary, for a given operand value, in one byte of the object program. The directive can have one or more operands. Delimit operands with commas. Two or more operands will be stored in contiguous bytes. In the operand field, you can write expressions and symbols with numerics assigned by the assembler and you can write hexadecimal, decimal, octal and binary numerics. You can assign a series of one or more null operands delimited by commas to an FCB directive. Zeroes (Ø) will be stored for the null operands. Undefined symbols are not allowed in the expressions of an FCB directive operand field. Figure 2-18 shows an example of how this directive is used. Underlined sections are FCB directive statements.

```
00001                           NAM     FCB


00003 0000 FF                   FCB     $FF
00004 0001 00           LABEL   FCB     ,$F,23,
      0002 0F
      0003 17
      0004 00
00005 0005 02                   FCB     %010,LABEL+1,*
      0006 02
      0007 07
00006 0008 05                   FCB     5
00007 0009 00                   FCB     ,,1
      000A 00
      000B 01
00008 000C 0A                   FCB     5*2

00010                           END

TOTAL ERRORS 00000
```

Figure 2-18 FCB Directive Example

Data Definition
FCC

2.7.4 FCC [Form Character Constant]

Format

Label	Operand
Δ	0

[<label>] FCC{d<ASCII character string>d / <decimal>,<ASCII character string>} [<comment>]

Note: ASCII character strings do not include 'CR'.

Function

Forms character constants

Explanation

The FCC directive converts character strings to 7 bit ASCII code. In this directive, you can use any character in the ASCII code from $20 (space) to $5F(-).

You can write in the FCC directive's operand field in either of 2 ways:

(1) <count decimal>,<ASCII character string>

Count is the number of characters to be formed. The character string starts after the operand's first comma. If the value of count is greater than the length of the character string, the code for spaces will be entered until the count value is reached. The maximum value of count is 255.

(2) d<ASCII character string>d

A character, number or symbol may be used for d. A character string delimited by d will be converted to ASCII code. If a number is used for d, the character string cannot start with ",".

If an operand character string contains 2 or more characters, the ASCII codes corresponding to the contiguous characters will be entered into contiguous areas. Figure 2-19 is an example of how the directive is used. The underlined sections are the statements of the FCC directive.

Note: The same characters as the delimiter symbol d can not be included in a character string.

(Example) FCC AABCA

(The delimiter symbol A is contained in the character string ABC.)

```
00001                           NAM     FCC

00003 0000 54       MSG1        FCC     /TEXT/
      0001 45
      0002 58
      0003 54
00004 0004 54       MSG2        FCC     9,TEXT
      0005 45
      0006 58
      0007 54
      0008 20
      0009 20
      000A 20
      000B 20
      000C 20
00005 000D 4D                   FCC     ?MORE TEXT?
      000E 4F
      000F 52
      0010 45
      0011 20
      0012 54
      0013 45
      0014 58
      0015 54

00007                           OPT     NOGEN
00008 0016 4E                   FCC     /NOGEN/

00010                           OPT     GEN
00011 001B 47                   FCC     /GEN/
      001C 45
      001D 4E
00012                           END

TOTAL ERRORS 00000
```

Figure 2-19 FCC Directive Example

2.7.5 FDB [Form Double Byte Constant]

Format

Label	Operand
Δ	0

[<label>] FDB {<expression> / <null>} ,{<expression> / <null>}...,{<expression> / <null>} [<comment>]

Function	Forms a 2-byte constant
Explanation	The FDB directive stores a 16-bit unsigned binary that is equivalent to the operand value in 2 bytes of the object program. When there are two or more operands the binary will be stored in contiguous areas. In the operand field, you can write symbols and expressions to which the assembler has assigned numeric values, and you can write binary, octal, decimal and hexadecimal numerics. FDB directives with one or more null operands delimited by commas will store zeroes (Ø) for the null operands. You may assign labels if you wish. FDB directive cannot have undefined symbols within their operand field expressions. Figure 2-20 is an example of how this directive is used. The statements of the FDB directive are underlined.

```
00001                           NAM     FDB


00003 0000 0002                 FDB     2
00004 0002 0000     LABEL       FDB     ,$F,$FF,$FFF,$FFFF
      0004 000F
      0006 00FF
      0008 0FFF
      000A FFFF
00005 000C 000C                 FDB     LABEL+10,LABEL+5,LABEL
      000E 0007
      0010 0002
00006                           END

TOTAL ERRORS 00000
```

Figure 2-20 FDB Directive Example

Assembly Control
NAM

2.7.6 NAM [Program Name]			
Format		Label	Operand
	NAM <program name> [<comment>]	X	O
Function	Used to specify the program name.		
Explanation	The NAM directive must always be written at the beginning of a source program. No labels may be attached to the NAM directive. Write the program name in the operand field using no more than 8 alphanumeric characters. The NAM directive displays the program name on the first line of each page in the list. The program name will also be in the object program's header record. See Figures 2-9 and 2-10 for object formats. Figure 2-21 shows how this directive is used. The statements in the NAM directive are underlined.		

```
00001                          NAM     NAM
00002                          OPT     O

00004                  *PROGRAM NAME
00005 0000 96 00       NAM01   LDA A   0
00006 0002 97 01               STA A   1
00007                          END

TOTAL ERRORS 00000
```

Figure 2-21 NAM Directive Example

Listing Control
OPT

2.7.7 OPT [Output Option]			
Format		Label	Operand
	OPT <option> [,<option>]·····	X	O
Function	Selects output		
Explanation	The OPT directive is used to give the programmer optional control of assembler output. Some options are reset to the default at the end of pass 1. To cancel control of those options, specify "NO" at the header. Table 2-7 lists the options available. Note the following points. (1) N, D and R mean: N - "NO" can be assigned to the header. D - Default. Selected when the operand is omitted. R - Reset at the end of pass 1. The option will be in effect at a point you specify. (2) Characters in parentheses are abbreviated forms of the option. (3) More detailed explanations of options marked with asterisks (*) will be given on subsequent pages. Figure 2-22 is an example of how to use XREF and SYMBOL options. The statements underlined are for that particular option.		

Table 2-7 Options

Option	Segment	Meaning
6301 *T={ } 6801	____	Selects 6301 or 6801 assembler. The default is T = 6301.
GENERATE (G)	N, D, R	Outputs expanded lines for FCC, FCB and FDB directives. When you specify "NO", expanded line will not be output.
LIST (L)	N, D, R	The option will begin to output the list at a specified point in time. If "NO" is specified, there will be no output after that point.
*MEMORY (M)	N	Outputs the object program directly to memory.
*OTAPE (O)	N, D	Outputs the object program tape.
PAGE (P)	N, D	Outputs the list in page format.
SYMBOL (S)	N	Outputs the symbol table list.
XREF (X)	N	Outputs the cross reference list.

```
PAGE  001   MOVE      6301 ASSEMBLER 1.0


00001                          NAM     MOVE
00002                          OPT     SYMBOL
00003                          OPT     XREF
00004 0400                     ORG     $400
00005 0400 0002     MOVBEG     RMB     2
00006 0402 0002     MOVDST     RMB     2
00007 0404 0002     SAVEX      RMB     2


00009      0406     MOVE       EQU     *
00010 0406 EC 00               LDD     0,X       * SET PARAMETER
00011 0408 FD 0400             STD     MOVBEG    *
00012 040B EC 02               LDD     2,X       *
00013 040D FD 0402             STD     MOVDST    *
00014 0410 A6 04               LDA A   4,X
00015                     *
00016 0412 FE 0400             LDX     MOVBEG
00017 0415 E6 00     MOV010    LDA B   0,X
00018 0417 FF 0404             STX     SAVEX
00019 041A FE 0402             LDX     MOVDST
00020 041D E7 00               STA B   0,X
00021 041F 08                  INX
00022 0420 FF 0402             STX     MOVDST
00023 0423 FE 0404             LDX     SAVEX
00024 0426 08                  INX
00025 0427 4A                  DEC A
00026 0428 26 EB               BNE     MOV010
00027 042A 39                  RTS
00028                          END

---

PAGE  002   MOVE      6301 ASSEMBLER 1.0


MOVBEG 0400  MOVDST 0402  MOVE   0406  MOV010 0415  SAVEX  0404


MOVBEG 0400  00005* 00011  00016
MOVDST 0402  00006* 00013  00019  00022
MOVE   0406  00009*
MOV010 0415  00017* 00026
SAVEX  0404  00007* 00018  00023

TOTAL ERRORS 00000
```

Figure 2-22 Example of XREF and SYMBOL Options Use

Assembly Control
OPT T

2.7.8 OPT T [Specify Assembly Language Type]

Format		Label	Operand
	T = {6801 / 6301}	X	O

Function

Converts the source program into a specified machine language.

Explanation

The OPT directive's T option directs the assembler to convert a source program into the machine language specified.

(1) When OPT T = 6801,
The source program will be converted into HD6801 machine language.

(2) When OPT T = 6301,
The source program will be converted into HD6301 machine language.

T = 6301 is the default so if you do not specify the T option, the source program will be converted into HD6301 machine language.

The T option must be placed prior to the statements of the executive instruction. If the T option is after those statements, it will be ignored and assembly will be performed by the default (6301).

The T option cannot be used more than once in one program. If it is, options specified afterwards will be cancelled.

Figure 2-23 is an example of how this option is used. The statements in the T option are underlined.

```
00001                          NAM     OPTTST
00002 1000                     ORG     $1000
00003                          OPT     X,T=6801

00005 1000 B6 1008             LDA A   WORK
00006 1003 97 10               STA A   $10
****ERROR  207
00007 1005 00 0010   AIM #$7F,$10

00009 1008 0001      WORK      RMB     1          WORK AREA
00010                          END
```

Figure 2-23 Example of T Option Use

Assembly Control
OPT M

2.7.9 OPT M [Output Object to Memory]				
		Label	Operand	
Format	OPT{MEMORY / M}	X	O	
Function	Outputs the object directly to memory.			
Explanation	The OPT directive's M option directs the assembler to output the object code directly to memory. If a RAM is not packaged in the respective area, error number 218 will be displayed and there will be no output to memory. Figure 2-24 gives an example of how this option is used. The statement specifying the M option is underlined.			

```
00001                              NAM     OPTMEM
00002 F800                         ORG     $F800
00003                              OPT     M

00005 F800 96 00    START  LDA A   0
00006 F802 97 01           STA A   1
00007 F804 7E F000         JMP     $F000

00009                              END     START

TOTAL ERRORS 00000
```

Figure 2-24 Example of MEMORY Option Use

2.7.10 OPT O [Select Object Output]

		Label	Operand
Format	OPT{$^{OTAPE}_{O}$}	X	O
Function	Selects output of an object program.		
Explanation	The OPT directive's O option is used in outputting object programs. The O option is a system default, so that if omitted, object program output will be selected. If you do not want the object program output, specify the NOO option to inhibit. The O option can be used only once, and you cannot use O and NOO together in the same program. Figure 2-25 shows an example of how this option is used. The statement specifying the O option is underlined.		

```
00001                          NAM     OPTOBJ
00002 1000                     ORG     $1000
00003                          OPT     O

00005 1000 96 00    START      LDA A   0
00006 1002 97 01               STA A   1
00007 1004 7E F000             JMP     $F000

00009                          END     START

TOTAL ERRORS 00000
```

Figure 2-25 Example of OTAPE Option Use

Assembly Control
ORG

2.7.11 ORG [Origin]			
Format	ORG <expression> [<comment>]	Label	Operand
		X	O
Function	Specifies the origin.		
Explanation	The ORG directive stores the values of operand field expressions into the program counter. Statements after the ORG directive are assigned to memory locations which start with those operand values. If ORG is not specified, zeroes (Ø) will be stored in the program counter. Do not assign labels to the ORG directive. Figure 2-26 gives an example of how this directive is used. Statements in the ORG directive are underlined.		

```
00001                          NAM     ORG
00002                          OPT     O


00004 0000 0001     BILL       RMB     1
00005      0001     JOHN       EQU     *


00007 0020                     ORG     $20
00008 0020 000A                RMB     10


00010 0001                     ORG     JOHN


00012                          END

TOTAL ERRORS 00000
```

Figure 2-26 Example of ORG Directive Use

Listing Control
PAGE

2.7.12 PAGE [Top of Page]			
Format	PAGE	Label	Operand
		X	X
Function	Advance to next page.		
Explanation	The PAGE directive is used to form feed to the beginning of the next page. This directive will not be displayed on the assembly listing. You cannot use a label or operand with it. Nor will the PAGE directive translate into a machine instruction.		

Reserve Area
RMB

2.7.13 RMB [Reserve Memory Byte]			
Format	[<label>] RMB <expression> [<comment>]	Label	Operand
		Δ	0
Function	Reserves bytes of memory area.		
Explanation	The RMB directive reserves an area in memory of a size specified (in bytes) by the operand field value. As a result, the location counter increments by operand field value. You can write numeric constants (binary, octal, decimal and hexadecimal), symbols and expressions in the operand field. The assembler converts the symbols and expressions to numerics. A memory area reserved using the RMB directive cannot be changed by a second use of the directive. You cannot write symbols previously referred to or undefined symbols in the operand field expression of an RMB directive. Figure 2-27 gives an example of how the directive is used. Statements in the RMB directive are underlined.		

```
00001                       NAM     RMB

00003 0000 0001   CLAB1     RMB     1          1BYTE RESERVED FOR CLAB1
00004 0001 0002   CLAB2     RMB     2          2BYTE RESERVED FOR CLAB2

00006 0003 0003             RMB     *-CLAB1    RESULT OF EXPRES. IS ABS

00008 0006 0006             RMB     *          ERROR - RESULT OF EXPRESSION


00010                       END

TOTAL ERRORS 00000
```

Figure 2-27 Example of RMB Directive Use

Listing Control
SPC

2.7.14 SPC [Space]			
Format	SPC <expression>	Label	Operand
		X	O
Function	Outputs a blank line		
Explanation	The SPC directive instructs as many blank lines as specified by the operand to be left in the assembly listing. SPC will not be displayed in the list. In the operand field, write the number of lines to be left blank in binary, octal, decimal or hexadecimal. You may write either symbols or expressions. If page change occurs halfway through execution of the SPC directive, blank lines will be placed only up to that page. The operand field expression of this directive cannot contain undefined symbols or forward reference symbols.		

2.8 Assembler Operational Examples

Figure 2-28 is an example of assembler operation.

```
① /1P
② /2L

---

PAGE  001   MOVE    6301   ASSEMBLER  1.0


00001                       NAM     MOVE
00002                       OPT     O
00003 0400                  ORG     $400
00004 0400 0002     MOVBEG  RMB     2
00005 0402 0002     MOVDST  RMB     2
00006 0404 0002     SAVEX   RMB     2


00008       0406    MOVE    EQU     *
00009 0406 EC 00            LDD     0,X       * SET PARAMETER
00010 0408 FD 0400          STD     MOVBEG    *
00011 040B EC 02            LDD     2,X       *
00012 040D FD 0402          STD     MOVDST    *
00013 0410 A6 04            LDA A   4,X
00014                 *
00015 0412 FE 0400          LDX     MOVBEG
00016 0415 E6 00    MOV010  LDA B   0,X
00017 0417 FF 0404          STX     SAVEX
00018 041A FE 0402          LDX     MOVDST
00019 041D E7 00            STA B   0,X
00020 041F 08               INX
00021 0420 FF 0402          STX     MOVDST
00022 0423 FE 0404          LDX     SAVEX
00023 0426 08               INX
00024 0427 4A               DEC A
00025 0428 26 EB            BNE     MOV010
00026 042A 39               RTS
00027                       END

TOTAL ERRORS 00000

---

③ /2TS00B00004D4F5645202020203D
S11E0406EC00FD0400EC02FD0402A604FE0400E600FF0404FE0402E70008FF6E
S10D04210402FE0404084A26EB3925
S9030000FC

TOTAL ERRORS 00000

/
```

Figure 2-28 Assembler Operational Example

Operations in Figure 2-28:

1 : In assembler command request status, key in command "1P" to execute pass 1. The "1P" command clears the symbol table and executes pass 1.

2 : Execute pass 2. The "2L" command outputs the assembly listing.

3 : Key in "2T" and output the object program. When performing paper tape output, key in "2T" and immediately turn the output device switch on. When paper tape output is complete, turn the output device switch off.

2.9 Assembler Error Messages

Assembler error messages will be displayed in this format:

****ERROR xxx

The error message number is displayed in xxx.

Table 2-8 lists the numbers, names and meaning of the assembler error messages.

Table 2-8 Assembler Error Messages

Error No.	Error Name	Description of Error
201	NAM directive error	NAM directive not in first statement of source program. Or, more than one NAM directive in the same program.
202	Label or operation code error	Label or operation code symbols start with non-alphabetic.
203	Statement error	Source statement has only a label or blank spaces.
204*	Syntax error	Error in source statement syntax.
205	Label error	The label field does not end with a blank space. Or an invalid character was used in the label field.

206	Dual definition symbol	An attempt was made to define a symbol more than once. The value first defined is the effective value. Or, the characters A, B or X have been used by themselves in the label.
207	Undefined operation code	An item, not defined as an instruction, was used in the operation code field.
208	Branching error	The operand value of a branching instruction was not within the 1-byte range. (* + 2) - 128 < D < (* + 2) + 127 *: Address of the first byte in the branching instruction. D: Address of the branch destination.
209	Invalid addressing mode	An addressing mode that does not conform with the operation code type was used in an operand.
210	Byte overflow or reserved word reference error	An operand value is outside the range of 0 - 255. Or, reserved word A, B or X was used in the operand of an FCB directive.
211	Undefined symbol	An undefined symbol was used in an operand.
212	Assembler directive syntax error	There is a syntactical error in the assembly directive operand.
213	EQU directive syntax error	The EQU directive is not labeled, or, it has a syntax error.
214	FCB directive syntax error	Syntactical error in FCB directive statement.
215	FDB directive syntax error	Syntactical error in FDB directive statement.
216	Assembler directive operand error	Operand error in the assembler directive.

217	OPT directive error	An undefined option was used in the OPT directive statement. Or, there is an error in the "OPT T =..." position.
218	Addressing error	The object program cannot be output to memory. Or, the memory to which output was attempted is not packaged.
220	Phasing error	An instruction word used in equivalent statements in pass 1 and pass 2 has two different addresses.
221	Symbol table overflow	The symbol table has overflowed. The defined symbol will not be cataloged after this error is output. All statements which refer to that symbol will also be in error.
223	Error in assembler directive label field	A label was assigned to a directive which must not have a label.
234	Dual definition symbol reference	A twice-defined symbol has been referred to.

* If there is a syntax error in an FCC directive, the error message will be written on the subsequent line of the source statement.

2.10 Assembler Commands

Table 2-9 lists the assembler commands.

Table 2-9 Assembler Commands

No.	Command	Description of Function
1	1S	Executes pass 1 without clearing the symbol table.
2	1P	Executes pass 1 after clearing the symbol table.
3	2L	Outputs the assembly listing only.
4	2T	Outputs the object program only.
5	2P	Outputs the assembly listing and the object program.
6	X	Returns control to the monitor.

3. Text Editor

3.1 General Description of Text Editor

The 6301 text editor programming system (afterwards, simply, text editor) inputs text on paper tape, and then edits and corrects that text.

3.2 Text Editor Features

(1) Editing and correcting of long text on a small, 3Kbyte buffer.

The text editor has functions for storing part of the text on a buffer in memory, correcting that text and then outputting it. After output terminates, the text editor reads the text remaining in the input device in the same way as previous storage, corrects the text and then outputs. It continues this same processing sequence over and over again. This segmented processing allows the text editor to handle large volumes of text.

(2) Serial execution of multiple commands.

You can execute commands either one at a time or sequentially.

(3) Repeating command execution

The text editor has functions to repeatedly execute commands, and you can repeat the same processing as many times as you specify.

(4) Specification of edit location by pointer

The text editor locates the pointer between two adjacent text characters. It inserts new text or deletes characters on the left or the right of the pointer. The text editor then moves the pointer to the next location requiring correction.

(5) Specifying edit location by character string

The text editor searches for a character string in the buffer, moves the pointer to the position directly after that character string and replaces it.

(6) Correcting key-in mistakes

When text or a command is incorrectly keyed in, the mistake is corrected by deleting a character previously keyed in or a line up to the previous command request.

3.3 Text Editor Input/Output

The text editor edits or corrects text input from paper tape by commands entered from the keyboard and outputs the results to paper tape. Figure 3-1 shows the flow of text editor I/O.

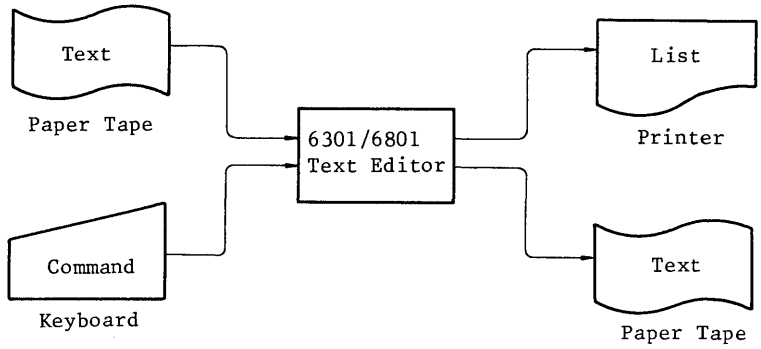


Figure 3-1 Text Editor Input/Output

3.3.1 Input to Text Editor

The text editor inputs commands keyed in at the keyboard and text punched on paper tape. For details on these commands see 3.6 "Text Editor Commands."

Text format

(1) Text is composed of characters in ASCII code.

(2) A line is a group of characters delimited by (CR) ($0D: carriage return).

(3) The final characters in text are (EOF) ($1A).

Figure 3-2 shows a text format

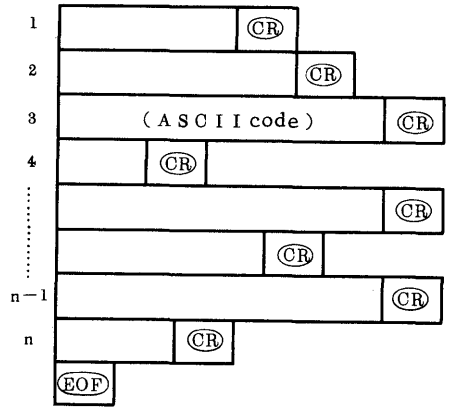


Figure 3-2 Text Format

3.3.2 Output from Text Editor

The text editor follows the directives of an entered command to output corrected text to paper tape and to output text lists.

Rules on text output

(1) Even if it has not been entered into input text, an LF will always be output prior to a CR.

(2) When text is output up to EOF, the text editor will output a feed (150 nulls) to the text.

3.4 Executing the Text Editor

If the EPROMs for the text editor are mounted on the evaluation kit, once the power supply is turned on and a key-in is made like that in Figure 3-3, the text editor enters command request status.

```
/E CR
6301 TEXT EDITOR  v.r
@
```

Figure 3-3 Text Editor Execution

Key to Figure 3-3.

①: Key in a monitor command to execute the text editor.

②: Displays the title of the text editor. V stands for the version number and r for the revision number.

③: The text editor enters command request status.

3.5 Operating the Text Editor

3.5.1 Edit Operation Flow

The text editor inputs text from paper tape. It edits and corrects text by command from the keyboard.

Figure 3-4 shows the flow of edit operations. The text editor repeatedly stores part of a continuous text in the buffer, edits buffer content and then outputs that content.

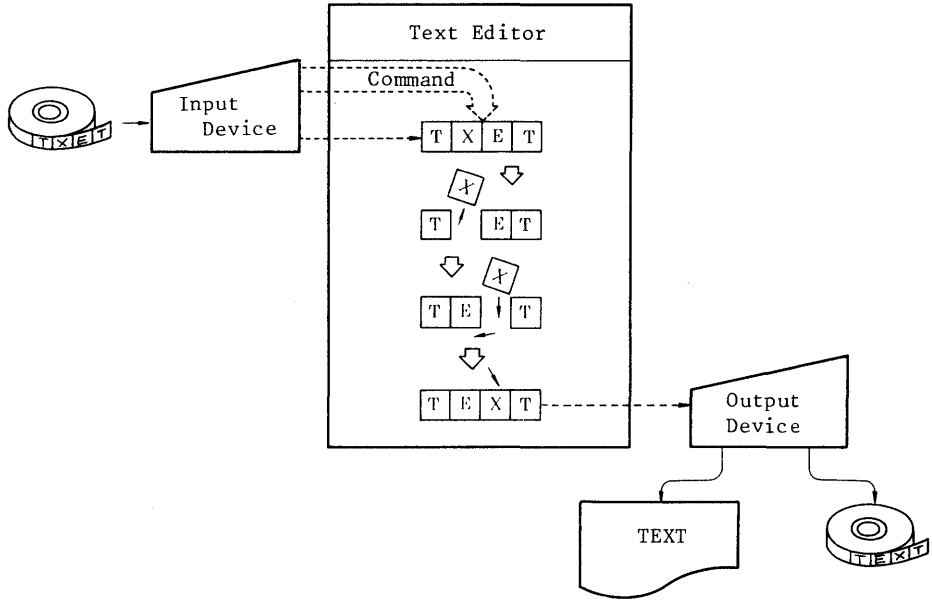


Figure 3-4 Flow of Editing Operations

3.5.2 Operational Procedures for Editing

Are:

(1) Performed by character or line unit.

(2) Performed for lines or characters located by a pointer. The pointer indicates where a character is in the buffer. As the example in Figure 3-5 shows, the pointer is considered to be between adjacent characters.

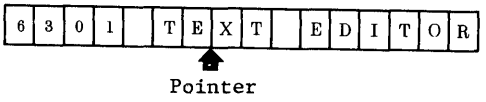


Figure 3-5 Pointer Position

(3) (LF) is not stored in the buffer, so it makes no difference whether input text includes an (LF) or not.

3.6 Text Editor Commands

This section explains the functions and methods used for each text editor command. Before that explanation some general points on command usage should be mentioned.

(1) Command request

The text editor displays a "@" at the left of the list whenever it is requesting a command. When a "@" appears, key in a command.

(2) Command execute directive

After keying in the command, key in (ESC) twice to end command input and begin command execution. (ESC) is an undisplayed character set, but when you key (ESC) in, the text editor displays "$".

Figure 3-6 is an example of command input. In this example, the pointer moves to the head of the buffer.

```
@ B (ESC)(ESC)
          ↓
      Displays "$"
```

Figure 3-6 Example of Command Input

(3) Rules on key-in

If the bell rings* during command input and the following message is displayed,

** BUFFER NEAR END

You will be able to key in only 10 more characters besides the 2 (ESC)s for directing command execution. Any more than 10 will not be accepted. Use (BS) and (CAN) (see (5)) to partially delete excess commands. Then begin execution.

At times you won't be able key in the C, I and S commands during text input. If you are unable to key in these commands and a command is then executed, you will be unable, as the next example shows, to perform the desired processing.

(Example)

```
@ IABCDEFGHIJKLMNO
** BUFFER NEAR END
P QRSTU$COTP$$
↑    ↑       ↑
|    |       └──────no keying in beyond this point
|    └After message is displayed only 12 characters can be
|     keyed in
└Message is displayed when this character is keyed in.
```

After "OTP" in the C command, an attempt was made to key in "(ESC) OPT" to replace "OPT" with "OTP". But, during C command input, the command ended and "OTP" was deleted because the buffer was full.

(4) Serial execution of commands

You can key in several different commands sequentially before issuing

---

*Differs according to console specifications.

the directive to execute a command. Figure 3-7 is an example of how this is done. In this example, text set in the input device is read and the second line from the header is deleted.

```
@ A B L K (ESC) (ESC)
            ↓
        Displays "$"
```

Figure 3-7 Example of Serial Command Execution

Section 3.7 discusses serial execution of commands in further detail.

(5) Correcting commands

If you notice any key-in errors in a command before issuing the execute directive, correct them using either method in Table 3-1.

Table 3-1 Methods of Correcting Commands

No.	Method	Name	Operation	Code	Description
1	Correction by (BS) Key	Back Space	(CTRL) + H	$08	Deletes the character just keyed in from the buffer and displays the deleted character. (Example) @ A B L L 2 L K (ESC) (ESC) ↑ (BS) In this example, commands A, B, 2L and K will each be executed.
2	Correction by (CAN) Key	Cancel	(CTRL) + X	$18	1 entire line will be deleted, a move made to the next line, and "@", requesting a new command, will display. (Example) @ A B 2 L K ↑ (CAN) @

The examples below explain each command used when text in Figure 3-8 is stored in the buffer. However, (CR) and (LF) indicate the key-in of (CR) and (LF).

```
NAM PGM(CR)(LF)
 OTP M MEMORY FILE OPTION(CR)(LF)
 OPT O OUTPUT OBJECT TAPES(CR)(LF)
 OPT S SELECT PRINTING SYMBOLS(CR)(LF)
 ORG 8192(CR)(LF)
 LDA B ADDR(CR)(LF)
COUNT EQU @8  @ INDICATES OCTAL(CR)(LF)
START LDS #STACK INZ STACK POINTER(CR)(LF)
 LDX ADDR(CR)(LF)
  LDA B  #COUNT IMMEDIATE ADDRESSING(CR)(LF)
 BACK LDA A 10 DIRECT ADDRESSING(CR)(LF)
  CMP A 2,X INDEXED ADDRESSING(CR)(LF)
  BEQ FOUND RELATIVE ADDRESSING(CR)(LF)
  DEX IMPLIED ADDRESSING(CR)(LF)
  DEC B  ACCUMULATOR ONLY ADDRESSING(CR)(LF)
  BNE BACK(CR)(LF)
  WAI  WAIT FOR INTERRUPT(CR)(LF)
  SPC 1(CR)(LF)
 FOUND JSR SUBRTN    JUMP TO SUBROUTINE(CR)(LF)
  JMP START EXTENDED ADDRESSING(CR)(LF)
 * COMMENT STATEMENT NOTE TRUNCATION 012345678901234567 89(CR)(LF)
 SUBRTN TAB COMMENT FIELD TRUNCATION0123456789(CR)(LF)
  ORA A BYTE    SET MOST SIGNIFICANT BIT(CR)(LF)
  RTS  RETURN FROM SUBROUTINE(CR)(LF)
  SPC 2(CR)(LF)
  RMB 20 SCRATCH AREA FOR STACK(CR)(LF)
 STACK RMB 1 START OF STACK(CR)(LF)
 BYTE FCB $80 FORM CONSTANT BYTE(CR)(LF)
  FCB $10,$4  $ INDICATES HEXADECIMAL(CR)(LF)
 ADDR FDB DATA FORM CONSTANT DOUBLE BYTE(CR)(LF)
 DATA FCC 'SET' FORM CONSTANT DATA STRING AS ASCII(CR)(LF)
  END(CR)(LF)
```

There are several errors in the above text. Those errors are corrected by the commands explained below.

Figure 3-8 Example of Input Text

Symbols used in each command:

Δ.....Means a space is keyed in.

_____ (underline).....sections keyed in by user.

3.6.1 A (text input)

Functions

Text is sent from an input device and stored in the buffer. Text is not displayed.

Explanation

(1) Text input terminates when any of the following conditions are satisfied. "@" then displays requesting the next command.

(a) When (EOF) ($1A: (CTRL) + Z) is input from tape.

(b) When 150 lines are read in.

(c) When the buffer is full.

(2) If any previously input text remains in the buffer, new text is read in after it.

(3) The following codes will not be input to the buffer.

(a) (NUL) (b) (DEL) (c) (LF)

(d) (ESC) (e) (BS) (f) (CAN)

(g) (EOF) (h) (DC1) (i) (DC2)

(j) (DC3) (k) (DC4)

(4) If there is space in the buffer, 150 lines or more can be input when the A command is repeated.

(5) The pointer will not move.

(6) If none of the conditions in (1) are satisfied, the input device enters input wait status even if all paper tape has been read in. At this time have an (EOF) read in.

Examples

① @A$$

@

①: Reads the content of paper tape into the buffer.

Moving the Pointer
B

3.6.2 B (moving pointer to head of buffer)	
Function	Moves the pointer to the head of the buffer.
Examples	① @T$$ OPT S SELECT PRINTING SYMBOLS ② @B$$ ③ @T$$ @ ① : Displays line indicated by pointer (in this case, line 4). ② : Moves pointer to head of buffer. ③ : Displays line indicated by pointer (1st line).

Edit Operations
Cstring1$string2

3.6.3 Cstring1$string2 (replacing a character string)	
Functions	Searches for a character string which is identical to "string1" then replaces that character string with "string2".
Explanation	(1) Execution of this command ends when either of the following occurs. (a) "string1" is found. "string1" is replaced by "string2", the pointer moves to a location after "string2" and "@" displays to request the next command.

(b) "string1" cannot be found from beginning to end of input text in the buffer.

After the message "CAN'T FIND 'string1'" is displayed, "@" is displayed requesting the next command. The pointer does not move.

(2) Key in 'ESC' once to delimit "string1" and "string2".

(3) If "string2" is omitted, "string1" will be deleted. When this happens, command input must be ended as "Cstring1 ESC ESC".

(4) "string1" and "string2" are ASCII code character strings of 16 characters or less not including the characters ESC and BREAK.

"string1" and string2" do not have to be the same length.

Examples

```
① @ B$$
② @ 5T$$
    NAM PGM
  * REVISION 1
   OTP M MEMORY FILE OPTION
   OPT  O OUTPUT OBJECT TAPES
   OPT S SELECT PRINTING SYMBOLS
③ @CSYMBOLS$OFΔSYMBOL$$
④ @COTP$OPT$$
  CAN'T FIND "OTP"
⑤ @B$$
⑥ @COTP$OPT$$
⑦ @B$$
⑧ @5T$$
    NAM PGM
  * REVISION 1
   OPT M MEMORY FILE OPTION
   OPT  O OUTPUT OBJECT TAPES
   OPT S SELECT PRINTING OF SYMBOLS

   @
```

	①: Moves the pointer to the head of the buffer. ②: Displays 5 lines after the line indicated by the pointer. ③: Searches for "SYMBOLS" and replaces it with "OFΔSYMBOLS". ④: Searches for "OTP". But, there is no "OTP" between the pointer location and the end of input text, and the message "CAN'T FIND "OTP"" is displayed. ⑤: Moves the pointer to the head of the buffer. ⑥: Searches for "OTP" and replaces it with "OPT". ⑦: Moves the pointer to the head of the buffer. ⑧: Displays 5 lines after the line (1st line) indicated by the pointer.

Edit Operations
nD

3.6.4 nD (deleting text by character units)	
Functions	From the location indicated by the pointer, deletes n characters from the buffer.
Explanation	(1) n is a decimal integer: -254≤n≤255. (2) No execution takes place if n = 0. (3) If n is negative, n characters to the left of the pointer location will be deleted. (4) If there are less than n characters from the pointer location to the head or foot of the input text, all characters between the head or foot will be deleted. (5) If n is omitted, it will be assumed as n = 1, if n is -D, it will be assumed as equivalent to n = -1.
Example	①@B$$ ②@4T$$ NAM PGM * REVISION 1 OTP M MEMORY FILE OPTION OPT O OUTPUT OBJECT TAPES

③ @STAPES$$
④ @-1D$$
⑤ @-4T$$
NAM PGM
* REVISION 1
OTP M MEMORY FILE OPTION
OPT O OUTPUT OBJECT TAPE
@

① : Moves pointer to head of buffer.
② : Display 4 lines from line indicated by the pointer (1st line).
③ : Searches for "TAPES". Moves pointer to next immediate position.
④ : Deletes character (S) to left of pointer location.
⑤ : Displays 4 lines above pointer location.

Text Input/Output
E

3.6.5 E (ending editing operations)	
Functions	Outputs entire buffer content to paper tape and continues to copy the tape remaining in the input device. When all copying ends, the trailer is output.
Explanation	(1) When all text corrections end, key in the E command. All buffer content and remaining text will be copied on paper tape. (2) If there is an EOF on the input tape, the EOF at the end of the output tape and the trailer (150 NULs) will be output and output will end. (3) Pointer does not move. (4) NULs will not be copied. (5) When the text in the buffer is output, LF will be assigned before CR; 6 NULs will then be output.

(6) The following operations differ according to whether the J command is executed.

(a) When not executed:

(i) Execute the E command. "PUNCH ON?" will display. Turn the output device switch on and key in characters other than (NUL), (DEL) or (LF). The keyed in characters will not be output, but buffer content and remaining text will be copied.

(ii) If there is an (EOF) on the tape, the copying of input tape will end and the trailer will be output. Since the tape stops when trailer output ends, turn the output device switch off and key in a character other than (NUL), (DEL) or (LF). "@" will be displayed and the system will enter command request status.

(b) When executed:

Execute the E command. Buffer content and remaining text will be output immediately. The trailer will also be output and the system will enter command request status.

Example

① @E$$
② PUNCH ON
⋮
③ @

① : Key in the E command to end edit operations.

② : Turn the output device switch on and key in any character except (NUL), (DEL) or (LF).

③ : After buffer content is output and text remaining in the input device is copied, the feed will be output. When this occurs, turn the output device switch off and key in characters other than (NUL), (DEL) or (LF). The text editor will then enter command request status.

3.6.6 F (outputting feed)

Functions Outputs 150 (NUL)s to paper tape in an output device.

Explanation

(1) Use the F command to output trailers or leaders to paper tape.

(2) Operations will differ depending on whether the J command has been executed.

(a) When not executed:

(i) Execute the F command. "PUNCH ON?" will display. Turn the output device switch on and key in characters other than (NUL), (DEL) or (LF). The keyed in characters will not be output, but 150 (NUL)s will.

(ii) When the tape stops, turn the output device switch off, and key in characters other than (NUL), (DEL), or (LF). The system will enter command request status and "@" will be displayed.

(b) When executed:

Execute the F command. The feed will be output immediately and the system will enter command request status.

Examples

① @F$$
② PUNCH ON?
③ @

① : Key in the F command to output feed (150 (NUL)s)
② : Turn the output device switch on, and key in any characters except (NUL), (DEL) or (LF).
③ : When feed output ends, turn the output device switch off and key in any characters except (NUL), (DEL) or (LF). The text editor will enter command request status.

3.6.7 Itext (inserting text)

Functions	Inserts text into the buffer in either line or character units.
Explanation	(1) Text will be inserted in the location indicated by the pointer. When insertion ends, the pointer moves to the end of the inserted text. (2) Except for the following, "text" is made up of characters in ASCII code. (a) (NUL) (b) (DEL) (c) (LF) (d) (RS) (e) (DC1) (f) (DC2) (g) (DC3) (h) (DC4) (i) (ESC) (j) (CAN)
Example	① @B$$ ② @2T$$ NAM PGM OTP M MEMORY FILE OPTION ③ @IΔ$$ ④ @L$$ ⑤ @IΔ*ΔREVISIONΔ1 (CR) $$ ⑥ @B$$ ⑦ @3T$$ NAM PGM * REVISION 1 OPT M MEMORY FILE OPTION @ ①: Moves the pointer to the head of the buffer. ②: 2 lines after line (1st line) indicated by pointer are displayed. ③: One space is inserted in the location indicated by the pointer. ④: Moves pointer to the next line.

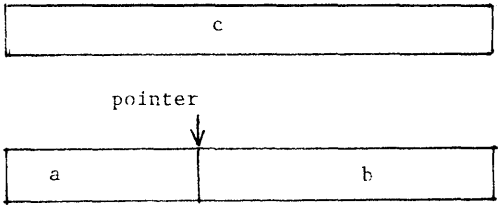
	⑤ : Insert "IΔ*ΔREVISIONΔ1 CR" into the location indicated by the pointer. ⑥ : Moves pointer to head of buffer. ⑦ : Displays 3 lines after the line (1st line) indicated by the pointer.

Editor Control
J

3.6.8 J (selecting output device)	
Function	Specifies console with packaged Automatic Device Control (ADC) function for controlling automatic punch output.
Explanation	(1) When you execute the J command there will be no temporary stop to turn the output device switch on or off or to output text to paper tape. The "PUNCH ON?" message will not be displayed. (2) At a console with a packaged ADC function, there is no need to turn the output device switch on and off for paper tape output. Just execute the J command. (3) Don't use the J command on a console that does not have the ADC function. If you do, punch errors may result. This is because you cannot easily time the on-off switching of the output device.
Example	① @J$$ ② @F$$ ③ @ ① : When you don't have to operate the output device on/off switch, key in the J command. ② : Key in the F command to output feed. ③ : "PUNCH ON?" will not be displayed, the feed will be automatically output, and the text editor will enter command request status.

3.6.9 nK (deleting text by line units)

Function	Deletes n lines of text from the buffer beginning at the position indicated by the pointer.
Explanation	(1) n is a decimal integer, $-254 \leq n \leq 255$. (2) There will be no execution when n = 0. However, if the pointer is within a line, that part of the line from the head to the pointer location will be deleted. (3) When n = 1, one line will be deleted. But, if the pointer is within a line, the line will be deleted from the pointer location to (CR). (4) When n is a negative number, n lines above the pointer location will be deleted. (5) When n = -1, one line above the pointer location will be deleted. But, if the pointer is in a line, the deletion will be from the head of the previous line to the pointer location. (6) If n is omitted, or assumed to be "+K", it will be regarded as equivalent to n = 1, and if it is assumed as "-K", it will be regarded as equivalent to n = -1.



a : deleted by "0K"
b : deleted by "1K"
c, a: deleted by "-1K"

(7) If the number of lines from the pointer location to the head or foot of the input text is less than n, only text from the pointer location to the head or foot will be deleted.

Example

```
① @B$$
② @7T$$
    NAM PGM
   * REVISION 1
    OTP M MEMORY FILE OPTION
    OPT  O OUTPUT OBJECT TAPE
    OPT S SELECT PRINTING SYMBOLS
    ORG 8192
    LDA B ADDR

③ @SLD$$
④ @OL$$
⑤ @K$$
⑥ @B$$
⑦ @7T$$
    NAM PAGM
   * REVISION 1
   OPT M MEMORY FILE OPTION
   OPT  O OUTPUT OBJECT TAPE
   OPT S SELECT PRINTING SYMBOLS
   ORG 8192
  COUNT EQU @8 @ INDICATES OCTAL
```

① : Moves pointer to head of buffer.

② : Displays 7 lines from the line indicated by the pointer (1st line).

③ : Searches for "LD". Immediately moves pointer directly behind "LD".

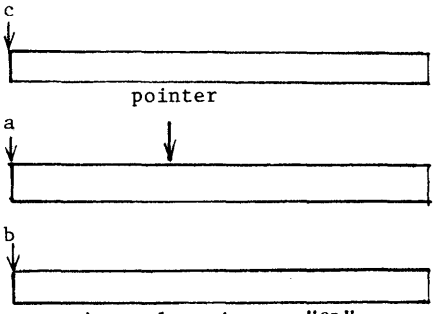
④ : Moves pointer to the head of the line.

⑤ : Deletes the line indicated by the pointer.

⑥ : Moves pointer to the head of the buffer.

⑦ : Displays 7 lines from the line indicated by the pointer (1st line).

Pointer Movement
nL

3.6.10 nL (moving pointer by line units)	
Function	Moves the pointer n lines
Explanation	(1) n is a decimal integer: -254≤n≤255. (2) When n = 0, the pointer will not even move one line. However, if the pointer is in a line, it moves to the head of that line. (3) If n = 1, the pointer moves to the next line. However, if the pointer is in a line, it moves to the head of the next line. (4) If n is a negative number, the pointer moves backward n lines. (5) If n = -1, the pointer moves backward 1 line. However, if the pointer is in a line, it moves to the head of the previous line . (6) If n is omitted, or assumed to be "+L", it will be regarded as equivalent to n = 1. If it is assumed to be "-L" it will be regarded as equivalent to n = -1.  a : pointer location at "0L" b : pointer location at "1L" c : pointer location at "-1L" (7) If the number of lines from pointer location to the head or foot of input text is less than n, the pointer moves to the head or foot.

Example	① @4T$$ NAM PGM OTP M MEMORY FILE OPTION OPT O OUTPUT OBJECT TAPES OPT S SELECT PRINTING SYMBOLS ② @3L$$ ③ @T$$ OPT S SELECT PRINTING SYMBOLS ④ @-2L$$ ⑤ @T$$ OTP M MEMORY FILE OPTION ⑥ @5M$$ ⑦ @T$$ M MEMORY FILE OPTION ⑧ @0L$$ ⑨ @T$$ OTP M MEMORY FILE OPTION @ ①: Displays 4 lines from pointer location ②: Moves pointer down 3 lines ③: Displays line indicated by pointer (4th line) ④: Moves pointer up 2 lines ⑤: Displays line indicated by pointer (2nd line) ⑥: Moves pointer 5 characters to the right ⑦: Displays 1 line from pointer location (8th and subsequent characters in the 2nd line) ⑧: Moves pointer to the head of the line. ⑨: Displays line indicated by pointer (2nd line)

Pointer Movement
nM

3.6.11 nM (moving pointer by character units)	
Functions	Moves the pointer n characters.
Explanation	(1) n is a decimal integer: $-254 \leq n \leq 255$. (2) If n = 0, the pointer will not move. (3) If n is negative, the pointer moves n characters to the

left.

(4) If n is omitted, or assumed to be "+M", it will be regarded as equivalent to n = 1. If it is assumed to be "-M", it will be regarded as equivalent to n = -1.

(5) If the characters between the pointer location and the head or foot of input text are less than n, the pointer moves to the head or the foot.

Example

```
①@4T$$
  NAM PGM
    OTP M MEMORY FILE OPTION
    OPT O OUTPUT OBJECT TAPES
    OPT S SELECT PRINTING SYMBOLS
②@3M$$
③@2T$$
    PGM
    OTP M MEMORY FILE OPTION
④@-1T$$
    NAM
⑤@5M$$
⑥@T$$
    OTP M MEMORY FILE OPTION
⑦@-1M$$
⑧@T$$
  @
```

① : Displays 4 lines from the line indicated by the pointer (in this case, the 1st line).

② : Moves the pointer 3 characters to the right.

③ : Displays from the pointer location to the next line.

④ : Displays from the head of the buffer to pointer location.

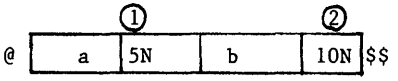
⑤ : Moves the pointer 5 characters to the right.

⑥ : Displays the line indicated by the pointer.

⑦ : Moves the pointer 1 character to the left (in front of (CR).

⑧ : Displays (CR) and (LF) only.

Editor Control
nN

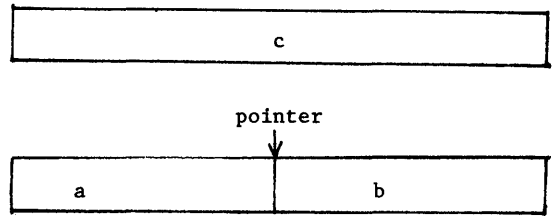
3.6.12 nN (repeat command execution)	
Functions	Repeatedly executes a command n times after command request or previous N command. Actually executes the command a total of n + 1 times.
Explanation	(1) n is a decimal integer: n≤255. (2) If n = 0, there is no repeat; the command will be executed only once. (3) Negative signs (-) are ignored, the numbers are regarded as positive. (4) If n is omitted, or assumed as "+N", it is regarded as equivalent to n = 1. (5) If the N command is keyed in more than once in the same command string, the execution will be as in the following diagram.  a : Repeated 5 times by N command in ①. b : Repeated 10 times by N command in ②. (6) If the buffer is filled during execution of the N command and the specified number of repeats cannot be performed, the message "CAN'T CONTINUE" will display and repeats will stop.
Example	① @B9LT$$ LDA B #COUNT IMMEDIATE ADDRESSING ② @CADDRESSINGADR3N$$ ③ @B9L6T$$ LDA B #COUNT IMMEDIATE ADR BACK LDA A 10'DIRECT ADR CMP A 2,X INDEXED ADR BEQ FOUND RELATIVE ADR

```
    DEX IMPLIED ADDRESSING
    DEC B  ACCUMULATOR ONLY ADDRESSING
  @
```

①: Moves pointer to the 10th line from the head of the buffer.
②: Searches for "ADDRESSING." Then executes the processing, once, to replace "ADDRESSING" with "ADR." Then executes the same process three times.
③: Moves the pointer to the 10th line from the head of the buffer and displays 6 lines.

Text Input/Output
nP

3.6.13 nP (outputting text)	
Function	Outputs n lines of buffer content to paper tape beginning at the line indicated by the pointer. Lines output are deleted from the buffer.
Explanation	(1) n is a decimal integer: $-254 \leq n \leq 255$. (2) If n = 0, there is no execution. But, if the pointer is in a line and output, the line, from the head to the pointer location, will be deleted from the buffer. (3) If n = 1, one line is output and deleted from the buffer. But if the pointer is within a line, the line, from pointer location to (CR), will be output and deleted from the buffer. (4) If n is negative, n lines above the location indicated by the pointer will be output and deleted from the buffer. (5) If n = -1, 1 line back will be output and deleted from the buffer. But if the pointer is within a line, that line, from the head of the above line to pointer location will be output and deleted from the buffer. (6) If n is omitted, or assumed to be "+P", it will be regarded as equivalent to n = 1; if it is assumed to be "-P" it will be regarded as equivalent to n = -1.



a : output by "OP"
b : output by "1P"
c, a : output by "-1P"

(7) If the number of lines from pointer location to head or foot of the input text is less than n, the lines from the pointer location to the head or foot will be output and deleted from the buffer.

(8) The following operations differ according to whether the J command is executed.

(a) When not executed:

(i) Execute the P command. "PUNCH ON?" displays. At this point, turn the output device switch on and key in any characters other than (NUL), (DEL) or (LF). The only output will be the specified number of lines, the keyed-in characters will not be displayed.

(ii) When the tape stops, turn the output device switch off. Key in any characters other than (NUL), (DEL) or (LF). The text editor will enter command request status and "@" will be displayed.

(b) When executed:

Execute the P command. The number of text lines specified will output immediately and the editor will enter command request status.

Examples

① @B5T$$

```
NAM PGM
 OTP M MEMORY FILE OPTION
 OPT  O OUTPUT OBJECT TAPES
 OPT S SELECT PRINTING SYMBOLS
```

```
   ORG 8192
② @3P$$
  PUNCH ON?
③ @B5T$$
   OPT S SELECT PRINTING SYMBOLS
   ORG 9182
   LDA B ADDR
   COUNT EQU @8 @ INDICATES OCTAL
   START LDS #STACK IN STACK POINTER
④ @2LT$$
   LDA B ADDR

⑤ @-2P$$
  PUNCH ON?
   ⋮
⑥ @B5T$$
   LDA B ADDR
  COUNT EQU @8  @ INDICATES OCTAL
  START LDS #STACK IN STACK POINTER
   LDX ADDR
    LDA B # COUNT IMMEDIATE ADDRESSING
  @
```

① : Displays 5 lines from the head of the buffer.

② : Outputs 3 lines from the location (1st line) indicated by the pointer.

③ : Displays 5 lines from the head of the buffer and confirms that they were output.

④ : Moves the pointer 2 lines down and displays line indicated by the pointer.

⑤ : Outputs 2 lines previous to the line indicated by the pointer.

⑥ : Displays 5 lines from the head of the buffer and confirms that they were output.

Pointer Movement
Sstring

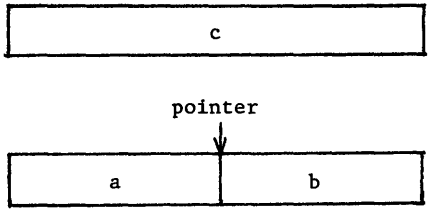
3.6.14 S string (searching for a character string)	
Function	Searches for the first character string which appears that is equal to "string".
Explanation	(1) Begins the search from the pointer location, and ends when the system enters a different status. (a) When a character string the same as "string" is found: The pointer locates itself after the final character in the string. The text editor displays "@" to request a command. (b) When the search is completed to the end of text in the buffer and "string" is not found: The message "CAN'T FIND "string"" and "@" for command request are displayed. If an equivalent string cannot be found, the pointer will not move. (2) "string" consists of not more than 16 characters in ASCII code. It does not include the words (ESC) or (BREAK).
Example	① @B$$ ② @SOPT S$$ ③ @T$$ OPT S SELECT PRINTING SYMBOLS ④ @SOTP$$ CAN'T FIND "OTP" ⑤ @B$$ ⑥ @SOTP$$ ⑦ @OL$$ ⑧ @T$$ OTP M MEMORY FILE OPTION @ ① : Moves pointer to head of buffer. ② : Searches for "OPTΔS".

	Since "@" requesting the next command is displayed, we know that "OPTΔS" was found and the pointer moved to the location behind it. ③ : Displays 1 line from location indicated by pointer. ④ : Searches for "OTP". Since "OTP" does not exist between the pointer location and the end of input text, the editor displays "CAN'T FIND "OTP"" and requests the next command. ⑤ : Moves pointer to head of buffer. ⑥ : Searches for "OTP". Since "@" requesting the next command is displayed, we know that "OTP" was found and the pointer moved to a location behind it. ⑦ : Moves pointer to head of line. ⑧ : Displays line (containing "OTP") indicated by pointer.

Text Input/Output
nT

3.6.15 nT (displaying text)	
Function	Displays n lines of buffer content from the location indicated by the pointer.
Explanation	(1) n is a decimal integer: $-254 \leq n \leq 255$. (2) If n = 0, no execution will be performed. However, if the pointer is within a line, the line will be displayed from the head to the pointer location. (3) If n = 1, one line is displayed. However, if the pointer is within a line, the line from pointer location to (CR) will be displayed. (4) If n is negative, n lines above the pointer will be displayed. (5) If n = -1, one line above will be displayed. However, if the pointer is within a line, the line from the head of the above line to the pointer location will be displayed.

(6) If n is omitted, or assumed to be "+T", it will be regarded as equivalent to n = 1; if it is assumed to be "-T" it will be regarded as equivalent to n = -1.



a : displayed by "0T"
b : displayed by "1T"
c, a : displayed by "-1T"

(7) (LF) is assigned after (CR).
(8) The pointer does not move.
(9) If the number of lines between the pointer location and the head or foot of input text is less than n, all input text up to the head or foot will be displayed.

Example

```
①@5T$$
   OPT S SELECT PRINTING SYMBOLS
   ORG 8192
   LDA B ADDR
  COUNT EQU @8 @ INDICATES OCTAL
  START LDS #STACK IN STACK POINTER
②@0T$$
③@-10T$$
  NAM PGM
   OTP M MEMORY FILE OPTION
   OPT  O OUTPUT OBJECT TAPES
④@T$$
   OPT S SELECT PRINTING SYMBOLS

  @
```

	①: Displays 5 lines from the pointer location. In this example, the pointer is at the head of the 4th line. ②: Displays no lines. ③: Since the number of lines from the pointer location to the head of the buffer is less than 10, all those lines are displayed. ④: Displays line indicated by pointer.

Editor Control
X

3.6.16 X (ending edit operations)	
Function	Moves control to monitor
Explanation	(1) If editing operations have ended or you want to cancel them, execute the X command to put the monitor in command request status.
Example	① @X$$ / ①: Key in the X command during editing, or when edit operations terminate. This cancels text editor execution and puts the monitor in command request status.

Pointer Movement
Z

3.6.17 Z (moving pointer to end of input text in buffer)	
Function	Moves pointer to end of input text.
Example	① @T$$ NAM PGM ② @Z$$ ③ @-1T$$ END @

	①: Displays lines indicated by pointer (in this case, the 1st line). ②: Moves pointer to end of input text. ③: Displays the line above the one (last line) indicated by pointer.

3.7 Serial Execution of Commands

You can execute commands serially using the text editor. To do this, key in the commands (more than one, of course) and then key (ESC) in twice.

Observe the following precautions:

(1) When a new command is entered after the character string in a C, I or S command, key in (ESC) once to separate the character string and subsequent command.

(2) The command string will be executed in sequence from left to right. If there is an error in one command string, subsequent commands will not be executed.

(3) At a certain point during serial key-in, a bell will ring and the following message will be displayed. The bell will not ring in some console I/O units.

```
@IABCDE

** BUFFER NEAR END
FGHIJKLMNOPQR
```

Will not be accepted when keyed in.

Only 12 more characters can be keyed in when this message appears.

Message appears when this character is keyed in.

When the message is displayed, you can only key in 10 characters with the exception of the 2 (ESC)s for executing the command. If you attempt to key in any more, they won't be accepted. You will have to key in (BS) or (CAN) to delete part of the commands and then execute. But, be careful when using a C, I or S command, a situation may arise where you are unable to input in the middle of text. If you execute without rectifying the situation, the desired processing in the following example cannot be performed.

(Example)

```
@IABCDEFGHIJKLMNO

** BUFFER NEAR END
PQRSTU$COTP$$

@
```

You cannot key in beyond this point.

An attempt was made to replace "OPT" with "OTP" by keying in "ESC OPT" after "OTP" in the C command. But the C command was discontinued due to a full buffer and "OTP" was deleted.

Figure 3-9 is an example of serial command execution.

```
① @B8T$$
  NAM PGM
   OTP M MEMORY FILE OPTION
   OPT  O OUTPUT OBJECT TAPES
   OPT S SELECT PRINTING SYMBOLS
   ORG 8192
   LDA B ADDR
  COUNT EQU @8  @ INDICATES OCTAL
  START LDS #STACK INZ STACK POINTER

② @2MIΔ$OLTCS$$
   NAM PGM
```

```
                    (BS)
(3) @SLD$OLKS@$DI9↓93$OLTCOTP$OPT$$
    COUNT EQU @3 @ INDICATES OCTAL

    CAN'T FIND "OTP"

(4) @BTSOTP$OPT (CAN)
(5) @BCOTP$OPT$BLIΔ*ΔREVISIONΔ1 (CR)
    $SSYMBOLS$-7MIOFΔ$$

(6) @B8TE$$
     NAM PGM
    * REVISION 1
     OPT M MEMORY FILE OPTION
     OPT  O OUTPUT OBJECT TAPE
     OPT S SELECT PRINTING OF SYMBOLS
     ORG 8192
    COUNT EQU @3  @ INDICATES OCTAL
    START LDS # STACK INZ STACK POINTER

    PUNCH ON?
```

Figure 3-9 Example of Serial Command Execution

Key to Figure 3-9:

① Displays 8 lines from the head of the buffer

② 2M.....Moves pointer 2 characters to the right.

IΔ$.....Inserts blank space into location indicated by pointer.

0L.....Moves pointer to the head of that line.

T.....Displays that line.

CS.....Deletes "S" in "TAPES."

③ SLD$.....Searches for "LD" and moves the pointer behind it.

0L.....Moves pointer to the head of the line.

K.....Deletes one line indicated by pointer.

S@$.....Searches for "@" and moves the pointer behind it.

D.....Deletes "8".

I9(BS)8$....."9" is deleted by (BS) since it was keyed in incorrectly. "3" is then inserted.

0L.....Moves pointer to head of line.

T.....Displays line indicated by pointer.

COTP$OPT....Searches for "OTP" to replace it with "OPT". But, there is no "OTP" between the pointer location and the end of input text in the buffer and "CAN'T FIND "OTP"" is displayed.

④ The command string is deleted by (CAN) and not executed.

⑤ B.....Moves pointer to head of buffer.

COTPOPT...Searches for "OTP" to replaces it with "OPT" The pointer moves directly behind the found location.

B.....Moves pointer to head of buffer.

L.....Moves pointer to head of next line.

IΔ*ΔREVISIONΔ1(CR)$

.....Inserts "Δ*ΔREVISIONΔ1 (CR) " into the location indicated by the pointer. When inserting a line in this way, put a (CR) at the end of the text.

SSYMBOLS$...Searches for "SYMBOLS" and moves the pointer to a location immediately after "SYMBOLS".

-7M.....Moves the pointer before "SYMBOLS".

IOFΔ.....Inserts "OFΔ" into location indicated by pointer.

⑥ B.....Moves pointer to head of buffer.

8T.....Displays 8 lines from the pointer location.

E.....Copies buffer content and remaining text. Ends text editing operations.

3.8 Text Editor Messages

Table 3-2 lists text editor messages.

Table 3-2 Text Editor Messages

Category	Message	Meaning
Execute Control Message	6301 TEXT EDITOR v.r	The head is displayed when control passes to the text editor. v stands for version number, r for revision number.
	@	Indicates that the system is in command request status.
Error Message	"x" ????	Incorrect characters for a command were keyed in.
	CAN'T FIND "string"	Specified character string can not be found.
	CAN'T CONTINUE	The buffer was filled during N command execution and further processing cannot be performed.
Warning Message	Bell rings ** BUFFER NEAR END	12 characters can be entered before buffer is filled.

3.9 Text Editor Commands

Table 3-3 is a complete list of commands in the text editor.

Table 3-3 Text Editor Commands

No.	Category	Command	Major Function	Section
1	Text Input/ Output	A	Inputs text from paper tape.	3.6.1
		E	Ends edit operations and creates output tape.	3.6.5
		nP	Outputs text in line units	3.6.13
		nT	Displays text in line units	3.6.15
		F	Outputs feed	3.6.6
2	Pointer Movement	B	Moves pointer to head of buffer	3.6.2
		nL	Moves pointer in line units	3.6.10
		nM	Moves pointer in character units	3.6.11
		Sstring	Searches in buffer for character string and moves pointer to location after string.	3.6.14
		Z	Moves pointer to end of text in buffer	3.6.17
3	Edit Operations	Cstring1$string2	Replaces one character string with another.	3.6.3
		nD	Deletes text in character units	3.6.4
		Itext	Inserts text	3.6.7
		nK	Deletes text in line units	3.6.9
4	Editor Control	nN	Repeats execution of a command string	3.6.12
		X	Ends text editor execution	3.6.16
		J	Selects output device	3.6.8

Appendix

A HD6301 and HD6801 Executive Instructions

Table A-1 lists the executive instructions for the HD6301 and Table A-2 lists them for the HD6801.

Explanation of symbols and abbreviations used in the appendix

(a) Operation symbols

( ) = indicates content, e.g., (ACCA): content of accumulator ACCA.

→ = send direction

∧ = logical AND

V = logical OR

⊕ = exclusive OR

+ = arithmetic addition

- = arithmetic subtraction

(b) Abbreviation symbols

OP = operation code (hexadecimal display)

~ = number of MPU cycles

= number of bytes in an instruction word

(c) Symbols for register in MPU

ACCA	= accumulator A	PC	= program counter, 16-bit
ACCB	= accumulator B	PCH	= upper 8 bits of program counter
ACCAB	= double accumulator	PCL	= lower 8 bits of program counter
CC	= condition code register	SP	= stack pointer, 16-bit
IX	= index register, 16-bit	SPH	= upper 8 bits of stack pointer
IXH	= upper 8 bits of index register	SPL	= lower 8 bits of stack pointer
IXL	= lower 8 bits of index register		

(d) Memory and address format

M = storage address

IMMED = immediate addressing

M + 1 = storage address + 1 to storage address M

MSP = storage address indicated by pointer

MSP+1 = storage address which adds 1 to storage address MSP indicated by stack pointer

MSP-1 = storage address which subtracts 1 from storage address MSP indicated by the stack pointer

(M) = complement of 1 in content of storage address M

Disp = displacement = M - (X)

IMM = immediate value

Mi = i bits in storage address M (i = 0~7)

DIRECT = direct addressing

INDEX = index addressing

EXTND = extended addressing

RELATIVE = relative addressing

IMPL = implied addressing

ACCX = accumulator addressing

(e) Meanings of bits 0 to 5 in the condition code register

C = carry and borrow

V = overflow display if 2's complement

Z = display if zero

N = display if negative

I = interrupt mask

H = carry from bit 3 to bit 4 (half carry)

(f) Symbols indicating changes in condition code register content

R = reset at any time

S = set at any time

↕ = set if true after test, anything else, clear

0 = no change by that instruction

① - ⑨ = set if true after test. Anything else, clear. Items 1 to 9 are explained below

①..result = 10000000 (binary display) ?

②..result ≠ 00000000 (binary display) ?

③..upper order 4 bit BCD (binary code decimal) display greater than 9?

④..operand = 10000000 (binary) ? (before execution)

⑤..operand = 01111111 (binary) ? (before execution)

⑥..N ⊕ C = 1 after shift?

⑦..highest order bit = 1? (after execution)

⑧..was there an overflow when subtracted (addition of 2's complementary) ?

⑨..result<Ø?

⑩ = loads from stack into condition code register

⑪ = set when interrupt occurs

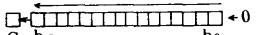
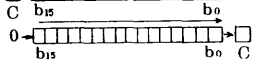
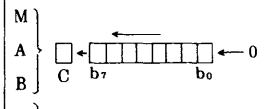
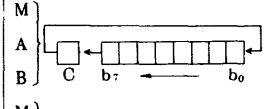
⑫ = set according to content of accumulator ACCA

Table A-1 HD6301 Executive Instructions

Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Model: IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPL, ACCX OP	IMPL, ACCX ~	IMPL, ACCX #	Condition Code 5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Operation by 16-bit Data		ADDD	Double Add without carry	(ACCAB)+(M:M+1)→ACCAB	C3	4	3	D3	5	2	E3	6	2	F3	6	3				●	●	↕	↕	↕	↕
		MUL	Multiply unsigned	(ACCA)*(ACCB)→ACCAB													3D	10	1	●	●	●	●	●	↕
		SUBD	Double Subtract without carry	(ACCAB)-(M:M+1)→ACCAB	83	4	3	93	5	2	A3	6	2	B3	6	3				●	●	↕	↕	↕	↕
		ABX	Add Acmltr B to Index Reg	(IX)+(ACCB)→IX													3A	3	1	●	●	●	●	●	●
Operation by 8-bit Register	Arithmetic Operation	ABA	Add Acmltrs	(ACCA)+(ACCB)→ACCA													1B	2	1	↕	↕	↕	↕	↕	↕
		ADCA	Add with carry	(ACCA)+(M)+(C)→ACCA	89	2	2	99	3	2	A9	4	2	B9	4	3				↕	●	↕	↕	↕	↕
		ADCB	Add with carry	(ACCB)+(M)+(C)→ACCB	C9	2	2	D9	3	2	E9	4	2	F9	4	3				↕	●	↕	↕	↕	↕
		ADDA	Add	(ACCA)+(M)→ACCA	8B	2	2	9B	3	2	AB	4	2	BB	4	3				↕	●	↕	↕	↕	↕
		ADDB	Add	(ACCB)+(M)→ACCB	CB	2	2	DB	3	2	EB	4	2	FB	4	3				↕	●	↕	↕	↕	↕
		SBA	Subtract Acmltrs	(ACCA)-(ACCB)→ACCA													10	2	1	●	●	↕	↕	↕	↕
		SBCA	Subtract with carry	(ACCA)-(M)-(C)→ACCA	82	2	2	92	3	2	A2	4	2	B2	4	3				●	●	↕	↕	↕	↕
		SBCB	Subtract with carry	(ACCB)-(M)-(C)→ACCB	C2	2	2	D2	3	2	E2	4	2	F2	4	3				●	●	↕	↕	↕	↕
		SUBA	Subtract	(ACCA)-(M)→ACCA	80	2	2	90	3	2	A0	4	2	B0	4	3				●	●	↕	↕	↕	↕
		SUBB	Subtract	(ACCB)-(M)→ACCB	C0	2	2	D0	3	2	E0	4	2	F0	4	3				●	●	↕	↕	↕	↕

Note: Execute instruction with an asterisk (*) to the right are for the HD3601 °

Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode														Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Operation by 8-bit Register	Arithmetic Operation (one operand)	CLR	Clear	00→M							6F	6	2	7F	6	3				●	●	R	S	R	R
		CLRA		00→ACCA													4F	2	1	●	●	R	S	R	R
		CLRB		00→ACCB													5F	2	1	●	●	R	S	R	R
		DAA	Decimal Adjust, A	Converts binary addition resultsto BCD													19	2	1	●	●	↕	↕	↕	③
		DEC	Decrement	(M)-1→M							6A	6	2	7A	6	3				●	●	↕	↕	④	●
		DECA		(ACCA)-1→ACCA													4A	2	1	●	●	↕	↕	④	●
		DECB		(ACCB)-1→ACCB													5A	2	1	●	●	↕	↕	④	●
		INC	Increment	(M)+1→M							6C	6	2	7C	6	3				●	●	↕	↕	⑤	●
		INCA		(ACCA)+1→ACCA													4C	2	1	●	●	↕	↕	⑤	●
		INCB		(ACCB)+1→ACCB													5C	2	1	●	●	↕	↕	⑤	●
		NEG	Complement, 2's (Negate)	00-(M)→M							60	6	2	0	6	3				●	●	↕	↕	①	②
		NEGA		00-(ACCA)→ACCA													40	2	1	●	●	↕	↕	①	②
		NEGB		00-(ACCB)→ACCB													50	2	1	●	●	↕	↕	①	②
	Comparison and Test	CBA	Compare Acmltrs	(ACCA)-(ACCB)													11	2	1	●	●	↕	↕	↕	↕
		CMPA	Compare	(ACCA)-(M)	81	2	2	91	3	2	A1	4	2	B1	4	3				●	●	↕	↕	↕	↕
		CMPB		(ACCB)-(M)	C1	2	2	D1	3	2	E1	4	2	F1	4	3				●	●	↕	↕	↕	↕
		TST	Test, Zero or Minus	(M)-00							6D	6	2	7D	6	3				●	●	↕	↕	R	R
		TSTA		(ACCA)-00													4D	2	1	●	●	↕	↕	R	R
		TSTB		(ACCB)-00													5D	2	1	●	●	↕	↕	R	R

Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode														Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
16-bit Operator	Shift and Rotate	ASLD(注)	Double Shift Left Arithmetic														05	3	1	●	●	↕	↕	⑥	↕
		(LSLD)	(Logical Double Shift Left)																						
		LSRD	Double Shift Right Logical														04	3	1	●	●	R	↕	⑥	↕
Operation by 8-bit Register	Shift and Rotate	ASL (LSL)	Shift Left Arithmetic (Logical Shift Left)								68	6	2	78	6	3				●	●	↕	↕	⑥	↕
		ASLA (LSLA)															48	2	1	●	●	↕	↕	⑥	↕
		ASLB (LSLB)															58	2	1	●	●	↕	↕	⑥	↕
		ASR	Shift Right Arithmetic								67	6	2	77	6	3				●	●	↕	↕	⑥	↕
		ASRA															47	2	1	●	●	↕	↕	⑥	↕
		ASRB															57	2	1	●	●	↕	↕	⑥	↕
		LSR	Shift Right Logical								64	6	2	74	6	3				●	●	R	↕	⑥	↕
		LSRA															44	2	1	●	●	R	↕	⑥	↕
		LSRB															54	2	1	●	●	R	↕	⑥	↕
		ROL	Rotate Left								69	6	2	79	6	3				●	●	↕	↕	⑥	↕
		ROLA															49	2	1	●	●	↕	↕	⑥	↕
		ROLB															59	2	1	●	●	↕	↕	⑥	↕
		ROR	Rotate Right								66	6	2	76	6	3				●	●	↕	↕	⑤	↕
		RORA															46	2	1	●	●	↕	↕	⑥	↕
		RORB															56	2	1	●	●	↕	↕	⑥	↕

* Mnemonics enclosed in parentheses ( ) may be used in the 6301 Assembler

Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode														Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL, ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Operation by 8-bit Register	Logic Operation	ANDA	And	(ACCA)·(M)→ACCA	84	2	2	94	3	2	A4	4	2	B4	4	3				●	●	↕	↕	R	●
		ANDB		(ACCB)·(M)→ACCB	C4	2	2	D4	3	2	E4	4	2	F4	4	3				●	●	↕	↕	R	●
		BITA	Bit Test	(ACCB)·(M)	85	2	2	95	3	2	A5	4	2	B5	4	3				●	●	↕	↕	R	●
		BITB		(ACCB)·(M)	C5	2	2	D5	3	2	E5	4	2	F5	4	3				●	●	↕	↕	R	●
		COM	Complement, 1'S	$(\overline{M})\rightarrow M$							63	6	2	73	6	3				●	●	↕	↕	R	S
		COMA		$(\overline{ACCA})\rightarrow ACCA$													43	2	1	●	●	↕	↕	R	S
		COMB		$(\overline{ACCB})\rightarrow ACCB$													53	2	1	●	●	↕	↕	R	S
		EORA	Exclusive OR	(ACCA)⊕(M)→ACCA	88	2	2	98	3	2	A8	4	2	B8	4	3				●	●	↕	↕	R	●
		EORB		(ACCB)⊕(M)→ACCB	C8	2	2	D8	3	2	E8	4	2	F8	4	3				●	●	↕	↕	R	●
		ORAA	Or, Inclusive	(ACCA)⊙(M)→ACCA	8A	2	2	9A	3	2	AA	4	2	BA	4	3				●	●	↕	↕	R	●
		ORAB		(ACCB)⊙(M)→ACCB	CA	2	2	DA	3	2	EA	4	2	FA	4	3				●	●	↕	↕	R	●
		AIM*	And Immediate	(M)·IMM→M				71	6	3	61	7	3							●	●	↕	↕	R	●
		OIM*	Or Immediate	(M)⊙IMM→M				72	6	3	62	7	3							●	●	↕	↕	R	●
		EIM*	Exclusive Or Immediate	(M)⊕IMM→M				75	6	3	65	7	3							●	●	↕	↕	R	●
		TIM*	Test Immediate	(M)·IMM				7B	6	3	6B	7	3							●	●	↕	↕	R	●
Operation in Bit Units	Logic Operation	BCLR*	Bit Clear	0 →Mi				71	6	3	61	7	3							●	●	↕	↕	R	●
		BSET*	Bit Set	1 →Mi				72	6	3	62	7	3							●	●	↕	↕	R	●
		BTGL*	Bit Toggle	Mi →Mi				75	6	3	65	7	3							●	●	↕	↕	R	●
		BTST*	Bit Test	Mi·1				7B	6	3	6B	7	3							●	●	↕	↕	R	●

Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode														Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
16-bit Operation	Load and Store	LDD	Double Load Acmltr, A.B	(M:M+1)→ACCAB	CC	3	3	DC	4	2	EC	5	2	FC	5	3				●	●	↕	↕	R	●
		STD	Double Store Acmltr, A.B	(ACCAB)→M:M+1				DD	4	2	ED	5	2	FD	5	3				●	●	↕	↕	R	●
Operation by 8-bit Register	Load and Store	LDAA	Load Acmltr.	(M)→ACCA	86	2	2	96	3	2	A6	4	2	B6	4	3				●	●	↕	↕	R	●
		LDAB		(M)→ACCB	C6	2	2	D6	3	2	E6	4	2	F6	4	3				●	●	↕	↕	R	●
		STAA	Store Acmltr.	(ACCA)→M				97	3	2	A7	4	2	B7	4	3				●	●	↕	↕	R	●
		STAB		(ACCB)→M				D7	3	2	E7	4	2	F7	4	3				●	●	↕	↕	R	●
	Send	TAB	Transfer Acmltrs.	(ACCA)→ACCB													16	2	1	●	●	↕	↕	R	●
		TBA		(ACCB)→ACCA													17	2	1	●	●	↕	↕	R	●
		PSHA	Push Data	(ACCA)→MSP (SP)-1→SP													36	3	1	●	●	●	●	●	●
		PSHB		(ACCB)→MSP (SP)-1→SP (SP)+1→SP													37	3	1	●	●	●	●	●	●
		PULA	Pull Data	(MSP)→ACCA (SP)+1→SP													32	4	1	●	●	●	●	●	●
		PULB		(MSP)→ACCB													33	4	1	●	●	●	●	●	●

Type		Mnemonic Code	Operation Content	Condition Determining Branch	Address Mode														Condition Code					
					RELATIVE			INDEX			EXTND			IMPL			IMPL, ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Branch and Jump Control	Conditional Branch	BCC (注)	Branch If Carry Clear	(C) = 0	24	3	1													●	●	●	●	●	●
		(BHS)	(Branch If Higher of Same)																					
		BCS (注)	Branch If Carry Set	(C) = 1	25	3	2													●	●	●	●	●	●
		(BLO)	(Branch If Lower)																					
		BEQ	Branch If = Zero	(Z) = 1	27	3	2													●	●	●	●	●	●
		BGE	Branch If ≧ Zero	(N) ⊕ (V) = 0	2C	3	2													●	●	●	●	●	●
		BGT	Branch If > Zero	(Z) ⊙ ((N)⊕(V)) = 0	2E	3	2													●	●	●	●	●	●
		BHI	Branch If Higher	(C) ⊙ (Z) = 0	22	3	2													●	●	●	●	●	●
		BLE	Branch If ≦ Zero	(Z) ⊙ ((N)⊕(V)) = 1	2F	3	2													●	●	●	●	●	●
		BLS	Branch If Lower or Same	(C)⊙(Z) = 1	23	3	2													●	●	●	●	●	●
		BLT	Branch If < Zero	(N)⊕(V) = 1	2D	3	2													●	●	●	●	●	●
		BMI	Branch If Minus	(N) = 1	2B	3	2													●	●	●	●	●	●
		BNE	Branch If Not Equal Zero	(Z) = 0	26	3	2													●	●	●	●	●	●
		BPL	Branch If Plus	(N) = 0	2A	3	2													●	●	●	●	●	●
		BVC	Branch If Overflow Clear	(V) = 0	28	3	2													●	●	●	●	●	●
		BVS	Branch If Overflow Set	(V) = 1	29	3	2													●	●	●	●	●	●
Unconditional Branch and jump		BRA	BRanch Always	None	20	3	2													●	●	●	●	●	●
		BRN	BRanch Never	Advances only by PC+2	21	3	2													●	●	●	●	●	●
		NOP	No Operation	Advances only by PC+1	01	2	1													●	●	●	●	●	●
		JMP	Jump	} See (Note)				6E	3	2	7E	3	3							●	●	●	●	●	●

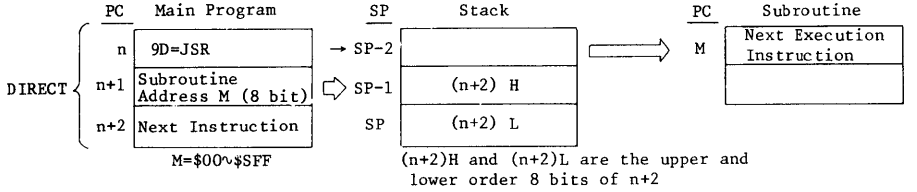
* Note: Mnemonics enclosed in parentheses ( ) may be used in the 6301 Assembler

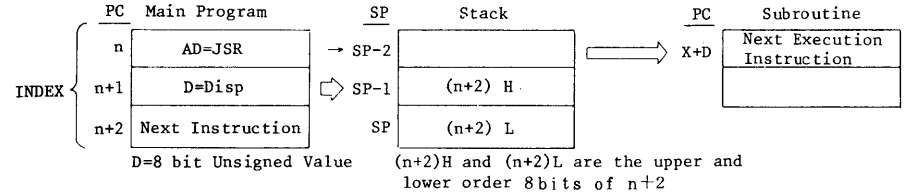
Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode													Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL, ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Index Register/Stack Pointer Control	Index Register	DEX	Decrement Index Reg	(IX)−1→IX													09	3	1	●	●	●	↕	●	●
		INX	Increment Index Reg	(IX)+1→IX													08	3	1	●	●	●	↕	●	●
		LDX	Load Index Reg	(M)→IXH, (M+1)→IXL	CE	3	3	DE	4	2	EE	5	2	FE	5	3				●	●	⑨	↕	R	●
		STX	Store Index Reg	(IXH)→M, (IXL)→M+1				DF	4	2	EF	5	2	FF	5	3				●	●	⑨	↕	R	●
		CPX	Compare Index Reg	(IXH)−(M), (IXL)−(M+1)	8C	4	3	9C	5	2	AC	6	2	BC	6	3				●	●	⑦	↕	⑧	↕
	Stack Pointer	DES	Decrement Stack Pointer	(SP)−1→SP													34	3	1	●	●	●	●	●	●
		INS	Increment Stack Pointer	(SP)+1→SP													31	3	1	●	●	●	●	●	●
		LDS	Load Stack Pointer	(M)→SPH, (M+1)→SPL	8E	3	3	9E	4	2	AE	5	2	BE	5	3				●	●	⑨	↕	R	●
		STS	Store Stack Pointer	(SPH)→M, (SPL)→M+1				9F	4	2	AF	5	2	BF	5	3				●	●	⑨	↕	R	●
	Send	TXS	Index Reg→Stack Pointer	(IX)−1→SP													35	3	1	●	●	●	●	●	●
		TSX	Stack Pointer→Index Reg	(SP)+1→IX													30	3	1	●	●	●	●	●	●
		PSHX	Push Index Reg	(IXL)→MSP; SP−1→SP													3C	4	1	●	●	●	●	●	●
		PULX	Pull Index Reg	(IXH)→MSP; SP−1→SP													38	5	1	●	●	●	●	●	●
				SP+1→SP; (MSP)→IXH																					
				SP+1→SP; (MSP)→IXL																					
		XGDX*	Exchang Aconltr D for Index Reg	(ACCAB)↔(IX)													18	2	1	●	●	●	●	●	●
Condition Code Register Control	Bit Control	CLC	Clear Carry	0→C													0C	2	1	●	●	●	●	●	R
		CLI	Clear Interrupt Mask	0→I													0E	2	1	●	R	●	●	●	●
		CLV	Clear Overflow	0→V													0A	2	1	●	●	●	●	R	●
		SEC	Set Carry	1→C													0D	2	1	●	●	●	●	●	S
		SEI	Set Interrupt Mask	1→I													0F	2	1	●	S	●	●	●	●
		SEV	Set Overflow	1→V													0B	2	1	●	●	●	●	S	●
	Byte Send	TAP	Acmltr A→CC Reg	(ACCA)→CC													06	2	1	⑫	⑫	⑫	⑫	⑫	⑫
		TPA	CC Reg→Acmltr A	(CC)→ACCA													07	2	1	●	●	●	●	●	●

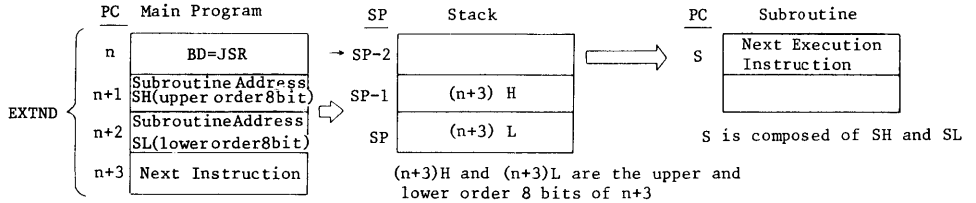
Type		Mnemonic Code	Operation Content	Conditions Determining Branch	Address Mode													Condition Code					
					RELATIVE			DIRECT			INDEX			EXTND			IMPL			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Branch and Jump Control	Subroutine Control	B S R	Branch To Subroutine	〈See Note〉	8D	6	2													●	●	●	●	●	●
		J S R	JumpTo Subroutine					9D	5	2	AD	6	2	BD	6	3				●	●	●	●	●	●
		R T S	Return From Subroutine														39	5	1	●	●	●	●	●	●
	Interrupt Control	R T I	Return From Interrupt														3B	10	1	⑩	⑩	⑩	⑩	⑩	⑩
		S W I	Soft Ware Interrupt														3F	12	1	●	S	●	●	●	●
		W A I	WAit for Interrupt														3E	9	1	●	⑪	●	●	●	●
		S L P*	Sleep	〈See Note〉													1A	41	1	●	●	●	●	●	●

(Notes)

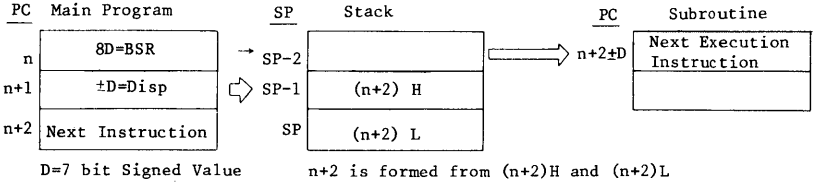
(1) JSR



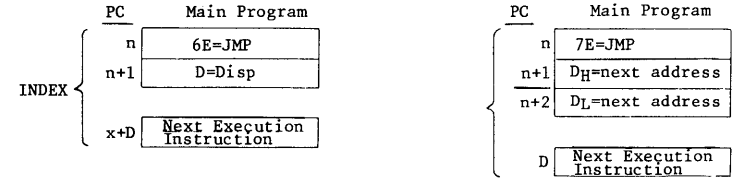




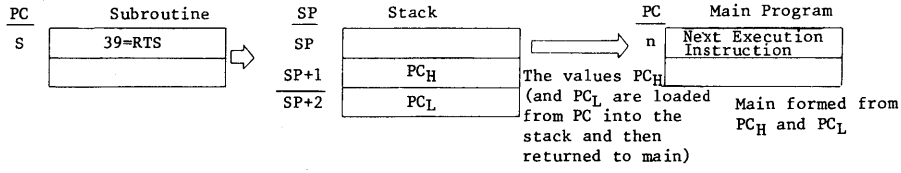
(2) BSR



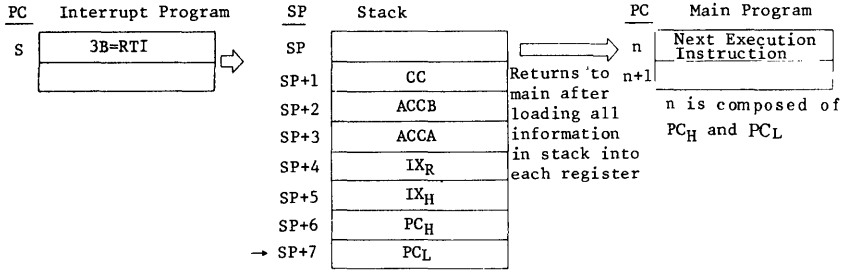
(3) JMP



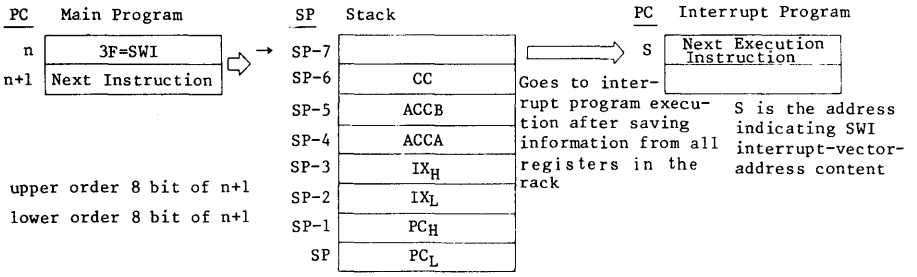
(4) RTS



(5) RTI



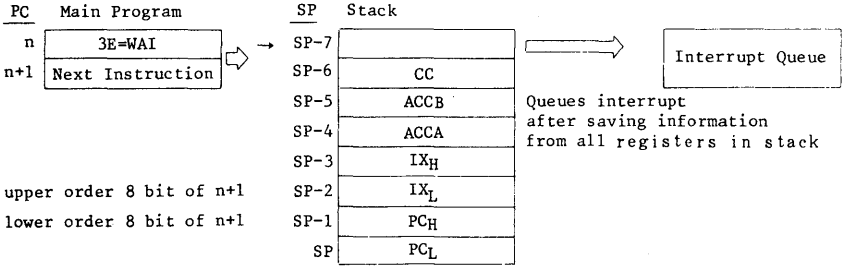
(6) SWI



PC_H: upper order 8 bit of n+1

PC_L: lower order 8 bit of n+1

(7) WAI



PC_H: upper order 8 bit of n+1

PC_L: lower order 8 bit of n+1

→ : Location of stack indicating value of SP after execution of each instruction

(8) SLP

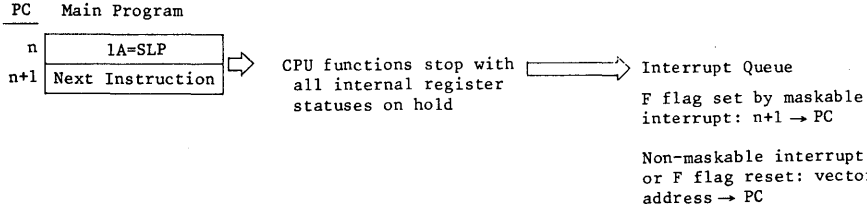


Table A-2 6801 Executive Instructions

Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode													Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL, ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Operation by 16-bit Data	Arithmetic Operation	ADDD	Double Add without carry	(ACCAB)+(M:M+1)→ACCAB	C3	4	3	D3	5	2	E3	6	2	F3	6	3				●	●	↕	↕	↕	↕
		MUL	Multiply unsigned	(ACCA)+(ACCB)→ACCAB													3D	10	1	●	●	●	●	●	↕
		SUBD	Double Subtract without carry	(ACCAB)-(M:M+1)→ACCAB	83	4	3	93	5	2	A3	6	2	B3	6	3				●	●	↕	↕	↕	↕
		ABX	Add Acmltr B to Index Reg	(IX)+(ACCB)→IX													3A	3	1	●	●	●	●	●	●
Operation by 8-bit Register	Arithmetic Operation	ABA	Add Acmltrs	(ACCA)+(ACCB)→ACCA													1B	2	1	↕	↕	↕	↕	↕	↕
		ADCA	Add with carry	(ACCA)+(M)+(C)→ACCA	89	2	2	99	3	2	A9	4	2	B9	4	3				↕	●	↕	↕	↕	↕
		ADCB		(ACCB)+(M)+(C)→ACCB	C9	2	2	D9	3	2	E9	4	2	F9	4	3				↕	●	↕	↕	↕	↕
		ADDA	Add	(ACCA)+(M)→ACCA	8B	2	2	9B	3	2	AB	4	2	BB	4	3				↕	●	↕	↕	↕	↕
		ADDB		(ACCB)+(M)→ACCB	CB	2	2	DB	3	2	EB	4	2	FB	4	3				↕	●	↕	↕	↕	↕
		SBA	Subtract Acmltrs	(ACCA)-(ACCB)→ACCA													10	2	1	●	●	↕	↕	↕	↕
		SBCA	Subtract with carry	(ACCA)-(M)-(C)→ACCA	82	2	2	92	3	2	A2	4	2	B2	4	3				●	●	↕	↕	↕	↕
		SBCB		(ACCB)-(M)-(C)→ACCB	C2	2	2	D2	3	2	E2	4	2	F2	4	3				●	●	↕	↕	↕	↕
		SUBA	Subtract	(ACCA)-(M)→ACCA	80	2	2	90	3	2	A0	4	2	B0	4	3				●	●	↕	↕	↕	↕
		SUBB		(ACCB)-(M)→ACCB	C0	2	2	D0	3	2	E0	4	2	F0	4	3				●	●	↕	↕	↕	↕

Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode														Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Operation by 8-bit Register	Arithmetic Operation (one operand)	CLR	Clear	00→M							6F	6	2	7F	6	3				●	●	R	S	R	R
		CLRA		00→ACCA													4F	2	1	●	●	R	S	R	R
		CLRB		00→ACCB													5F	2	1	●	●	R	S	R	R
		DAA	Decimal Adjust, A	Convarts binary addition results to BCD													19	2	1	●	●	↕	↕	↕	③
		DEC	Decrement	(M)-1→M							6A	6	2	7A	6	3				●	●	↕	↕	④	●
		DECA		(ACCA)-1→ACCA													4A	2	1	●	●	↕	↕	④	●
		DECB		(ACCB)-1→ACCB													5A	2	1	●	●	↕	↕	④	●
		INC	Increment	(M)+1→M							6C	6	2	7C	6	3				●	●	↕	↕	⑤	●
		INCA		(ACCA)+1→ACCA													4C	2	1	●	●	↕	↕	⑤	●
		INCB		(ACCB)+1→ACCB													5C	2	1	●	●	↕	↕	⑤	●
		NEG	Complement, 2's (Negate)	00-(M)→M							60	6	2	0	6	3				●	●	↕	↕	①	②
		NEGA		00-(ACCA)→ACCA													40	2	1	●	●	↕	↕	①	②
		NEGB		00-(ACCB)→ACCB													50	2	1	●	●	↕	↕	①	②
	Comparison and Test	CBA	Compare Acmltrs	(ACCA)-(ACCB)													11	2	1	●	●	↕	↕	↕	↕
		CMPA	Compare	(ACCA)-(M)	81	2	2	91	3	2	A1	4	2	B1	4	3				●	●	↕	↕	↕	↕
		CMPB		(ACCB)-(M)	C1	2	2	D1	3	2	E1	4	2	F1	4	3				●	●	↕	↕	↕	↕
		TST	Test, Zero or Minus	(M)-00							6D	6	2	7D	6	3				●	●	↕	↕	R	R
		TSTA		(ACCA)-00													4D	2	1	●	●	↕	↕	R	R
		TSTB		(ACCB)-00													5D	2	1	●	●	↕	↕	R	R

Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode														Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
16-bit Operator	Shift and Rotate	ASLD (注)	Double Shift Left Arithmetic	C b_{15} b_0 ← 0													05	3	1	●	●	↕	↕	⑥	↕
		(LSLD)	(Logical Double Shift Left)																					
		LSRD	Double Shift Right Logical	0 → b_{15} b_0 C													04	3	1	●	●	R	↕	⑥	↕
Operation by 8-bit Register	Shift and Rotate	ASL (注) (LSL)	Shift Left Arithmetic (Logical Shift Left)	M, A, B: C b_7 b_0 ← 0							68	6	2	78	6	3				●	●	↕	↕	⑥	↕
		ASLA (LSLA)															48	2	1	●	●	↕	↕	⑥	↕
		ASLB (LSLB)															58	2	1	●	●	↕	↕	⑥	↕
		ASR	Shift Right Arithmetic	M, A, B: b_7 b_0 C							67	6	2	77	6	3				●	●	↕	↕	⑥	↕
		ASRA															47	2	1	●	●	↕	↕	⑥	↕
		ASRB															57	2	1	●	●	↕	↕	⑥	↕
		LSR	Shift Right Logical	M, A, B: 0 → b_7 b_0 C							64	6	2	74	6	3				●	●	R	↕	⑥	↕
		LSRA															44	2	1	●	●	R	↕	⑥	↕
		LSRB															54	2	1	●	●	R	↕	⑥	↕
		ROL	Rotate Left	M, A, B: C b_7 b_0							69	6	2	79	6	3				●	●	↕	↕	⑥	↕
		ROLA															49	2	1	●	●	↕	↕	⑥	↕
		ROLB															59	2	1	●	●	↕	↕	⑥	↕
		ROR	Rotate Right	M, A, B: C b_7 b_0							66	6	2	76	6	3				●	●	↕	↕	⑤	↕
		RORA															46	2	1	●	●	↕	↕	⑥	↕
		RORB															56	2	1	●	●	↕	↕	⑥	↕

* Mnemonics enclosed in parentheses ( ) may be used in the 6801 Assembler

Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode														Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Operation by 8-bit Register	Logic Operation	ANDA	And	(ACCA)∧(M)→ACCA	84	2	2	94	3	2	A4	4	2	B4	4	3				●	●	↕	↕	R	●
		ANDB		(ACCB)∧(M)→ACCB	C4	2	2	D4	3	2	E4	4	2	F4	4	3				●	●	↕	↕	R	●
		BITA	Bit Test	(ACCB)∧(M)	85	2	2	95	3	2	A5	4	2	B5	4	3				●	●	↕	↕	R	●
		BITB		(ACCB)∧(M)	C5	2	2	D5	3	2	E5	4	2	F5	4	3				●	●	↕	↕	R	●
		COM	Complement, 1'S	$(\overline{M})\rightarrow M$							63	6	2	73	6	3				●	●	↕	↕	R	S
		COMA		$(\overline{ACCA})\rightarrow ACCA$													43	2	1	●	●	↕	↕	R	S
		COMB		$(\overline{ACCB})\rightarrow ACCB$													53	2	1	●	●	↕	↕	R	S
		EORA	Exclusive OR	(ACCA)⊕(M)→ACCA	88	2	2	98	3	2	A8	4	2	B8	4	3				●	●	↕	↕	R	●
		EORB		(ACCB)⊕(M)→ACCB	C8	2	2	D8	3	2	E8	4	2	F8	4	3				●	●	↕	↕	R	●
		ORAA	Or, Inclusive	(ACCA)∨(M)→ACCA	8A	2	2	9A	3	2	AA	4	2	BA	4	3				●	●	↕	↕	R	●
		ORAB		(ACCB)∨(M)→ACCB	CA	2	2	DA	3	2	EA	4	2	FA	4	3				●	●	↕	↕	R	●
16-bit Operation	Load and Store	LDD	Double Load Acmltr, A.B	(M:M+1)→ACCAB	CC	3	3	DC	4	2	EC	5	2	FC	5	3				●	●	↕	↕	R	●
		STD	Double Store Acmltr, A.B	(ACCAB)→M:M+1				DD	4	2	ED	5	2	FD	5	3				●	●	↕	↕	R	●

Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode													Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Operation by 8-bit Register	Load and Store	LDAA	Load Acmltr.	(M)→ACCA	86	2	2	96	3	2	A6	4	2	B6	4	3				●	●	↕	↕	R	●
		LDAB		(M)→ACCB	C6	2	2	D6	3	2	E6	4	2	F6	4	3				●	●	↕	↕	R	●
		STAA	Store Acmltr.	(ACCA)→M				97	3	2	A7	4	2	B7	4	3				●	●	↕	↕	R	●
		STAB		(ACCB)→M				D7	3	2	E7	4	2	F7	4	3				●	●	↕	↕	R	●
	Send	TAB	Transfer Acmltrs.	(ACCA)→ACCB													16	2	1	●	●	↕	↕	R	●
		TBA		(ACCB)→ACCA													17	2	1	●	●	↕	↕	R	●
		PSHA	Push Data	(ACCA)→MSP (SP)−1→SP													36	3	1	●	●	●	●	●	●
		PSHB		(ACCB)→MSP (SP)−1→SP													37	3	1	●	●	●	●	●	●
		PULA	Pull Data	(SP)+1→SP (MSP)→ACCA													32	4	1	●	●	●	●	●	●
		PULB		(SP)+1→SP (MSP)→ACCB													33	4	1	●	●	●	●	●	●

Type		Mnemonic Code	Operation Content	Conditions Determining Branch	Address Mode														Condition Code					
					RELATIVE			INDEX			EXTND			IMPL			IMPL ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Branch and Jump Control	Conditional Branch	BCC* (BHS)	Branch If Carry Clear (Branch If Higher or Same)	(C)=0	24	3	2													●	●	●	●	●	●
		BCS* (BLO)	Branch If Carry Set (Branch If Lower)	(C)=1	25	3	2													●	●	●	●	●	●
		BEQ	Branch If = Zero	(Z)=1	27	3	2													●	●	●	●	●	●
		BGE	Branch If ≧ Zero	(N) ⊕ (V) = 0	2C	3	2													●	●	●	●	●	●
		BGT	Branch If > Zero	(Z) ∨ ((N)⊕(V))=0	2E	3	2													●	●	●	●	●	●
		BHI	Branch If Higher	(C) ∨ (Z) = 0	22	3	2													●	●	●	●	●	●
		BLE	Branch If ≦ Zero	(Z) ∨ ((N)⊕(V))=1	2F	3	2													●	●	●	●	●	●
		BLS	Branch If Lower or Same	(C)∨(Z)=1	23	3	2													●	●	●	●	●	●
		BLT	Branch If < Zero	(N)⊕(V)=1	2D	3	2													●	●	●	●	●	●
		BMI	Branch If Minus	(N)=1	2B	3	2													●	●	●	●	●	●
		BNE	Branch If Not Equal Zero	(Z)=0	26	3	2													●	●	●	●	●	●
		BPL	Branch If Plus	(N)=0	2A	3	2													●	●	●	●	●	●
		BVC	Branch If Overflow Clear	(V)=0	28	3	2													●	●	●	●	●	●
		BVS	Branch If Overflow Set	(V)=1	29	3	2													●	●	●	●	●	●
Unconditional Branch and Jump		BRA	BRanch Always	None	20	3	2													●	●	●	●	●	●
		BRN	BRanch Never	Advances only by PC+2	21	3	2													●	●	●	●	●	●
		NOP	No Operation	Advances only by PC+1	01	2	1													●	●	●	●	●	●
		JMP	Jump	} See <Note>				6E	3	2	7E	3	3							●	●	●	●	●	●

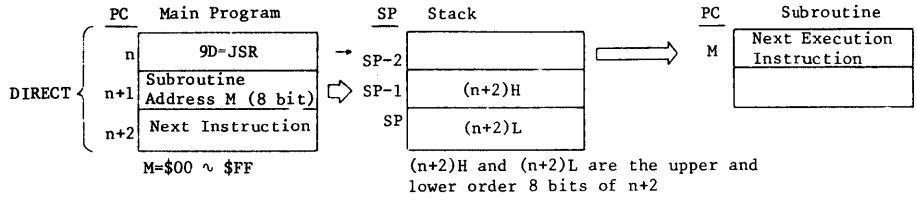
* Note: Mnemonics enclosed in parentheses ( ) may be used in the 6801 Assembler

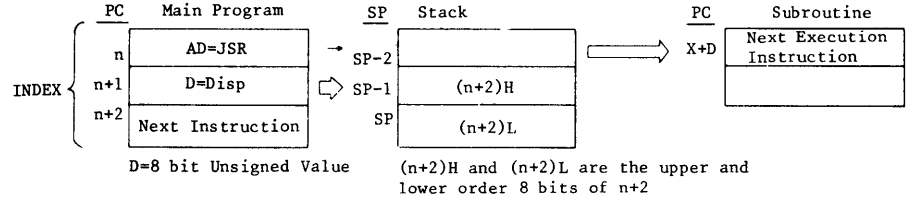
Type		Mnemonic Code	Operation Content	Logic/Arithmetic Operation	Address Mode													Condition Code					
					IMMED			DIRECT			INDEX			EXTND			IMPL, ACCX			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Index Register/Stack Pointer Control	Index Register	DEX	Decrement Index Reg	(IX)-1→IX													09	3	1	●	●	●	↕	●	●
		INX	Increment Index Reg	(IX)+1→IX													08	3	1	●	●	●	↕	●	●
		LDX	Load Index Reg	(M)→IXH, (M+1)→IXL	CE	3	3	DE	4	2	EE	5	2	FE	5	3				●	●	⑨	↕	R	●
		STX	Store Index Reg	(IXH)→M, (IXL)→M+1				DF	4	2	EF	5	2	FF	5	3				●	●	⑨	↕	R	●
		CPX	Compare Index Reg	(IXH)-(M), (IXL)-(M+1)	8C	4	3	9C	5	2	AC	6	2	BC	6	3				●	●	⑦	↕	⑧	↕
	Stack Pointer	DES	Decrement Stack Pointer	(SP)-1→SP													34	3	1	●	●	●	●	●	●
		INS	Increment Stack Pointer	(SP)+1→SP													31	3	1	●	●	●	●	●	●
		LDS	Load Stack Pointer	(M)→SPH, (M+1)→SPL	8E	3	3	9E	4	2	AE	5	2	BE	5	3				●	●	⑨	↕	R	●
		STS	Store Stack Pointer	(SPH)→M, (SPL)→M+1				9F	4	2	AF	5	2	BF	5	3				●	●	⑨	↕	R	●
	Send	TXS	Index Reg→Stack Pointer	(IX)-1→SP													35	3	1	●	●	●	●	●	●
		TSX	Stack Pointer→Index Reg	(SP)+1→IX													30	3	1	●	●	●	●	●	●
		PSHX	Push Index Reg	(IXL)→MSP; SP-1→SP													3C	4	1	●	●	●	●	●	●
		PULX	Pull Index Reg	(IXH)→MSP; SP-1→SP SP+1→SP; (MSP)→IXH SP+1→SP; (MSP)→IXL													38	5	1	●	●	●	●	●	●
Condition Code Register Control	Bit Control	CLC	Clear Carry	0→C													0C	2	1	●	●	●	●	●	R
		CLI	Clear Interrupt Mask	0→I													0E	2	1	●	R	●	●	●	●
		CLV	Clear Overflow	0→V													0A	2	1	●	●	●	●	R	●
		SEC	Set Carry	1→C													0D	2	1	●	●	●	●	●	S
		SEI	Set Interrupt Mask	1→I													0F	2	1	●	S	●	●	●	●
		SEV	Set Overflow	1→V													0B	2	1	●	●	●	●	S	●
	Byte Send	TAP	Acmltr A→CC Reg	(ACCA)→CC													06	2	1	⑫	⑫	⑫	⑫	⑫	⑫
		TPA	CC Reg→Acmltr A	(CC)→ACCA													07	2	1	●	●	●	●	●	●

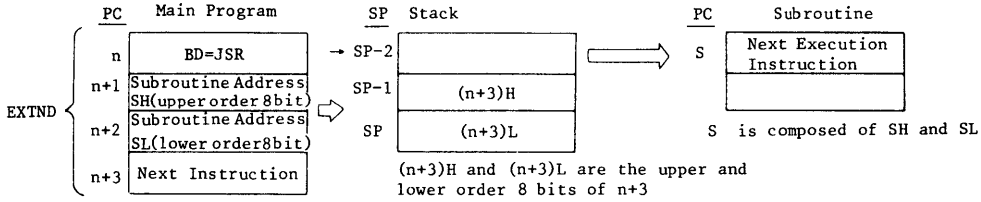
Type		Mnemonic Code	Operation Content	Conditions Determining Branch	Address Mode														Condition Code					
					RELATIVE			DIRECT			INDEX			EXTND			IMPL			5	4	3	2	1	0
					OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Branch and Jump Control	Subroutine Control	B S R	Branch To Subroutine	〈See Note〉	8D	6	2													●	●	●	●	●	●
		J S R	JumpTo Subroutine					9D	5	2	AD	6	2	BD	6	3				●	●	●	●	●	●
		R T S	Return From Subroutine														39	5	1	●	●	●	●	●	●
	Interrupt Control	R T I	Return From Interrupt														3B	10	1	⑩	⑩	⑩	⑩	⑩	⑩
		S W I	Soft Ware Interrupt														3F	12	1	●	S	●	●	●	●
		W A I	WAit for Interrupt														3E	9	1	●	⑪	●	●	●	●

(Notes)

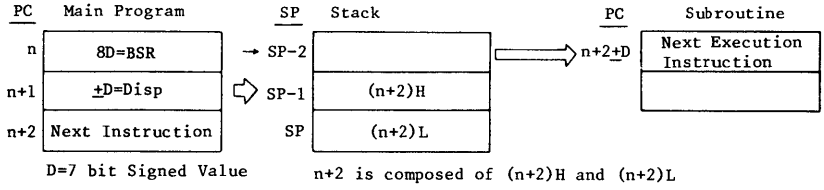
(1) JSR





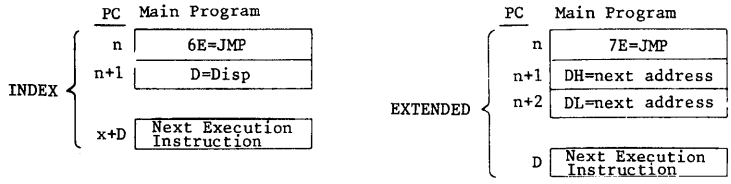


(2) BSR

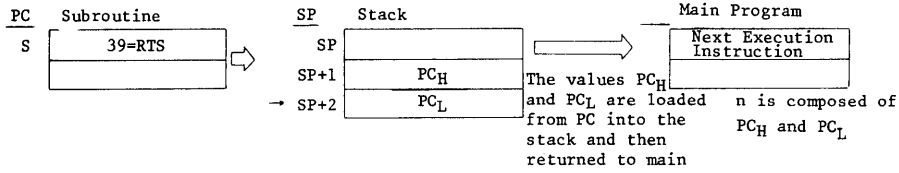


→ : Stack location indicating value of SP after each instruction is executed.

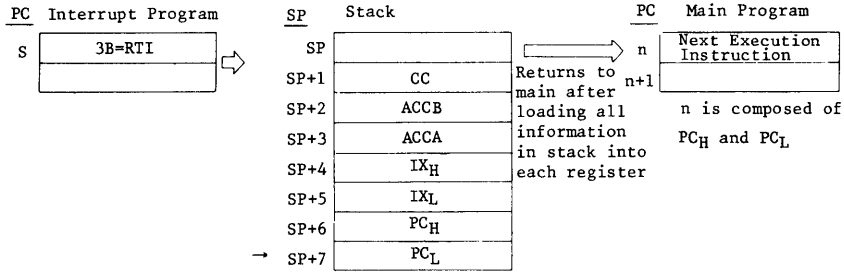
(3) JMP



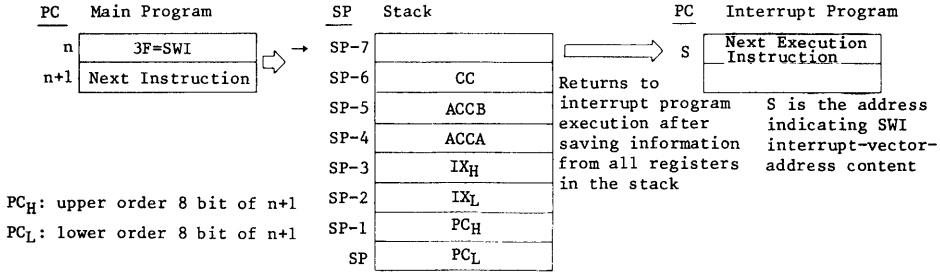
(4) RTS



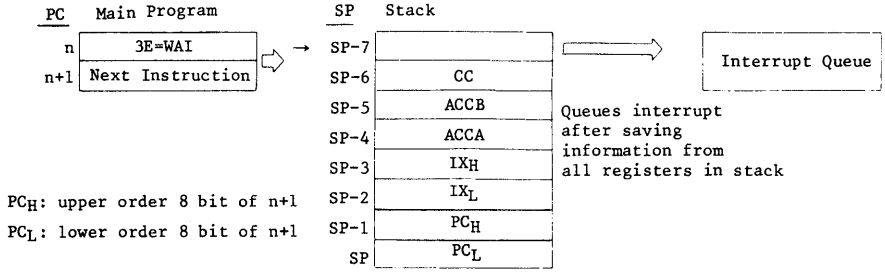
(5) RTI



(6) SWI



(7) WAI



B ASCII CODE TABLE

Table B-1 ASCII Code

Parity Bit →	b_7								
	b_6	○	○	○	○	●	●	●	●
	b_5	○	○	●	●	○	○	●	●
	b_4	○	●	○	●	○	●	○	●
b_3 b_2 b_1 b_0	MSB / LSB	0	1	2	3	4	5	6	7
○ ○ ○ ○	0	NUL	DC_0	SP	0	@	P	`	p
○ ○ ○ ●	1	SOM	X-ON	!	1	A	Q	a	q
○ ○ ● ○	2	EOA	TAPE	"	2	B	R	b	r
○ ○ ● ●	3	EOM	X-OFF	#	3	C	S	c	s
○ ● ○ ○	4	EOT	~~TAPE~~	$	4	D	T	d	t
○ ● ○ ●	5	WRU	ERROR	%	5	E	U	e	u
○ ● ● ○	6	RU	SYNC	&	6	F	V	f	v
○ ● ● ●	7	BELL	LEM	(APOS)	7	G	W	g	w
● ○ ○ ○	8	FE_0	CAN	(	8	H	X	h	x
● ○ ○ ●	9	TAB	S_1	)	9	I	Y	i	y
● ○ ● ○	A	LF	EOF	*	:	J	Z	j	z
● ○ ● ●	B	VT	ESC	+	;	K	[	k	{
● ● ○ ○	C	FF	S_4	,	<	L	\	l	|
● ● ○ ●	D	CR	S_5	-	=	M	]	m	}
● ● ● ○	E	SO	S_6	.	>	N	^	n	~
● ● ● ●	F	SI	S_7	/	?	O	_	o	RUB OUT

Characters within the double lined border may be used in comments or character constants

Hexadecimal-Decimal Conversion Table (1)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895

Hexadecimal-Decimal Conversion Table (2)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
40	1024	1026	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	0756	1757	1758
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

Hexadecimal-Decimal Conversion Table (3)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1865	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1928	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1958	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	3534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

Hexadecimal-Decimal Conversion Table (5)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

D EPROM MOUNTING METHOD

The EPROMs for the 6301/6801 Assembler-Text Editor (S31MIX1-R/S61MIX2-R) are mounted on the main module (H62EV02). Figure D-1 and Table D-1 show the proper locations.

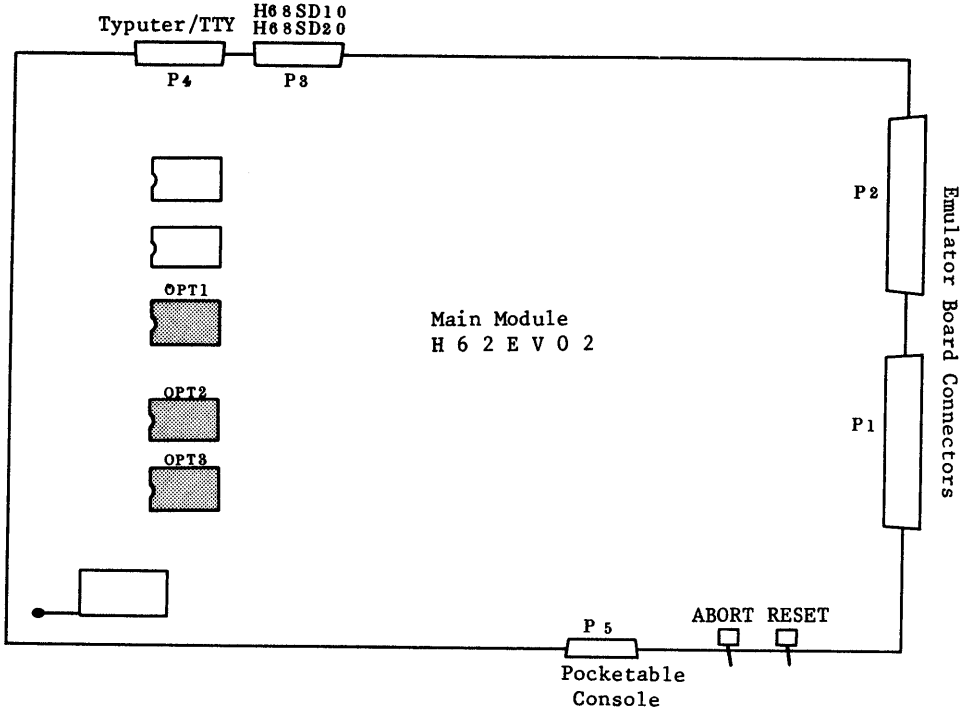


Figure D-1 EPROM Package Locations

Table D-1 EPROM and Socket Match

Location	EPROM Name
OPT 1	S31MIX1-R or S61MIX2-R
OPT 2	S31MIX1-R or S61MIX2-R
OPT 3	S31MIX1-R or S61MIX2-R

The following points are important in mounting and replacing EPROMs.

(1) Always turn the power off when removing an EPROM from the board.

(2) Mount the EPROM in the proper position as shown in Figure D-2.

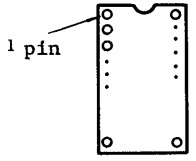


Figure D-2 EPROM Mounting Direction

If the EPROM were mounted backwards and the power applied, the EPROM will be destroyed. Always make sure that EPROMs are plugged in in the correct direction.

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 #U24

Printed in U.S.A.