POCKET COMPUTER

PC-1350

MACHINE LANGUAGE REFERENCE MANUAL

SHARP

FOREWORD

Since the release of the PC-1350 on market, we have had great number of questions from users regarding the machine language of the PC-1350.

To meet with such demand from ardent users, we are now sending this text for study of the machine language of the Sharp's original design SC61860 Microprocessor in concern with the PC-1350 system. Because the text is edited on the basis of user questions, it may not support quality as a guidebook. In such an event, you are suggested to make reference to microprocessor guidebooks published on market, in addition to this text.

Your opinions and questions are welcome through our products distributor.

NOTE: Machine language program, which controls hardware directly, gives you more various functions than BASIC programs. However, you should check your machine language program enough to make no error before executing it because single wrong key operation may upset the program or occasionally make the machine break down. Sharp Corporation assumes no liability or responsibility of any kind arising from the use of programs or program materials or any part thereof.

Contents

INTRODUCTION

For many programmers there comes a time when, regardless of the size or sophistication of the machine they program, they become dissatisfied with the exclusive use of a high-level programming language such as BASIC. Perhaps they want to make more efficient use of the available memory, they want to decrease the execution time of programs or perhaps they simply want to understand more about how the machine solves the problems presented to it. Whatever the cause, the programmer will need to learn about the assembler language or machine language of the particular machine being programmed.

This manual has been written to introduce the PC-1350 assembler and machine language, the command language for the ESR-H central processing unit.

While this manual provides much information about the PC-1350 and its resident BASIC, it was not intended to be a technical reference manual.

The material here assumes little beyond a working knowledge of BASIC and the operation of the PC-1350. Fundamental mathematical concepts, such as binary number systems, are reviewed in the context of their application to machine code programming. Likewise, fundamental machine code concepts are reviewed in the context of their application to the ESR-H language. This manual provides all the information needed to write a program in mnemonics, translate it into machine language and enter it into memory.

The transition from BASIC to machine language programming can be difficult. Machine code commands, being closer to what the machine understands, are even further from natural languages than the high-level language BASIC. In fact, many BASIC commands require more than ten or even twenty lines of machine code to accomplish similar actions. Also, space must be thought of differently at the machine code level. One must deal with fixed registers, fixed addresses, and the particular protocols for moving information from one location to another. However, the skills one developed while programming in BASIC, or which are developed programming in almost any computer language, will be invaluable in making the transition. With a bit of patience and study, you will become an able programmer for the ESR-H.

TERMS AND CONCEPTS

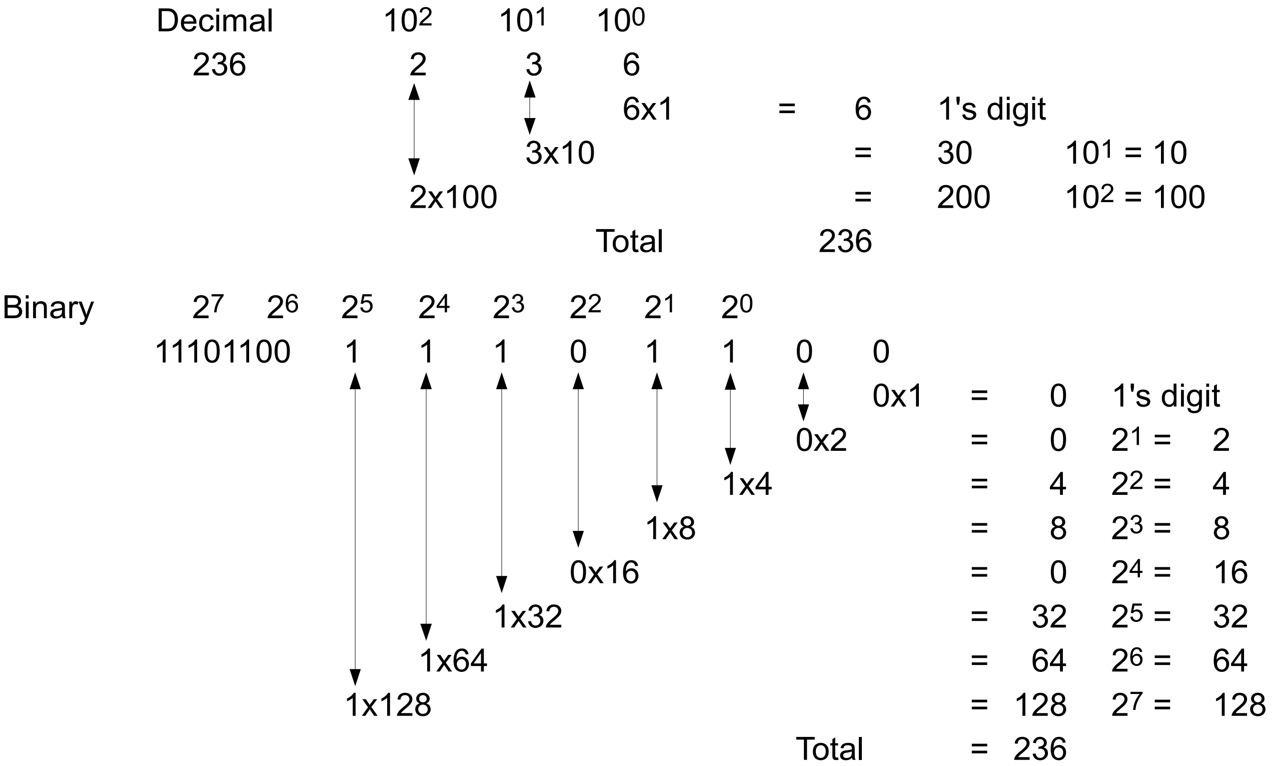
The Binary and Hexadecimal Number System

Memory in a computer consists of groups of binary digits, called "bits". A binary digit can have one of only two different values, 0 or 1. In the PC-1350, as in many other computers, 8 bits are grouped together to form one memory position, called a "byte". The left-most digit of a byte is called the "high-order digit" or "most significant bit" and the right-most is called the "low-order digit" or "least significant bit" .

Each byte of memory has a unique location, and the description of that location is called an "address". Some addresses, those in internal memory can be described with 1 byte (or sometimes even less than 1 byte) of information. Others, in external memory, require 2 bytes. Any byte of memory can contain several different kinds of information, but it is always in binary form, a series of eight 0's and 1's. The interpretation of the pattern of 0's and 1's in a particular byte is determined by the internal logic or programming of the machine or by an external program. More will be said about memory addresses and the kinds of information that can be stored in memory in a later section.

Since the only kind of numbers the computer can recognize are binary ones, any communication with the machine must be done using binary numbers. Every digit of a number in our familiar decimal system represents a power of 10. Likewise, each of the eight bits of a binary byte represents a power of 2.

The following illustration shows a decimal and a binary number having the same value 236, and what each digit of the two numbers represents.



To convert a decimal number to binary, the following method of successive divisions by 2 can be used.

236	2	=	118	Remainder	0 → 0	(lowest bit)
118	2	=	59	Remainder	0 → 0	
59	2	=	29	Remainder	1 → 1	
29	2	=	14	Remainder	1 → 1	
14	2	=	7	Remainder	0 → 0	
7	2	=	3	Remainder	1 → 1	
3	2	=	1	Remainder	1 → 1	
1	2	=	0	Remainder	1 → 1	(highest bit)

The binary equivalent is 11101100.

Binary representation of numbers, with its series of zeros and ones, can be very confusing to humans. Because of this, various alternate ways of representing binary numbers are often used. One of these alternate notations, hexadecimal, is used in programming the PC-1350.

To convert an 8 bit binary number into hexadecimal, the 8 bits are first divided into 2 groups of four bits, then each group of 4 bits is assigned a single digit value. The result of this is a 2 digit number which has the same value as the 8 digit binary number. In order to represent each of all the possible values (0-15) of a 4 digit binary number with single digit, we need 16 distinct characters, one for each of the 16 values.

0000	=	0	1010	=	10	Decimal representation requires 2 digits for these values
0001	=	1	1011	=	11	
0010	=	2	1100	=	12	
0011	=	3	1101	=	12	
0100	=	4	1110	=	14	
0101	=	5	1111	=	15	
0110	=	6				
0111	=	7				
1000	=	8				
1001	=	9				

As can be seen in the table above, the decimal digits 0-9 are not sufficient to represent all of the binary combinations of 4 digits; another 6 characters are needed. Any characters could be used, but the standard for hexadecimal in computers is to use the alphabetic characters A-F. 16 is the "base" of the hexadecimal system, just as is 10 (with 10 distinct digits) for the decimal system and 2 (with 2 distinct digits) for the binary system.

The 16 digits of the hexadecimal system and their binary and decimal equivalents are:

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

It is important to remember that all of the numbers, in spite of their different appearance, in a single row across the 3 columns have the same actual value. 1110 = E = 14, but when one is working with these numbers E is much easier to keep track of than is 1110, especially when it is surrounded by other similar numbers. 23F016 is considerably less confusing to the human eye and brain than is 001000111111000000010110, which is the only form of the numbers that makes sense to the computer.

Binary Arithmetic

The rules for binary arithmetic are similar to those of decimal arithmetic. Addition can be summarized as follows:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10

Here 10, the binary equivalent of decimal 2, can be thought of as a 0 and a "carry".

If for example, we add 3 and 1 in binary,

$$\begin{array}{rcrcrcrl} & & 1 & & 1 & & 1 & \\ 11 & & 11 & & 11 & & 11 & \\ \underline{+\,01} & \longrightarrow & \underline{01} & \longrightarrow & \underline{01} & \longrightarrow & \underline{01} & \\ & & 0 & & 00 & & 100_2 & = 4_{10} \end{array}$$

we first add the one's place column. The total is 10_2, so we put a 0 in the sum's one's place and carry 1 into the two's place (the second column). The second column is then added, with again a result of 10, so a 0 is put in the two's place column and a 1 is carried to the four's place column, giving us the result of 100 (base 2) or 4 (base 10).

With eight bits, it is possible to represent numbers from 00000000 to 11111111, or in decimal, 0 to 255 (= 2^8 -1). With two bytes, of 16 bits, we can represent numbers from 0 to 65535 (= 2^{16} - 1). In order to represent negative numbers, we treat the high-order bit as a "sign bit". With single byte numbers, since this bit cannot now be used as a part of the numeric representation, the range of the number becomes - 127 to + 127. With two byte numbers, the range becomes - 32767 to + 32767.

Binary - 3 = 1000011, + 3 = 00000011

Binary - 03 = 100000000000011, + 3 = 0000000000000011

One of the most commonly used forms of representation for negative binary numbers is what is known as "Two's Complement Representation". This representation allows us to add a negative number, i.e. subtract, using the addition command. A "one's complement" of a binary number is formed by reversing all of its digits. For example, the number-5, in one's complement form would be:

```
00000101  (5)
11111010  (- 5)  One's Complement
```

By adding 1 to a one's complement representation of a negative number, we get the "two's complement" form of the number.

```
11111010             One's complement form
      +1
--------
11111011  =          Two's complement form of - 5
```

If we use the two's complement representation for negative numbers, we can use the same simple addition rules for subtraction and for addition of negative numbers. Take, for example, the subtraction 7-5. First, the five is put in two's complement form, then it is "added" to 7.

```
   00000111   =    7
   11111011   =   -5         (Two's complement form)
(1)00000010   =    2         (plus a binary carry)
```

If we ignore the carry, the answer is 2.

One consideration with this form of representation is that the result of an addition of 2 single byte numbers may require more than 7 bits. This condition is called "overflow" , since the extra bit required to represent the results "overflows" into the high order sign bit. An overflow beyond the entire 8 bits of a byte is called a "carry". The extra bit of a carry is lost, but the occurrence of a carry causes the Carry Status Flag to be set to 1 to alert the programmer to the condition. An overflow into the high order sign bit will produce a false sign in the result of a binary addition under two conditions.

1. If both are positive and one or both have a large value.

sign bit		
(0) 1111111	+ 127	(Largest positive number which can be represented in 7 bits)
+ (0)0000010	+ 2	
(1)0000001	- 127	(False negative, interpreted as a 2's complement because of the 1 in the sign bit)

The result has a false negative sign. Any combination which would have a result of more than + 127 (for a single byte number) would cause this error condition.

2. If both are negative and one or both have a large value.

sign bit		
(1)0000001	- 127	(Largest negative number which can be represented in 7 bits in 2's complement notation)
+ (1)1111100	- 2	(in 2' s complement notation)
(1)(0)1111101	+ 125	(False positive, not interpreted as a 2's complement because of the 0 in the sign bit)

Carry is lost,
Carry Flag set

The result has a false positive sign. Any combination which would have a result of more than -127 (for a single byte number) would cause this error condition.

The programmer must check for these two error conditions by testing the Carry Flag and the sign bits themselves when they suspect that the result of an operation might cause an overflow error.

Logical and Bit Shift Operations

In addition to binary addition and subtraction. there are several binary logical operations and bit shifts which should be understood by the programmer.

Logical OR—The logical OR operation compares bit by bit all 8 bits of 2 individual bytes and produces a result based on the following conditions:

If both bits are 0, result = 0
If either bit is 1, result = 1

All of the possible combinations and results are:

Byte A 1 Bit	Byte B 1 Bit	Result
0	0	0
0	1	1
1	0	1
1	1	1

This operation can be used to place a 1 bit in selected location(s) of a byte. If we want to add a negative sign bit to a positive number. for example, to change 5 to 7, we can do the following:

A	00000101	5
OR with B	00000010	2
Result	00000111	7

Only the 2's position bit has been changed.

Logical AND—The logical AND operation compares each of the 8 bits of two bytes and produces a result based on the following conditions:

If both bits are 1, result = 1
If either bit is 0, result = 0

The possible combinations and results are:

Byte A 1 Bit	Byte B 1 Bit	Result
0	0	0
0	1	0
1	0	0
1	1	1

This operation can be used to remove or test for a 1 bit in selected location(s) of a byte. If we want to change the 7 we produced in the OR example back into a 5, we could do the following:

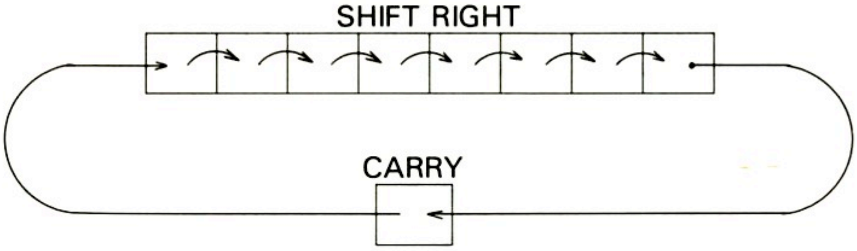
A	00000111	7
AND with B	11111101	253
Result	00000101	5

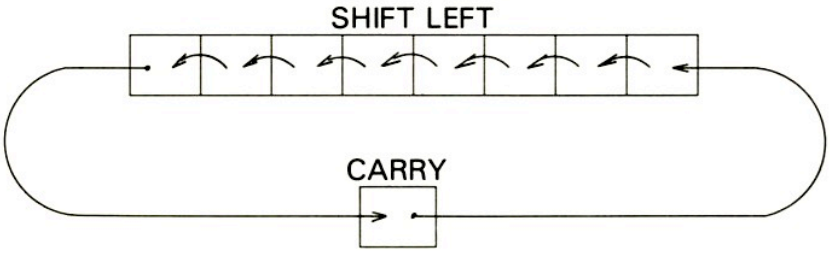
Again, only the 2's position bit has been changed.

Bit Shift Operations—Two instructions that shift the bits of a single byte to the right or left are provided in the PC-1350 instruction set.

1. Shift Right—Each bit of a byte is shifted one bit position to the right. The Least Significant Bit, which is pushed out of the byte, is stored in the Carry Flag Position and the previous contents of the Carry Flag is stored in the Most Significant Bit of the byte. This operation gives a result that is the same as dividing by two, and is useful for division routines.

2. Shift Left—Each bit of a byte is shifted 1 bit position to the left. The Most Significant Bit, which is pushed out of the byte is stored in the Carry Flag Position and the previous contents of the Carry Flag is stored in the Least Significant Bit of the byte. This operation gives a result that is the same as multiplying by two and is useful for multiplication routines.





Binary Coded Decimal

Another type of representation of numbers that provides greater accuracy for such applications as accounting, where more precision is necessary, is called BCD or Binary Coded Decimal. The decimal numbers 0-9 can be represented in binary in four bits, one half byte (called a "nibble"). Since only a half byte is needed, two decimal numbers can be coded into each byte. This representation of decimal numbers is called "Packed BCD". Some of the binary values that can be expressed in 4 bits, that is, binary 10-15, are not needed to express the decimal digits 0-9. These unneeded values are not used in BCD and can cause some problems in BCD arithmetic. However, the BCD instructions in the PC-1350 instruction set automatically make the necessary adjustments so the programmer need not worry about them. The BCD values 0-9 are shown in the chart below:

BIN	DEC	BCD	BIN	DEC	BCD	BIN	DEC	BCD
0000 =	0	0	0101 =	5	5	1010 =	10	Not Used
0001 =	1	1	0110 =	6	6	1011 =	11	
0010 =	2	2	0111 =	7	7	1100 =	12	
0011 =	3	3	1000 =	8	8	1101 =	13	
0100 =	4	4	1001 =	9	9	1110 =	14	
						1111 =	15	

A number expressed in BCD must be limited to a fixed number of digits, in the PC-1350 it is 10 digits. In order to represent numbers that are larger than the largest number, or in the case of fractions, smaller than the smallest number that can be expressed in 10 digits, a representation called Floating Point is used. Essentially, what this format allows is the elimination of the need to represent zeros on either side of the decimal point and subsequently the elimination of the bytes needed to hold these zeros.

Equivalent numbers can be represented by shifting the location of the decimal point and multiplying them by 10 to the appropriate power. Thus the decimal number 23,000.00 could be represented as:

	2300.00	x 10^1
or	230.00	x 10^2
or	23.00	x 10^3
or	2.30	x 10^4

Numbers to the right of the decimal point are represented by exponents with a minus sign. The number .00023 could be represented as:

$.0023 \times 10^{-1}$
or $.023 \times 10^{-2}$
or $.23 \times 10^{-3}$
or 2.3×10^{-4}

All of these combinations are possible, but in the PC-1350 the number is represented with the decimal point to the right of the left-most digit:

2.3×10^{4}

$2.3 \text{ X } 10^{-4}$

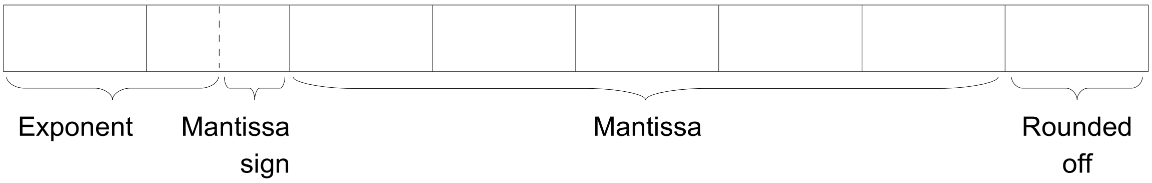
Variable and Program Structure

The internal format of numbers and variables is described in the following paragraphs.

(1) Internal format of numbers

A number is represented using 8 bytes. A numeric value consists of an exponent, mantissa sign, and mantissa.

Numbers from -9.999999999 x 10^{99} to 9.999999999 x 10^{99} can be represented.



i) Exponent

- The exponent is represented using two decimal digits.
  The most significant digit is always zero for positive numbers.
- Negative numbers are represented using a complement.
  901 (10^{-99}) to 099 (10^{99})

ii) Mantissa sign

- Zero is used when the mantissa is positive.
- Eight is used when the mantissa is negative.

iii) Computation correction

- Computation correction is performed only during computation. Normally, it is reset after rounding off.

(Example) Assume that a number is stored in 6CF0H to 6CF7H (fixed variable B)

6CF0H							6CF7H	
00H	30H	15H	00H	00H	00H	00H	00H	1500
00H	00H	12H	34H	56H	00H	00H	00H	1.23456
99H	70H	12H	34H	56H	78H	90H	00H	0.00123456789
00H	88H	12H	34H	00H	00H	00H	00H	-1.234X10^{8}

Table 1

The same internal format is used for numbers in operation registers in the CPU.

(2) Internal format of character strings

- When a character string is stored in a variable other than a fixed variable (including A() arrays), the ASCII code of the contents of the character string is stored directly.
- When a character string is stored in a fixed variable, the character variable code (F5H) is set at the beginning. The remainder is stored in ASCII code.

(Example) Assume that character string PC1350 is stored in Z$ (6C30H to 6C37H).

6C30H 6C37H

F5H	50H	43H	31H	33H	35H	30H	00H
Character string code	P	C	1	3	5	0	

- When character string operations are processed in a CPU operation register, the internal format is not the same as in the case of a variable; character string information is represented using 8 bytes (4 bytes of actual data), and the actual character string exists in the address indicated by the character string information.

•	•	•	•	D0H	ADDH	ADDL	LENGTH
Invalid data				D0H	Beginning of the character string		Length of the character string

D0H: character string identification code

The length of the character string can be from 01H to 50H.

Beginning of the character string: The address in the string buffer can be used to indicate the beginning of the character string. The acceptable range is from 6E60H to 6EAFH.

(Example) Assume that character string information is contained in operation register X, and the actual character string SHARP exists in the string buffer.

10H 17H (RAM in CPU)

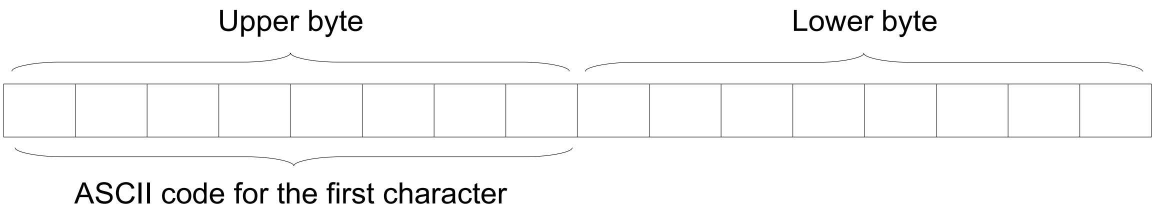
•	•	•	•	D0H	60H	6EH	05H

6E60H 6E64H

53	48H	41H	52H	50H
S	H	A	R	P

(3) Variable name configuration

The name of the variables created in variable area such as AB$ or X(5,5) are represented using two bytes which indicate the ASCII code of the variable name, whether it is numeric or character, and whether it is an array or not.



Lower Byte

i) When the variable name is a single character (array only)

Number array	→	80H is stored.
Character array	→	A0H is stored.

ii) When the variable name consists of two or more characters

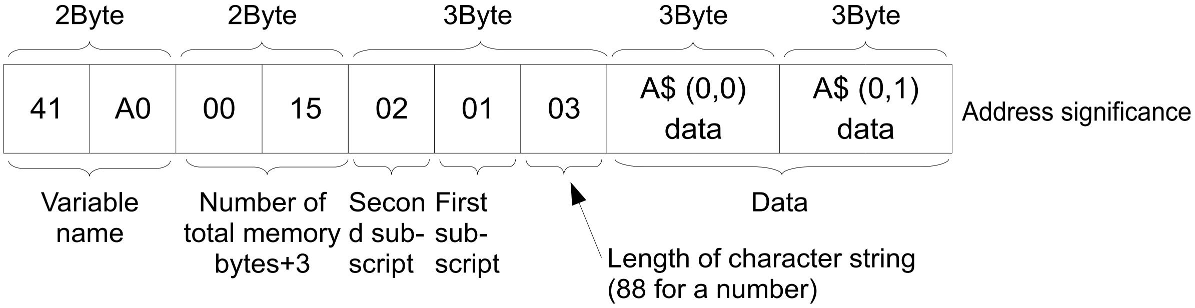
If the variable is character, 40H is added to the ASCII code of the second character. If the variable is an array, 80H is added.

Variable name	Code
B1	4231H
CC	4343H
D (2)	4480H
EE (1)	45C5H
F$ (1)	46A0H
GG$	4787H
ZZ$ (2)	5A1AH

Table 2

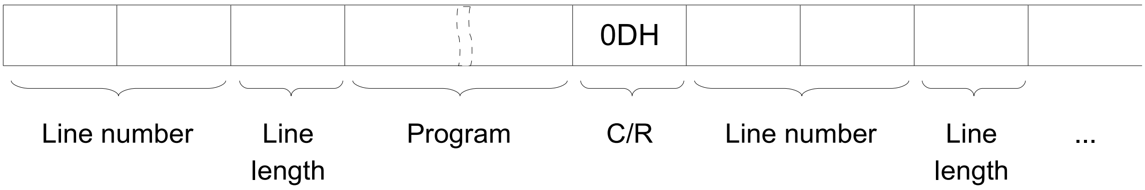
When an A() array is used as an extension for fixed variables A through Z, the variable name (code) is 4000H.

(Example) Assume that A$ (1,2) * 3 is declared for an array.



(4) Program configuration

Each line of a program is represented by a line number, line length, program, and end code.



For the following program (with no RAM card)

```
10  PRINT  A
20  END
```

the following data is stored.

Address	Data	
6030H	FFH	… Code indicating the beginning of the BASIC program
31H	00H	10
32H	0AH	
33H	03H	… Line length
34H	DEH	… PRINT
35H	41H	… A
36H	0DH	… C/R
37H	00H	20
38H	14H	
39H	02H	… Line length
3AH	D8H	… END
3BH	0DH	… C/R
603CH	FFH	… Code indicating the end of the BASIC program

Table 3

(5) Reserved area configuration

Reserved area consists of address 6F6FH through 6FFEH in system RAM. Reserved contents are catalogued in the following format.

i) Reserved key code

There are 18 reserved keys. Each reserved key is catalogued using a reserved key code.

Reserved key	Code
A	81H
B	82H
C	83H
D	84H
F	86H
G	87H
H	88H
J	8AH
K	8BH

Reserved key	Code
L	8CH
M	8DH
N	8EH
S	F3H
V	F6H
X	F8H
Z	FAH
spc	F1H
=	F4H

Table 4

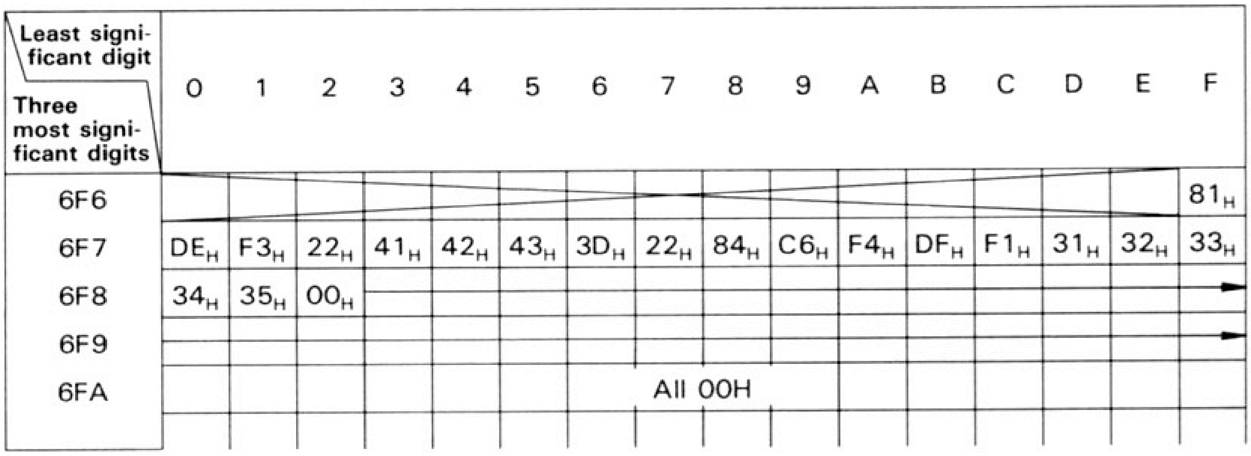
ii) Reserved contents are written after each reserved key code. Delimiters are not inserted between reserved programs. Reserved programs are written in the order they are catalogued. If a program is re-catalogued, the previous program is deleted, and the new program is added at the end of the catalogue.

iii) If NEW is executed in the reserved mode, the reserved area is filled with hexadecimal zeros. Therefore, unused area will contain 00H.

(Example) Assume that the following contents are catalogued in reserved area.

Catalog sequence	Reserved area	Catalogued contents
1	A	PRINT
2	S	"ABC="
3	D	GOTO
4	=	INPUT
5	SPC	12345

Table 5



Least significant digit Three most significant digits	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
6F6																81_H
6F7	DE_H	$F3_H$	22_H	41_H	42_H	43_H	$3D_H$	22_H	84_H	$C6_H$	$F4_H$	DF_H	$F1_H$	31_H	32_H	33_H
6F8	34_H	35_H	00_H													
6F9																
6FA								All 00H								

Table 6 System RAM

SYSTEM CONFIGURATION

The SHARP PC-1350 Pocket Computer is divided into four functional blocks and some support devices. The four functional blocks are: the central processing unit (CPU), the random access memory (RAM), the read-only memory (ROM), and the I/O interface. These functional blocks are connected by three buses: the 16-bit address bus, the 8-bit bidirectional data bus, and the control bus. Figure 1 shows the configuration of the SHARP PC-1350 Pocket Computer.

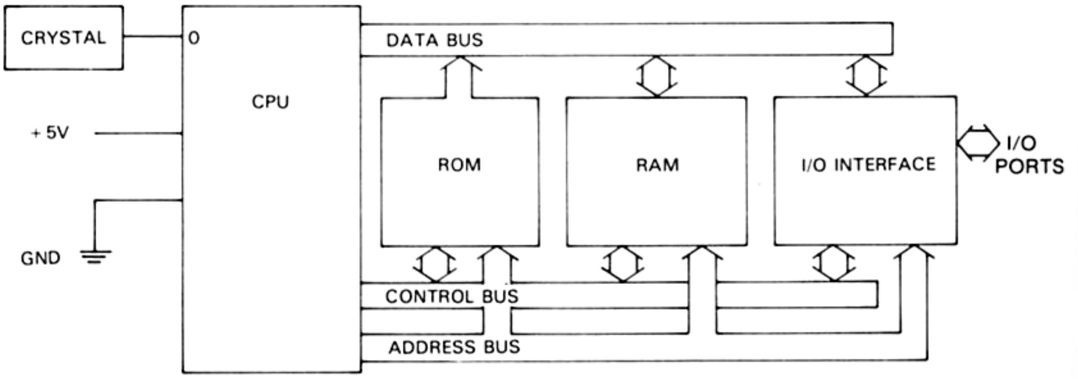


Fig. 1 A Simplified Diagram of System Architecture

The PC-1350 operates on dc 5-volt power supply and runs on the 768 KHz basic system dock. The basic system clock is generated in the CPU. Its clock frequency is derived from the 768 KHz quartz crystal which is external to the CPU.

The CPU controls the flow of data to and from, and between the other system blocks. It places one byte of data or code at a time on the data bus from one memory or I/O block (RAM, ROM, or I/O interface) and takes it into itself (fetch) or further stores it in another memory or I/O location in the same or another system block via the data bus (move).

The location (or address) of the data that the CPU is to read, store, or move is designated by the 16-bit address bus. This address is generated by the CPU. The PC-1350 address bus can address up to 64K main memory locations.

The control bus carries various control signals generated by the CPU. The CPU controls the overall timing of the system operations using the control signals placed on the control bus.

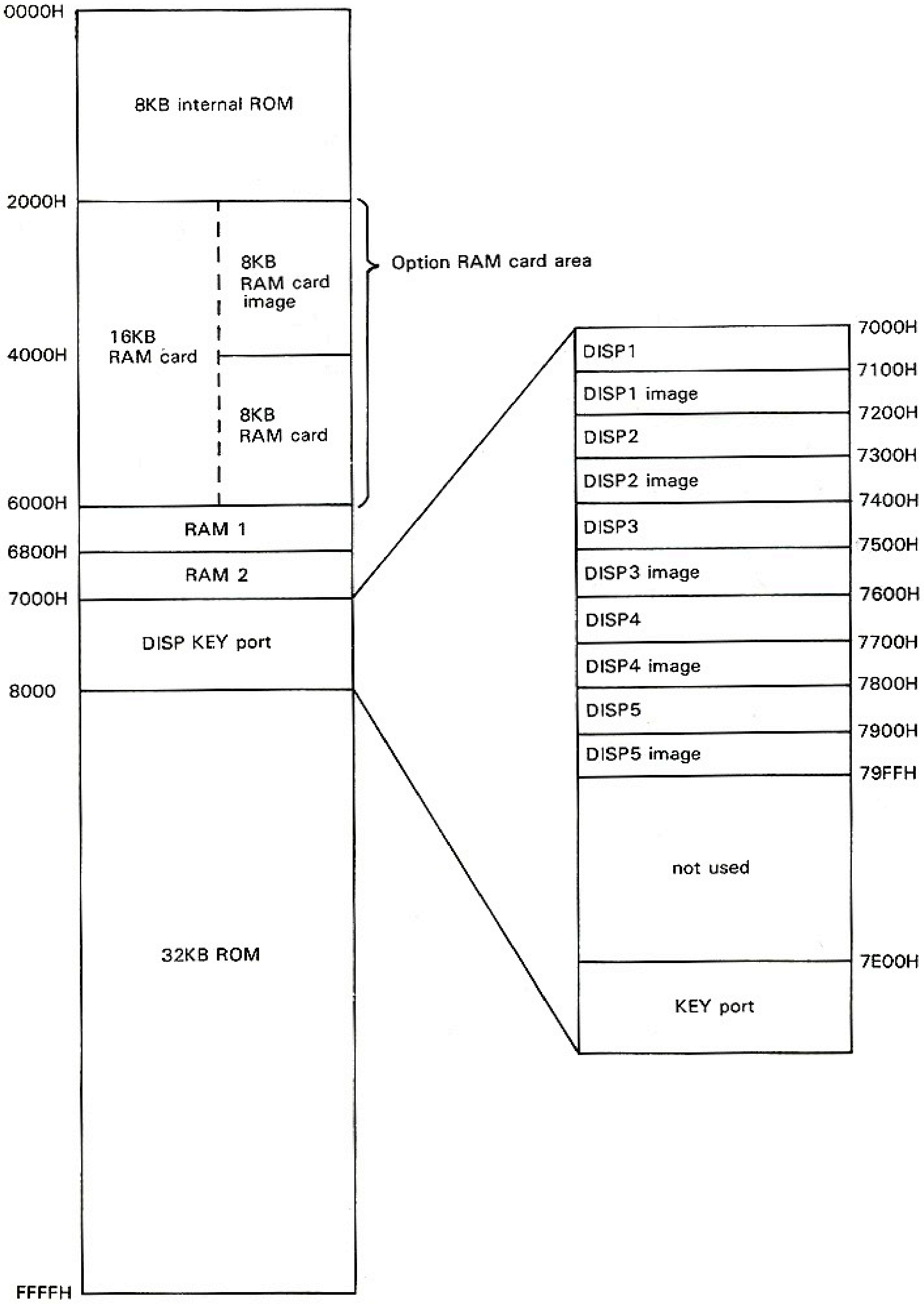
ROM stores data which can be read but which cannot be altered. It is used primarily to store program code. The ROM block shown in the figure is 32K bytes and contains the PC-1350 BASIC interpreter. The CPU also incorporates 8K bytes of internal ROM which holds the PC-1350 command interpreter.

RAM is a memory device which data can be written into and read from. It is used to hold intermediate values of computations and BASIC variables during execution. User programs are also loaded in RAM for execution.

The I/O interface block consists of interfaces for the keyboard, the LCD display, the CE-126P printer, and the CE-127R cassette recorder. This block is connected to the CPU and other system blocks through the address, data, and control buses. Except the LCD display, all I/O interfaces are controlled by programs written in either BASIC or machine language. The LCD display can and should be controlled only by means of machine-language programs because of its complexity.

System Memory Map

The system memory map is shown below.



For details see Appendices

Fig. 2 System Memory Map

The CPU

The CPU is the center of the PC-1350 Pocket Computer. It fetches instruction code, interprets it, and, depending on the instruction, loads data from memory, performs arithmetic operations on the data, and stores the processing results in memory.

The CPU is made up of the arithmetic/logical unit (ALU), the data pointer (DP) register, the program counter (PC), the 96-byte internal RAM, general-purpose registers, and the control unit which controls the internal operation of the CPU. Figure 3 gives a schematic diagram of the PC-1350 CPU.

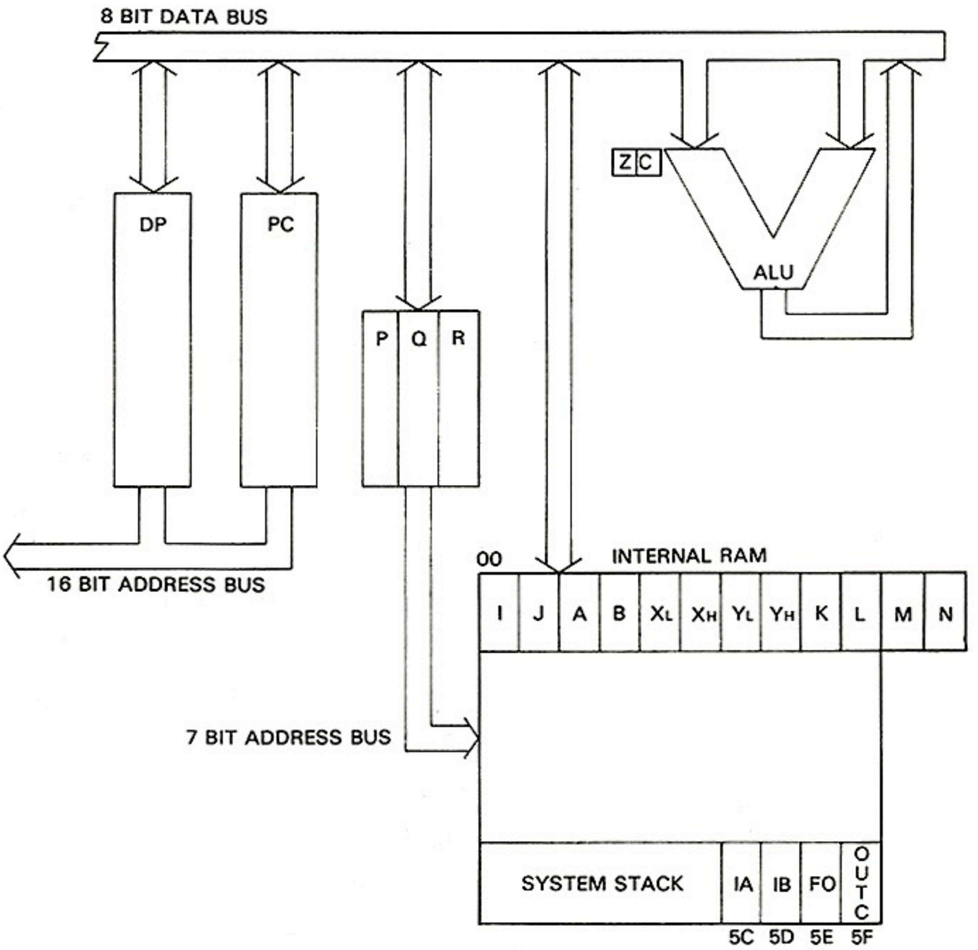


Fig. 3 A Simplified Diagram of the ESR-H CPU

In addition to the 16-bit address bus, 8-bit data bus, and control bus, which connect the CPU to the other system blocks, the CPU has a 7-bit internal bus. This internal bus is used to address the location of the internal RAM which is used as internal registers.

The ALU performs arithmetic and logical operations. It takes one or two operands, performs an arithmetic or logical operation, and stores the result in a register, usually the A register in the internal RAM.

There are two flags in the CPU, the carry (C) flag and the zero (Z) flag, which are affected by the operation of the ALU. The Z flag is set (loaded with a 1) if the result of an operation is zero and reset (loaded with a 0) if it is nonzero. The C flag is set if the operation generates a carry and reset otherwise. These flags can be tested, set, and reset directly by user programs.

The flags are used to control the flow of program execution. They are examined by conditional instructions that cause execution to branch to a different portion of the program depending on their state.

Not all PC-1350 instructions affect the flags. Which instructions affect flag(s) and which instructions do not are described in the description of the individual PC-1350 Instructions.

The DP register is 2 bytes wide and used to address a location in external memory.

All load and store instructions are performed on the memory location designated by the DP register. The DP register can be incremented, decremented, or loaded with an immediate value or the data that is moved from the X or Y register in the internal RAM.

The PC is a 2-byte register which contains the address of the instruction to be executed next. It is incremented sequentially to point to the next instruction as instructions are executed. The PC may be loaded directly with a nonsequential address by a flow-controlling instruction such as JUMP or CALL.

The P, Q and R registers are used to address the internal RAM. The P and Q registers normally designate internal registers in the internal RAM. The R register holds the value that points to the top of the system stack in the internal RAM. These registers are 7 bits wide, which is adequate to address the 98-byte internal RAM.

The internal RAM contains 12 internal registers including an accumulator, the work area, the system stack area, and the I/O port registers. The internal registers are named I, J, A, B, Xl, Xh, Yl, Yh, K, L, M, and N. They are arranged in the internal RAM as shown in Table 7.

ADDRESS	REGISTER
00	I
01	J
02	A
03	B
04	XL
05	XH
06	YL
07	YH
08	K
09	L
0A	M
0B	N

Table 7 Internal Ram Registers

The I and J registers are 1 byte wide and used as index registers. They contain a byte offset with respect to a base address. The I and J registers are assumed by block move instructions as holding the number of bytes to be moved.

The 1-byte A register functions as an accumulator. It is used to store the result of an arithmetic or logical operation performed in the ALU. Most data movement operations (as directed by load and store instructions) are carried out via the A register. The B register is a l-byte spare register and used in the same way as the A register.

The X and Y registers are used as address pointers. They are 2 bytes wide with the lower order byte occupying the lower address in the internal RAM. The X register is typically used by the IXL instruction to point to the address whose contents are to be loaded into the accumulator (A register). The Y register is typically used by the IYS instruction to point to the address in which the data in the accumulator is to be stored.

The K, L, M, and N registers are 1-byte general-purpose registers. They may be used to hold intermediate values of computations.

The internal RAM contains four I/O port registers at locations 5C, 5D, 5E, and 5F in hexadecimal. These registers hold a l-byte data which is to be sent to an I/O device with the OUTA, OUTB, OUTF, or OUTC instructions.

The internal RAM also has a system stack. The system stack is of the last-in first-out (LIFO) structure. The top of the stack is always pointed to by the R register. Data may be pushed into and popped out of the stack area, 2 bytes at a time. The first data that is pushed is placed at the bottom of the stack and the latest data, which is to be popped out of the stack first, is placed at the top of the stack. The stack starts at internal RAM address 5B in hexadecimal and grows downward or toward the lowest address in the internal RAM. The stack is used to hold temporary data and the return address of subroutines. The PUSH, POP, CALL, and RTN instructions, when executed, automatically increment or decrement the contents of the R register that points to the top of the stack.

The Instruction Execution Cycle

This section describes how an instruction is executed in the CPU. Understanding the basic mechanism of instruction execution will help the user construct programs in PC-1350 machine language.

As the execution of an instruction starts, the CPU places the contents of the PC on the address bus. The program code addressed by the address data on the address bus is then placed on the data bus. The CPU fetches the code on the data bus into one of its registers called the instruction register (IR). The control unit of the CPU interprets the code and generates internal and external control signals in the sequence established by the code to perform the specified operation.

After the instruction is fetched, the PC is automatically incremented by one to point to the next address. If the instruction requires the second and third operand bytes (e.g., LIA or LIDP), the CPU reads them and the PC is incremented accordingly to designate the instruction to be executed next. Thus, when the execution of an instruction is completed, the PC points to the next sequential instruction.

The above steps are represented in terms of machine cycles of the CPU. Different instructions require different number of machine cycles and therefore take different times to execute. The number of machine cycles that each instruction requires is stated in the individual instruction descriptions that are given in a later section.

BASIC Program Areas

A BASIC program is stored in RAM memory starting at address 6030 in hexadecimal. The location of the program are as that are used by BASIC programs are illustrated in Figure 4.

USER AREA

6000	
	Header & System Pointers
602F	
6030	BASIC program Source
6900	Recommended Machine Code Starting Point
6C2F	Array Storage
6C30	
	Predefined Variables
6CFF	
6D00	
	Various System Pointer
6F6E	
6F6F	
	Reserve Key Information
6FFF	

Fig. 4

Space for simple and dimensioned BASIC variables are dynamically allocated in memory starting at address 6C2F. This area extends toward the lowest address of memory. In Figure 5, this variable area starts at address 6C2F and ends at 6900. User supplied machine-language programs should be placed somewhere between these two areas. In the example shown in Figure 5, a machine-language program can start at address 6900 provided that the BASIC program does not extend beyond this address. Also, the BASIC variable area must not grow beyond the last address of the machine-language program.

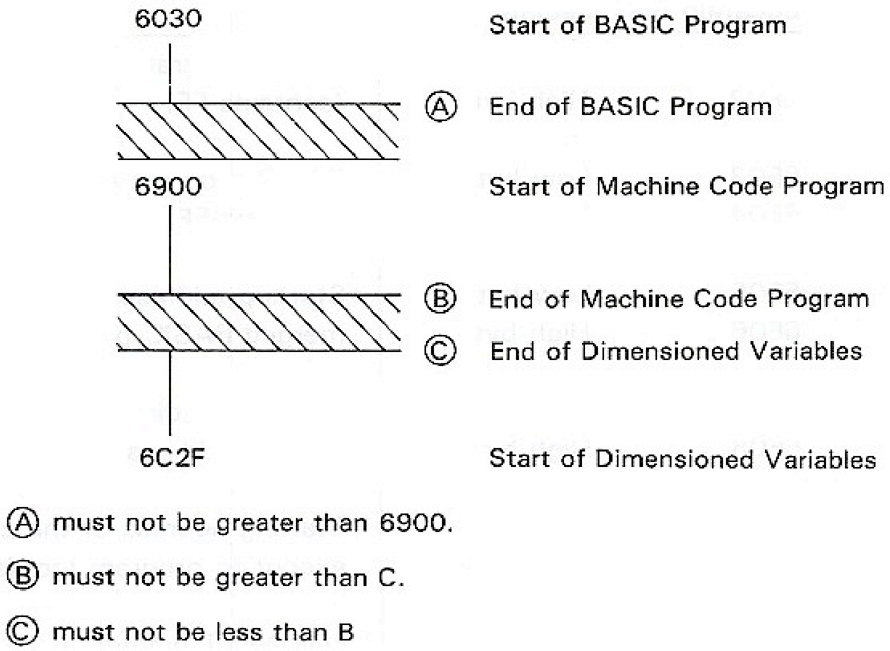

6030
Start of BASIC Program
Ⓐ End of BASIC Program
6900
Start of Machine Code Program
Ⓑ End of Machine Code Program
Ⓒ End of Dimensioned Variables
6C2F
Start of Dimensioned Variables
Ⓐ must not be greater than 6900.
Ⓑ must not be greater than C.
Ⓒ must not be less than B

Fig. 5

A system memory area starting at 6F01 in hexadecimal contains the locations of the BASIC program areas. They are listed in Table 8.

Location (hexadecimal)		Contents	
6F01	Low byte	BASIC program starting address	
6F02	High byte	(contains FF)	
6F03	Low byte	BASIC program ending address	
6F04	High byte	(contains FF)	
6F05	Low byte	Starting address of the last	
6F06	High byte	merged BASIC program (contains FF)	
6F07	Low byte	Simple and dimensioned variables	
6F08	High byte	starting address	
6F1C	Low byte	Starting address of the currently	
6F1D	High byte	executing program (contains FF)	
6F2B		FOR-NEXT pointer	Current top address
6F2C		GOSUB pointer	
6E06		FOR-NEXT stack area	
6E5F			
7090		GOSUB stack area	
70A3			

Table 8 BASIC Program Area Control Table

MACHINE-LANGUAGE PROGRAMMING

PC-1350 BASIC provides one function and four statements to facilitate the user to handle machine-language programs from his or her BASIC programs and pass information between them. They are the PEEK function and the POKE, CALL, CSAVE M, and CLOAD M statements. The PEEK function reads the contents of a memory location. It is used to receive argument information. The POKE statement loads a byte of information into a memory location. It can be used to place machine-language code directly into desired locations in the user area. The CALL statement transfers CPU control to a user-supplied machine-language program. The CSAVE M statement saves a machine-language program onto cassette tape and the complementary CLOAD M statement loads a machine-language program into memory from cassette tape. This section shows with examples how to load and run user-supplied machine-language programs using these BASIC facilities.

Using the PEEK Function

The PEEK function is used to read the contents of memory locations. It takes one argument which evaluates to an address expression. The general format is shown below.

```
PEEK < expression >
```

< expression> specifies the memory location to be peeked. It must be an address expression which is evaluated to a hexadecimal value from &2000 to &FFFF. The PEEK function returns the contents of the memory location specified in < expression > in the form of a decimal number. The memory contents may be viewed by displaying them on the screen with the PRINT statement.

The PEEK function may be used to find the address of memory areas. For example, the function call

```
PEEK &6F01
```

should return a decimal number of 48 (30 in hexadecimal). A subsequent PEEK call with the next memory address (&6F02) as its argument should return 96 (60 in hexadecimal). Since address data is 2 bytes long and always stored in memory with the lower order byte first on the PC-1350, these two bytes form an address value of &6030 in hexadecimal, which is the starting address of the user BASIC program (see Figure 5).

The sample program given below illustrates the use of the PEEK function. This program takes a dump of contiguous memory locations. When executed, this program asks for the beginning address of the location you want to look into and the number of bytes. When you press the RETURN key, the program will display on the screen addresses and their contents in hexadecimal repeatedly for the number of bytes you specified.

```
 10:  INPUT "ADDR?"; A
 20:  INPUT "BYTES?"; B
 30:  FOR I=0 TO (B-1)
 40:  X=PEEK (A+I)
 50:  P = INT (X/16)
 60:  Q = (X -16*P)
 70:  IF P>9 THEN LET P=P+ 7
 80:  IF Q>9 THEN LET Q=Q+7
 90:  P=P+48:Q=Q+48
100:  PRINT (A+I);" "; CHR$ P; CHR$ Q
110:  NEXT I
120:  END
```

Variable A holds the starting address you entered. Variable B holds the number of bytes to be looked into and is used as the control variable for the FOR loop formed by lines 30 through 110. In the FOR loop, the memory contents are placed in X and then divided into two hexadecimal values and stored in P and Q. On lines 70 and 80, a check is made to determine if the hexadecimal number fall between A and F. On line 90, the hexadecimal numbers are converted to ASCII code. Line 100 prints the address and its contents. The CHR$ function converts the ASCII code to its character representation.

Given a starting address of &6030 and a byte count of 17, the above program will give the following display (provided that only the above program is loaded in the BASIC program area):

```
24624  FF
24625  00
24626  0A
24627  0B
24628  DF
24629  22
24630  41
24631  44
24632  44
24633  52
24634  3F
24635  22
24636  3B
24637  41
24638  0D
24639  00
24640  14
```

The first code FF in the above display identifies the beginning of your BASIC program. The next two bytes indicate the line number of the first statement in binary. The following code B is the length of that statement. DF is the internal code for the BASIC statement INPUT. 22 represents a double quotation mark (" ") and 41 represents the letter A. The subsequent several codes are alphabetic characters. 0D at address 24638 identifies the end of the statement. 00 and 14 form a pointer to the beginning of the next tine.

Reference: Program Line Format

BASIC program lines are stored in the BASIC program area in memory in the format shown below.

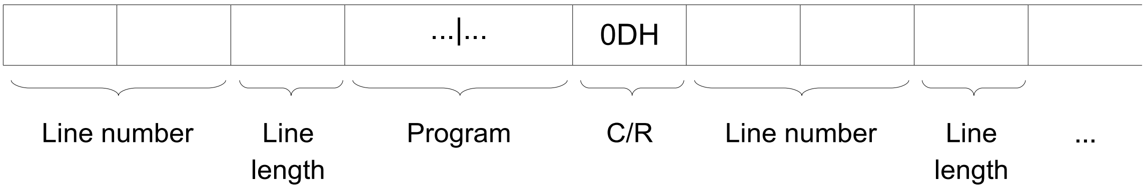


Fig. 6

As shown above figure, a BASIC program line consists of the line number, line length, program line, and termination code fields.

Using the POKE Statement

The POKE statement loads specified memory location(s) with data byte(s). It is typically used to bury machine-language code into the BASIC program area.

The POKE statement has two formats:

1. POKE expression, expression
2. POKE expression, expression, ... , expression

The POKE statement in format 1 stores the value of the second expression in the memory location designated by the first expression. The POKE statement in format 2 stores the values of the second and subsequent expressions in the contiguous memory locations starting at the address designated by the first expression. The second and subsequent expressions must be evaluated to values between 0 and 255.

Enter the following POKE statement for an example:

```
POKE &6900,&12,&06,&02,&D7,&37
```

The results of this statement can be examined using the preceding sample program for the PEEK statement. A sample run is given below.

Sample Run

```
ADDR?  &6900
BYTES? 5
26880  12
26881  06
26882  02
26883  07
26884  37
```

Running a Machine-language Program

User-supplied machine-language programs are executed as subroutines which are called from BASIC programs. A machine-language program can be loaded into a free BASIC program area for execution using, for example, the POKE statement. A machine-language program is started by a BASIC program by transferring CPU control with the CALL statement.

The CALL statement takes one argument which specifies the starting address of the machine-language program to be executed and transfers control to that address. For example, the statement

```
CALL &6900
```

initiates the execution of the machine-language program in the program area starting at address &6900.

Since a machine-language program is a subroutine, it must end with a RTN machine-language instruction (a machine-language program can have more than one RTN instruction). When a RTN instruction is encountered during execution, CPU control is returned to the BASIC program, immediately following the CALL statement that called the machine-language program.

Several examples for running machine-language programs are given in the following sections.

The CSAVE M Statement

The CSAVE M statement saves the contents of the specified memory area onto cassette tape in the machine-language format. It has the following general format:

```
CSAVE M "filename"; < expression-1 > , < expression-2 >
```

"filename" is the name of the memory data with which data is to be recorded on cassette tape. < expression-1 > identifies the beginning of the memory area and < expression-2 > the end of the memory area. These parameters must be evaluated to address values. They are required and you must specify both parameters.

For the tape formats of the BASIC programs and data, see Appendixes.

The CLOAD M Statement

The CLOAD M statement loads cassette tape data into memory in the machine-language format in the same location at which it was saved. It has the following general format:

```
CLOAD M "filename"; < expression >
```

"filename" is the name of the data stored on cassette tape. You cannot omit the filename. < expression > must be evaluated to an address value. If it is specified for a machine-language program, execution starts at this address immediately after the program is loaded (auto start feature).

Sample Program 1: Simple Program

The simple machine-language program given below loads a number 6 (6 is arbitrarily chosen) into the accumulator and places it in memory location 6DF0 in hexadecimal. This location is selected because it is the address that follows immediately the end of the BASIC variable area and will do no harm to the PC-1350 when the location is altered. Here is the example machine-language program:

```
LIA     06
LIDP    6DF0
STD
RTN
```

The LIA instruction loads the accumulator A with immediate number (06). The LIDP instruction loads the DP register with 2-byte address data (6DF0). STD stores the data in the A register in the memory location pointed to by the contents of the DP register. The last instruction, RTN, returns control to the BASIC program that called this program. When this program is executed successfully, a 06 will be placed in memory location 6DF0.

To run the above program, you must "hand-assemble" it, that is, you must represent the program in code that the PC-1350 can understand. A table such as shown below will help you assemble the machine-language program.

Addr	Machine Code	label	OP Code	Operand(s)	Comments
6900			LIA	06	Data to be stored
			LIDP	6DF0	
			STD		
			RTN		

Table 9

The Addr column contains the addresses in memory where the machine codes are to be placed. The Machine Code column contains hand-assembled codes. The use of the Label column will be described later. The OP Code column holds the machine-language instructions in mnemonic form. One machine-language instruction must be placed in each row. The Operand(s) column contains the operand(s) on which the operation specified by the OP code is to be performed. Some instructions take one operand and other instructions two. There are instructions, such as STD and RTN, which take no operand. The Comments column may contain any remarks you want to make. You should write down here what the instruction does for what purposes so that you can later recall what is going on with the program.

After completing the Addr, OP Code, and Operand(s) columns, translate the instructions into machine code referring to the PC-1350 instruction descriptions given in the later portion of this manual. For example, the instruction LIA 06 can be translated into 0206. 02 is the machine code of the LIA instruction and 06 is its operand. Because this instruction takes up two bytes of memory, the address of the next instruction must be 6902. Place this address value in the second row of the Addr column. LIDP is translated into 106DF0 where 6DF0 is the operand of the LIDP instruction and designates a memory location. This instruction occupies 3 bytes, so the next instruction starts at address 6905. STD is 1 byte long and has a machine code of 52. Write 52 in the row labeled 6905. Finally, fill the next row; write down 6906 in the Addr colurnn and 37, which is the machine code of the RTN instruction, in the Machine Code column.

Addr	Machine Code	label	OP Code	Operand(s)	Comments
6900	0206		LIA	06	Data to be stored
6902	106DF0		LIDP	6DF0	
6905	52		STD		
6906	37		RTN		

Table 10

When the table is completed, load the machine-language program into memory using a BASIC program. Because this program is fairly short, you could do it with a single POKE statement. The sample code below uses two POKE statements for readability.

```
200: POKE &6900,&02,&06,&10,&6D,&F0
210: POKE &6905,&52,&37
220: END
```

Enter and run the above program. If the PEEK program which is discussed previously is still in memory, you can check to see how the program is loaded in memory.

After making sure that the program code is loaded properly in memory, enter the following program code:

```
300: POKE &6DF0,0
310: PRINT "BEFORE "; PEEK &6DF0
320: CALL &6900
330: PRINT "AFTER "; PEEK &6DF0,0
340: END
```

The above program initializes the "interface" address to 0 and prints its contents before and after a calI to the sample machine-language program.

If the program executes successfully, the BEFORE value should be 0 and the AFTER value should be 6.

Sample Program 2: Converting Binary Numbers to Hexadecimal Numbers

The second sample program converts binary data to a hexadecimal number. The program is basically identical to the program lines 50 through 90 of the previous PEEK program, though a different algorithm is used. The program includes some additional machine-language programming principles.

The program can be divided into several code segments. The first code segment starts at address 6900 and ends at address 690C. It places F5 in the first byte position of the preallocated variable Y. F5 identifies that Y is a character variable (see the description on the BASIC internal variable structure).

The first six instructions load the 16-bit Y register in the internal RAM with the memory address one byte less than the address of the beginning of the preallocated variable Y. Although the DP register is normally used to point to memory locations, it is hard to update its contents. To update the DP register contents, it is most easy to load the DP register with the contents of the X or Y register which is easy to update. The PC-1350 has many instructions which load the DP register with the contents of the X or Y register after incrementing or decrementing the register.

So the Y register is first initialized (6900-6908). The LIA instruction at address 6909 loads the required byte (F5) into the A register. The IYS instruction at address 690B increments the Y register, loads the incremented Y register value into the DP register and stores the contents (F5) of the A register in the memory address pointed to by the DP register, that is, the beginning of the preallocated variable Y (because of the auto increment feature).

Load and run the program segment you constructed so far. Examine the contents of the Y$ variable with the previous PEEK program to see whether F5 is placed in the correct memory location. Do not forget to clear Y$ with the assignment statement Y = 0 or other BASIC statements. This stepwise programming, that is, writing and testing a program in small chunks, is recommended for building good programs.

Addr	Machine Code	Label	Mnemonic	Operand(s)	Comments
6900	12 06		LIP	06	Address of YL
02	02 37		LIA	37	
04	DB		EXAM		
05	50		INCP		address of YH
06	02 6C		LIA	6C	
08	D8		EXAM		6C38=Y$
09	02 F5		LIA	F5	char variable header
08	26		IYS		store in Y$
690C	10 6D F0		LIDP	6DF0	"window" address
690F	57		LDD		get byte
6910	34		PUSH		save copy
6911	58		SWP		set up high nibble
6912	78 69 1D		CALL	(1)	convert high nibble
6915	58		POP		get copy
6916	78 69 1D		CALL	(1)	convert low nibble
6919	02 00		LIA	00 00	= end of string
691B	26		IYS		place null in Y$
691C	37		RTN		return to basic
691D	64 0F	(1)	ANIA	0F	mask off top nibble
1F	34		PUSH		save copy
20	75 0A		SBIA	0A	will set carry if result is negative
22	3A 06		JRC	(2)	if number is decimal, jump, if hex, continue
24	58		POP		get binary value
25	74 37		ADIA	37	add ALPHA offset
27	2C 04		JR	(3)	
29	58	(2)	POP		get binary value
2A	74 30		ADIA	30	add NUMERAL offset
2C	26	(3)	IYS		store HEX CHAR in Y$
2D	37		RTN		return calling routine

Table 11 Binary to Hexadecimal Conversion

The next two instructions (addresses 690C-690F) load the contents of the window into the accumulator (A). The PUSH instruction saves the copy of the accumulator onto the stack. Do not forget to pop out this data at a later time; otherwise, the correct return address could not be set up when a later RTN instruction is executed.

The SWP instruction exchanges the higher and lower nibbles of the byte in the accumulator. The subroutine at addresses 691D through 6920 converts the lower nibble in the accumulator into a hexadecimal character.

The CALL instruction is used to invoke a subroutine in external RAM memory (CALL may be used to calI a program in memory below 1FFF). The CALL instruction, like the BASIC CALL statement, requires an absolute address argument. To give the correct address argument to the CALL instruction during hand assembly, leave two bytes of space after the operation code of the CALL instruction (78). When the address of the target subroutine is later established, fill this space with that address. In this example, the two CALL instructions at addresses 6912 and 6916 invoke the subroutine that starts at address 691D.

The first subroutine call converts the higher nibble of the accumulator to its hexadecimal character representation and places it into memory addressed by the Y register. The subsequent POP instruction gets the copy of the byte saved by the PUSH instruction at address 6910. The second subroutine call now converts the lower nibble of the byte to its hexadecimal character representation. The two instructions at ad dresses 6919 and 691B place a null character (00) in memory after the two hexadecimal characters. A null character identifies the end of a character string. The subsequent RTN instruction returns control to the BASIC program that called this machine-language program.

The subroutine between 6910 and 6920 does the binary-to-hexadecimal conversion. The subroutine first tests the given nibble to see whether it is greater than 9. If it is smaller than or equal to 9, the subroutine adds a constant 30 in hexadecimal to the nibble to put it in the range 30 to 39 in hexadecimal, which correspond to the ASCII numeric characters 0 to 9. If the nibble is greater than 9, the subroutine adds a hexadecimal constant 37 to put the nibble in the range of 41 to 46 which correspond to the ASCII letters A to F.

The ANlA instruction at address 691D masks off the higher nibble to leave the lower nibble in the accumulator. Subsequently, the nibble is saved for later processing. The nibble in the accumulator is then checked whether it is greater than 9 by subtracting 10 (0A in hexadecimal). If the nibble is smaller than or equal to 9, an offset for letters is added to the nibble. If the nibble is greater than 9, which is indicated by the carry flag being set, the program jumps to the instruction identified by label (2) to bypass the above-mentioned conversion step. The JRC* instruction tests the carry flag and, if it is set, causes control to transfer to the location (address 6929 in this example) identified by the operand field of the instruction.

The code from addresses 6924 to 6928 converts a hexadecimaI number to a ASCII letter by adding an offset for letters (37 hex). The POP instruction at address 6924 restores the hexadecimaI number into the accumuIator.

The code from addresses 6929 to 692B simply converts a hexadecimaI number to its ASCII equivalent by adding an offset for ASCII code (30 hex).

The result of the conversion is stored in the BASIC variable Y$ by the IYS instruction at address 692C. The subroutine is then exited by the RTN instruction at address 692D.

* Relative versus absolute jumps

The PC-1350 has two types of jump instructions: absolute jumps and relative jumps. The absolute jump instructions require a 2-byte operand while the relative jump instructions require a 1-byte operand. In either case, the value of the operand is algebraically added to the PC during execution so that execution continues at a nonsequential address.

To determine the value of the operand of a relative instruction, find the address of the destination and subtract from it the address where the operand is to be placed, that is, the address of the relative instruction plus 1. In this sample program, the address of the operand is 6923 and the destination address is found to be 6929 in hexadecimal, so the value of the operand is ca1culated as 6929 - 6923 = 6.

Relative jumps are trickier to calculate and are more likely subject to errors than absolute jumps. A thumb of rule is to first write a program using only absolute jumps. After the program is extensively tested and proved to run normally, substitute relative jumps for absolute jumps. The advantages of relative jumps are that they take less memory and execute a little faster than absolute jumps and that they need not be modified when a program is to be relocated in memory.

The sample program below illustrates how to use the machine-language program from a BASIC program. The program asks for the starting address of the data to be converted to ASCII characters. Then it fetches a byte from the specified data area with the PEEK function, places it in the window at 6DF0 with the POKE statement, and calls the machine-language program at address 6900 with the CALL statement. The PRINT statement on line 70 displays the results of the conversion stored in string variable Y$. The program repeats the above sequence for the specified number of bytes.

```
10:  INPUT "ADDR?"; A
20:  INPUT "BYTES?"; B
30:  FOR I=0 TO (B-1)
40:  X=PEEK (A+I)
50:  POKE &6DF0, X
60:  CALL &6900
70:  PRINT (A+I);" ";Y$
80:  NEXT I
90:  END
```

The machine-program can be loaded into memory using the BASIC program given below.

```
400:  POKE &6900,&12,&06,&02,&37,&DB,&50
410:  POKE &6906,&02,&6C,&DB,&02,&F5,&26
420:  POKE &690C&10,&6D,&F0,&57,&34,&58,&78,&69,&1D
430:  POKE &6915,&5B,&78,&69,&1D,&02,&00,&26,&37
440:  POKE &691D,&64,&0F,&34,&75,&0A,&3A,&06
450:  POKE &6924,&5B,&74,&37,&2C,&04
460:  POKE &6929,&5B,&74,&30,&26,&37
470:   END
```

PC-1350 I/O

The PC-1350 is provided with several I/O devices. The I/O devices include a keyboard, a liquid Crystal Display (LCD), a serial interface, and a cassette record.

The following sections describe the PC-1350 I/O devices with sample driver programs.

LCD Display

The PC-1350 employs an LCD display consisting of 150 dots horizontally and 32 dots vertically. Each dot on the screen is mapped into a bit in video memory; that is, a dot is turned on by setting on the corresponding bit in video memory. The video memory starts at address 7000 in hexadecimal. A vertically aligned 8 dots form a display pattern as shown in the figure below. A display pattern is represented by a number consisting of 8 video memory bits with each bit assigned a weight as shown in the figure. The LCD video RAM map is shown below.

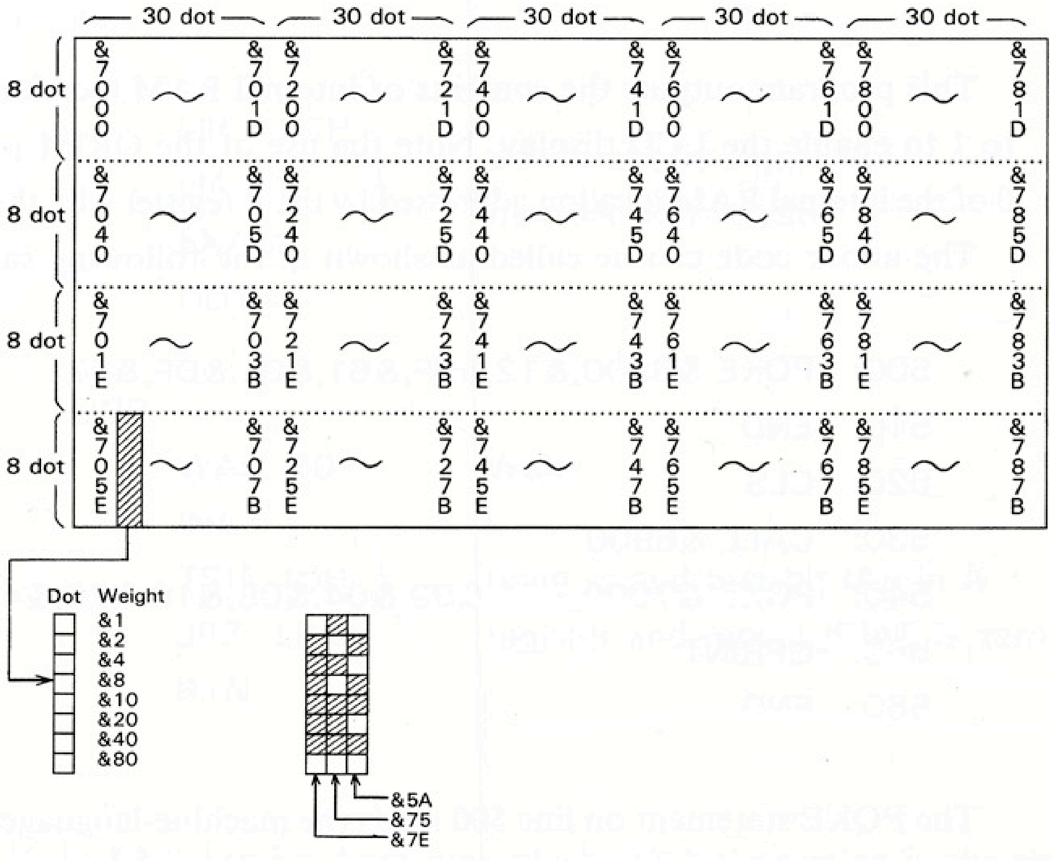


Fig. 7 LCD Video RAM Map

The LCD display is controlled by the lowest order bit of internal RAM location 5F in hexadecimal. Turning on bit 0 of address 5F enables the display and turning off bit 0 disables the display. The program given below turns on the LCD display.

Addr	Machine Code	OP Code	Operand(s)
6900	125F	LIP	5F
6902	6101	ORIM	1
6904	DF	OUTC	
6905	37	RTN	

Table 12

This program outputs the contents of internal RAM location 5F with its bit 0 set to 1 to enable the LCD display. Note the use of the ORIM instruction. It sets bit 0 of the internal RAM location addressed by the P register with the other bits left intact.

The above code can be called as shown in the following sample program:

```
500: POKE &6900,&12,&5F,&61,&01,&DF,&37
510: END
520: CLS
530: CALL &6900
540: POKE &7000,&01,&02,&04,&08,&10,&20,&40,&80,&00,&FF
550: GPRINT
560: END
```

The POKE statement on line 500 loads the machine-language program in memory at address 6900 in hexadecimal. The CALL statement on line 530 calls the machine-language program to enable the LCD display. The data to be displayed on the screen is placed on the screen by the POKE statement on line 540. The GPRINT statement on line 550 turns on the screen accordingly.

The Keyboard

The PC-1350 keyboard has two groups of keys. One group of keys are scanned and read via the input/output lines IA1 through IA8 (the keys that form a triangle in the key matrix diagram given at the end of this manual). The other group of keys are scanned by the scan signals K01 through K07 that are sent from an I/O port under program control, and read from lines IA1 through IA7 (see the key matrix diagram).

Key data from the first key group can be read by first sending the contents of the CPU IA port register at internal RAM address 5CH as a strobe signal and taking in the contents of the same IA port into the A register with an INA instruction. For example, the following machine-language code can be used to read only the ENT key data:

```
          LIP     5CH
          LIA     8          Sends strobe signal IA4 from IA port register.
          EXAM
          OUTA
LOOP
          WAIT    30         Wait.
          INA
          TSIA    10H         Read in and test bit IA5 in A
          JRZ     LOOP       register and repeat if IA5 is zero.
          RTN
```

If the ENT key has been pressed, the INA instruction in the above code should place code 18 in hexadecimal in the A register. This is because the INA instruction reads in bit IA4 that has been set as a strobe signal. Also note that bit 8 (most significant bit) of the IA port at address 5CH corresponds to IA8 (most significant bit) of the CPU A register and that bit 1 (least significant bit) of the IA port corresponds to IA1 of the CPU A register.

The second group of keys are scanned by strobe signals K01 through K07. These signals are generated by writing appropriate scan data into the key port that is located in the RAM are a between 7E00H through 7FFFH. Keyed in data can be read by taking in the contents of the IA port that are connected to lines IA1 through IA8 into the A register using an INA instruction.

Since the key port address is duplicated in memory addresses between 7E00H through 7FFFH, you can specify any address within this area as the key port. The relationship between the strobe data bits and the lines K01 through K07 is shown below.

	MSB			LSB
Data bits	Bit 8	Bit 7	◄——►	Bit 1
K01-K07 bits		K07	◄——►	K01

Here are sample programs for generating scan signals.

Example 1: Generating K02

```
LIDP    7E00H
LIA     02H
STD
```

Example 2: Generating K06

```
LIDP    7F00H
LIA     20H
STD
```

The Serial Interface

The PC-1350 has one serial interface as a serial port. It uses asynchronous (start/stop) communication and supports only the half-duplex mode. The major specifications of the PC-1350 serial interface are given below.

1. Communication mode:	Start-stop system, Half-duplex mode
2. Baud rate*:	300, 600, 1200 bauds
3. Data length *:	7 and 8 bits
4. Parity*:	Odd, even, none
5. Stop bits*:	1 or 2
6. Connector:	Dedicated 15-pin connector (see figure below.)
7. Output level:	C-MOS level (4 to 6 volts)
8. Interface signals:	Input: RD, CD, CD Output: SD, RS, RR, ER Others: SG, (FG), VC

*: Items marked by an asterisk are software programmable.

(1) Serial Interface Connector

The PC-1350 is furnished with a 15-pin connector for the serial interface. It is located on the right side panel of the main unit. The serial interface connector is shown below.

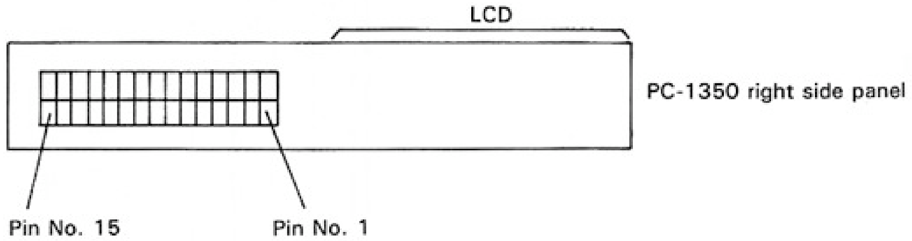


Fig. 8 Serial Interface Connector

The PC-1350 uses eight pins out of the 15 pins. The pin assignments and their definitions are summarized in the table below.

No.	Signal Name	Symbol	Direction	Description
1	Frame Ground	FG		Protective ground.
2	Transmit Data	SD	Output	Output data.
3	Receive Data	RD	Input	Input data.
4	Request to Send	RS	Output	Set on in the transmit mode and off in the receive mode.
5	Clear to Send	CS	Input	Response signal to send request. A 1 indicates that the PC-1350 can send data.
7	Signal Ground	SG		Signal ground. Gives the reference voltage for the interface signals.
8	Carrier Detect	CD	Input	A 1 enables the PC-1350 for reception and a 0 disables the PC-1 350 for reception.
11	Receive Ready	RR	Output	When set to 1, indicates that the PC-1350 is ready for reception.
14	Equipment Ready	ER	Output	When the serial port is selected, set on to indicate that the PC-1350 is ready for communication.
10&13		VC		Power source.

Table 13

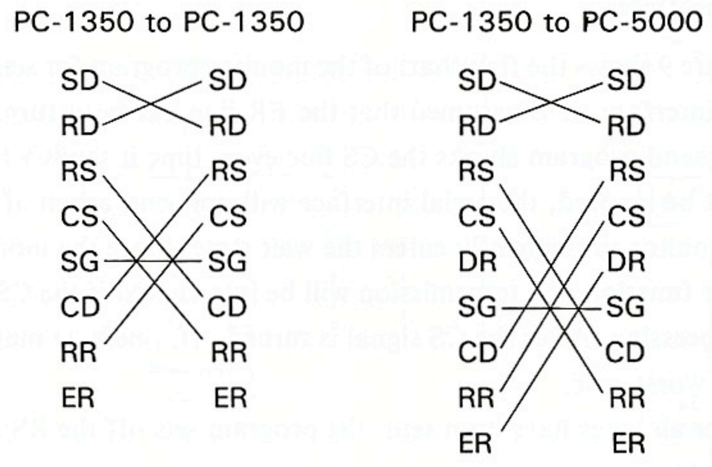
Notes:

1. Other pins are not connected inside the PC-1350.
2. Pins are at the VC level when they are set on and at the SG level when they are off.
3. Applying a source voltage beyond the permissible range (i.e., voltage difference across VC and SG) may cause damage to the PC-1350 electronics as it is made up of C-MOS components. Take extreme care when connecting the connector to an external device.

(2) Connection

• Connecting a PC-1350 to another PC-1350 or a Sharp PC-5000

The wiring diagrams for connecting your PC-l350 to another PC-1350 and a Sharp PC-5000 are shown below.



Caution: A voltage level shifter is required when connecting your PC-1350 to a terminal device other than PC-1350 (e.g., PC-5000). An attempt to connect the PC-l350 to such a device may damage the PC-l350 interior.

(3) Programming the Serial Interface

Before communicating with an external device through the serial interface, it is necessary to execute an OPEN statement. Executing an OPEN statement sets the ER line to ON. The ER line remains on until a CLOSE statement is executed.

• Sending Data

Figure 9 shows the flowchart of the monitor program for sending data from the serial interface. It is assumed that the ER line has been turned on.

The send program checks the CS line every time it sends a byte. The CS signal cannot be ignored; the serial interface will not send a byte if the CS line is off. The monitor subsequently enters the wait state. Since the monitor does not have a timer function, the transmission will be interrupted if the CS line is set off during processing. After the CS signal is turned off, one byte may probably be sent, in the worst case.

After all bytes have been sent, the program sets off the RS line and terminates processing.

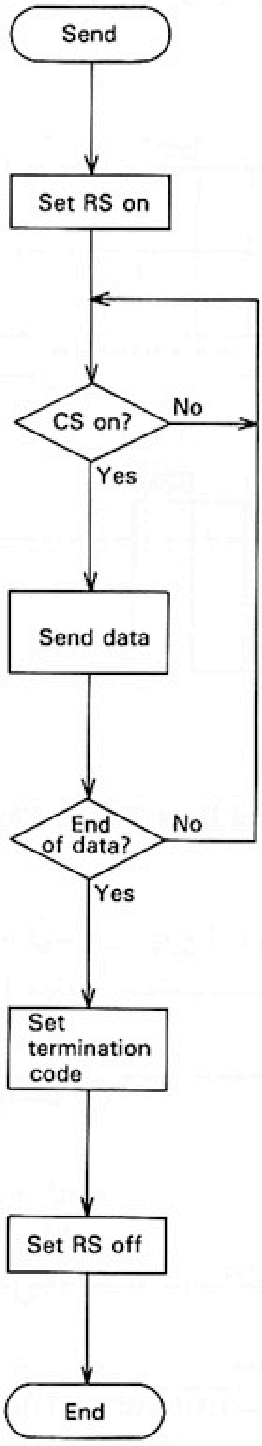
• Receiving Data

Figure 10 shows the flowchart of the monitor program for receiving data at the serial interface. It is assumed that the ER line has been turned on.

The receive program checks the CD tine every time it receives a byte. The CD signal cannot be ignored. Since the monitor does not have a timer function, the receive processing will be interrupted if the CD line is set off during processing. The monitor subsequently enters the wait state.

The program turns off the RR line to signal the termination of receive processing to the counterpart device when it receives a termination code or when it cannot continue processing due to an error such as parity or framing error.

In either send or receive mode, one device is put into the wait state if the other device terminates processing. The device that is held in the wait state can be reset by pressing the BREAK key.



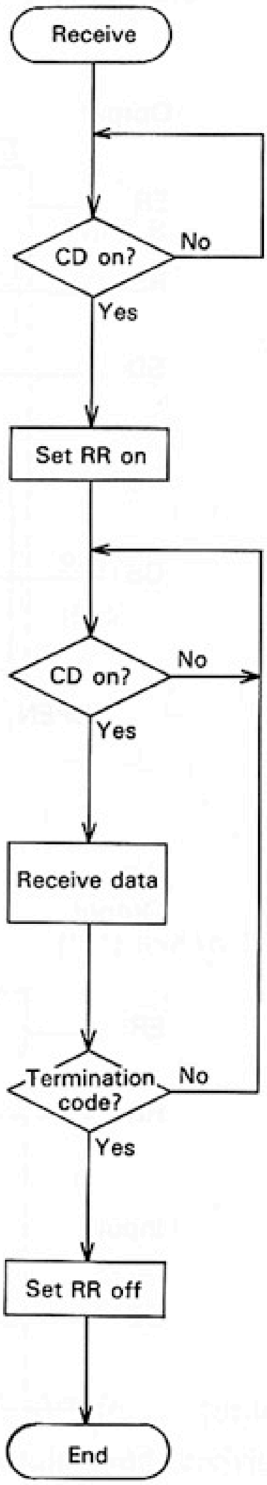


Fig. 9 Send Processing

Fig. 10 Receive Processing

The send and receive timing charts are shown in Figures 11 and 12.

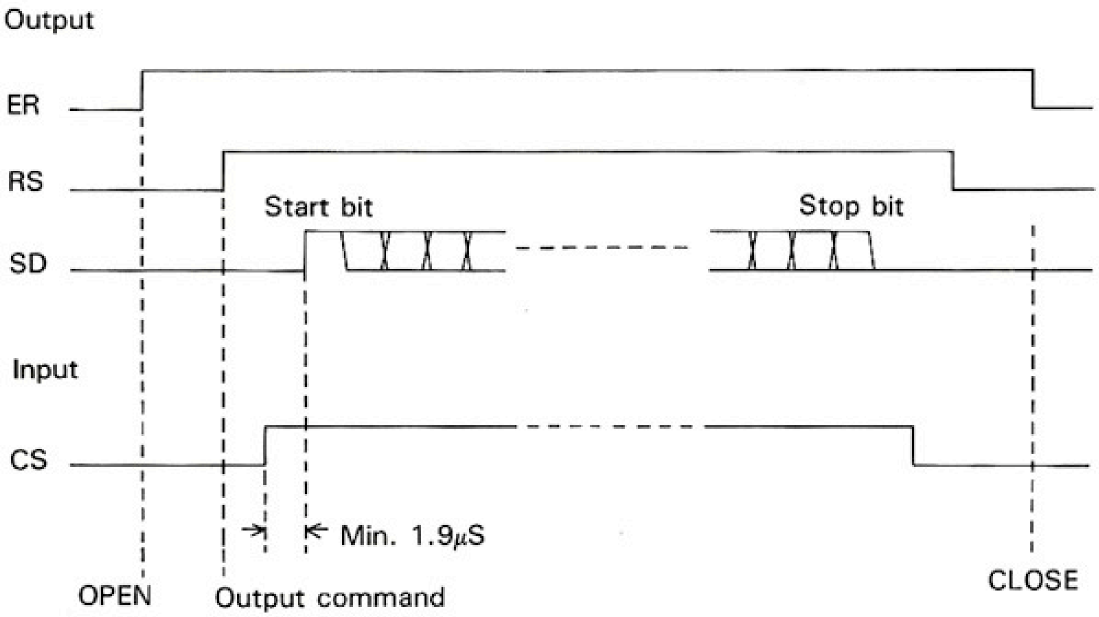


Fig. 11 Send Data Timing Chart

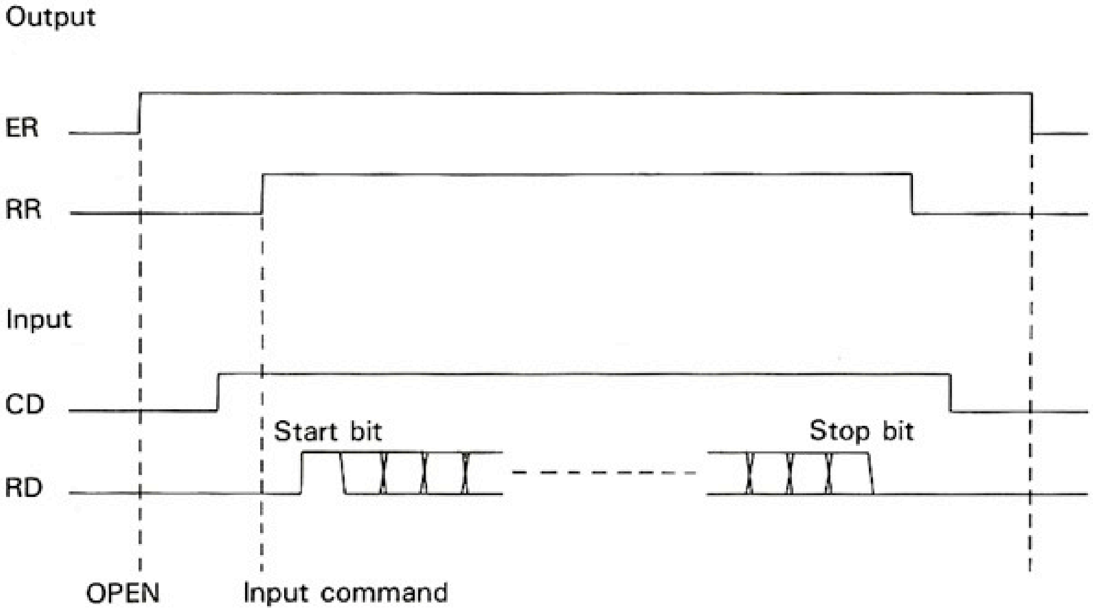


Fig.12 Receive Data Timing Chart

Figure 13 shows the relationship between the serial interface signals and the bit definitions of the I/O ports B and F in the CPU internal RAM.

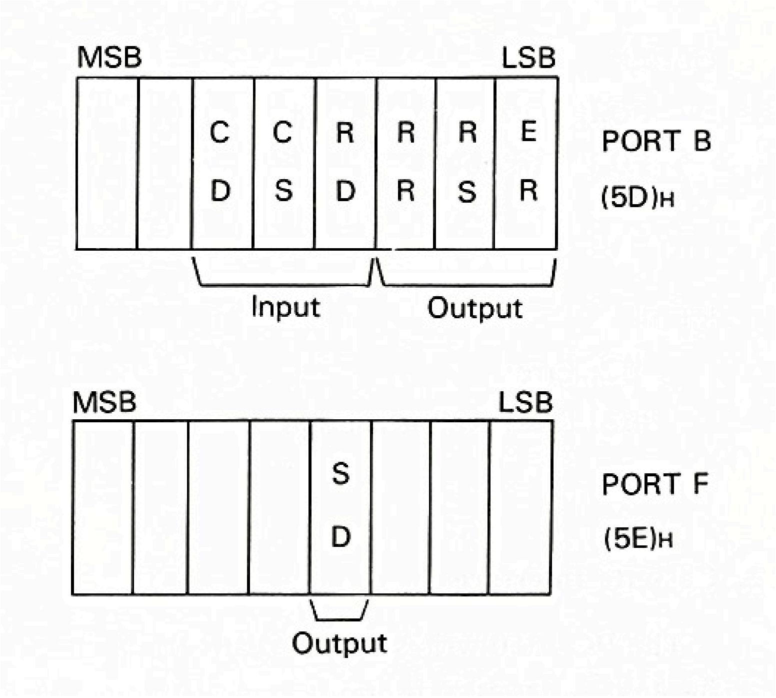


Fig. 13

The following sample programs set the send data (SD) line to 1 (low level) and 0 (high level) :

```
Setting SD high          Setting SD low
LIP     5EH              LIP     5EH
ORIM    8                ANIM    F7H
OUTF                     OUTF
```

Note: Refer to the instructions manual for the PC-1350 for the formats of the send and receive data and the procedures for sending and receiving data via the serial interface using the BASIC LPRINT, PRINT # 1, and LLIST statements.

Other I/O Interfaces

1. Printer Interface

The following I/O ports are used to interface to the optional thermal printer Model CE-126P:

Output
F02: Data
F03: Strobe

Input
IB7: Error
IB8: Data Acknowledge

2. Cassette Interface

The PC-1350 writes to and reads from the cassette recorder through the XIN (input) and XOUT (output) terminals.

3. Programming Note

When controlling I/O port register F in the CPU internal RAM with the OUTF instruction, be sure to set its bit 0 to 0.

THE PC-1350 INSTRUCTIONS LIST

There are 123 machine language instructions for the PC-1350 included here. The instructions may occur in the following sizes and formats:

1. 1 byte instructions
A. 8 bit operation code
B. 2 bit operation code, 6 bit operand

2. 2 byte instructions
A. 8 bit operation code, 8 bit operand
B. 7 bit operation code, 9 bit operand
C. 3 bit operation code, 13 bit operand

3. 3 byte instructions-8 bit operation code, 16 bit operand
4. More than 3 byte instructions-8 bit operation code 3 or more byte operand

Detailed information about each instruction is given in this section of the manual.

At the end of the section, the summarized information is listed alphabetically and by hexadecimal operation code. In the instruction detail list, the instructions are grouped by similarity of function and the following information is provided for each:

1. Machine Language mnemonic code
2. A description and diagram of the actions performed by the instruction
3. The number of bytes required
4. The number of bits in the operation code and the operand(s)
5. The appearance of the instruction in memory in binary and hexadecimal
6. Cycles - The number of machine cycles required for execution of the instruction. This information can be used to select the faster instructions if more than one instruction or set of instructions could be used to obtain the desired results.
7. Flags - Indicates whether the execution of the instruction will affect the Carry (C) or the Zero (Z) flag.
8. Other - Indicates other changes that may occur during the execution of the instruction, such as changes in the contents of registers.

Abbreviations used in the instruction descriptions

Registers

A	Accumulator
B	Extra Accumulator
DP	Data Pointer-External RAM address
I	Block Operation Register
J	Block Operation Register
K	General Purpose-For programmers use
L	General Purpose-For programmers use
M	General Purpose-For programmers use
N	General Purpose-For programmers use
P	Internal RAM address pointer
PC	Program Counter
Q	Internal RAM address pointer
R	Stack Pointer
X	External RAM address pointer
Y	External RAM address pointer

If the letter appears as above, the contents of the register is being specified.

If the letter appears within parentheses, the contents of the memory address that is stored in the register is being specified.

Operands

ℓ	6 bit literal = an address in internal RAM between 00 and 3F.
n	7 bit literal value (for registers P and Q) 8 bit literal value (for other registers)
nm	16 bit literal value

1. Move Data Instructions

1.1. Load Immediate

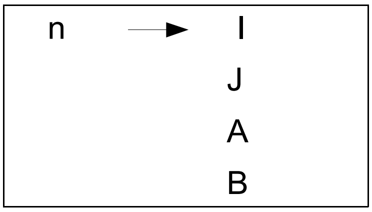
The value represented by the operand (n,m,l) is moved into the specified register.

LIr n

Load the value of n into r(egister)

r = an 8 bits register

n = 8 bits



	Mnem	Appearance in Memory								Byte
Op Code	LIr	0	0	0	0	0	0	←r→		1
Operand	n	←			n				→	2
		8	7	6	5	4	3	2	1	

if r =	2 Low Bits of Op Code		Hex Op Code
I	0	0	00
J	0	1	01
A	1	0	02
B	1	1	03
	2	1	

Cycles 4

Flags None

Other None

LIr n

Load the value of n into r(egister)

r = a 7 bits internal RAM address register

n = 7 bits

n → P
Q

	Mnem		Appearence in Memory								Byte
Op Code	LIr	→	0	0	0	1	0	0	1	r	1
Operand	n	→	•	◄			n			►	2
			8	7	6	5	4	3	2	1	

if r =	Low Bits of Op Code	Hex Op Code
P	0	12
Q	1	13
	1	

Cycles 4

Flags None

Other None

LIDP nm

Load the value of nm into 16 bit DP register

nm = 16 bits

n → DP_H m → DP_L

	Mnem		Appearence in Memory								Op Code	Byte
Op Code	LIr	→	0	0	0	1	0	0	0	0	10	1
Operand	n	→	◄			n				►		2
	m	→	◄			m				►		3
			8	7	6	5	4	3	2	1		

Cycles 8

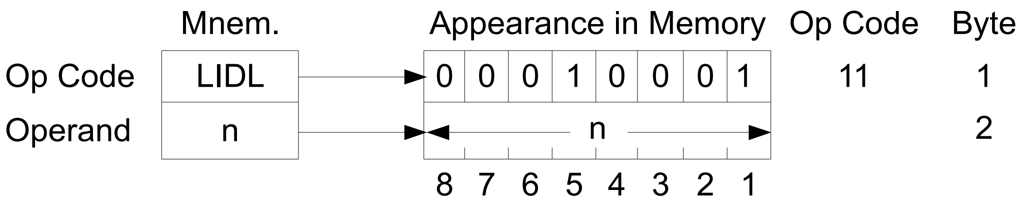
Flags None

Other None

LIDL n

Load the value of n into the low order byte of the DP register

$\boxed{n \rightarrow DP_L}$



Cycles 5

Flags None

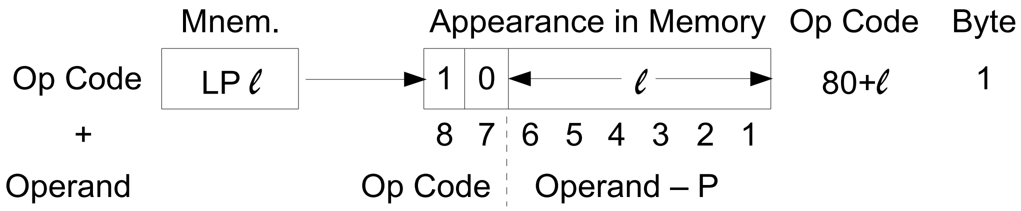
Other None

LP ℓ

Load the value ℓ onto P register

ℓ = 6 bits (00 – 3F)

$\boxed{\ell \rightarrow P}$



Cycles 2

Flags None

Other None

1.2. Load/Store a register into/from the accumulator

LDr

Load the contents of r(egister) into the accumulator.
(register A).
r = a 7 bit internal RAM address register

P → A
Q
R

	Mnem.	Appearance in Memory								Byte
Op Code	LDr	0	0	1	0	0	0	←r→		1
		8	7	6	5	4	3	2	1	

if r =	2 Low Bits of Op Code		Hex Op Code
P	0	0	20
Q	0	1	21
R	1	0	22
	2	1	

Cycles 2
Flags None
Other None

STr

Store the contents of the accumulator (register A) into r(egister)

r = an 7 bits internal RAM address register

A	→	P
		Q
		R

	Mnem		Appearance in Memory								Byte
Op Code	STr	→	0	0	1	1	0	0	←r→		1
			8	7	6	5	4	3	2	1	

if r =	2 Low Bits of Op Code		Hex Op Code
P	0	0	30
Q	0	1	31
R	1	0	32
	2	1	

Cycles	2
Flags	None
Other	None

1.3. Move data between memory and the accumulator

Move the contents of the accumulator or the address in a register to/from the address in a register of the accumulator.

LDM

Load the contents of the address in the P register into the accumulator.

(P) → A

	Mnem.		Appearance in Memory								Op Code	Byte
Op Code	LDM	→	0	1	0	1	1	0	0	1	59	1
			8	7	6	5	4	3	2	1		

Cycles	2
Flags	None
Other	None

LDD

Load the contents of the address in the DP register into the accumulator.

$\boxed{(DP) \rightarrow A}$

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	LDD	0	1	0	1	0	1	1	1	57	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	None
Other	None

STD

Store the contents of the accumulator in the address in the DP register.

$\boxed{A \rightarrow (DP)}$

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	STD	0	1	0	1	0	0	1	0	52	1
		8	7	6	5	4	3	2	1		

Cycles	2
Flags	None
Other	None

1.4. Move data from one memory address to another

Move the contents of the address in a register to the address in another register.

MVMD

Move the contents of the DP register address to the P register address.

$\boxed{(DP) \rightarrow (P)}$

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	MVMD →	0	1	0	1	0	1	0	1	55	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	None
Other	None

MVDM

Move the contents of the P register address to the DP register address.

$\boxed{(P) \rightarrow (DP)}$

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	MVDM →	0	1	0	1	0	0	1	1	53	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	None
Other	None

1.5. Exchange data between two registers

Exchange the data in the accumulator with that in the address in the DP register or register B.

EXAM

Exchange the contents of the address in register P with the contents of the accumulator.

$\boxed{A \leftrightarrow (P)}$

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	EXAM	1	1	0	1	1	0	1	1	DB	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	None
Other	None

EXAB

Exchange the contents of register B with the contents of the accumulator.

$\boxed{A \leftrightarrow B}$

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	EXAB	1	1	0	1	1	0	1	0	DA	1
		8	7	6	5	4	3	2	1		

Cycles	5
Flags	None
Other	None

1.6. Block move of data in memory

Move the contents of one or more bytes of memory to another area of memory.

MVW

MVB

Move the contents of d+1 bytes starting with the address in the Q register into the d+1 bytes starting with the address in register P.

d must be stored in register I or J

if d = 0, 1 byte will be moved

```
 I   ──►  d
 J

REPEAT
   (Q) → (P), P+1, Q+1, d–1
UNTIL d = FF
```

	Mnem.	Appearance in Memory	Op Code	Byte
If	MVW	0 0 0 0 1 0 0 0	08	1
		8 7 6 5 4 3 2 1		

Then d = the value stored in register I

	Mnem.	Appearance in Memory	Op Code	Byte
If	MVB	0 0 0 0 1 0 1 0	0A	1
		8 7 6 5 4 3 2 1		

Then d = the value stored in register J

Cycles	5 + 2d
Flags	None
Other	P and Q registers are incremented

MVWD

MVBD

Move the contents of d+1 bytes starting with address in the DP register into the d+1 bytes starting with the address in the P register.

d must be stored in register I or J

if d=0, 1 byte will be moved

```
 I    ──►   d
 J

REPEAT
   (DP) → (P), DP+1, P+1, d–1
UNTIL d = FF
```

	Mnem.	Appearance in Memory								Op Code	Byte
If	MVWD	0	0	0	1	1	0	0	0	18	1
		8	7	6	5	4	3	2	1		

Then d = the value stored in register I

	Mnem.	Appearance in Memory								Op Code	Byte
If	MVBD	0	0	0	1	1	0	1	0	1A	1
		8	7	6	5	4	3	2	1		

Then d = the value stored in register J

Cycles	5 + 2d
Flags	None
Other	P and DP registers are incremented

1.7. Block exchange of data in memory

Exchange the contents of one or more bytes of memory with the contents of the same number of bytes in another area of memory

EXW

EXB

Exchange the contents of d+1 bytes starting with the address in the Q register with the contents of the d+1 bytes starting with the address of the P register.

d must be stored in register I or J

if d=0, 1 byte will be exchanged

```
 I    ──►  d
 J

REPEAT
   (P) ↔ (Q), P+1, Q+1, d–1
UNTIL d = FF
```

	Mnem.	Appearance in Memory								Op Code	Byte
If	EXW	0	0	0	0	1	0	0	1	09	1
		8	7	6	5	4	3	2	1		

Then d = the value stored in register I

	Mnem.	Appearance in Memory								Op Code	Byte
If	EXB	0	0	0	0	1	0	1	1	0B	1
		8	7	6	5	4	3	2	1		

Then d = the value stored in register J

Cycles	6 + 3d
Flags	None
Other	P and Q registers are incremented

EXWD

EXBD

Exchange the contents of d+1 bytes starting with address in the DP register into the d+1 bytes starting with the address in the P register.

d must be stored in register I or J

if d=0, 1 byte will be moved

```
 I    ──►   d
 J


REPEAT
   (DP) ↔ (P), DP+1, P+1, d–1
UNTIL d = FF
```

	Mnem.		Appearance in Memory								Op Code	Byte
If	EXWD	──►	0	0	0	1	1	0	0	1	19	1
			8	7	6	5	4	3	2	1		

Then d = the value stored in register I

	Mnem.		Appearance in Memory								Op Code	Byte
If	EXBD	──►	0	0	0	1	1	0	1	1	1B	1
			8	7	6	5	4	3	2	1		

Then d = the value stored in register J

Cycles	7 + 6d
Flags	None
Other	P and DP registers are incremented

1.8. Increment or decrement a register

Add or subtract 1 from the contents of the register specified by the instruction.

INCP

Add 1 to the contents of register P.

$$\boxed{P +1 \rightarrow P}$$

	Mnem.		Appearance in Memory								Op Code	Byte
Op Code	INCP	→	0	1	0	1	0	0	0	0	50	1
			8	7	6	5	4	3	2	1		

Cycles	2
Flags	None
Other	None

DECP

Substrac 1 to the contents of register P.

$$\boxed{P -1 \rightarrow P}$$

	Mnem.		Appearance in Memory								Op Code	Byte
Op Code	DECP	→	0	1	0	1	0	0	0	1	51	1
			8	7	6	5	4	3	2	1		

Cycles	2
Flags	None
Other	None

INCr

Increment the contents of r(egister) by 1.

I	+1 →	I_1	C,Z
J		J_1	
A		A_1	
B		B_1	
K		K_1	
L		L_1	
M		M_1	
N		N_1	

	Mnem.		Appearance in Memory	Byte
Op Code	INCr	→	← See below →	1
			8 7 6 5 4 3 2 1	

Cycles	4
Flags	C, Z
Other	Contents of Q register change

if r =

	Mnem.		Appearance in Memory								Op Code
I	INCI	→	0	1	0	0	0	0	0	0	40
J	INCJ	→	1	1	0	0	0	0	0	0	C0
A	INCA	→	0	1	0	0	0	0	1	0	42
B	INCB	→	1	1	0	0	0	0	1	0	C2
K	INCK	→	0	1	0	0	1	0	0	0	48
L	INCL	→	1	1	0	0	1	0	0	0	C8
M	INCM	→	0	1	0	0	1	0	1	0	4A
N	INCN	→	1	1	0	0	1	0	1	0	CA
			8	7	6	5	4	3	2	1	

DECr

Decrement the contents of r(egister) by 1.

I	–1	→	I_1	C, Z
J			J_1	
A			A_1	
B			B_1	
K			K_1	
L			L_1	
M			M_1	
N			N_1	

	Mnem.		Appearance in Memory	Byte
Op Code	DECr	→	◄— See below —►	1
			8 7 6 5 4 3 2 1	

Cycles 4

Flags C, Z

Other Q register changes

if r =

	Mnem.		Appearance in Memory								Op Code
I	DECI	→	0	1	0	0	0	0	0	1	41
J	DECJ	→	1	1	0	0	0	0	0	1	C1
A	DECA	→	0	1	0	0	0	0	1	1	43
B	DECB	→	1	1	0	0	0	0	1	1	C3
K	DECK	→	0	1	0	0	1	0	0	1	49
L	DECL	→	1	1	0	0	1	0	0	1	C9
M	DECM	→	0	1	0	0	1	0	1	1	4B
N	DECN	→	1	1	0	0	1	0	1	1	CB
			8	7	6	5	4	3	2	1	

1.9. Increment or decrement an external memory address register and move the address from register to DP register

Add or subtract 1 to or from the address register X or Y, move the contents of X or Y to the DP register.

Ir

Add 1 to the memory address in r(egister) and store the incremented address in the DP register

X → DP
Y
DP+1 → DP, X Y

if r =

	Mnem.	Appearance in Memory								Op Code	Byte
X	IX	0	0	0	0	0	1	0	0	04	1
		8	7	6	5	4	3	2	1		

	Mnem.	Appearance in Memory								Op Code	Byte
Y	IY	0	0	0	0	0	1	1	0	06	1
		8	7	6	5	4	3	2	1		

Cycles	6
Flags	None
Other	Q register changes

Dr

Subtract 1 to the memory address in r(egister) and store the decremented address in the DP register

X → DP
Y
DP–1 → DP, X Y

if r =

	Mnem.	Appearance in Memory								Op Code	Byte
X	DX	0	0	0	0	0	1	0	1	05	1
		8	7	6	5	4	3	2	1		

	Mnem.	Appearance in Memory								Op Code	Byte
Y	DY	0	0	0	0	0	1	1	1	07	1
		8	7	6	5	4	3	2	1		

Cycles 6
Flags None
Other Q register changes

1.10. Increment or decrement register X and load the contents of the register X address into the accumulator

Add or subtract 1 from or to the address in register X, load the new address into the DP register and load the contents of new address into the accumulator.

IXL

X → DP	a. Add 1 to the address in register X
DP+1 → DP, X	b. Load the incremented address into the DP register.
(DP) → A	c. Move the contents of the address in the DP register into the accumulator

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	IXL	0	0	1	0	0	1	0	0	24	1
		8	7	6	5	4	3	2	1		

Cycles 7
Flags none
Other Q register changes

DXL

X ⟶ DP	a. subtract 1 to the address in register X
DP–1 → DP, X	b. Load the decremented address into the DP register.
(DP) → A	c. Move the contents of the address in the DP register into the accumulator

	Mnem.		Appearance in Memory	Op Code	Byte
Op Code	DXL	→	0 0 1 0 0 1 0 1	25	1
			8 7 6 5 4 3 2 1		

Cycles	7
Flags	none
Other	Q register changes

1.11. Increment or decrement register Y and store the contents of the accumulator register into the address in the Y register

Add or subtract 1 from or to the address in register Y, load the new address into the DP register and store the contents of the accumulator into the new address.

IYS

Y → DP	a. Add 1 to the address in register Y
DP+1 → DP, Y	b. Load the incremented address into the DP register.
A → (DP)	c. Move the contents of the accumulator into the DP register address.

	Mnem.		Appearance in Memory	Op Code	Byte
Op Code	IYS	→	0 0 1 0 0 1 1 0	26	1
			8 7 6 5 4 3 2 1		

Cycles	6
Flags	none
Other	Q register changes

DYS

Y → DP	a. Subtract 1 from the address in register Y
DP–1 → DP, Y	b. Load the decremented address into the DP register.
A → (DP)	c. Move the contents of the accumulator into the DP register address.

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	DYS	0	0	1	0	0	1	1	1	27	1
		8	7	6	5	4	3	2	1		

Cycles	6
Flags	none
Other	Q register changes

1.12. Fill a block of memory with a single value

Fill either an internal RAM or an external memory block with the value in the accumulator.

FILM

Store the value in the accumulator into the d+1 bytes of the internal RAM starting with the address in the P register.

d must be stored in register I

if d=0, one byte will be filled

```
  I → d
REPEAT
   A → (P), P+1, d–1
UNTIL d = FF
```

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	FILM	0	0	0	1	1	1	1	0	1E	1
		8	7	6	5	4	3	2	1		

Cycles	5+d
Flags	none
Other	P register changes

FILD

Store the value in the accumulator into the d+1 bytes of the external RAM starting with the address in the DP register.

d must be stored in register I

if d=0, one byte will be filled

```
  I → d
REPEAT
   A → (DP), DP+1, d–1
UNTIL d = FF
```

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	FILD	0	0	0	1	1	1	1	1	1F	1
		8	7	6	5	4	3	2	1		

Cycles	4+3d
Flags	none
Other	DP register changes

2. Arithmetic, Logical and Shift Instructions

2.1. Add/Subtract Immediate, Accumulator

Add or subtract the value n to/from the accumulator.

ADIA n

Add the value n to the accumulator

A + n → A	C,Z

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	ADIA	0 1 1 1 0 1 0 0	74	1
Operand	n	◄— n —►		2
		8 7 6 5 4 3 2 1		

Cycles	4
Flags	C, Z
Other	none

SBIA n

Subtract the value n from the accumulator

A – n → A	C,Z

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	SBIA	0 1 1 1 0 1 0 1	75	1
Operand	n	◄— n —►		2
		8 7 6 5 4 3 2 1		

Cycles	4
Flags	C, Z
Other	none

2.2. Add/Subtract Immediate, Memory

Add or subtract the value n to/from the register P memory address.

ADIM n

Add the value n to the register P memory address

(P) + n → (P)   C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	ADIM	0	1	1	1	0	0	0	0	70	1
Operand	n	n									2
		8	7	6	5	4	3	2	1		

Cycles	4
Flags	C, Z
Other	none

SBIM n

Subtract the value n to the register P memory address

(P) – n → (P)   C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	SBIM	0	1	1	1	0	0	0	1	71	1
Operand	n	n									2
		8	7	6	5	4	3	2	1		

Cycles	4
Flags	C, Z
Other	none

2.3. Byte Binary Addition or Subtraction

Add or subtract the contents of the address in the P register to/from the accumulator and store the results in the address in the P register.

ADM

Add the contents of the address in the P register to the accumulator and store the results in the address in the P register.

(P) + A → (P)   C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	ADM	0	1	0	0	0	1	0	0	44	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	C, Z
Other	none

SBM

Subtract the contents of the address in the P register from the accumulator and store the results in the address in the P register.

(P) – A → (P)   C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	SBM	0	1	0	0	0	1	0	1	45	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	C, Z
Other	none

2.4. Byte Binary Addition or Subtraction with carry

Add or subtract the contents of the address in the P register to/from the accumulator with carry, and store the results in the address in the P register.

ADCM

Add the contents of the address in the P register and the carry to the accumulator and store the results in the address in the P register.

(P) + A + C → (P)   C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	ADCM	1	1	0	0	0	1	0	0	C4	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	C, Z
Other	none

SBCM

Subtract the contents of the address in the P register and the carry to the accumulator and store the results in the address in the P register.

(P) – A – C → (P)   C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	SBCM	1	1	0	0	0	1	0	1	C5	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	C, Z
Other	none

2.5. 2 Byte Binary Addition or Subtraction

Add or subtract the contents of the address in the P register to/from registers A and B (accumulator and spare register) and store the results in the address in the P register.

ADB

Add the contents of the address in the P register to register A and B and store the results in the address in the P register.

[P +1 , P] + [B,A] → [P + 1,P]   C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	ADB	0	0	0	1	0	1	0	0	14	1
		8	7	6	5	4	3	2	1		

Cycles	5
Flags	C, Z
Other	P register changes

SBB

Subtract the contents of the address in the P register from register A and B and store the results in the address in the P register.

[P + 1 , P] – [B,A] → [P + 1,P]   C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	SBB	0	0	0	1	0	1	0	1	15	1
		8	7	6	5	4	3	2	1		

Cycles	5
Flags	C, Z
Other	P register changes

2.6. 2 Block BCD Addition or Subtraction

Add or subtract d+1 bytes starting with the address in the P register to/from the accumulator or the address in the Q register

ADN

Add the contents of the accumulator to the block of d+1 bytes of memory starting with the address in the P register as the right-most digit.

the contents of I are stored in d

```
   I → d
REPEAT
   (P) + A → (P)  (BCD), P – 1, d – 1
UNTIL d = FF
```

	Mnem.		Appearance in Memory	Op Code	Byte
Op Code	ADN	→	0 0 0 0 1 1 0 0	0C	1
			8 7 6 5 4 3 2 1		

Cycles	7 + 3d
Flags	C, Z
Other	P register changes

SBN

subtract the contents of the accumulator from the block of d+1 bytes of memory starting with the address in the P register as the right-most digit.

the contents of I are stored in d

```
   I → d
REPEAT
   (P) – A → (P)  (BCD), P – 1, d – 1
UNTIL d = FF
```

	Mnem.		Appearance in Memory	Op Code	Byte
Op Code	SBN	→	0 0 0 0 1 1 0 1	0D	1
			8 7 6 5 4 3 2 1		

Cycles	7 + 3d
Flags	C, Z
Other	P register changes

ADW

Add the contents of the block of d+1 bytes of memory starting with the address in the Q register as its right-most (low order) digit to the d+1 bytes of memory starting with the address in the P register as the right-most digit.

the contents of I are stored in d

```
I → d
REPEAT
(P) + Q → (P)  (BCD), P – 1,  Q – 1, d – 1
UNTIL d = FF
```

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	ADW	0	0	0	0	1	1	1	0	0E	1
		8	7	6	5	4	3	2	1		

Cycles 7 + 3d
Flags C, Z
Other P & Q register change

SBW

subtract the contents of the block of d+1 bytes of memory starting with the address in the Q register as its right-most (low order) digit to the d+1 bytes of memory starting with the address in the P register as the right-most digit.

the contents of I are stored in d

```
I → d
REPEAT
(P) – Q → (P)  (BCD), P – 1,  Q – 1, d – 1
UNTIL d = FF
```

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	SBW	0	0	0	0	1	1	1	1	0F	1
		8	7	6	5	4	3	2	1		

Cycles 7 + 3d
Flags C, Z
Other P & Q register change

2.7. Block Shift 4 bits

Shift 4 bits of d+1 bytes starting with the address in the P register 4 bits to the right or left

SRW

Shift d+1 bytes 4 bits (one BCD digit) to the right starting with the address in the P register

the contents of I are stored in d

```
I → d
REPEAT
Shift P 4 bits to right, P + 1,  d – 1
UNTIL d = FF
```

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	SRW →	0 0 0 1 1 1 0 0	1C	1
		8 7 6 5 4 3 2 1		

Cycles	5 + d
Flags	none
Other	P register changes

SLW

Shift d+1 bytes 4 bits (one BCD digit) to the left starting with the address in the P register

the contents of I are stored in d

```
I → d
REPEAT
Shift P 4 bits to left, P – 1,  d – 1
UNTIL d = FF
```

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	SLW →	0 0 0 1 1 1 0 1	1E	1
		8 7 6 5 4 3 2 1		

Cycles	5 + d
Flags	none
Other	P register changes

2.8. Logical OR

OR the contents of the accumulator or a memory location with an immediate value, or the contents of an address with the accumulator.

ORIA n

OR the contents of the accumulator with the immediate value n.

A ∨ n → A Z

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	ORIA	0 1 1 0 0 1 0 1	65	1
Operand	n	◄— n —►		2
		8 7 6 5 4 3 2 1		

Cycles 4
Flags Z
Other None

ORIM n

OR the contents of the address in the P register with the immediate value n.

(P) ∨ n → (P) Z

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	ORIM	0 1 1 0 0 0 0 1	61	1
Operand	n	◄— n —►		2
		8 7 6 5 4 3 2 1		

Cycles 4
Flags Z
Other None

ORID n

OR the contents of the address in the DP register with the immediate value n.

(DP) ∨ n → (DP)   Z

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	ORID	1 1 0 1 0 1 0 1	D5	1
Operand	n	◄── n ──►		2
		8 7 6 5 4 3 2 1		

Cycles	6
Flags	Z
Other	R – 1 used for temporary storage

ORMA

OR the contents of the address in the P register with the contents of the accumulator.

(P) ∨ A → (P)   Z

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	ORMA	0 1 0 0 0 1 1 1	47	1
Operand	n	◄── n ──►		2
		8 7 6 5 4 3 2 1		

Cycles	3
Flags	Z
Other	None

2.9. Logical AND

AND the contents of the accumulator or a memory location with an immediate value, or the contents of an address with the accumulator.

ANIA n

AND the contents of the accumulator with the immediate value n.

A ∧ n → A   Z

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	ANIA	0 1 1 0 0 1 0 0	64	1
Operand	n	n		2
		8 7 6 5 4 3 2 1		

Cycles	4
Flags	Z
Other	None

AMIM n

AND the contents of the address in the P register with the immediate value n.

(P) ∧ n → (P)   Z

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	ANIM	0 1 1 0 0 0 0 0	60	1
Operand	n	n		2
		8 7 6 5 4 3 2 1		

Cycles	4
Flags	Z
Other	None

ANID n

AND the contents of the address in the DP register with the immediate value n.

$(DP) \wedge n \rightarrow (DP)$ Z

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	ANID	1 1 0 1 0 1 0 0	D4	1
Operand	n	← n →		2
		8 7 6 5 4 3 2 1		

Cycles	6
Flags	Z
Other	R – 1 used for temporary storage

ANMA

AND the contents of the address in the P register with the contents of the accumulator.

$(P) \wedge A \rightarrow (P)$ Z

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	ANMA	0 1 0 0 0 1 1 0	46	1
Operand	n	← n →		2
		8 7 6 5 4 3 2 1		

Cycles	3
Flags	Z
Other	None

2.10. Bit Text Immediate

AND the contents of the accumulator or a memory position with the value n, store the result in the Zero Flag.

TSIA n

AND the contents of the accumulator with the value n and store the result in the Zero Flag.

A ∧ n Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	TSIA	0	1	1	0	0	1	1	0	66	1
Operand	n	n									2
		8	7	6	5	4	3	2	1		

Cycles	4
Flags	Z
Other	None

TSIM n

AND the contents of the address in the P register with the value n and store the result in the Zero Flag.

(P) ∧ n Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	TSIM	0	1	1	0	0	0	1	0	62	1
Operand	n	n									2
		8	7	6	5	4	3	2	1		

Cycles	4
Flags	Z
Other	None

TSID n

AND the contents of the DP register address with the value n and store the result in the Zero Flag.

(DP) ∧ n	Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	TSID	1	1	0	1	0	1	1	0	D6	1
Operand	n	n									2
		8	7	6	5	4	3	2	1		

Cycles	6
Flags	Z
Other	R – 1 used for temporary storage

2.11. Compare Immediate

Compare the accumulator or a memory location with the immediate value n.

Compare the contents of internal memory with the accumulator.

CPIA n

Compare the accumulator with the immediate value n.

A – n	C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	CPIA	0	1	1	0	0	1	1	1	67	1
Operand	n	n									2
		8	7	6	5	4	3	2	1		

Cycles	4		
Flags	A < n	C = 1	Z =0
	A = n	C = 0	Z =1
	A > n	C = 0	Z =0
Other	None		

CPIM n

Compare the contents of the address in the P register with the value n.

(P) – n	C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	CPIM	0	1	1	0	0	0	1	1	63	1
Operand	n	n									2
		8	7	6	5	4	3	2	1		

Cycles	4		
Flags	(P) < n	C = 1	Z =0
	(P) = n	C = 0	Z =1
	(P) > n	C = 0	Z =0
Other	None		

CPMA n

Compare the contents of the address in the P register with the accumulator.

(P) – A	C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	CPMA	1	1	0	0	0	1	1	1	C7	1
		8	7	6	5	4	3	2	1		

Cycles	3		
Flags	(P) < A	C = 1	Z =0
	(P) = A	C = 0	Z =1
	(P) > A	C = 0	Z =0
Other	None		

SWAP

Exchange the contents of the 4 right-most bits of accumulator with the contents of the 4 left-most bits.

$A_{1-4} \leftrightarrow A_{5-8}$ C,Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	SWAP →	0	1	0	1	1	0	0	0	58	1
		8	7	6	5	4	3	2	1		

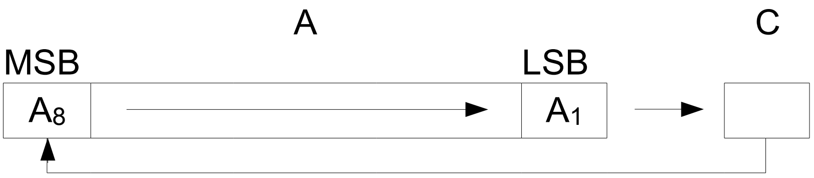
Cycles	2
Flags	None
Other	None

2.12. Shift Bits of a Byte

Shift the 8 bits of a byte 1 bit position to the right or left.

SR

Shift the 8 bits of a byte 1 bit position to the right. The original LSB goes to the Carry Flag, original contents of the Carry Flag goes to the byte's MSB.

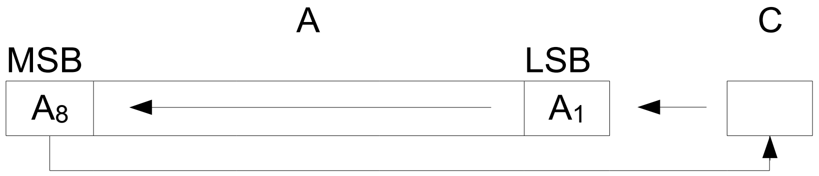


	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	SR →	1	1	0	1	0	0	1	0	D2	1
		8	7	6	5	4	3	2	1		

Cycles	2
Flags	C
Other	None

SL

Shift the 8 bits of a byte 1 bit position to the left. The original LSB goes to the Carry Flag, original contents of the Carry Flag goes to the byte's MSB.



	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	SL	0	1	0	1	1	0	1	0	5A	1
		8	7	6	5	4	3	2	1		

Cycles	2
Flags	C
Other	None

2.13. Set or Reset The Carry Flag

Set he Carry Flag to either 0 or 1

SC

store a 1 in the Carry Flag. Note that Zero Flag is set.

1 → C
1 → Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	SC	1	1	0	1	0	0	0	0	D0	1
		8	7	6	5	4	3	2	1		

Cycles	2
Flags	C, Z is set to 1
Other	None

RC

store a 0 in the Carry Flag. Note that Zero Flag is set.

0 → C
1 → Z

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	RC	1	1	0	1	0	0	0	1	D1	1
		8	7	6	5	4	3	2	1		

Cycles	2
Flags	C is set to 0 Z is set to 1
Other	None

3. Jump Instructions

Jump n bytes from the address of the Op Code in the plus or minus direction based on the stated conditions.

3.1. Jump Relative

JRcP n

Jump n bytes from the address of the Op Code in the P(lus) direction based on the stated conditions.

c = condition (see below)

IF condition = true	THEN PC + n + 1 → PC
IF condition = false	THEN PC + 2 → PC

	Mnem.	Appearance in Memory	Byte
Op Code	JRcP	see below	1
Operand	n	n	2
		8 7 6 5 4 3 2 1	

if c =

	Mnem.	8	7	6	5	4	3	2	1	Op Code
Z = 1	JRZP	0	0	1	1	1	0	0	0	38
Z = 0	JRNZP	0	0	1	0	1	0	0	0	28
C =1	JRCP	0	0	1	1	1	0	1	0	3A
C =0	JRNCP	0	0	1	0	1	0	1	0	2A
unconditional	JRP	0	0	1	0	1	1	0	0	2C

Cycles	7 if condition is met 4 if condition is not met
Flags	None
Other	None

JRcM n

Jump n bytes from the address of the Op Code in the M(inus) direction based on the stated conditions.

c = condition (see below)

IF condition = true	THEN PC + 1 - n → PC
IF condition = false	THEN PC + 2 → PC

	Mnem.	Appearance in Memory	Byte
Op Code	JRcM	see below	1
Operand	n	n	2
		8 7 6 5 4 3 2 1	

if c =

	Mnem.	Appearance in Memory								Op Code
Z = 1	JRZM	0	0	1	1	1	0	0	1	39
Z = 0	JRNZM	0	0	1	0	1	0	0	1	29
C =1	JRCM	0	0	1	1	1	0	1	1	3B
C =0	JRNCM	0	0	1	0	1	0	1	1	2B
unconditional	JRM	0	0	1	0	1	1	0	1	2D
		8	7	6	5	4	3	2	1	

Cycles	7 if condition is met 4 if condition is not met
Flags	None
Other	None

3.2. Jump Absolute

Jump to the absolute address nm based on the stated condition.

JPc

c = condition (see below)

IF condition = true	THEN n → PC_H	m → PC_L
IF condition = false	THEN PC + 3 → PC	

	Mnem	Appearence in Memory	Byte
Op Code	JPc	see below	1
Operand	n	n	2
Operand	m	m	3
		8 7 6 5 4 3 2 1	

if c =

	Mnem.	8	7	6	5	4	3	2	1	Op Code
Z = 1	JPZ	0	1	1	1	1	1	1	0	7E
Z = 0	JPNZ	0	0	1	1	1	1	0	0	7C
C =1	JPC	0	1	1	1	1	1	1	1	7F
C =0	JPNC	0	1	1	0	1	1	0	1	7D
unconditional	JP	0	1	1	0	1	0	0	1	79

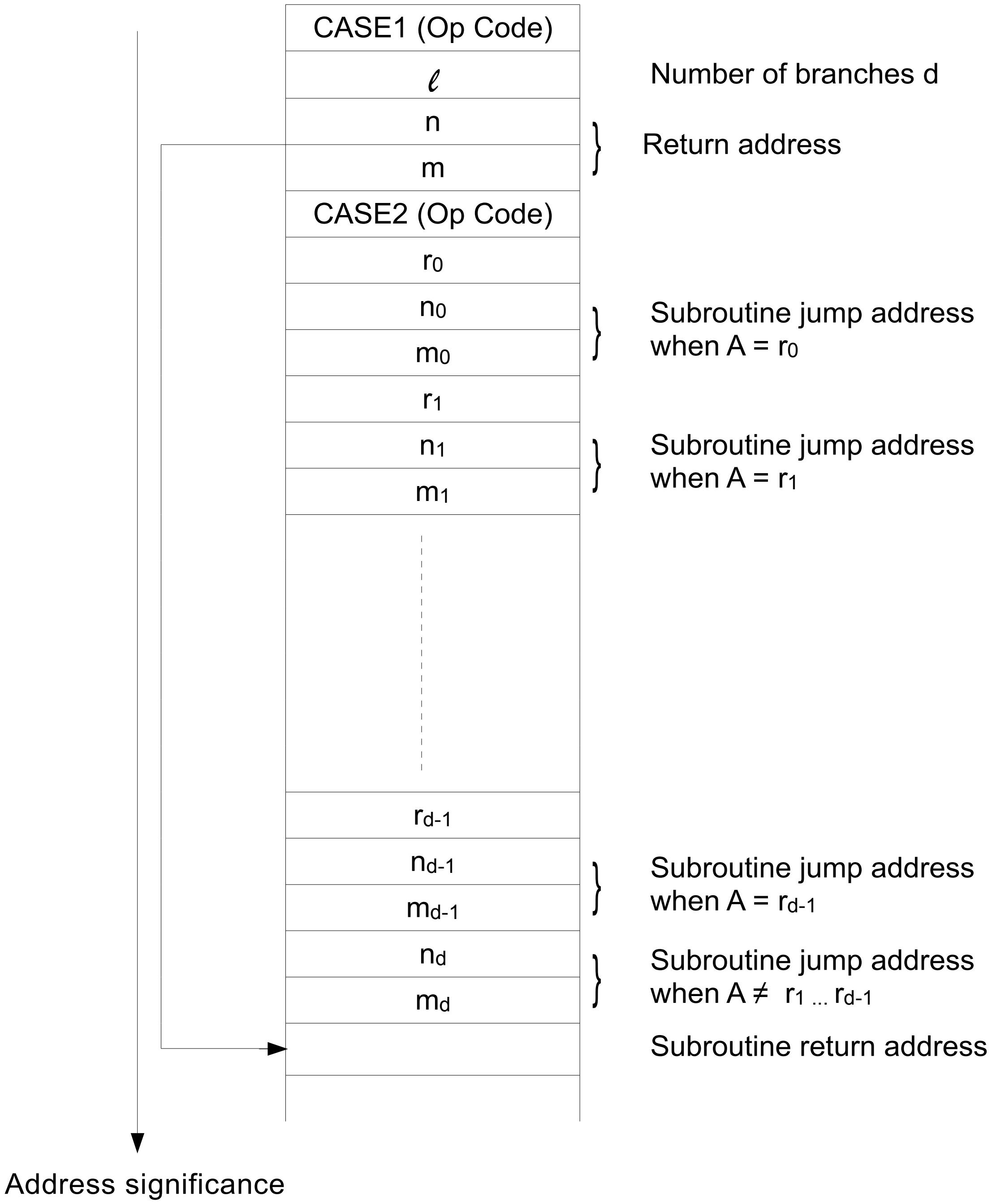
Appearance in Memory

Cycles	6
Flags	None
Other	None

3.3. CASE1 CASE2 This is a conditional branching instruction which compares the contents of register A with data following CASE2 and jump to a subroutine.

CASE1 CASE2

The memory configuration is shown below.


CASE1 (Op Code)
ℓ
n
m
CASE2 (Op Code)
r_0
n_0
m_0
r_1
n_1
m_1
r_{d-1}
n_{d-1}
m_{d-1}
n_d
m_d
Number of branches d
Return address
Subroutine jump address when A = r_0
Subroutine jump address when A = r_1
Subroutine jump address when A = r_{d-1}
Subroutine jump address when A ≠ $r_1 \ldots r_{d-1}$
Subroutine return address
Address significance

r_0 to r_{d-1} : Data compared with the contents of register A (1byte)

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	CASE1	0	1	1	1	1	0	1	0	7A	1
	CASE2	0	1	1	0	1	0	0	1	69	2
		8	7	6	5	4	3	2	1		

Cycles	8 (CASE1), 5 + 7d (CASE2)
Flags	None
Other	None
Total bytes	4 + 3d

4. Other Instructions

PUSH

Put the contents of the accumulator onto the stack.

R – 1 → R
A → (R)

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	PUSH	0	0	1	1	0	1	0	0	34	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	None
Other	Register R is decremented

POP

Pop the contents of the top of the stack into the accumulator.

(R) → A
R +1 → (R)

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	POP	0	1	0	1	1	0	1	1	5B	1
		8	7	6	5	4	3	2	1		

Cycles	2
Flags	None
Other	Register R is incremented

LOOP n

Decrement the top of the stack. If the Carry Flag equals 1, then execute the next instruction. If the Carry Flag equals 0, then make a relative jump from the address of the LOOP opcode to the address (PC+1+n). Note: One stack space is used for calculation.

```
 (R ) – 1 → R
  IF C = 0  THEN  PC + 1 – n → PC
  IF C = 1  THEN  PC + 2 → PC
```

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	LOOP	0 0 1 0 1 1 1 1	2F	1
Operand	n	◄── n ──►		2
		8 7 6 5 4 3 2 1		

Cycles	10 if C = 0
Flags	7 if C = 1
	C, Z
Other	1 stack space is used for calculation

LEAVE

Zero to top of stack.

Store a zero to the top of the stack.

```
 0 → (R)
```

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	LEAVE	1 1 0 1 1 0 0 0	D8	1
		8 7 6 5 4 3 2 1		

Cycles	2
Flags	None
Other	None

CAL In

Call subroutine. Store the address in the PC+2, the next command after the call on the stack. Jump to the absolute address In, an address in the first 8k bytes of memory (00001FFF).

PC + 2 → (R - 1, R - 2)
R - 2 → R
000ℓ → PC_H
n → PC_L

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	CAL ℓ	1 1 1 ← ℓ →	E0 + ℓ	1
Operand	n	← n →		2
		8 7 6 5 4 3 2 1		

Cycles 7
Flags None
Other None

CALL nm

Call subroutine. Store the address in the PC+2, the next command after the call on the stack. Jump to the absolute address n, anywhere in the 64k bytes.

PC + 3 → (R - 1, R - 2)
R - 2 → R
n → PC_H
m → PC_L

	Mnem	Appearence in Memory	Op Code	Byte
Op Code	CALL	0 1 1 1 1 0 0 0	78	1
Operand	n	← n →		2
	m	← m →		3
		8 7 6 5 4 3 2 1		

Cycles 8
Flags None
Other None

RTN

Return from subroutine.

Pop the address on the stack into the PC.

$$\boxed{\begin{array}{l}(R) \rightarrow PC_L \\ (R + 1) \rightarrow PC_H \\ R + 2 \rightarrow R\end{array}}$$

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	RTN	0	0	1	1	0	1	1	1	37	1
		8	7	6	5	4	3	2	1		

Cycles	4
Flags	None
Other	None

NOPW

No operation, 2 cycles.

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	NOPW	0	1	0	0	1	1	0	1	4D	1
		8	7	6	5	4	3	2	1		

Cycles	2
Flags	None
Other	None

NOPT

No operation, 3 cycles.

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	NOPT	1 1 0 0 1 1 1 0	CE	1
		8 7 6 5 4 3 2 1		

Cycles	3
Flags	None
Other	None

WAIT n

No operation, 6 + n cycles.

	Mnem	Appearence in Memory	Op Code	Byte
Op Code	WAIT	0 1 0 0 1 1 1 0	4E	1
Operand	n	◄— n —►		2
		8 7 6 5 4 3 2 1		

Cycles	6 + n
Flags	None
Other	None

OUTC

Write the contents of the internal RAM address 5F to the control port. Bit 1 controls the LCD display. (0=DISPLAY OFF, 1=DISPLAY ON). Note: it is important to preserve the rest of the byte when altering bit 1.

(5F) → control port

	Mnem.	Appearance in Memory	Op Code	Byte
Op Code	OUTC	1 1 0 1 1 1 1 1	DF	1
		8 7 6 5 4 3 2 1		

Cycles	2
Flags	None
Other	None

(5F)

	BZ3	BZ2	BZ1	OFF	HLT	CL	DIS
8	7	6	5	4	3	2	1

1 DISPLAY ON
0 DISPLAY OFF

COUNTER RESET

CLOCK STOP

POWER OFF

CONTROL BIT OF Xout AND Xin (see below)

BZ3	BZ2	BZ1	Xout	Xin
0	0	0	LOW	Not Active
0	0	1	HIGH	Not Active
0	1	0	2kHz	Not Active
0	1	1	4kHz	Not Active
1	0	0	LOW	Active
1	0	1	HIGH	Active
1	1	x	Xin → Xout	Active

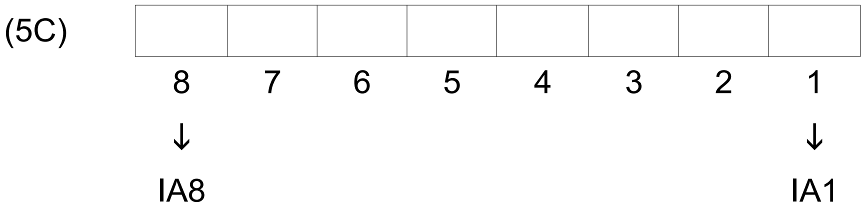
OUTA

Each bit of (5C) appears to the IA port.

(5C) → IA port

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	OUTA	0	1	0	1	1	1	0	1	5D	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	None
Other	None



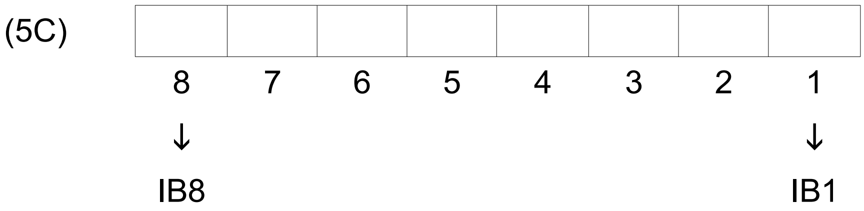
OUTB

Each bit of (5D) appears to the IB port.

(5D) → IB port

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	OUTB	1	1	0	1	1	1	0	1	DD	1
		8	7	6	5	4	3	2	1		

Cycles	2
Flags	None
Other	None



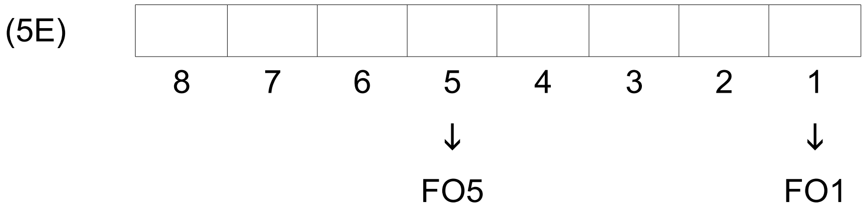
OUTF

Each bit of (5E) appears to the FO port.

(5E) → FO port

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	OUTF	0	1	0	1	1	1	1	1	5F	1
		8	7	6	5	4	3	2	1		

Cycles	3
Flags	None
Other	None



INA

To make IA port input terminal, set the corresponding bit(s) of RAM (5C) "0" and then execute OUTA command.

IA port → A (Accumulator)

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	INA	0	1	0	0	1	1	0	0	4C	1
		8	7	6	5	4	3	2	1		

Cycles	2
Flags	Z
Other	None

INB

To make IB port input terminal, set the corresponding bit(s) of RAM (5D) "0" and then execute OUTB command.

IB port → A (Accumulator)

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	INB	1	1	0	0	1	1	0	0	CC	1
		8	7	6	5	4	3	2	1		

Cycles	2
Flags	Z
Other	None

TEST n

n SET the bit of the operand which is required to test.

n → TEST

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	TEST	0	1	1	0	1	0	1	1	6B	2
Operand	n	X in				K on					
		8	7	6	5	4	3	2	1		

Cycles	4
Flags	None
Other	None

ex)

TEST
80 Judge the input level of Xin
IF Xin = 1 Z = 0
IF Xin = 0 Z = 1

TEST
08 Judge the input level of Kon
IF Kon = 1 Z = 0
IF Kon= 0 Z = 1

TEST 2 msec Counter test
02

TEST Approx. 0.5 sec Counter test
01

CUP

This instruction test the input status of XI. Register I contains the number of tests to be perform. If XI goes high during a test, execution of the instruction is terminated.

```
I → d
REPEAT
        P + 1, d – 1
UNTIL
        d = FF OR XI = 1
```

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	CUP	0	1	0	0	1	1	1	1	4F	1
		8	7	6	5	4	3	2	1		

Cycles	1 + 4d
Flags	Z
Other	P register changes

CDN

This instruction test the input status of XI. Register I contains the number of tests to be perform. If XI goes low during a test, execution of the instruction is terminated.

```
I → d
REPEAT
        P + 1, d – 1
UNTIL
        d = FF OR XI = 0
```

	Mnem.	Appearance in Memory								Op Code	Byte
Op Code	CDN	0	1	1	0	1	1	1	1	6F	1
		8	7	6	5	4	3	2	1		

Cycles	1 + 4d
Flags	Z
Other	P register changes

APPENDIXES

Specifications

Since this unit employs CMOS parts, input voltage to the input terminal cannot exceed the allowable range (SG-VC). Specifications for serial interface input (RD, CS, and CD) and output terminals (SD, RS. RR. and ER) are shown below.

Item	Condition	Specification	
		MIN.	MAX.
Input high level voltage V_{IH}		1.3V	
Input low level voltage V_{IL}			1.0V
Output high level voltage V_{OH}	Io = - 400μA	2.4V	
Output low level voltage V_{OL}	Io = 2mA		0.4V

Temperature range: 0 to 40°C
Power: Vc = 4. 5V

OP Codes in Alphabetic order

Mnemonic	Function	Bytes	Cycles	Hex. op. code
ADB	[P + 1,P] + [BA] → [P + 1,P]	1	5	14
ADCM	(P) + A + C →(P)	1	3	C4
ADIA n	A + n → A	2	4	74
ADIM n	(P) + n → (P)	2	4	70
ADM	(P) + A → (P)	1	3	44
ADN	(P) + A → (P) (BCD)	1	7+3d	0C
ADW	(P) + (Q) → (P) (BCD)	1	7+3d	0E
ANIA	A ∧ n → A	2	4	64
ANID	(DP) ∧ n → (DP)	2	6	D4
ANIM n	(P) ∧ n → (P)	2	4	60
ANMA	(P) ∧ A →(P)	1	3	46
CAL In	PC + 2 → (R –1, R – 2) R – 2 → R, In → PC	2	7	E0 + I
CALL nm	(PC + 3) → (R – 1,R – 2) R – 3 → R, nm → PC	3	8	78
CPIA n	A – n C,Z	2	4	67
CPIM n	(P) – n C,Z	2	4	63
CPMA	(P) – A C,Z	1	3	C7
DECA	A – 1 → A	1	4	43
DECB	B – 1 → B	1	4	C3
DECI	I – 1 → I	1	4	41
DECJ	J – I → J	1	4	C1
DECK	K – 1 → K	1	4	49
DECL	L –1 → L	1	4	C9
DECM	M – 1 → M	1	4	4B
DECN	N– 1 → N	1	4	CB
DECP	P– 1→ P	1	2	51
DX	X –1 → X, X → DP	1	6	05
DXL	X –1 → X, X → DP, (DP) → A	1	7	25
DY	Y – 1 → Y, Y → DP	1	6	07
DYS	Y – 1 → Y, Y → DP, A → (DP)	1	6	27
EXAB	A ↔ B	1	3	DA
EXAM	A ↔ (P)	1	3	DB
EXB	(P) ↔ (Q)	1	6+3d	0B
EXBD	(P) ↔ (DP)	1	7 + 6d	1B
EXW	(P) ↔ (Q)	1	6 + 3d	09
EXWD	(P) ↔ (DP)	1	7 + 6d	19
FILD	A → (DP), DP + 1 → DP	1	4+3d	1F

Mnemonic	Function	Bytes	Cycles	Hex. op. code
FILM	A → (P), P +1 → P	1	5+d	1E
INA	IA port → A	1	2	4C
INB	IB port → A	1	2	CC
INCA	A + 1 → A	1	4	42
INCB	B + 1 → B	1	4	C2
INCI	I + 1 → I	1	4	40
INCJ	J + 1 → J	1	4	C0
INCK	K + 1 → K	1	4	48
INCL	L + 1 → L	1	4	C8
INCM	M + 1 → M	1	4	4A
INCN	N + 1 → N	1	4	CA
INCP	P + 1 → P	1	2	50
IY	Y + 1 → Y, Y → DP	1	6	06
IYS	Y + 1 → Y, Y → DP A → (DP)	1	6	26
IX	X + 1 → X x → DP	1	6	04
IXL	X + 1 → X, X → DP (DP) → A	1	7	24
JP nm	n → PC_H , m → PC_L	3	6	79

JPC nm	IF C = 1 THEN n → PC_H , m → PC_L
	IF C = 0 THEN PC + 3 → PC
		3	6	7F

JPNC nm	IF C = 0 THEN n → PC_H , m → PC_L
	IF C = 1 THEN PC + 3 → PC
		3	6	70

JPNZ nm	IF Z = 0 THEN n → PC_H , m → PC_L
	IF Z = 1 THEN PC + 3 → PC
		3	6	7C

JPZ nm	IF Z = 1 THEN n → PC_H , m → PC_L
	IF Z = 0 THEN PC + 3 → PC
		3	6	7E

JRCM n	IF C =1 THEN PC + 1 – n → PC
	IF C =0 THEN PC + 2 → PC
		2	7/4	3B

JRCP n	IF C = 1 THEN PC + 1 + n → PC
	IF C = 0 THEN PC + 2 → PC
		2	7/4	3A

JRM n	PC + 1 - n → PC	2	7	2D

JRNCM n	IF C = 0 THEN PC +1 - n → PC
	IF C = 1 THEN PC + 2 → PC
		2	7/4	2B

JRNCP n	IF C = 0 THEN PC + 1 + n → PC
	IF C = 1 THEN PC + 2 → PC
		2	7/4	2A

JRNZM n	IF Z = 0 THEN PC + 1 - n → PC
	IF Z = 1 THEN PC + 2 → PC
		2	7/4	29

JRNZPn	IF Z = 0 THEN PC + 1 + n → PC
	IF Z = 1 THEN PC + 2 → PC
		2	7/4	28

Mnemonic	Function	Bytes	Cycles	Hex. op. code
JRP n	PC + 1 + n → PC	2	7	2C

JRZM n	IF Z = 1 THEN PC + 1 - n → PC
	IF Z = 0 THEN PC + 2 → PC
		2	7/4	39

JRZP n	IF Z = 1 THEN PC + 1 + n → PC
	IF Z = 0 THEN PC + 2 → PC
		2	7/4	38

LEAVE	0 → (R)	1	2	D8
LDD	(DP) → A	1	3	57
LDM	(P) → A	1	2	59
LDP	P → A	1	2	20
LDQ	Q → A	1	2	21
LDR	R → A	1	2	22
LIA n	n → A	2	4	02
LIB n	n → B	2	4	03
LIDL n	n → DPL	2	5	11
LIDP nm	n → DPH, m → DPL	3	8	10
LII n	n → I	2	4	00
LIJ n	n → J	2	4	01
LIP n	n → P	2	4	12
LIQ n	n → Q	2	4	13

LOOP	(R) - 1 → (R)
	IF C = 0 THEN PC + 1 - n → PC
	IF C = 1 THEN PC + 2 → PC
		2	10/7	2F

LP ℓ	ℓ → p	1	2	80+ℓ
MVB	(Q) → (P)	1	5 + 2d	0A
MVBD	(DP) → (P)	1	5 + 4d	1 A
MVDM	(P) → (DP)	1	3	53
MVMD	(DP) → (P)	1	3	55
MVW	(Q) → (P)	1	5+2d	08
MVWD	(DP) → (P)	1	5+4d	18
NOPT	NOP	1	3	CE
NOPW	NOP	1	2	4D
ORIA n	A V n → A	2	4	65
ORID n	(DP) V n → (DP)	2	6	D5
ORIM n	(P) V n → (P)	2	4	61
ORMA	(P) V A → (P)	1	3	47
OUTA	(5C) → IA port	1	3	5D
OUTB	(5D) → IB port	1	2	DD
OUTF	(5E) → FO port	1	3	5F

Mnemonic	Function	Bytes	Cycles	Hex. op. code
OUTC	(5F) → CONTROL	1	2	DF
POP	(R) → A R + 1 → R	1	2	5B
PUSH (DO)	R - 1 → R A → (R)	1	3	34
RC	0 → C 1 → Z	1	2	D1
RTN	(R) → PCL (R + 1) → PC R + 2 → R	1	4	37
SBB	[P + 1, P] – [BA] → [P + 1, P]	1	5	15
SBCM	(P) – A – C → (P)	1	3	C5
SBIA n	A – n → A	2	4	75
SBIM n	(P) – n → (P)	2	4	71
SBM	(P) – A → P)	1	3	45
SBN	(P) – A → (P) (BCD)	1	7+3d	0D
SBW	(P) – (Q) → (P) (BCD)	1	7+3d	0F
SC	1 → C, 1 → Z	1	2	D0
SL	1 bit shift left	1	2	5A
SLW	4 bit shift left	1	5+d	1D
SR	1 bit shift right	1	2	D2
SRW	4 bit shift right	1	5 + d	1C
STD	A → (DP)	1	2	52
STP	A → P	1	2	30
STQ	A → Q	1	2	31
STR	A → R	1	2	32
SWP	A1 – A4 ↔ A5 – A8	1	2	58
TEST n	n → TEST	2	4	6B
TSIA n	A ∧ n Z	2	4	66
TSID n	(DP) ∧ n Z	2	6	D6
TSIM n	(P) ∧ n Z	2	4	62
WAIT n	NOP	2	6+n	4E
CUP	Test to see if XI is high	1	1+4d	4F
CDN	Test to see if XI is low	1	1+4d	6f
CASE1 CASE2	Conditional branching	4 + 3d	8 5 + 7d	7A 69

Machine Code

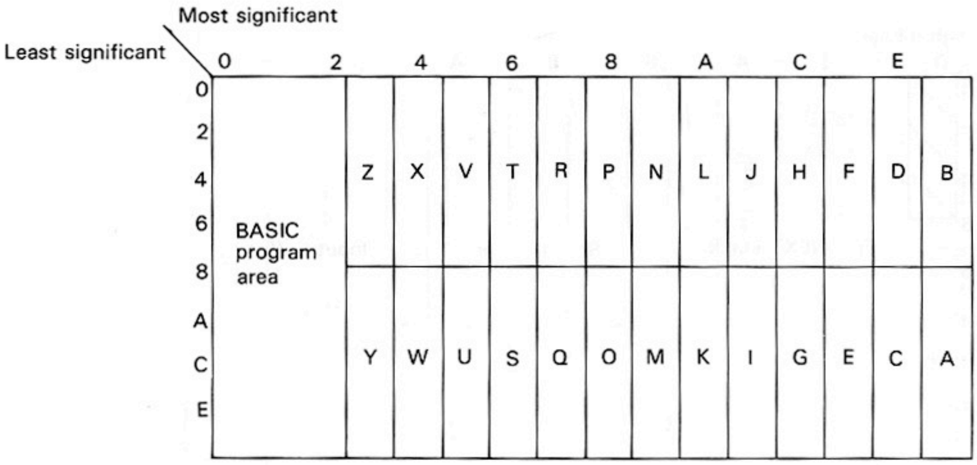
MS LS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	LII n	LIDP nm	LDP	STP	INCI	INCP	ANIM n	ADIM n					INCJ	SC		
1	LIJ n	LIDL n	LDQ	STQ	DECI	DECP	ORIM n	SBIM n					DECJ	RC		
2	LIA n	LIP n	LDR	STR	INCA	STD	TSIM n						INCB	SR		
3	LIB n	LIQ n			DECA	MVDM	CPIM n						DECB	WRIT		
4	IX	ADB	IXL	PUSH	ADM	READM	ANIA n	ADIA n					ADCM	ANID n		
5	DX	SBB	DXL	DATA	SBM	MVMD	ORIA n	SBIA n		LP	I		SBCM	ORID n		
6	IY		IYS		ANMA	READ	TSIA n						TSMA	TSID n	CAL	In
7	DY		DYS	RTN	ORMA	LDD	CPIA n						CPMA	CPID n		
8	MVW	MVWD	JRNZP n	JRZP n	INCK	SWAP		CALL nm					INCL	LEAVE		
9	EXW	EXWD	JRNZM n	JRZM n	DECK	LDM	CASE 2	JP nm					DECL			
A	MVB	MVBD	JRNCP n	JRCP n	INCM	SL		CASE 1					INCN	EXAB		
B	EXB	EXBD	JRNCM n	JRCM n	DECM	POP	TEST n						DECN	EXAM		
C	ADN	SRW	JRP n		INA			JPNZ nm					INB			
D	SBN	SLW	JRM n		NOPW	OUTA		JPNC nm						OUTB		
E	ADW	FILM			WAIT n			JPZ nm					NOPT			
F	SBW	FILD	LOOP n		CUP	OUTF	CDN	JPC nm						OUTC		

Internal Representation of BASIC

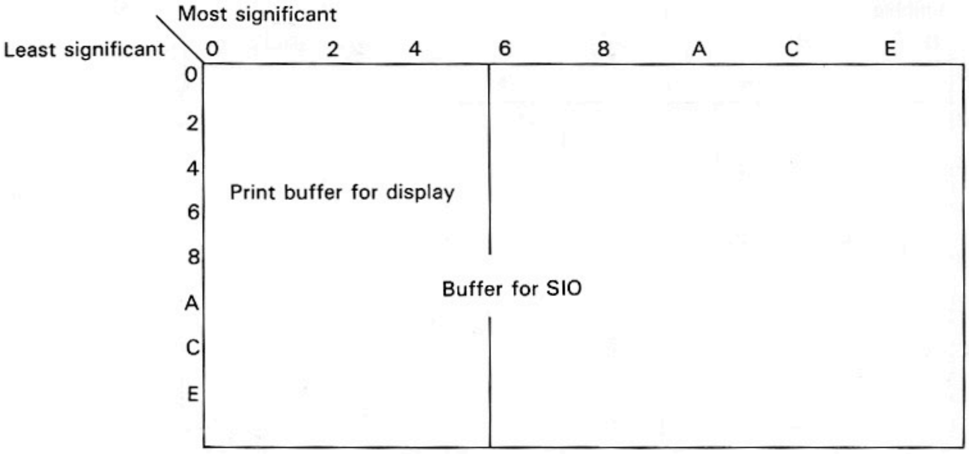
MS LS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NULL		SPACE	0	@	P	`	p		~	RND	RUN	RANDOM	TO	GOSUB	
1			!	1	A	Q		q		LN	AND	NEW	DEGREE	STEP	AREAD	
2			"	2	B	R	b	r		LOG	OR	CONT	RADIAN	THEN	LPRINT	
3			#	3	C	S	c	s		EXP	NOT	PASS	GRAD	ON	RETURN	
4			$	4	D	T	d	t		SQR	ASC	LIST	BEEP	IF	RESTORE	
5			%	5	E	U	e	u		SIN	VAL	LLIST	WAIT	FOR	CHAIN	♠
6			&	6	F	V	f	v		COS	LEN	CSAVE	GOTO	LET	GCURSOR	♥
7			'	7	G	W	g	w		TAN	PEEK	CLOAD	TRON	REM	GPRINT	♦
8			(	8	H	X	h	x		INT	CHR$	MERGE	TROFF	END	LINE	♣
9			)	9	I	Y	i	y		ABS	STR$	~	CLEAR	NEXT	POINT	■
A			*	:	J	Z	j	z		SGN	MID$	~	USING	STOP	PSET	□
B			+	;	K	[	k	{		DEG	LEFT$	OPEN	DIM	READ	PRESET	π
C			,	<	L	\	l	|		DMS	RIGHT$	CLOSE	CALL	DATA	BASIC	√
D	CR		_	=	M	]	m	}		ASN	INKEY$	SAVE	POKE	PAUSE	TEXT	
E			.	>	N	^	n	~		ACS	PI	LOAD	CLS	PRINT	OPEN$	
F			/	?	O	_	o			ATN	MEM	CONSOLE	CURSOR	INPUT	~	BEGIN-END

Memory Map (I)

6C00H ~ 6CFFH

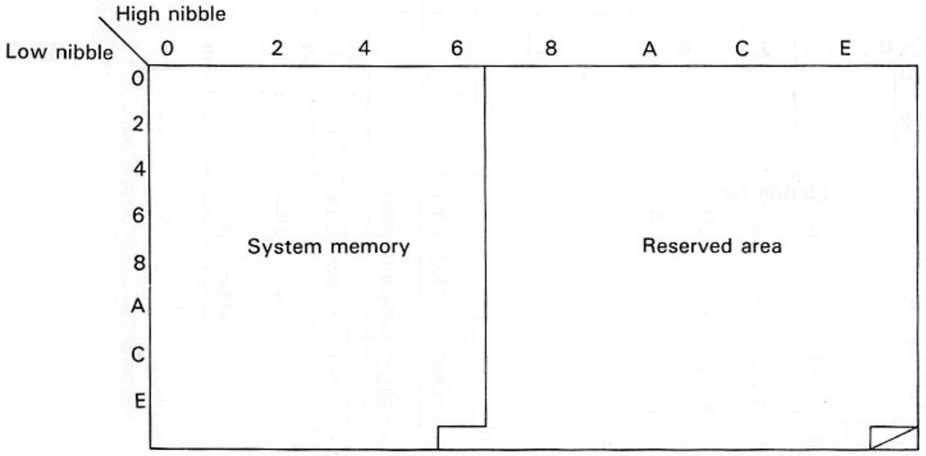


6D00H ~ 6DFFH

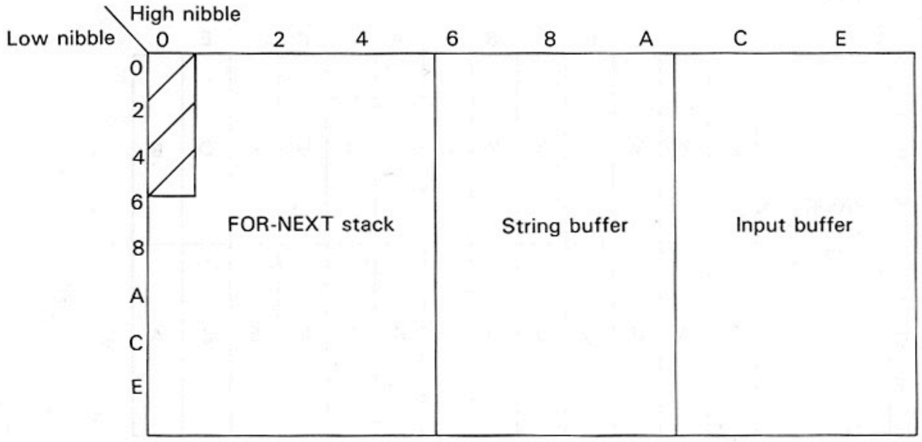


SIO: Serial Interface

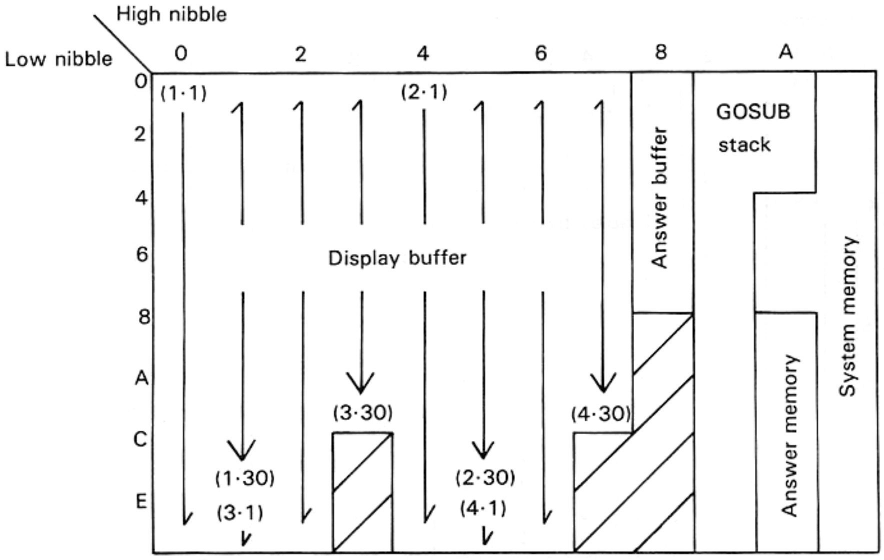
6F00H ~ 6FFFH



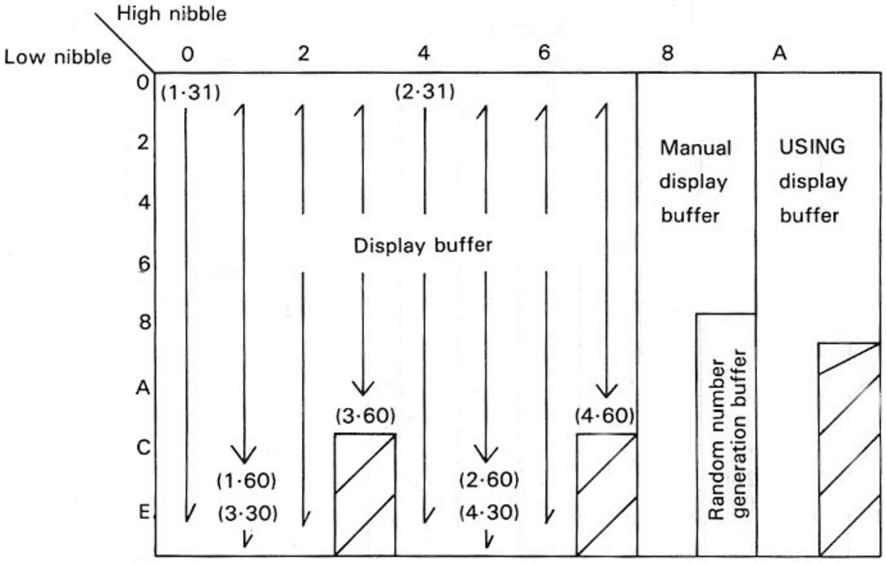
6E00H ~ 6EFFH



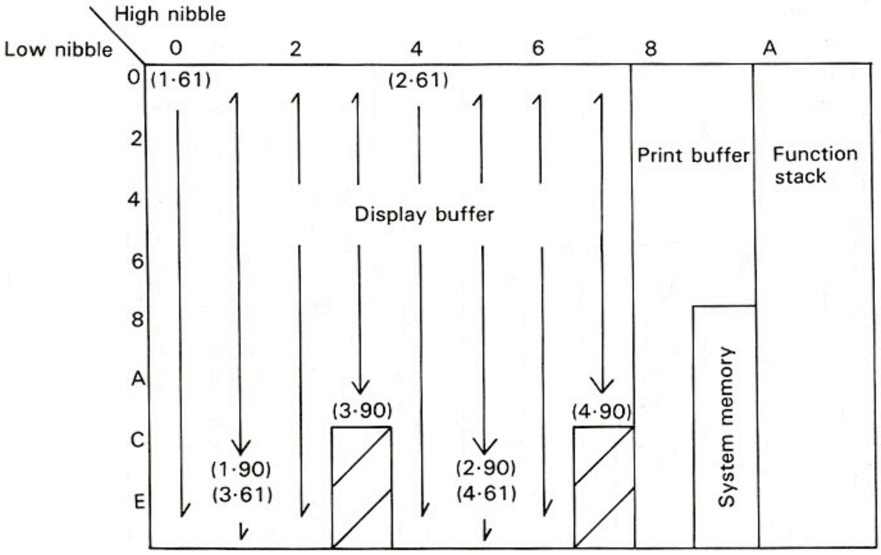
7000H ~ 70BFH



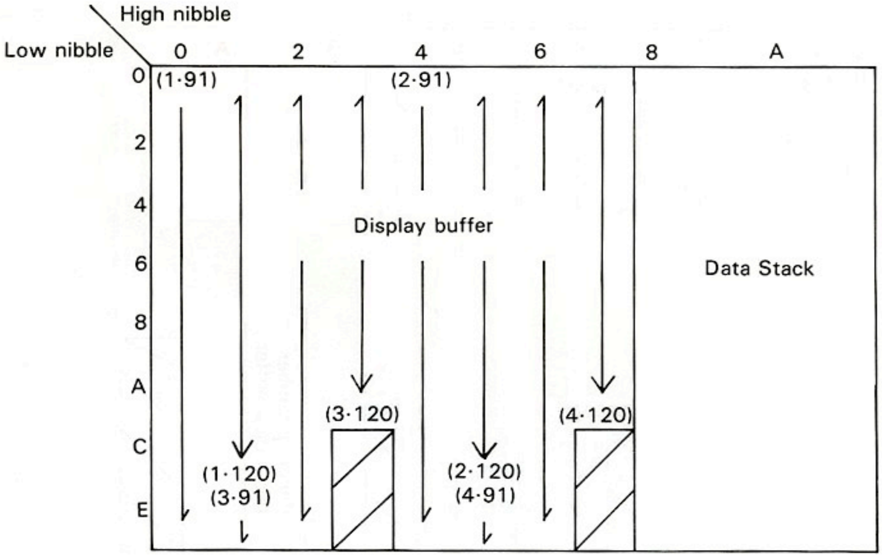
7200H ~ 72BFH



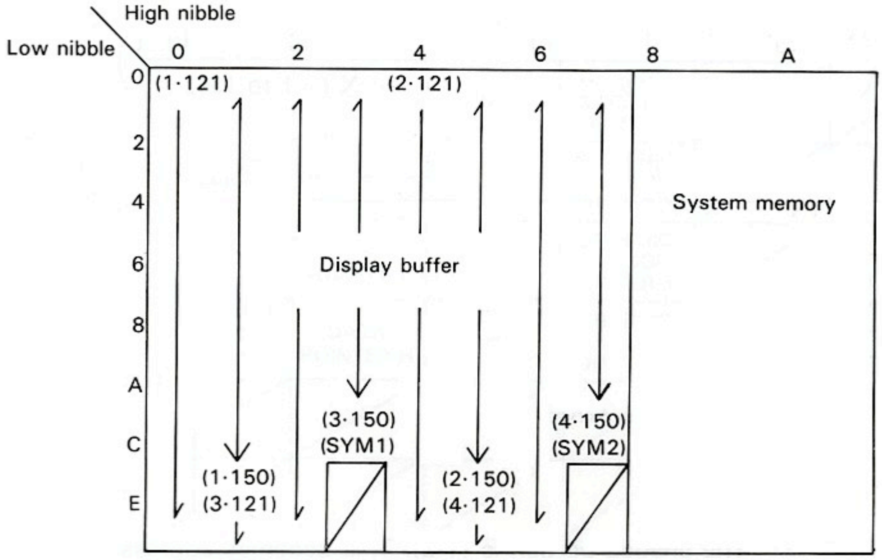
7400H ~ 74BFH



7600H ~ 76BFH



7800H ~ 78BFH

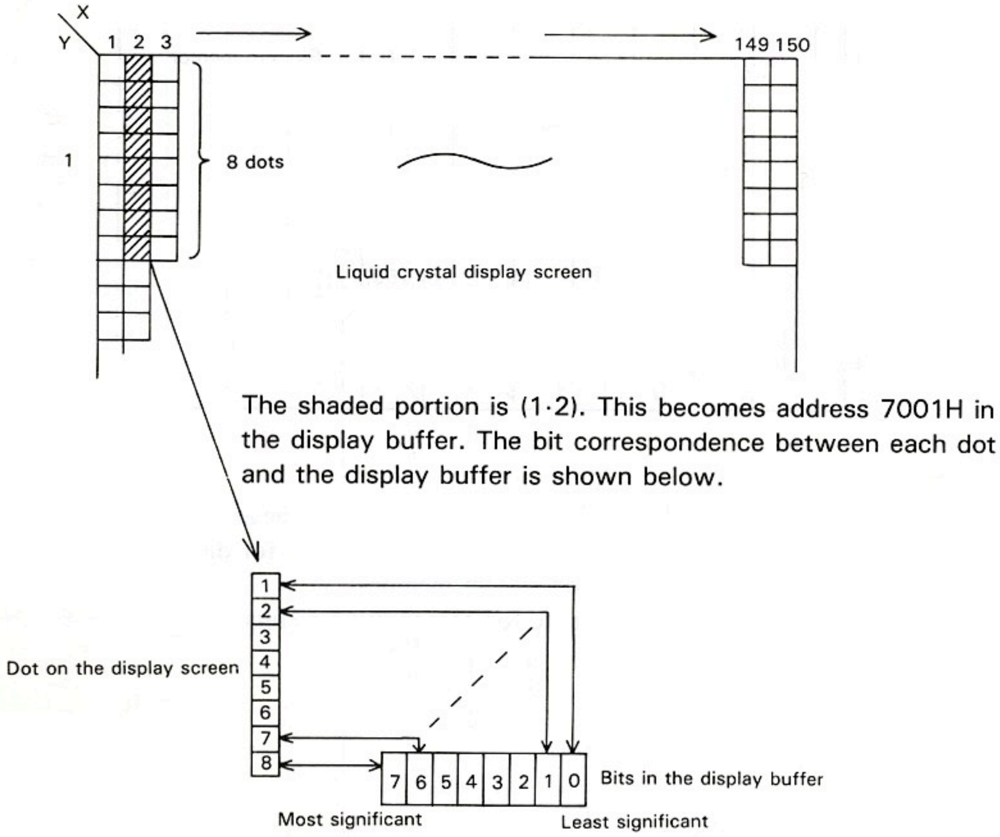


- A to Z (6C30H to 6CFFH) are numerical variables, but they use the same area as character variables A$ to Z$.
- Part of the SIO buffer (6D00H to 6DFFH) is used as a print buffer for display.
- Refer to memory map (II) for information on system memory.
- The answer buffer is used as temporary storage for computation results.
- Answer memory stores the last answer.

Notes on the display buffer

i) Numbers in parentheses, such as (1·31) in the display buffer diagram, indicate the display position on the screen.

Vertical position in units of 8 dots • Horizontal dot position
y (= 1 to 4) X (= 1 to 150)


X
Y
1 2 3
149 150
1
8 dots
Liquid crystal display screen
Dot on the display screen
1
2
3
4
5
6
7
8
7 6 5 4 3 2 1 0
Bits in the display buffer
Most significant
Least significant

The shaded portion is (1·2). This becomes address 7001H in the display buffer. The bit correspondence between each dot and the display buffer is shown below.

ii) (SYM1) at address 783CH in the display buffer indicates the symbolic contents of the left side of the display screen.

Bit	7	6	5	4	3	2	1	0
783CH	SML		PRO	RUN			DEF	SHIFT

(SYM2) at address 787CH is not displayed, but it indicates the angle mode.

Bit	7	6	5	4	3	2	1	0
787CH						GRAD	GRADIAN	DEGREE

Memory Map (II)

(1) System Memory detail

6FXXH

MS LS	0	1	2	3	4	5	6
0					GRAPHIC CURSOR POINTER X_L		
	TEXT TOP L				GRAPHIC CURSOR POINTER X_H		
2	TEXT TOP H		DATA POINTER L		GRAPHIC CURSOR POINTER Y_L		
	TEXT END L		DATA POINTER H		GRAPHIC CURSOR POINTER Y_H		
4	TEXT END H						
	MERGE TEXT TOP L			BLINK CHARACTER			
6	MERGE TEXT TOP H						
	VARIABLE POINTER L						
8	VARIABLE POINTER H			INPUT BUFFER CURSOR POINTER			
				USING F/F			
A			INPUT BUFFER POINTER	USING M			
			FOR POINTER	USING m			
C		EXCLUSIVE TEXT TOP L	GOSUB POINTER	USING &		PREVIOUS ORIGIN POINTER X_L	
		EXCLUSIVE TEXT H	DATA STACK POINTER			PREVIOUS ORIGIN POINTER X_H	
E			FUNCTION STACK POINTER			PREVIOUS ORIGIN POINTER Y_L	
			STRING BUFFER POINTER			PREVIOUS ORIGIN POINTER Y_H	

70BXH

0		AUTO POWER OFF COUNTER L
		AUTO POWER OFF COUNTER M
2		AUTO POWER OFF COUNTER H
		WAIT COUNTER L
4		WAIT COUNTER H
6		
8		
A		
C		
E		

749XH

8	PREVIOUS OLD ADDRESS L
	PREVIOUS OLD ADDRESS H
A	BREAK ADDRESS L
	BREAK ADDRESS H
C	ERROR ADDRESS L
	ERROR ADDRESS H
E	

78XXH

MS LS	8	9	A	B
0	DISPLAY POINTER Y			
	DISPLAY POINTER X			EOT CODE
2				BAUD RATE
				SIO CONDITION F/F
4				CONSOLE VALUE
6				
8				
A				
	CURSOR POINTER X			
C	CURSOR POINTER Y			
	BLINK CURSOR L			
E	BLINK CURSOR H			

(2) Detail of System Memory in CPU

10H–3FH

MS LS	1	2	3
0	XREG	ZREG	
2			
4			ERL
6			
8	YREG	WREG	CURRENT TOP L
			CURRENT TOP H
A			SEARCH ADDRESS L
			SEARCH ADDRESS H
C			SEARCH LINE L
			SEARCH LINE H
E			CURRENT LINE L
			CURRENT LINE H

Address	Name	Contents
6F01H	TEXT TOP L	Beginning of BASIC program
6F02H	TEXT TOP H	
6F03H	TEXT END L	End of BASIC program
6F04H	TEXT END H	
6F05H	MERGE TEXT TOP L	Beginning of the program block last merged
6F06H	MERGE TEXT TOP H	
6F1CH	EXECUTIVE TEXT TOP L	Beginning of the program currently being executed.
6F1DH	EXECUTIVE TEXT TOP H	
6F2AH	INPUT BUFFER POINTER	Input buffer pointer
6F2DH	DATA STACK POINTER	Data stack pointer
6F2EH	FUNCTION STACK POINTER	Function stack pointer
78B1 H	EOT CODE	EOT code (SIO)
78B2H	BAUD RATE	Baud rate (SIO)
78B3H	SIO CONDITION F/F	Interface condition F/F (SIO)
78B4H	CONSOLE VALUE	Console value (SIO)
(10H - 17H)	XREG	Operation register
(18H - 1FH)	YREG	
(20H - 27H)	ZREG	
(28H - 2FH)	WREG	
6F40H	GRAPHIC CURSOR POINTER XL	Graphic cursor pointer XLH: horizontal YLH: vertical (- 32768 to 32767)
6F41 H	GRAPHIC CURSOR POINTER XH	
6F42H	GRPAHIC CURSOR POINTER YL	
6F43H	GRAPHIC CURSOR POINTER YH	
788BH	CURSOR POINTER X	Cursor pointer (X: 0 to 23, Y: 0 to 3)
788CH	CURSOR POINTER Y	
6F5CH	PREVIOUS ORIGIN POINTER XL	End point coordinates of the LINE instruction previously executed. (- 32768 to 32767)
6F5DH	PREVIOUS ORIGIN POINTER XH	
6F5EH	PREVIOUS ORIGIN POINTER YL	
6F5FH	PREVIOUS ORIGIN POINTER YH	
6F38H	INPUT BUFFER CURSOR POINTER	Cursor pointer in the input buffer.
7880H	DISPLAY POINTER Y	Pointer indicating display position (X: 0 to 23, Y: 0 to 3)
7881H	DISPLAY POINTER X	
70B4H	WAIT COUNTER H	Wait counters
70B3H	WAIT COUNTER L	
6F36H	BLINK CHARACTER	Character code of blinking character
788CH	BLINK CURSOR H	Position of blinking cursor (address in

788DH	BLINK CURSOR L	display buffer)
6F2BH	FOR POINTER	Stack pointer of FOR-NEXT
6F2CH	GOSUB POINTER	GOSUB pointer
6F2FH	STRING BUFFER POINTER	String buffer pointer
6F39H	USING F/F	USING format (whether decimal points or commas are used)
6F3AH	USING M	Integer part of USING
6F3CH	USING &	USING for character string
6F3BH	USING m	USING decimal point
6F08H	VARIABLE POINTER H	Variable pointers
6F07H	VARIABLE POINTER L	
(34H)	ERL	Error number when an error occurred
(3FH)	CURRENT LINE H	Current line number
(3EH)	CURRENT LINE L	
(38H)	CURRENT TOP H	Beginning of the program containing the current line
(39H)	CURRENT TOP L	
7499H	PREVIOUS OLD ADDRESS H	Address of the previous line
7498H	PREVIOUS OLD ADDRESS L	
(3BH)	SEARCH ADDRESS H	Address of the line found in a search
(3AH)	SEARCH ADDRESS L	
(3DH)	SEARCH UNE H	Line number found after search
(3CH)	SEARCH UNE L	
749BH	BREAK ADDRESS H	Break address
749AH	BREAK ADDRESS L	
749DH	ERROR ADDRESS H	Error addresses
749CH	ERROR ADDRESS L	
6F23H	DATA POINTER H	Data text pointers
6F22H	DATA POINTER L	
70B2H	AUTO POWER OFF COUNTER H	Auto power off counters
70B1H	AUTO POWER OFF COUNTER M	
70B0H	AUTO POWER OFF COUNTER L	

Addresses enclosed in parentheses are within the CPU.

System Subroutines

The following subroutines can be used when a program is written in machine language.
The entry address of a subroutine may be different for different ROM versions.
The ROM version can be determined by checking the contents of address FFF0H.

Version 0	Contents of FFF0H is CEH (= 206)
Version 1	Contents of FFF0H is 03H

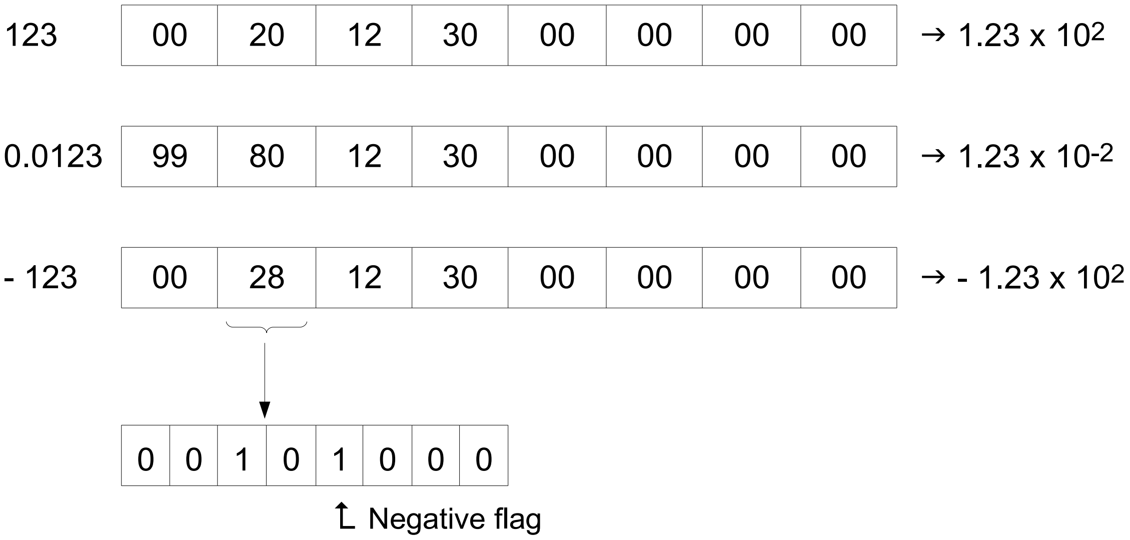
The entry addresses for Version 1 are used in this manual. The entry addresses for Version 0 are indicated by brackets. When Version 0 is not mentioned, and the address is not enclosed in brackets, the same entry number can be used for both versions.

(1) Operation subroutines

① Entry preparation

Numbers must be stored in decimal format in operation registers X (10H to 17H) and Y (l8H to 1FH) the case of a single variable function, use operation register X only.

Operation register format



123	00	20	12	30	00	00	00	00	→ 1.23 x 10^{2}
0.0123	99	80	12	30	00	00	00	00	→ 1.23 x 10^{-2}
- 123	00	28	12	30	00	00	00	00	→ - 1.23 x 10^{2}

0	0	1	0	1	0	0	0

② Entry addresses

	Operation		Version 0	Version 1
Two-variable functions	Addition	Y + X → X	8962H	8AB8H
	Subtraction	Y - X → X	8979H	8ACFH
	Multiplication	Y * X → X	8983H	8AD9H
	Division	Y / X → X	898DH	8AE3H
	Exponentiation	Y ^ X → X	8996H	8AECH
Single-variable functions	Square root	SQR X → X	89B3H	8B09H
	Logarithm	LN X → X	899DH	8AF3H
		LOG X → X	89A5H	8AFBH
	Exponent	EXP X → X	89ACH	8B02H
	Trigonometric function	SIN X → X	89B8H	8B10H
		COS X → X	89C1H	8B17H
		TAN X → X	89C8H	8B1EH
	Inverse trigonometric function	ASN X → X	89CFH	8B25H
		ACS X → X	89D6H	8B2CH
		ATN X → X	89DDH	8B33H
	DMS conversion	DEG X → X	89EBH	8B41H
		DMS X → X	89F2H	8B48H
	Absolute value	ABS X → X	8E9FH	8FF4H
	Integer	INT X → X	8E7BH	8FD0H
	Sign	SGN X → X	8A00H	8B56H
	Random number	RND X → X	89F9H	8B4FH

(2) Comparison operations

< Numeric comparison>

① Entry preparation

Numbers are stored in decimal format in operation registers X and Y.

② Entry address

Operation	Version 0	Version 1
Y <> X	8B0FH	8C65H
Y < X	8A85H	8BDBH
Y > X	8AB5H	8C0BH
Y = X	8AFBH	8C51H
Y <= X	8A1FH	8B75H
Y >= X	8A2CH	8B82H

③ Condition satisfied

When the condition is satisfied
The value 1 is stored in operation register X.

XREG	00	00	10	00	00	00	00	00	→ 1

When the condition is not satisfied
The value 0 is stored in operation register X.

XREG	00	00	00	00	00	00	00	00	→ 0

< Character string comparison>

① Entry preparation

The following values are stored in operation registers X and Y.

	XReg address	YReg address
D0H	14H	1CH
Starting address of character string (least significant digits)	15H	1DH
Starting address of character string (most significant digits)	16H	1EH
Length of character string	17H	1FH

The string buffer (6E60H to 6EAFH) can be used to store addresses for the character string.

Store 60H in the string buffer pointer (6F2FH).

② Entry address

Operation	Version 0	Version 1
y <> X	8B0AH	8C60H
y < X	8A34H	8B8AH
Y > X	8A36H	8B8CH
Y = X	8ABDH	8C13H
Y <= X	8A18H	8B6EH
Y>= X	8A1AH	8B70H

③ Condition satisfied

The value 1 is stored in operation register X.

XREG	00	00	10	00	00	00	00	00	→ 1

Condition not satisfied
The value 0 is stored in operation register X.

XREG	00	00	00	00	00	00	00	00	→ 0

(3) Character string operation functions

1) STR$

- Entry condition

  i) The decimal number in internal format to be converted is stored in operation register X.
  ii) 60H is stored in the string buffer pointer (6F2FH).

- Entry address

  8CFCH [8BA6H]

- Exit status

  i) The converted character string information is stored, in internal character string format, in operation register X in the CPU.
  ii) The actual character string is stored in the string buffer.

2) CHR$

- Entry condition

  Same as described in (1). The valid range is 0≤number≤255.

- Entry address

  8C94H [8B3EH]

- Exit status

  i) CARRY = 0
  Same as described in (1) STR$.
  ii) CARRY = 1
  The number to be converted does not satisfy the following expression:
  0≤number≤255

3) VAL

- Entry condition

Store character string information for the character string to be converted (which exists in the string buffer) in operation register X using internal character format.

- Entry address

  8D58H [8C02H]

- Exit status

  i) CARRY = 0

  The converted decimal number is stored in internal format in operation register X.

  ii) CARRY = 1

  The number cannot be converted to a decimal number in internal format.

4) ASC

- Entry condition

  Same as described in 3) VAL.

- Entry address

  8C74H [881EH]

- Exit status

  The converted decimal number is stored in internal format in operation register X.

(4) Key scan

- The number of the currently pressed key is stored in ACC.

- Entry address

  0436H

Carry	ACC
0	No key
1	00-3FH (40H if two or more keys are pressed)

- The contents of registers B, K, L, M, and N are unpredictable.
- Strobe signals ($K0_1$ through $K0_6$ and IA_1 through IA_6) are all low upon return.

Note: A key number is indicated using a 1-byte binary number (0H to 40H). Values in the key code table correspond to key numbers. Refer to the key matrix and key code tables.

(5) Search function

1) Conversion of a decimal number in internal format to binary representation (2 bytes)

- Entry condition

The number in internal format to be converted is stored in operation register X in the CPU.

- Entry address

The entry address depends on the number to be converted (XReg).

(XReg)	Entry address
- 32768 ≤ (XReg) ≤ 32767	162FH
0 ≤ (XR.g) ≤ 65535	163AH

- Exit status

i) CARRY = 0

The converted value is stored in 19H (most significant byte) and 18H (least significant byte) in the CPU.

ii) CARRY = 1

Error. The value of register X does not fall within the range shown above.

Note: If the entry address is 162FH, the number is converted to a signed binary number. Numbers from - 32768 to 32767 are converted to binary numbers from 8000H to 7FFFH.

2) Conversion of a binary number (2 bytes) to a decimal number. in internal format

- Entry condition

The 2-byte binary number to be converted is stored in 19H (most significant byte) and 18H (least significant byte) in the CPU.

- Entry address

i) The stored binary number is converted directly.

11B0H

ii) The stored binary number is considered to be a signed binary number and is converted as such.

11B7H

- Exit status

The converted decimal number in internal format is stored in operation register X in the CPU.

3) Program line number search

- Entry condition

  Store the line number to be searched for (in 2-byte binary format) is stored in 19H (most significant byte) and 18H (least significant byte) in the CPU.

- Entry address

  B8F4H [B6E1H]

- Exit status

i) CARRY = 0

The specified line was found. The following data is stored in 3AH through 3DH in the CPU.

3AH — Address of the line number (least significant)
3BH — Address of the line number (most significant)
3CH — Line number (most significant)
3DH — Line number (least significant)

ii) CARRY = 1

The specified line number could not be found. 3CH and 3DH in the CPU indicate the following.

When both 3CH and 3DH are 0:

The entire program was searched, but the specified line could not be found.

When either 3CH or 3DH is not 0:

A line number greater than the specified line number was found.

Note: In using this subroutine, line numbers can be specified in internal format. In this case, the entry condition and entry address are as follows (the exit status is the same).

- Entry condition

i) The line number to be searched for is stored in operation register X in the CPU.
ii) The contents of 36H in the CPU and XXXXXXIX are ORed.

- Entry address

  B8EBH [B6D8H]

4) Variable address search (simple variable)

- Entry condition

i) The variable name to be searched for is stored in 0AH (first byte of the variable name) and 0BH (second byte of the variable name).

ii) Zero is stored in 33H in the CPU.

• Entry address
1AEDH

• Exit status
i) (Starting address of the variable contents)-l is stored in 06H and 07H (YL and YH) in the CPU.
ii) The length of the specified variable is stored in 02H (register A) in the CPU.

Note: This subroutine does not have error detection capability. Therefore, the specified variable must be defined.

5) Variable address search II (array variable)

• Entry condition
i) The name of the variable to be searched for is stored in 0AH (first byte of the variable name) and 0BH (second byte of the variable name) in the CPU.
ii) The subscript of the array to be searched for is stored in 0CH and 0DH in the CPU in binary format.

	1-dimensional array	2-dimensional array
0CH	First subscript	Second subscript
0DH	0	First subscript

iii) Zero is stored in 33H in the CPU.

• Entry address
17F5H

• Exit status
i) CARRY =0
Normal termination.
- (Starting address of variable contents)-l is stored in 06H and 07H (YL and YH) in the CPU.
- The unit length of the specified array variable is stored in 02H (register A) in the CPU.

ii) CARRY = 1
An error was detected. The following errors may be encountered:
- The specified array variable is not defined.
- The specified subscript does not fall within the subscript range declared at array definition.

(6) Display

1) One-line and full-screen display

Write the code of the character to be displayed in the address corresponding to the appropriate line of the print buffer. Satisfy the entry condition, and calI this subroutine. The contents will then be displayed on the liquid crystal screen.

Print butter address

First line of display	6D00H	~	6D17H
Second line of display	6D18H	~	6D2FH
Third line of display	6D30H	~	6D47H
Fourth line of display	6D48H	~	6D5FH

24 characters

• Entry condition

KR (register K ← 0)

D_2 bit of 788FH ← 1 (788FH value ORed with xxxxx1xx)

ACC ← 0 to 4

• 0 to 3 indicates the single line to be displayed. 0 for the first line, 1 for the second, and so on.

• 4 indicates that the full screen is to be displayed.

• Entry address

Version 1 D534H [D2B6H]

2) Scroll up

When this subroutine is called, the display image is scrolled up. The contents of the print buffer are also scrolled up, and the space code is stored in the fourth level of the print buffer.

• Entry

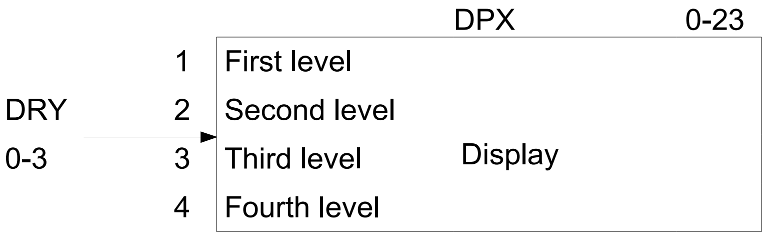
ACC ← 4

• Entry address

Version 1 E23CH [DEADH]

3) Single character display

This subroutine displays the character stored in ACC. The display position is determined by DPY and DPX.
The contents of the print buffer do not change.



DPX and DPY are used to determine the position the character is to displayed. The DPY address is 7880H, and the DPX address is 7881H.

- Entry condition
  D_0 of 788FH ← 1 (788FH value ORed with xxxxxxx1)
  Set DPY (0 to 3) and DPX (0 to 23)
  ACC ← Character code

- Entry address
  Version 1 E983H [E549H]

Note: DPY: Display Pointer Y
DPX: Display Pointer X

4) Set the LCD RAM address corresponding to the position indicated using DPY and DPX in Y_{LH}

The value stored in Y_{LH} is (display starting address for LCD RAM)-l. That is, when IYS is performed by this subroutine, the contents of ACC are stored in the first address of the LCD position indicated by DPY and DPX.

- Entry
  DPY (0 to 3)
  DPX (0 to 23)

- Entry address
  1CEFH

5) Display off

Display is terminated.

- Entry
  None.

- Entry address
  04ADH

6) Display on

Display is activated. Values in L©D RAM must be prepared for display.

- Entry
  None.

- Entry address
  04B1H

7) Print buffer clear

The print buffer (96 bytes) is cleared.

- Entry
  None.

- Entry address
  1E0CH

Note: These subroutines do not have entry check capabilities. The user is responsible for the validity of entry values.

(7) Serial interface (SIO)

1) Open serial interface circuit

- Entry condition
  None.

- Entry address
  FC7BH [FA67H]

- Exit status
  Only the ER signal is high. All other signals remain low.

2) Close serial interface circuit

- Entry condition
  None.

- Entry address
  FC97H [FA83H]

- Exit status

AlI signals on the serial port are low.

3) CS signal monitor

- Entry condition
  None.

- Entry address
  1E4BH

- Exit status
  i) CARRY = 0 CS signal is high.
  ii) CARRY = 1 The BREAK key was pressed.

Note: Control is not returned from this subroutine until one of the two conditions above is satisfied.

4) CD signal monitor

- Entry condition
  None.

- Entry address
  1E60H

- Exit status
  i) CARRY =0 CS signal is high.
  ii) CARRY = 1 The BREAK key was pressed.

Note: Control is not returned from this subroutine until one of the two conditions above is satisfied.

5) Get interface condition

- Entry condition
  None.

- Entry address
  1E43H

- Exit status
  The interface condition stored in external RAM is obtained in the CPU.
  EOT code (78B1H) to 0DH in the CPU
  Baud rate (78B2H) to 0EH in the CPU
  Condition (78B3H) to 0FH in the CPU

This subroutine must be executed before any of the system subroutines (6), (7), and (8) is used.
However, since the contents of 0DH through 0FH in the CPU do not change, they do not have to be set by this subroutine if these subroutines are used in succession.

6) Output 1 byte

- Entry condition
  Subroutine (5) must have been executed beforehand. Output data must also be stored in register B in the CPU.

- Entry address
  F316H [EF2DH]

- Exit status
  None.

7) Input 1 byte

- Entry condition
  Subroutine (5) must have been executed beforehand.

- Entry address
  F22AH [EE27H]

- Exit status
  i) Input data is stored in register B in the CPU.
  ii) CARRY is changed.
  CARRY = 0 Data input ended.
  CARRY = 1 Contents of 35H in the CPU indicate the following
  XX1XXXXX The BREAK key was pressed.
  XX0XXXXX A parity frame error occurred.
  If CARRY = l, the RR signal of the serial port goes low.

Note: If the input byte matches the termination code or the end of text code, the RR signal in the serial port goes low. (However, if the end code is CR + LF, this subroutine (7) is used repeatedly, and the 2 bytes are checked for a match.)

8) Output the termination code

- Entry condition
  Subroutine (5) must have been executed beforehand.

- Entry address
  F1FAH [EDF7H]

- Exit status
  i) CARRY = 0 The termination code was output.
  ii) CARRY = 1 The BREAK key was pressed.

Note: Subroutine (3) is called from subroutine (8). Therefore, control is not returned from this subroutine until the condition for subroutine (3) is satisfied.

9) Conversion of internal format (number) to ASCII sequence

- Entry condition
  The number in internal format to be converted is stored in operation register X in the CPU.

- Entry address
  F1B8H [EDB5H]

- Exit status
  The number converted to ASCII sequence is stored from the beginning of the SIO buffer (6D00H). The ENTER code is stored after it. The number always converted to exponential format.

10) Output SIO buffer contents

- Entry condition
  The contents to be output are input to the SIO buffer (6D00H to 6DFFH), and the ENTER code (0DH) is input to the contents.

- Entry address
  F217H [EE14H]

- Exit status
  i) CARRY = 0 AlI contents and the termination code were output.
  ii) CARRY = 1 The BREAK key was pressed.

Note: Subroutine (10) uses subroutine (3). Therefore, control is not returned from subroutine (10) until the condition from subroutine (3) is satisfied.

(8) Printer

1) Printing characters

① Entry preparation

- Connect the CE-126P to the main unit.
- Reset the printer.
- Entry address
  A467H [A2BAH]
- Store the 24 digit code to be printed in registers X, Y, and Z.

② Print execution

- Entry address
  8054H

Note: Since 24 digits are printed per line, the terminal head cannot be stopped while printing a line.

2) Paper feed

① Entry preparation

- Connect the CE-126 to the main unit.
- Reset the printer.
- Entry address
  A467H [A2BAH]
- Store 24 digits of spaces (20H) in registers X, Y, and Z.
- Entry address
  8054H

(9) Cassette

1) Remote on

- Entry address
  8048H

2) Remote off

- Entry address
  804BH

3) Header output

- Entry preparation
  - Store 0H in internal RAM (31H).
  - Store the file name (a maximum of 7 bytes) in operation register Z.

(Example) File name ABC12

ZReg	F5	41	42	43	31	32	00	00
		A	B	C	1	2		

Filename

If the file name is not specified, 00 is used.

ZReg	F5	00	00	00	00	00	00	00

- Entry address
  9CF5H [9B5CH]

4) Header input

- Entry preparation
  Store the file name in operation register Z.

(Example) File name ABC12

ZReg	F5	41	42	43	31	32	00	00
		A	B	C	1	2		

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{Filename}}$$

If the file name is not specified, 00 is used.

ZReg	F5	00	00	00	00	00	00	00

- Entry address
  9D1DH [9B84H]

If a file name is specified, the program keeps searching for the file until it is found. If the file is found, and asterisk is displayed at the bottom right corner of the display screen.

5) Save one character

- Entry preparation
  Store data in ACC.

- Entry address
  CFA5H [CD8DH]

Data Recording Formats

PC-1350 BASIC stores BASIC programs and data on cassette tape in various formats. This appendix section shows the tape formats supported by PC-1350 BASIC.

(1) BASIC and Reserved Program Tape Formats

The tape formats for the BASIC and reserved programs are shown below.

1. **Without a password**

	5	6	7	1	2	1	2	1	...	2	3'	3"	1	

Note: 3" is not subject to sum checking.

2. **With a password**

	5	6'	7	1	4	1	2	1	2	1	...	2	3'	3"	1	

(2) BASIC Data Tape Format

The tape format for BASIC-created data is shown below.

	5	8	7	1	11	1	9	1	...	9	1	3	10	11'	1

9'	1	9'	1	...	9'	1	3	10	11"	1	9"	1	9"

1	...	9"	1	3

Legends:

1: Check sum code
2: BASIC program (120 bytes) or reserved program (80 bytes)
3: End of file code (F0H)
3', 3": End of file code (FFH)
4: Password
5: Filler (all is recorded for 8 seconds)
6: ID code identifying a BASIC or reserved program without a password (70H)
7: ID code identifying a BASIC or reserved program with a password (71H)
8: ID code identifying memory data (74H)
9: Memory data block (8 bytes) represented by A through Z or A(n).
9': Array variable data (8 bytes)
9": Symbol variable data (8 bytes)
10: Filler (all is recorded for 2 seconds)
11: Label for a static variable (5 bytes)
11': Label for an array variable (5 bytes)
l1": Label for a simple variable (5 bytes)

(3) Machine-language Program Tape Format

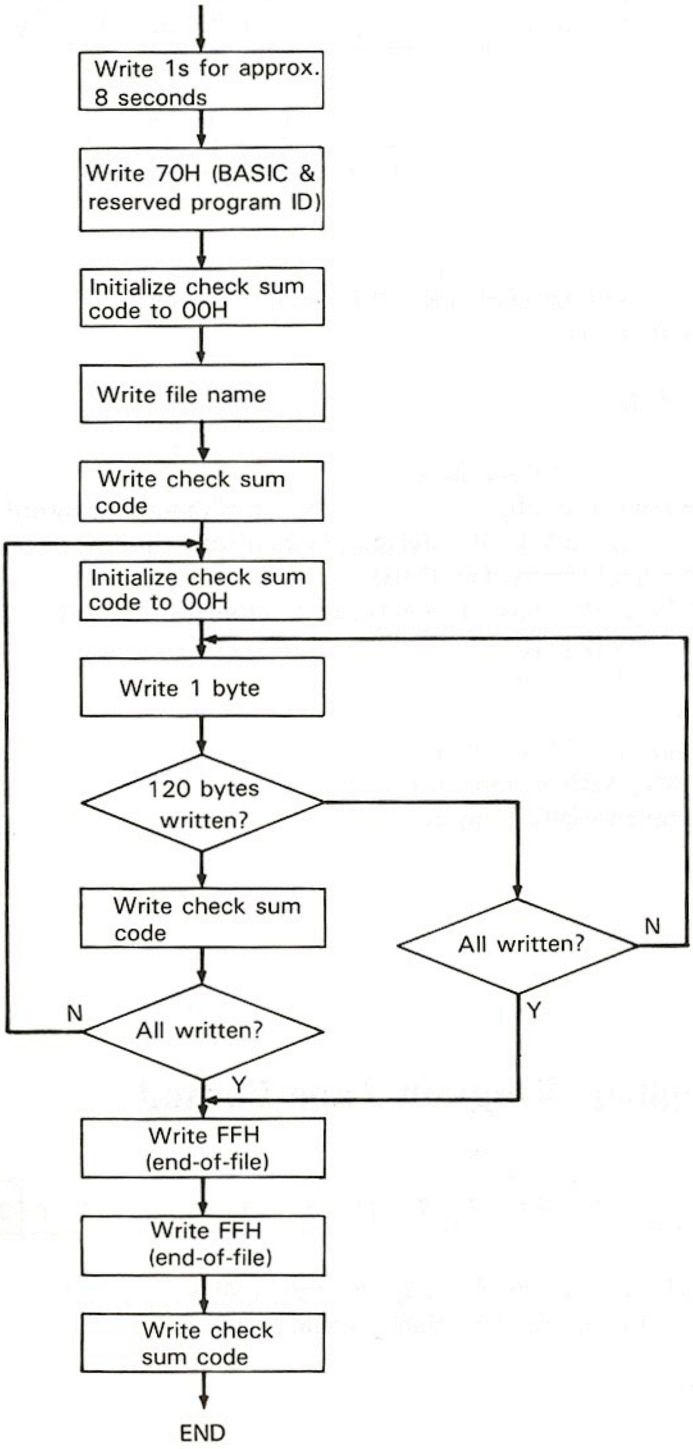
	5	12	7	1	13	1	2	1	2	1	...	2	1	3'	3"	1	

12: ID code identifying a machine-language program (76H)
13: Starting address and length of machine-language data

Recording Procedures

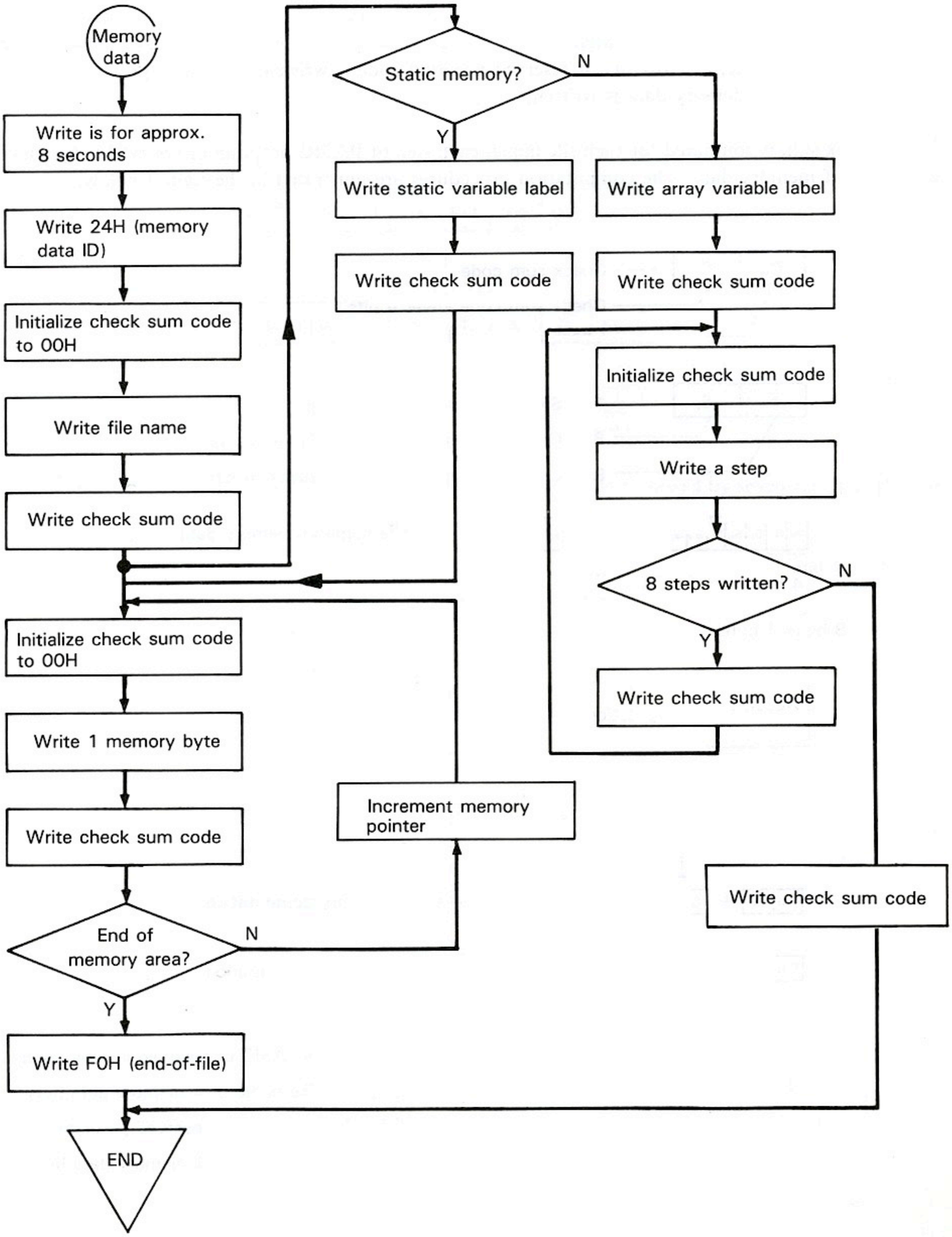
The flowcharts given below show how PC-1350 programs and data are recorded on cassette tape. By following these procedures, you could record your PC-1350-compatible programs and data using your machine-language programs.

(1) Recording BASIC or Reserved Programs



BASIC and Reserved Program Recording Flowchart

2) Recording Memory Data



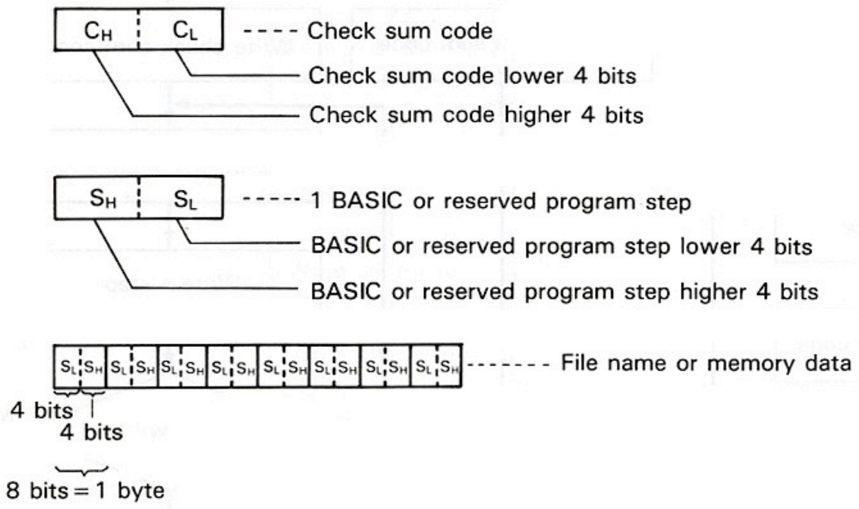
Memory Data Recording Flowchart

(3) Check Sum

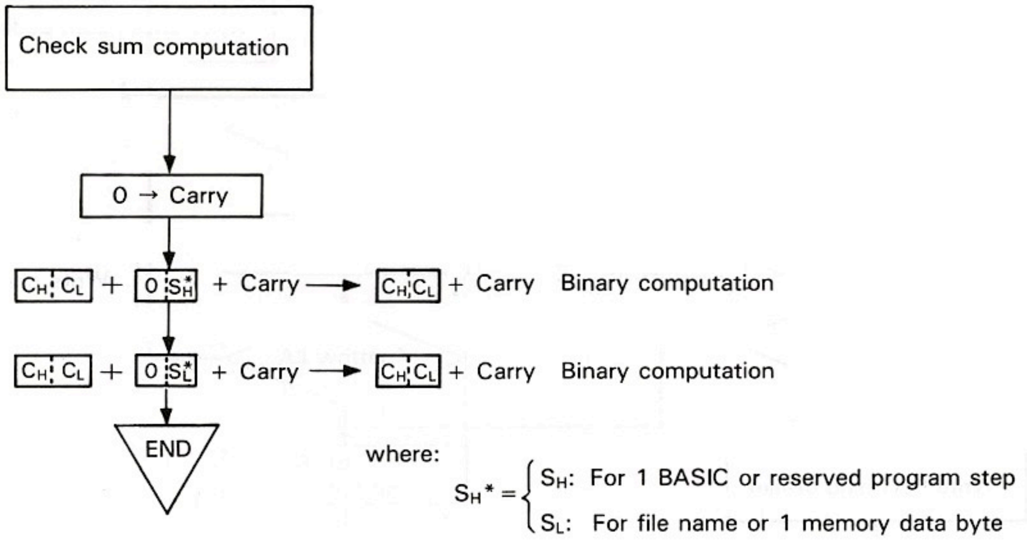
The check sum code is initialized at the following timing:

1. Before the file name is written.
2. Before 120 steps of BASIC or reserved program code is written.
3. Before any memory data is written.

The check sum is computed for each file name, each step of BASIC program or reserved program code, and each byte of memory data. The computation procedures are illustrated in the figures below.



C_H
C_L
---- Check sum code
Check sum code lower 4 bits
Check sum code higher 4 bits
S_H
S_L
----- 1 BASIC or reserved program step
BASIC or reserved program step lower 4 bits
BASIC or reserved program step higher 4 bits
S_L S_H
------- File name or memory data
4 bits
4 bits
8 bits = 1 byte



Check sum computation
0 → Carry
C_H C_L + 0 S_H^* + Carry → C_H C_L + Carry
Binary computation
C_H C_L + 0 S_L^* + Carry → C_H C_L + Carry
Binary computation
END

where:

$$S_H^* = \begin{cases} S_H\text{: For 1 BASIC or reserved program step} \\ S_L\text{: For file name or 1 memory data byte} \end{cases}$$

$$S_L^* = \begin{cases} S_L\text{: For 1 BASIC or reserved program step} \\ S_H\text{: For file name or 1 memory data byte} \end{cases}$$

(4) File Name Format

A file name consists of up to seven characters (or steps) preceded by a l-byte ID code F5H. File names shorter than seven characters are extended with codes 00H to form 7-character file names. For example, file names 'PROGRAM' and 'DATA' are recorded on cassette tape in the following formats:

'PROGRAM' Code	M	A	R	G	O	R	P	F5$_H$
	D 4	1 4	2 5	7 4	F 4	2 5	0 5	5 F

'DATA' Code	00$_H$	00$_H$	00$_H$	A	T	A	D	F5$_H$
	0 0	0 0	0 0	1 4	4 5	1 4	4 4	5 F

If no file name is specified, the default file name consisting of code F5H followed by seven zero (00H) codes is created.

(5) Memory Data Format (Static Variables)

AIl static memory variables are eight bytes long. They are recorded on cassette tape in the following formats:

1. String Variable

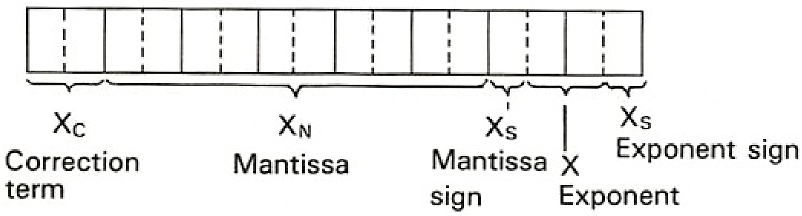
String variables are recorded in the same format as file names. For example, 'BOOK' is recorded as follows:

'BOOK' Code	00_H	00_H	00_H	K	O	O	B	$F5_H$

0 0	0 0	0 0	B 4	F 4	F 4	2 4	5 F

2. Numeric Variable

A PC-1350 BASIC numeric variable is divided into four fields as shown in the figure below.



Example:

π = 3141592654

0 0	4 5	6 2	9 5	1 4	1 3	0 0	0 0

$-123x10^{10}=-1.23x10^{12}$

0 0	0 0	0 0	0 0	0 3	2 1	8 2	1 0

$0.0789=7.89x10^{-2}$

0 0	0 0	0 0	0 0	0 9	8 7	0 8	9 9

The sign is stored in Xs. A 0 in Xs identifies a positive number and an 8 in Xs identifies a negative number. X is 2 digits long and stores the exponent portion of the number and Xs stores its sign. Numbers are stored in numeric variables in scientific notation. If the absolute value of a number is smaller than l, Xs and X are offset by a factor of 1000.

(6) Recording a BASIC Program Statement

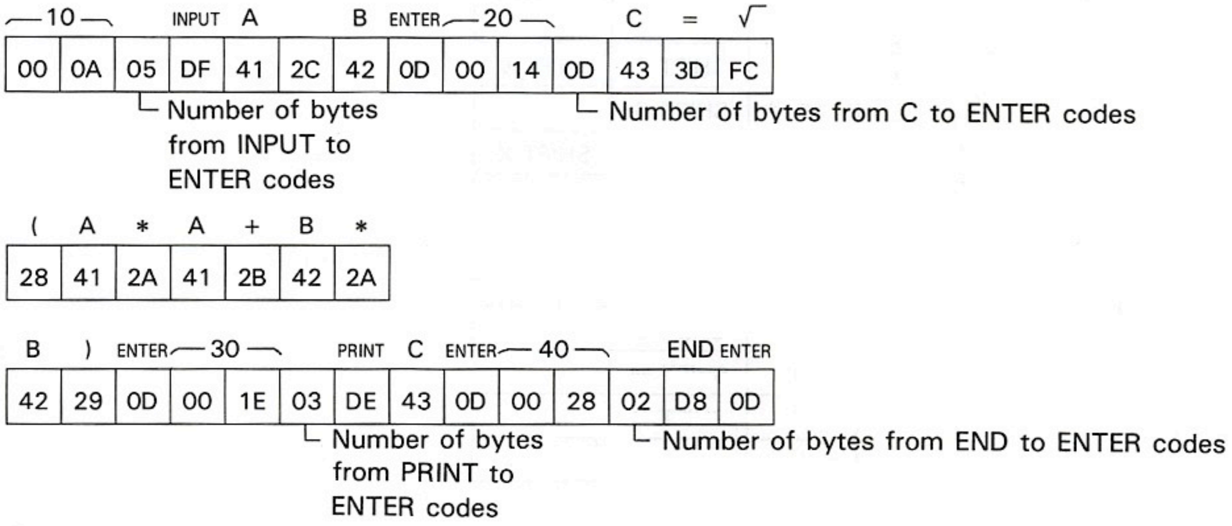
Each line number of a BASIC statement takes up 2 steps of program memory. For example, line numbers 1, 12, and 123 are stored as shown below.

Line No. 1	00	01
Line No. 12	00	0C
Line No. 123	00	78

These memory steps are followed by the number of bytes representing the statement up to an ENTER code. For example, the BASIC program code

```
10:  INPUT A,B
20:  C =√(A*A+B*B)
30:  PRINT C
40:  END
```

is stored in the BASIC program area as shown below.



10			INPUT	A		B	ENTER	20			C	=	√
00	0A	05	DF	41	2C	42	0D	00	14	0D	43	3D	FC

05: Number of bytes from INPUT to ENTER codes

0D: Number of bytes from C to ENTER codes

(	A	*	A	+	B	*
28	41	2A	41	2B	42	2A

B	)	ENTER	30			PRINT	C	ENTER	40			END	ENTER
42	29	0D	00	1E	03	DE	43	0D	00	28	02	D8	0D

03: Number of bytes from PRINT to ENTER codes

02: Number of bytes from END to ENTER codes

(7) Recording a Reserved Program

For example, the reserved program memory contains the following data when the Z key is assigned to RUN and the A key to SIN A:

SHIFT Z	RUN	SHIFT A	SIN	A			
F A	B 0	8 1	9 5	4 1			

Reserved codes such as SHIFT Z and SHIFT A occupy one byte of memory. The table below lists the PC-1350 reserved codes.

	8	**F**
0		
1	SHIFT A	SHIFT SPC
2	SHIFT B	
3	SHIFT C	SHIFT S
4	SHIFT D	SHIFT =
5		
6	SHIFT F	SHIFT V
7	SHIFT G	
8	SHIFT H	SHIFT X
9		
A	SHIFT J	SHIFT Z
B	SHIFT K	
C	SHIFT L	
D	SHIFT M	
E	SHIFT N	
F		

(8) Recording a File Name and Memory Data

File names and memory data (static memory) are recorded in the following sequences:

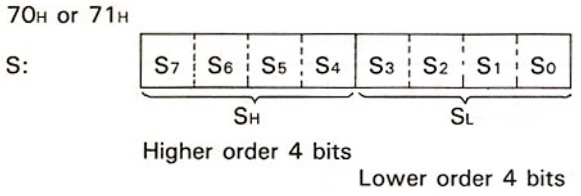
File Name: 'PROGRAM'	M	A	R	G	O	R	P	F5$_H$
	D 4	1 4	2 5	7 4	F 4	2 5	0 5	5 F

String variable: 'BOOK'	00$_H$	00$_H$	00$_H$	K	O	O	B	F5$_H$
	0 0	0 0	0 0	B 4	F 4	F 4	2 4	5 F

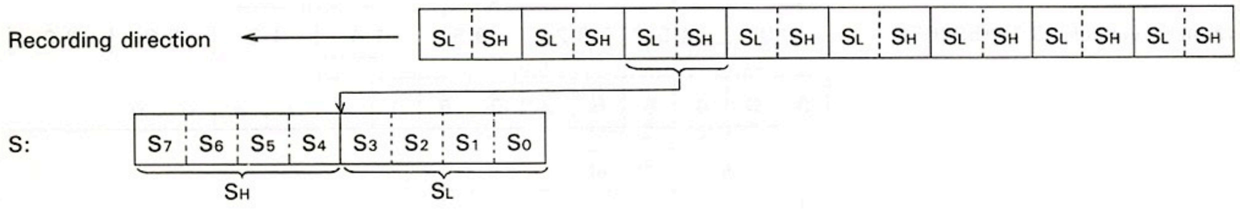
Numeric variable: 3.141592654	00	45	62	95	14	13	00	00
	0 0	4 5	6 2	9 5	1 4	1 3	0 0	0 0

(9) Recording a Data Byte

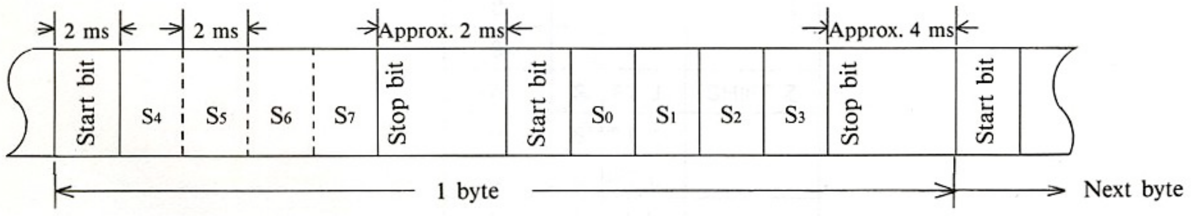
A BASIC or reserved program byte, a check sum code, a BASIC or reserved program ID code (70H or 71H), a memory data ID code (74H), and a end-of-file code (F0H or FFH) are recorded in the following format on cassette tape:



File name and memory data bytes are recorded in the following format:



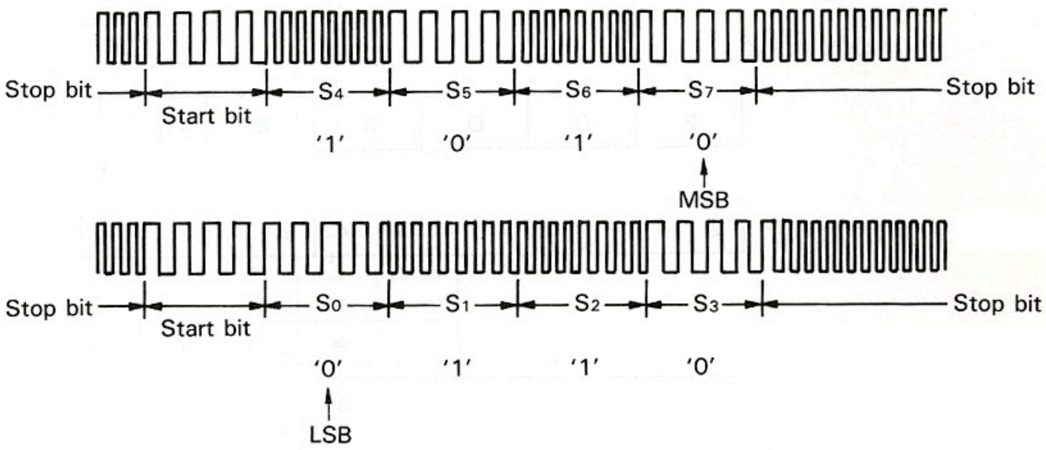
Each byte is recorded in the following format:



The interval between a start bit and the following data bit is approximately 2 milliseconds.

(10) Recording Waveform

The waveform of a recording signal is shown below. A start bit or a 0 data bit is represented by four 2-kHz pulses per the 2-ms data interval and a 1 data bit is represented by eight 4-kHz pulses per the 2-ms data interval. The figure below shows the waveform of the recording signal for the 1-byte data whose bit state is (01010110).



(11) Recording Variable Labels

1. Static variable label

Recording direction ←	E F	0 0	0 0	0 0	0 0

2. Array variable label

	TOTAL LENGTH	TOTAL LENGTH	DIM 1	DIM 2	LENGTH
Recording direction ←	L H	L H	L H	L H	L H

3. Simple variable label

The format of a simple variable label is identical to that of an array variable label except that DIM#1 and DIM#2 in the above figure are reversed.

(12) Recording Array Variable Data

One byte of array variable data is recorded in the lower-nibble-first format, which is the same as the format of static memory data. The format of a block (8 to 80 bytes long) of array variable data, however, differs from that of memory data. See the figures below.

Example:

BOOK BOOK

	B	O	O	K	B	O	O	K
Recording direction ←	2 4	F 4	F 4	B 4	2 4	F 4	F 4	B 4

π = 3141592654

Recording direction ←	0 0	0 0	1 3	1 4	9 5	6 2	4 5	0 0

(13) Recording a Password

A password is recorded in the same format as a file name. For example, the password 'PASS' is recorded as shown below.

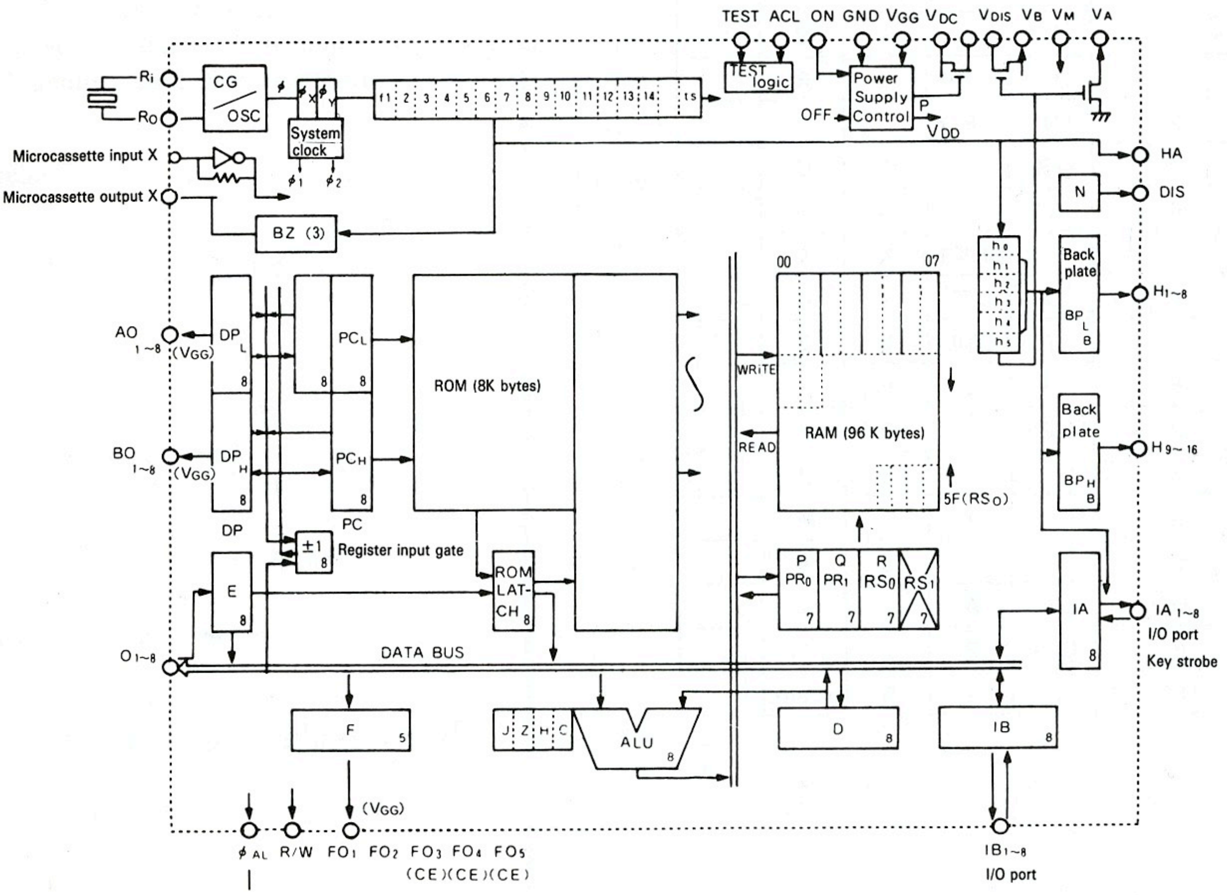
'PASS' ←	00H	00H	00H	S	S	A	P	F5H
Recording direction ←	0 0	0 0	0 0	3 5	3 5	1 4	0 5	5 F

Key Code Table

	0	1	2	3	4
0	Y	L	*	8	
1	H	ENTER	/	R	
2	N	P	(	F	
3	DEL	=	W	V	
4	INS		S	0	
5		SHIFT	X	1	
6	U	DEF	+	4	
7	J	SML	3	7	
8	M	,	6	T	
9	MODE	:	9	G	
A		:	E	B	
B	I	)	D	►	
C	K	Q	C	◄	
D	SPC	A	.	↓	
E	CLS	Z	2	↑	
F	O	–	5	ON/BRK	

CPU Internal Block Diagram and Pin Signals

CPU (SC61860A13) 8-bit C-MOS CPU



Address latch

F01 .... External ROM CE
F02 .... DOUT (external interface)
F03 .... BUSY (external interface)
F04 .... SD of SIO (SIO terminal)
F05 .... 32K byte program ROM CE

IB1 – SIO ER
IB2 – SIO RS
IB3 – SIO RR
IB4 – SIO RD
IB5 – SIO CS
IB6 – SIO CD
} SIO terminal

IB7 – ACK
IB8 – DIN
} External interface

LSI Explanation

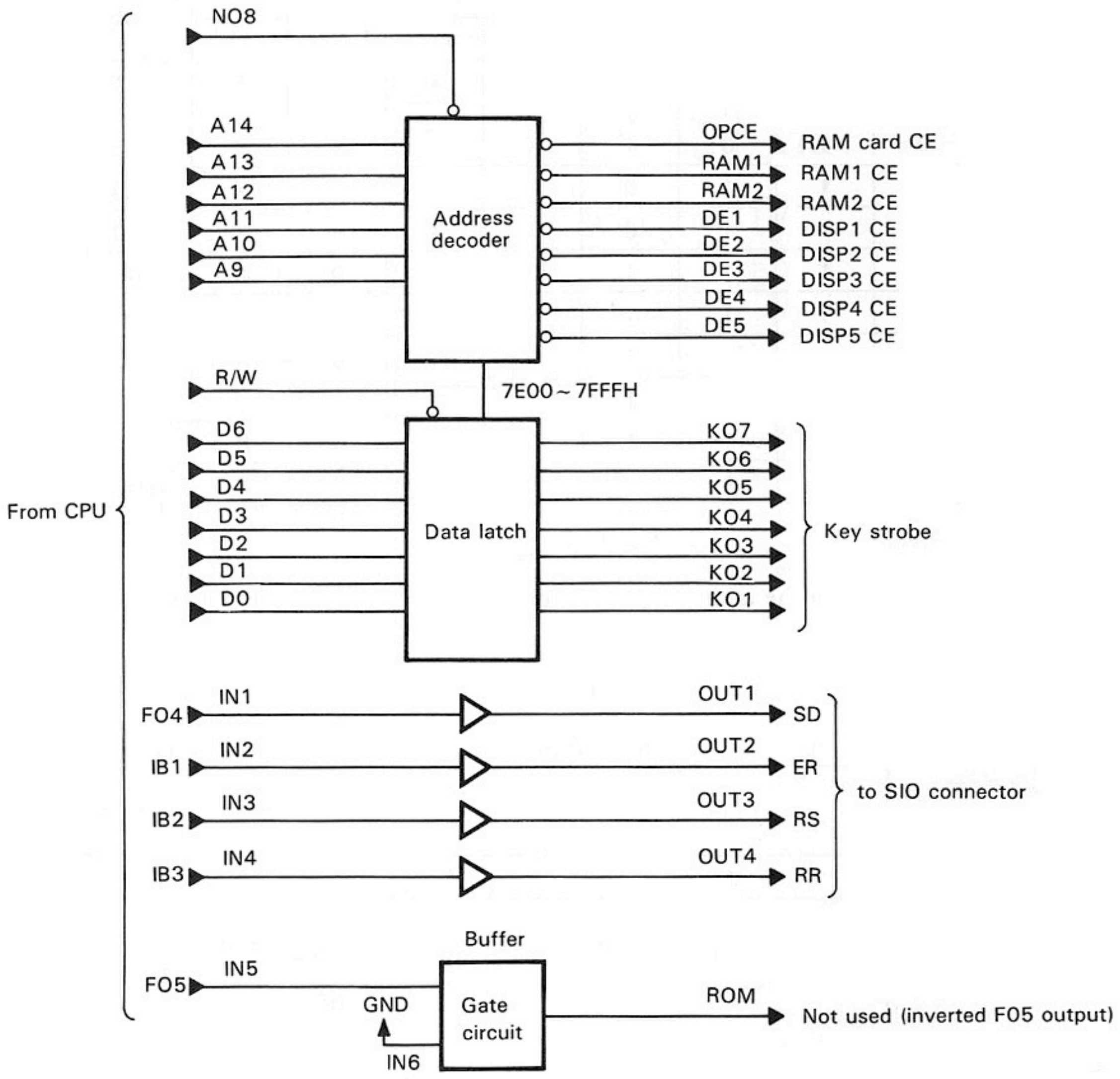
Terminal CPU signals (SC61860A13)

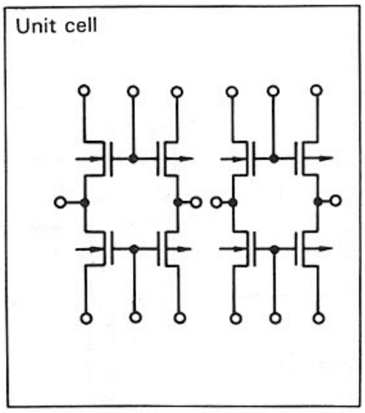
Pin number	Signal name	Input/Output	Explanation Stand-by = power off
1	A01	Output	Address bus, high du ring stand-by.
2	R/W	Output	Write clock, normally high.
3	ØAL	Output	Low order bit address latch signal. The clock is used to latch the low order 8 bits in the 16-bit address signal on the data bus line, when a large-capacity ROM is used. Normally high.
4	TES	Input	Input terminal for test purposes. Normally low.
5	Ø1	Input	Input terminal for oscillation circuit.
6	Ø0	Output	Output terminal from oscillation circuit.
7	RES	Input	Reset input terminal. Reset at high. Normally low.
8	Xin	Input	Input (MT in) for microcassette signal from CE-124 option.
9	ON	Input	ON (BREAK) key input terminal, normally low.
10	Xout	Output	Output (MT out1) for microcassette signal to CE-124 option and buzzer.
11	Dis	Output	LCD driver control signal.
12	HA	Output	LCD driver clock signal, low during stand-by. 2KHz pulse generated during display.
13	IA8	Input/Output	Key input/strobe signal, low during stand-by. Pulse is generated when key is pressed.
14	IA7	Input/Output	Key input/strobe signal, low during stand-by. Pulse is generated when key is pressed.
15	IA6	Input/Output	Key input/strobe signal, low du ring stand-by. Pulse is generated when key is pressed.
16	IA5	Input/Output	Key input/strobe signal, low du ring stand-by. Pulse is generated when key is pressed.
17	IA4	Input/Output	Key input/strobe signal, low during stand-by. Pulse is generated when key is pressed.
18	IA3	Input/Output	Key input/strobe signal, low during stand-by. Pulse is generated when key is pressed.
19	IA2	Input/Output	Key input/strobe signal, low during stand-by. Pulse is generated when key is pressed.
20	IA1	Input/Output	Key input/strobe signal, low du ring stand-by. Pulse is generated when key is pressed.
21	IB8	Input	ACK signal that enables the CPU to read data through the I/0 port (PCU).
22	IB7	Input	Serial data input signal from Din (data in) PCU (bit by bit serial handshake).
23	IB6	Input	Detection of remote transmission request from CD of SIO.
24	IB5	Input	Detection of remote acknowledgement from CS of SIO.
25	IB4	Input	Received data of RD of SIO.
26	IB3	Output	Transmission of received OK from main unit for RR of SIO.
27	IB2	Output	Transmission of main unit transmission request for RS of SIO.
28	IB1	Output	Becomes high by execution of SIO ER OPEN instruction.
29	VM	Input	LCD power supply
30	VA	Input	LCD power supply
31	GND	Input	Power supply
32	H 1	Output	LCD backplate signal, high during stand-by and 4-level pulse du ring display.
33	H2	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
34	H3	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
35	H4	Output	LCD backplate signal, high during stand-by and 4-level pulse du ring display.
36	H5	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
37	H6	Output	LCD backplate signal, high during stand-by and 4-level pulse during display ..

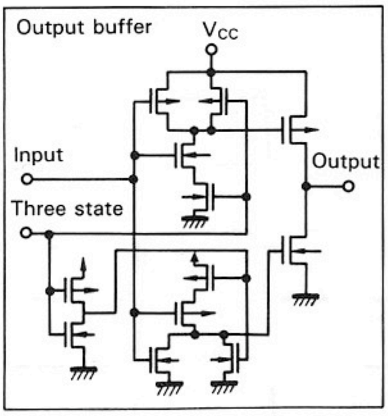
38	H7	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
39	H8	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
40	H9	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
41	H10	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
42	H11	Output	LCD backplate signal, high during stand-by and 4-level pulse du ring display.
43	H12	Output	LCD backplate signal, high during stand-by and 4-level pulse du ring display.
44	H13	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
45	H14	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
46	H15	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
47	H16	Output	LCD backplate signal, high during stand-by and 4-level pulse during display.
48	VB	Input	LCD power supply. High at stand-by. Vb at clock stop.
49	VDIS	Input	LCD power supply. High at stand-by and low when clock stops.
50	Vcc	Input	LCD power supply, always low.
51	Voc	Output	LCD power supply. High at stand-by and low when clock stops.
52	VGG	Input	Power supply, always low.
53	08	Input/Output	Data bus, normally high.
54	07	Input/Output	Data bus, normally high.
55	06	Input/Output	Data bus, normally high.
56	05	Input/Output	Data bus, normally high.
57	04	Input/Output	Data bus, normally high.
58	03	Input/Output	Data bus, normally high.
59	02	Input/Output	Data bus, normally high.
60	01	Input/Output	Data bus, normally high.
61	F05	Output	Chip enable for 32K ROM.
62	F04	Output	SD transmission data for SIO. Low at stand-by (buffering by gate array).
63	F03	Output	Busy interface output port.
64	F02	Output	Data output port Dout (data out) to peripheral.
65	F01	Output	Chip enable output for application ROM (in RAM card connector).
66	B08	Output	Enable signal of RAM, DISP-LSI, etc.
67	B07	Output	(A14) address bus line, high at stand-by.
68	B06	Output	(A13) address bus line, high at stand-by.
69	B05	Output	(A12) address bus line, high at stand-by.
70	B04	Output	(A11) address bus line, high at stand-by.
71	B03	Output	(A10) address bus line, high at stand-by.
72	B02	Output	(A9) address bus line, high at stand-by. _
73	B01	Output	(A8) address bus line, high at stand-by.
74	A08	Output	(A7) address bus line, high at stand-by.
75	A07	Output	(A6) address bus line, high at stand-by.
76	A06	Output	(A5) address bus line, high at stand-by.
77	A05	Output	(A4) address bus line, high at stand-by.
78	A04	Output	(A3) address bus line, high at stand-by.
79	A03	Output	(A2) address bus line, high at stand-by.
80	A02	Output	(A1) address bus line, high at stand-by.

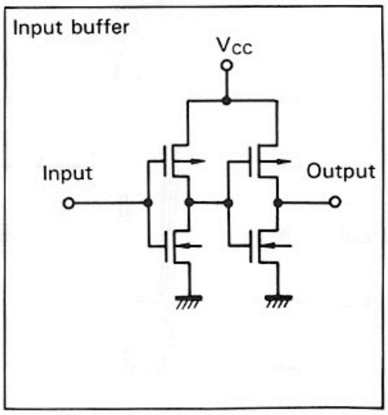
Gate Array (SC60220)

This LSI decodes CS (chip select) of various LSI's (e.g., RAM and DISP), and performs buffering of key strobe generation circuit and SIO output signals.









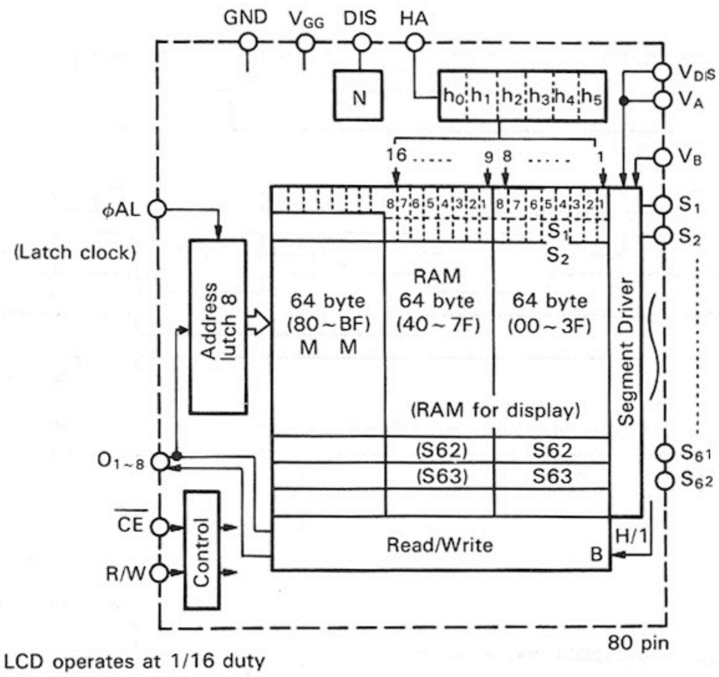
The function of the address decode is shown below.

Output	Address								Address
	B08	A14	A13	A12	A11	A10	A9	A8	
RAM card CE	0 0	0 1	1 0	0 ≀ 1	X X	X X	X X	X X	2000H ≀ 5FFFH
RAM 1 CE	0	1	1	0	0	X	X	X	6000H~67FFH
RAM 2 CE	0	1	1	0	0	X	X	X	6800H~6FFFH
DISP1 CE	0	1	1	1	0	0	0	X	7000H~71FFH
DISP2 CE	0	1	1	1	0	0	1	X	7200H~73FFH
DISP3 CE	0	1	1	1	0	1	0	X	7400H~75FFH
DISP4 CE	0	1	1	1	0	1	1	X	7600H~77FFH
DISP5 CE	0	1	1	1	1	0	0	X	7800H~79FFH
KEY port CE	0	1	1	1	1	1	1	X	7E00H~7FFFH

X: Either value

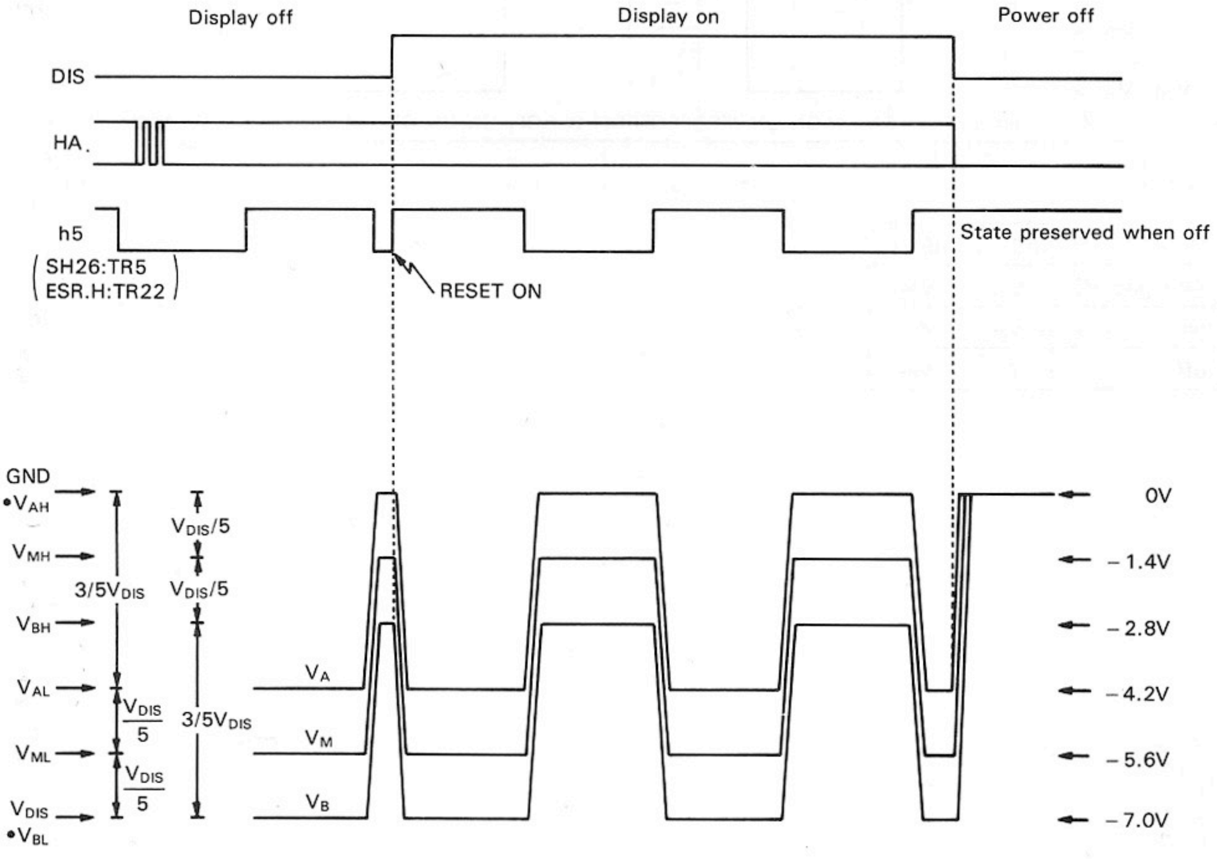
Key strobe output writes to the address space from 7E00H to 7FFFH.

Explanation of Display LSI (SC43537)



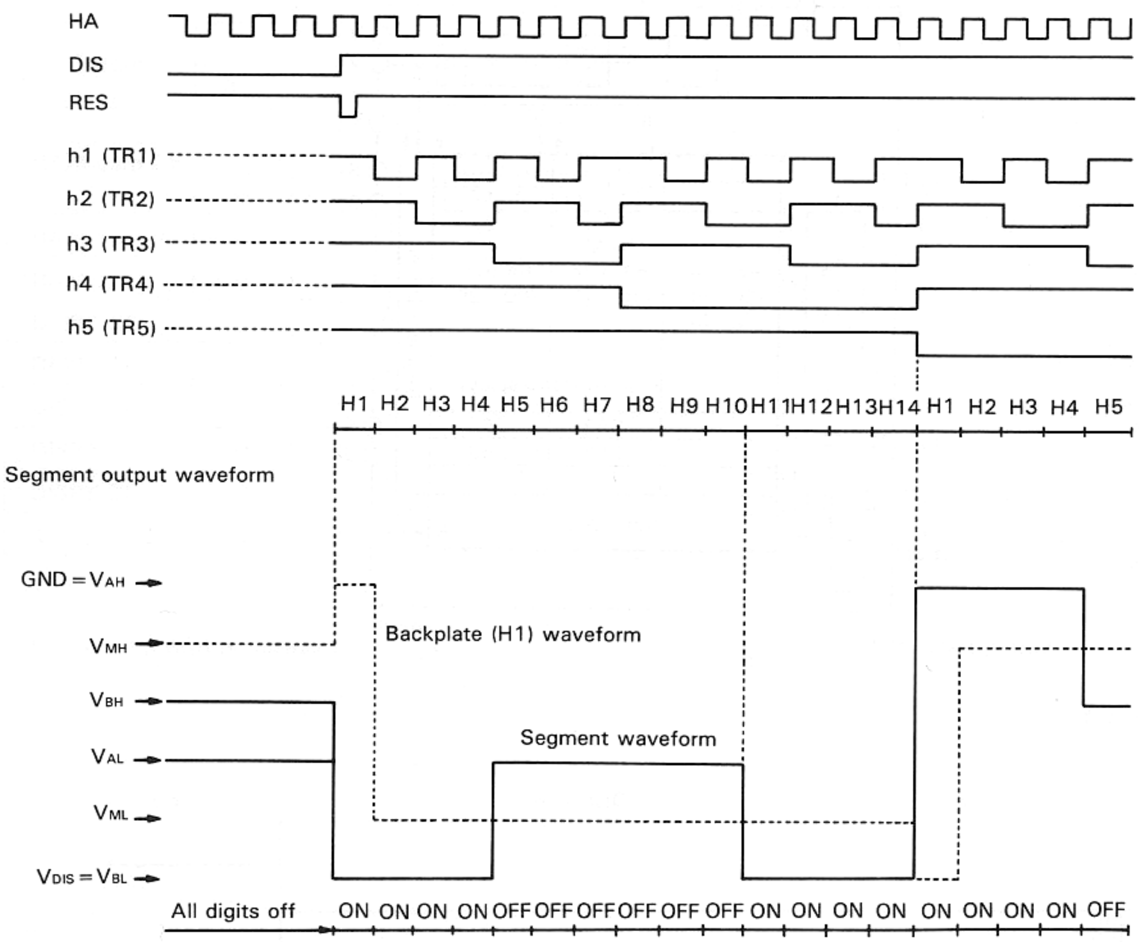
LCD operates at 1/16 duty

- **Timing diagram**



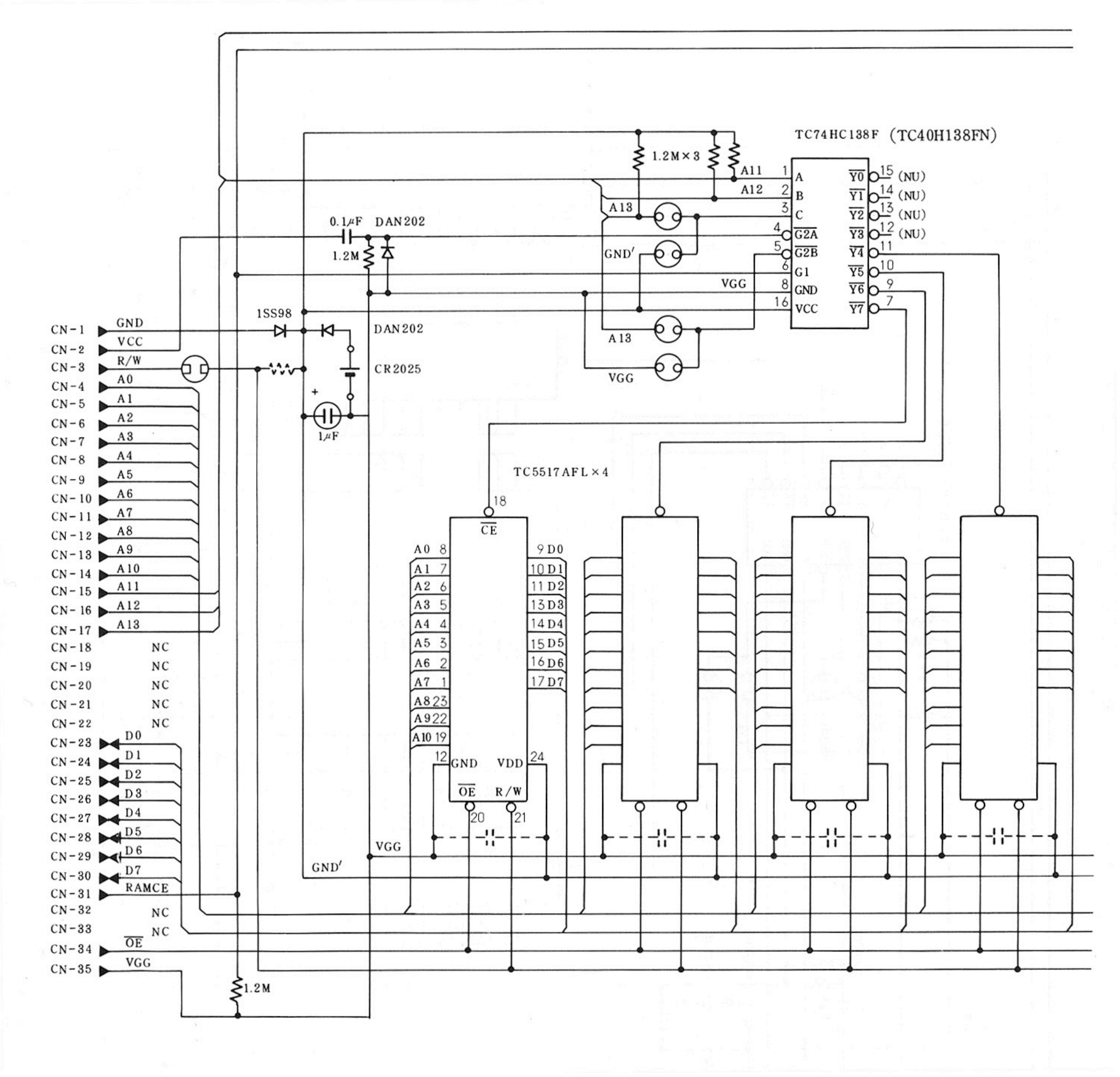
(Note) voltage level when $V_{DIS} = 7.0V$

• Counter and segment waveform

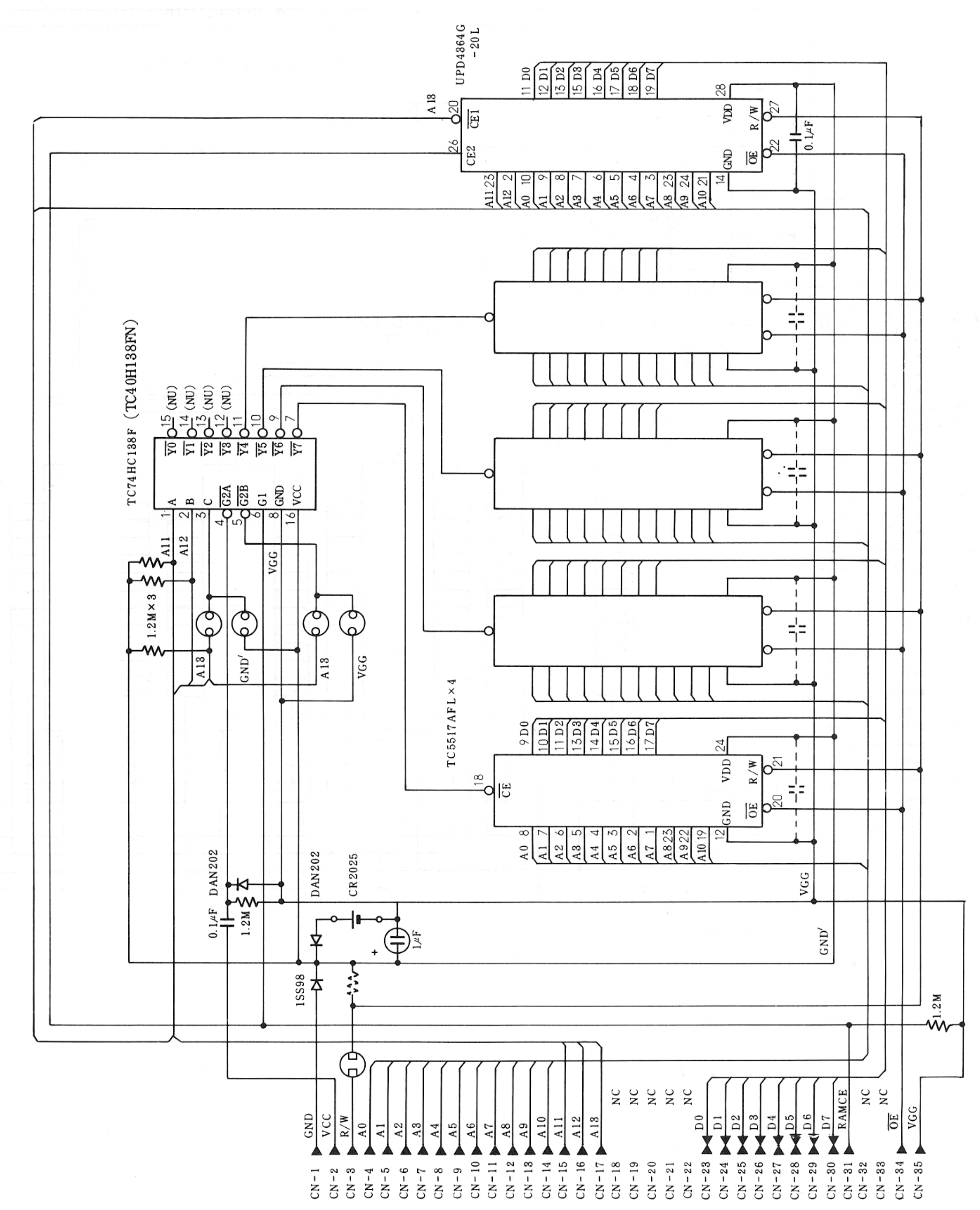


	h5=1	h5=0
All digits off	VAL	VBH
on	VBL	VAH
off	VAL	VBH

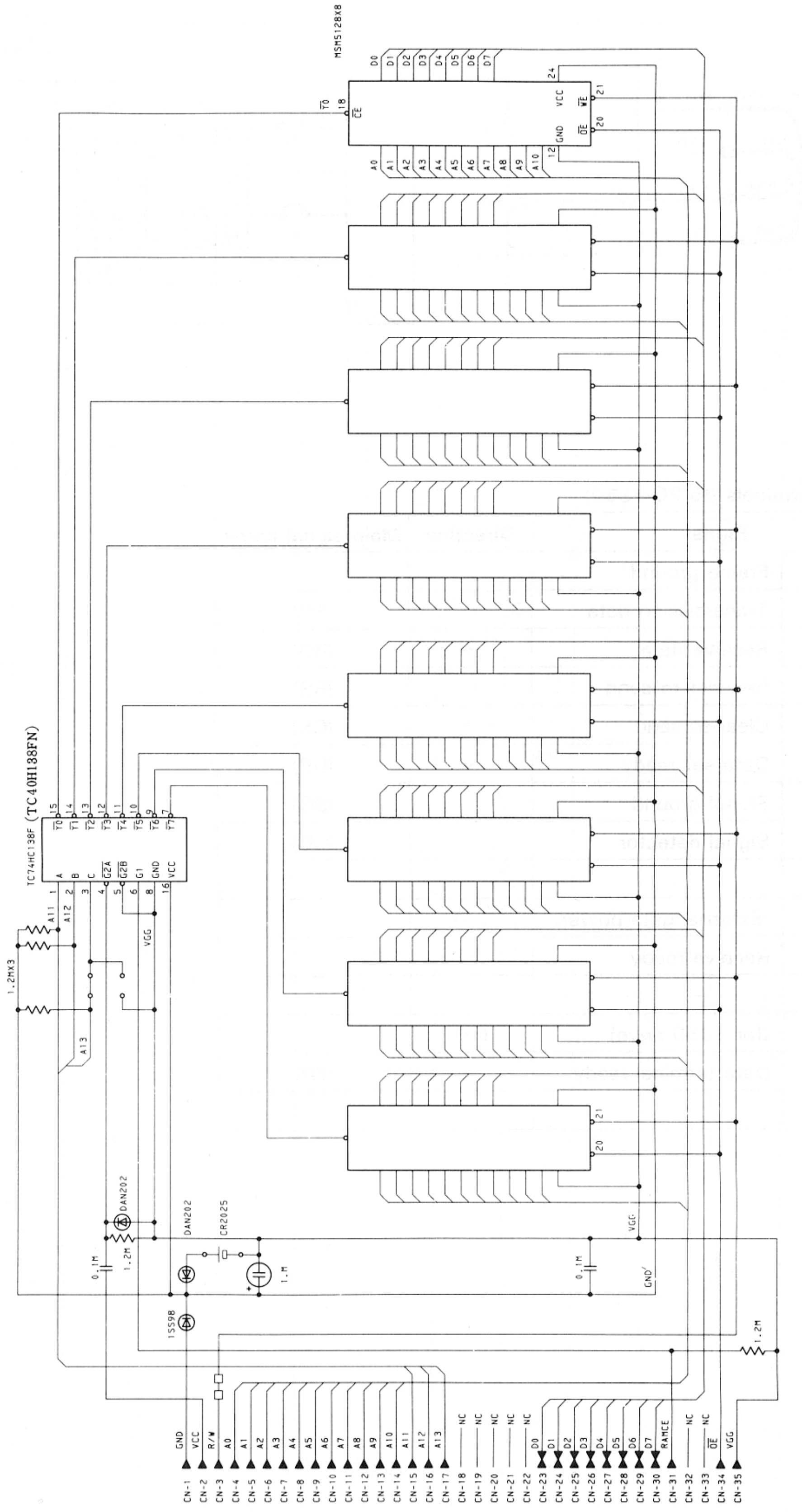
CE-201 M Circuit Diagram (flat LSI)



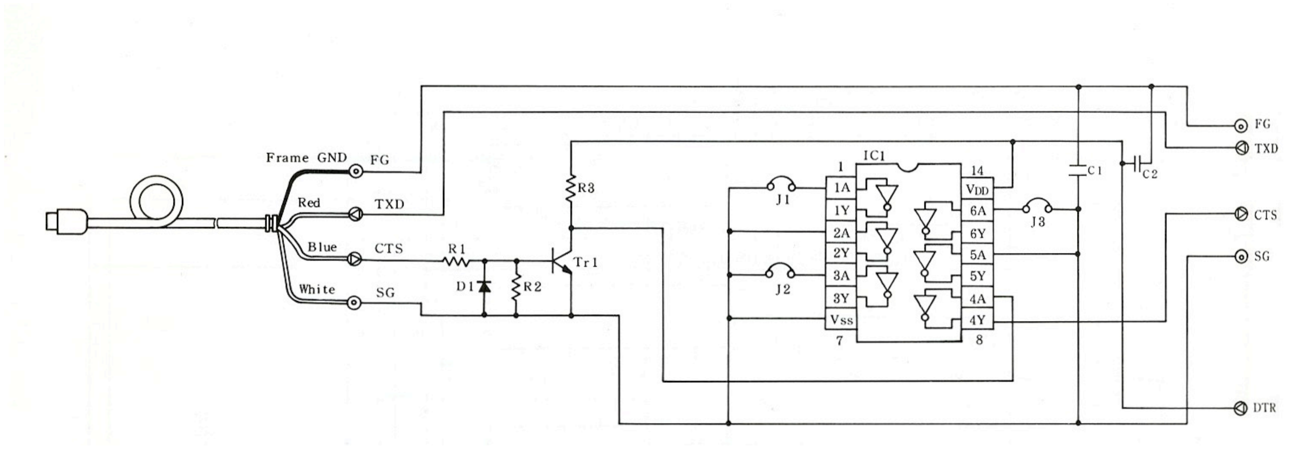
CE-202 M Circuit Diagram (flat LSI)



CE-202 M Circuit Diagram (P-COS)



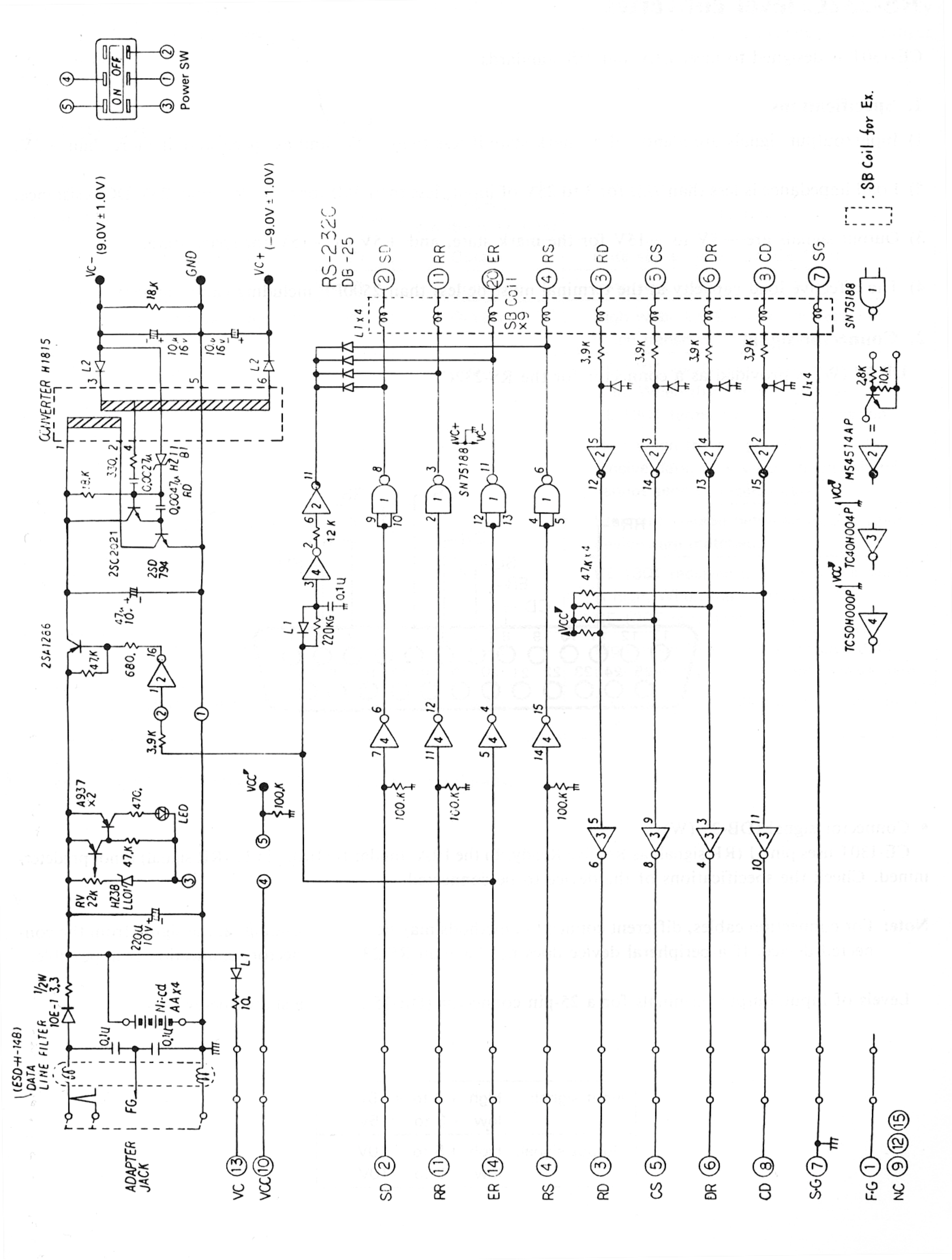
CE-516L Circuit Diagram



15-pin connector terminals (to PC-1350)

Pin No.	Signal		Direction	Main signal name
1	FG	Frame ground		
2	TXD	Transmission data	←	(SD)
3	RXD	Receive data	→	(RD)
4	RTS	Request to send	←	(RS)
5	CTS	Clear to send	→	(CS)
6	DSR	Data set ready	→	(DR)
7	SG	Signal ground		(SG)
8	CD	Signal detector	→	(CD)
9	(NC)			
10	Vcc	(External gate power)	(←)	
11	RR	Receive ready	←	
12	(NC)			
13	Vc	(for 1350 drive)	(→)	
14	DTR	Data terminal ready	→	(ER)
15	(NC)			

CE-130T Circuit Diagram



RS-232C level converter

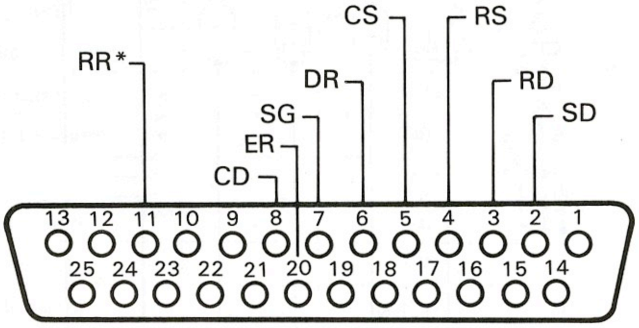
CE-l30T is designed to meet EIA and JIS standards.

1. Specifications

1) Input/output signals are identified as mark state if less than -3V, and as space state if more than +3V.
2) Load impedance is less than 7kΩ for 3 to 25V of input, less than 3kΩ for input less than 25V DC resistance.
3) Output signals are -5V to -15V for the mark state, and +5V to +15V for space state.
4) The effective load capacity at the terminal must be less than 2500PF including cable capacity.

2. Connector signal

DB-25 (W) is provided as a connector for the RS-232C.



• Connector signal (DB-25 (W))

CE-l30T uses pin 11 (RR signal) as Receive Ready. In the EIA standards (JIS), pin 11 (RR signal) is not predetermined. Check the specifications of the device to be connected.

Note: For connection cables, different connection methods may be used depending on the signal from the connected device. If a peripheral device does not have an RS-232C connector, connection is impossible.

Levels of input/output terminals for a 25-pin connector (DB-25 (W)) are shown below.

Input signal:	high +3 to +15V low -3 to -15V
Output signal:	high +5 to +10V low -5 to -10V

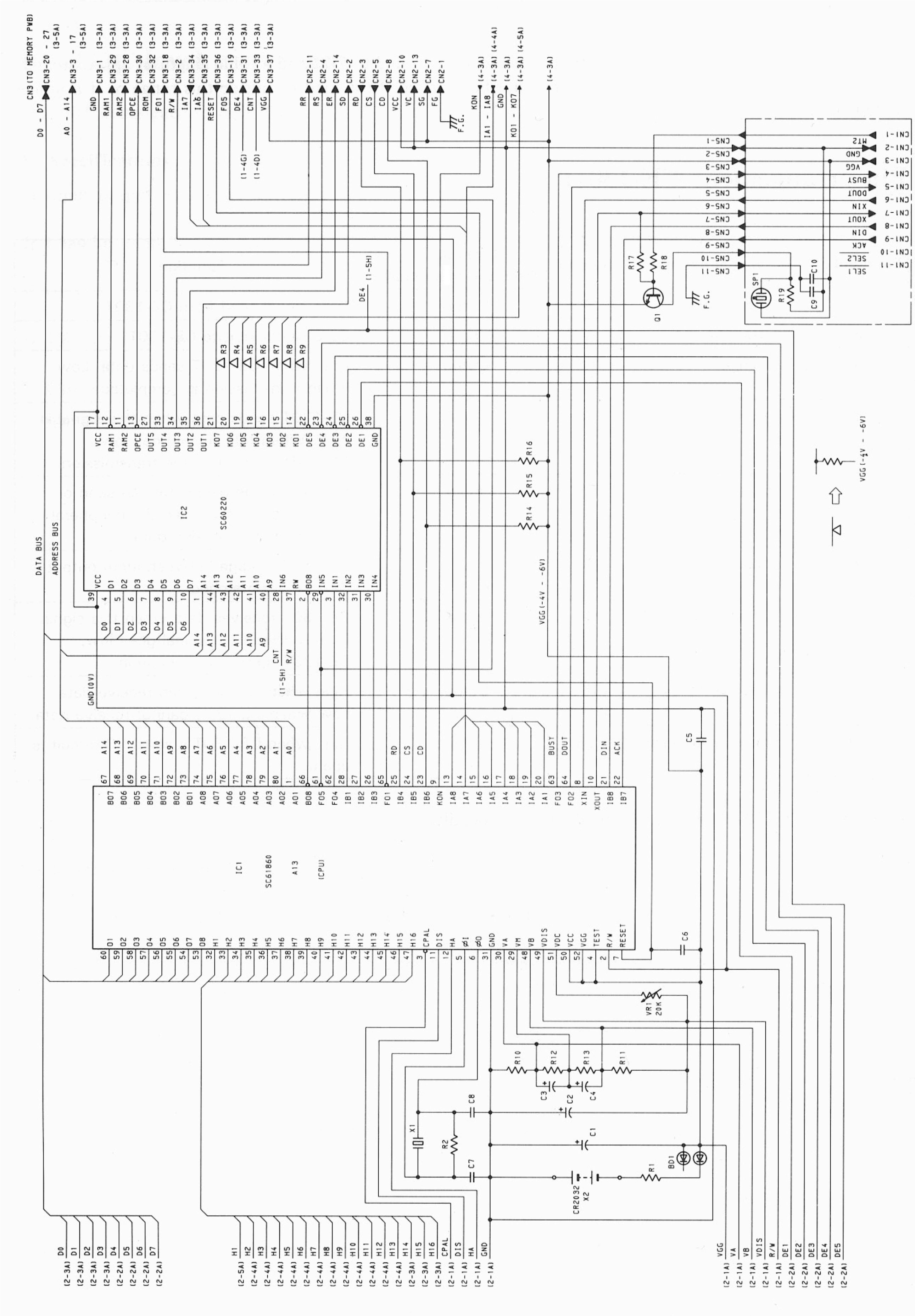
The values were obtained using an output signal load of from 3 to 7KΩ and a cable length of approximately 1 meter. Therefore, the above conditions may not be satisfied if the load is outside this range or if a longer cable is used.

Pin arrangement CE-130-T

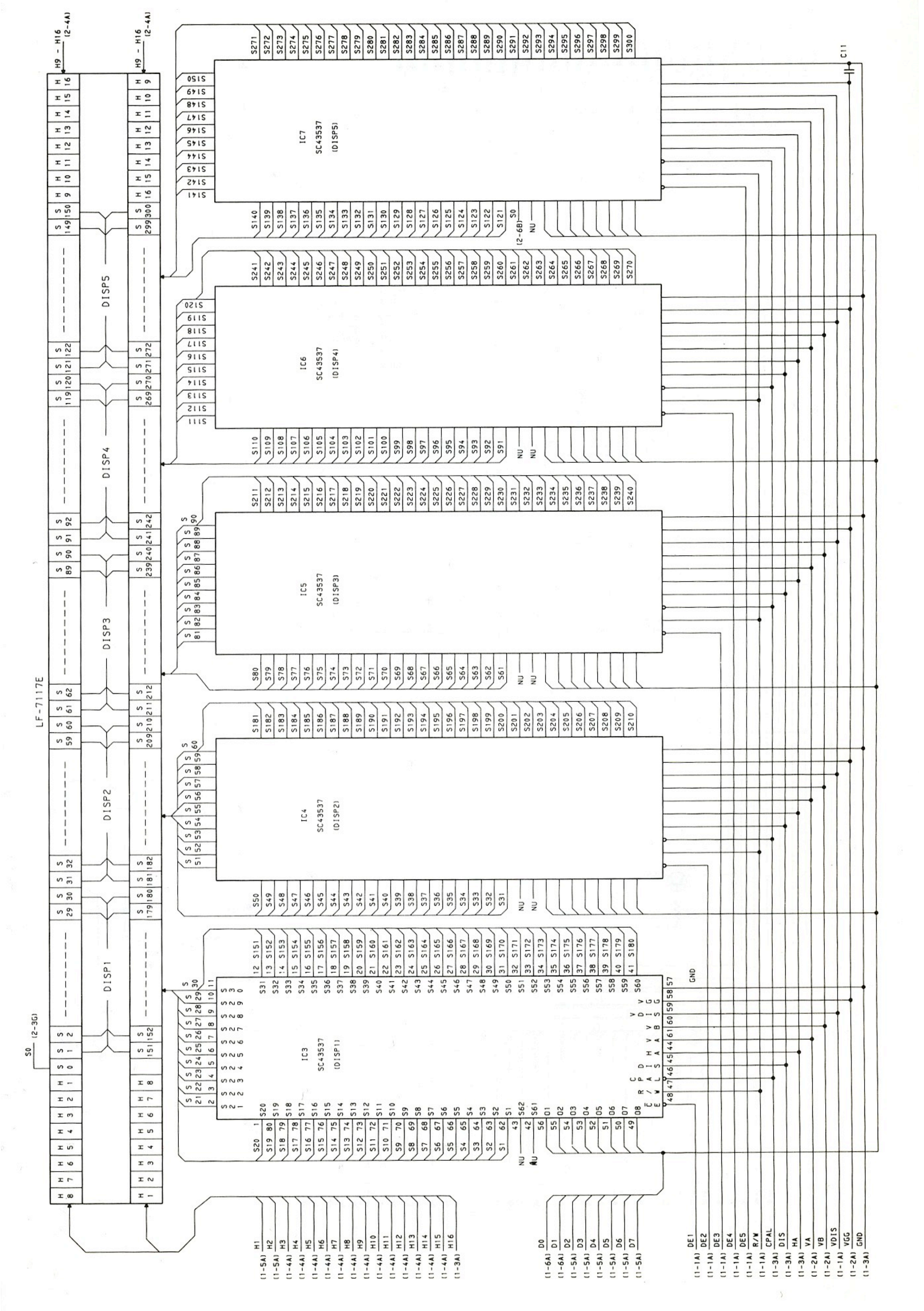
Pin number	Signal name	Symbol	Direction (from the main Unit)	Function
2	Send data	SD	Output	Data signal send from CE-130T
3	Receive data	RD	Input	Data signal sent to CE-130T
4	Transmission request	RS	Output	High when CE-130T sends data. Low when transmission is completed.
5	Transmission possible	CS	Input	CE-130T sends data when this signal is high. When this signal goes low, C E-1 30T terminates the transmission.
6	Data set ready	DR	Input	High when the peripheral can send or receive data. Low when the peripheral cannot send or receive data.
7	Signal ground	SG		Standard voltage between input/output devices are matched.
8	Carrier detection	CD	Input	CE-130T receives data when this signal is high. When the signal goes low, CE-130T terminates reception.
11	Receive ready	RR	Output	High when CE-130T can receive data. Low when CE-130T cannot receive data.
20	Data terminal ready	ER	Output	High when CE-130T's serial I/O circuit is open.

Refer to the CE-l30T Operation Manual for more details.

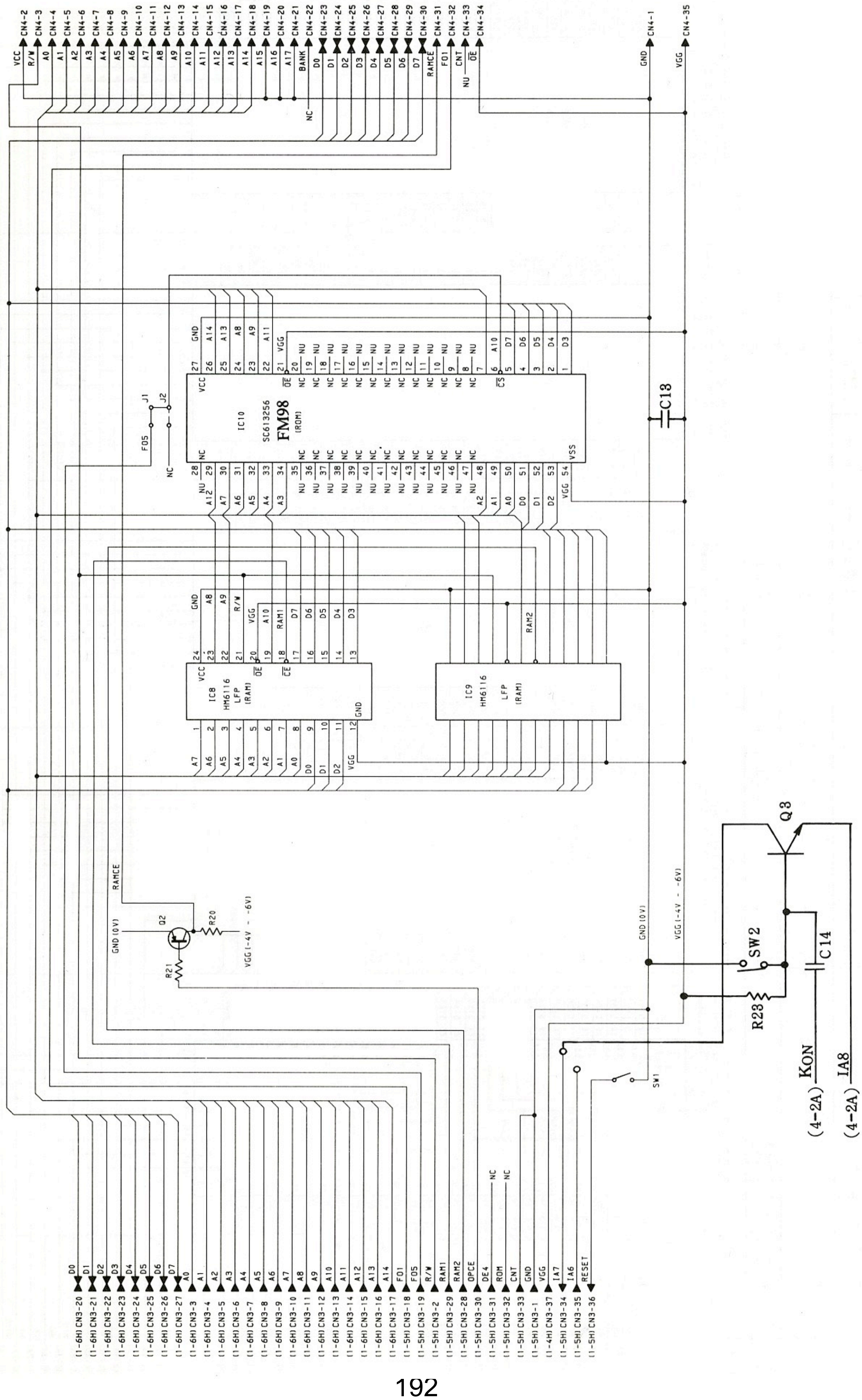
CIRCUIT DIAGRAM (1. PC-1350 CPU Circuit)



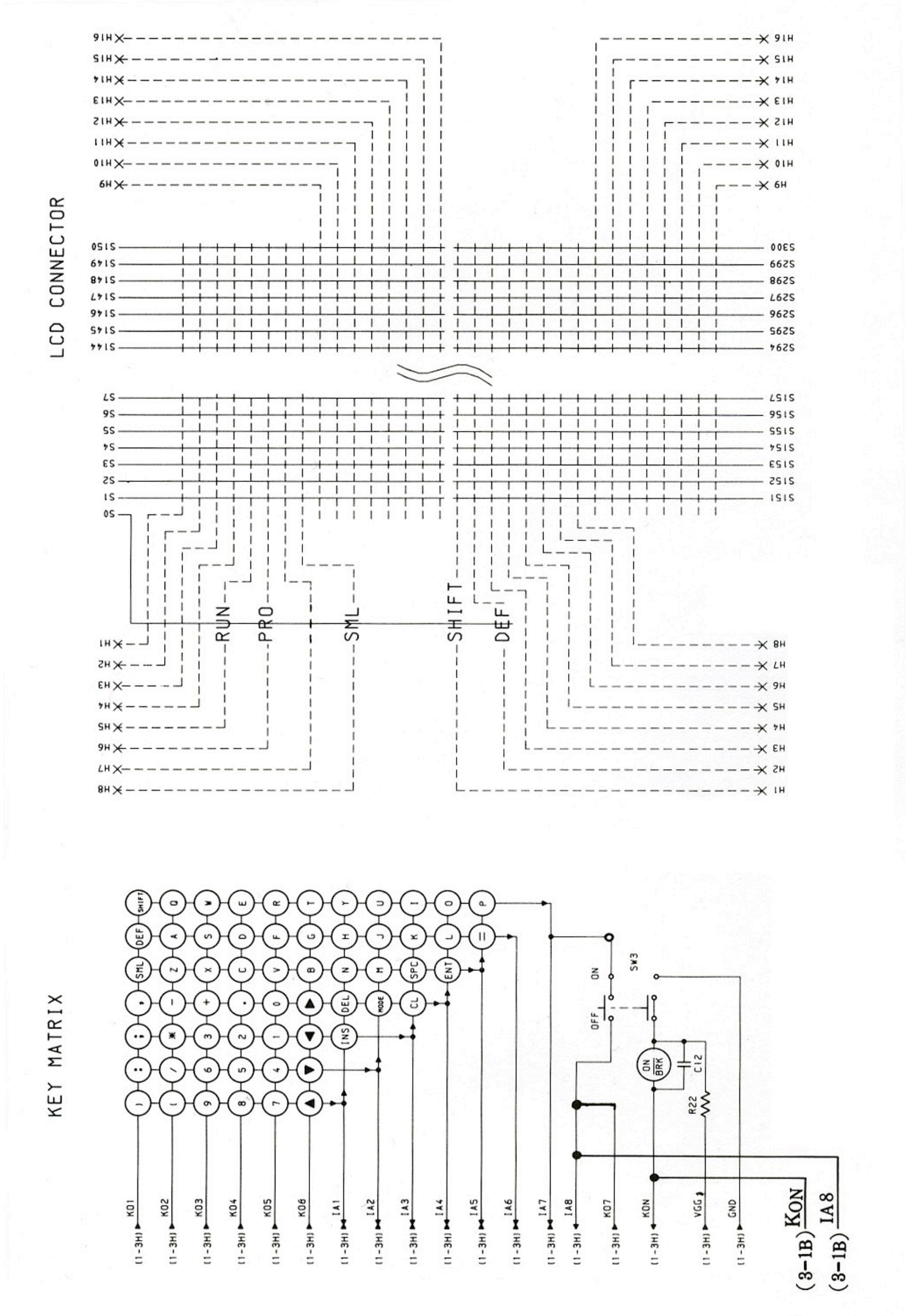
CIRCUIT DIAGRAM (2. PC-1350 Display Unit)



CIRCUIT DIAGRAM (3. PC-1350 Memory Circuit)



CIRCUIT DIAGRAM (1. PC-1350 Key/LCD Matrix)


LCD CONNECTOR
H16
H15
H14
H13
H12
H11
H10
H9
S150
S149
S148
S147
S146
S145
S144
S7
S6
S5
S4
S3
S2
S1
S0
S300
S299
S298
S297
S296
S295
S294
S157
S156
S155
S154
S153
S152
S151
RUN
PRO
SML
SHIFT
DEF
H1
H2
H3
H4
H5
H6
H7
H8
KEY MATRIX
SHIFT
Q
W
E
R
T
Y
U
I
O
P
DEF
A
S
D
F
G
H
J
K
L
=
SML
Z
X
C
V
B
N
M
SPC
ENT
,
−
+
.
0
▶
DEL
MODE
CL
;
*
3
2
1
▼
INS
:
/
6
5
4
◀
)
(
9
8
7
▲
ON
OFF
SW3
ON BRK
C12
R22
KO1
KO2
KO3
KO4
KO5
KO6
IA1
IA2
IA3
IA4
IA5
IA6
IA7
IA8
KO7
KON
VGG
GND
(1-3H)
KON
IA8
(3-1B)
(3-1B)