CASIO PERSONAL COMPUTER

FP-200

TECHNICAL BOOK

MACHINE LANGUAGE AND COMMUNICATIONS

CASIO®

CASIO PERSONAL COMPUTER

FP-200

TECHNICAL BOOK

MACHINE LANGUAGE AND COMMUNICATIONS

PREFACE

The fundamental principle employed throughout this book in the learning of machine language and RS-232C communications is that the reader is required to create test programs for himself in order to accumulate practical knowledge step by step.

Part 1 of this book is entitled "A 8085 Machine Language primer". It encourages the reader to solve problems through experimentation. Part 2, entitled "RS-232C Communications", describes the methods of connecting the FP-200 with other personal computers via an RS-232C interface. It is important to master the techniques of RS-232C communications, since the FP-200 can be used as an efficient data entry machine. Once RS-232C communication techniques have been mastered, it is possible to collect data with an FP-200 in the field and then process the data on another personal computer back in the office. The authors hope this book will help the reader to make the most of his FP-200 and achieve a deeper understanding of computers in general.

April 1984

CONTENTS

Part 1. A 8085 Machine Language Primer

1. Introduction *10*
2. Why learn a machine language now? (Machine language processing speed) *12*
   1) Why use machine language? *12*
   2) Machine language processing speed *12*
3. Use of binary, decimal and hexadecimal numbers *17*
   1) Binary numbers and decimal numbers *17*
   2) Hexadecimal numbers *18*
   3) BASIC and decimal numbers *18*
   4) Getting accustomed to hexadecimal numbers *19*
4. Creating a disassembler *24*
   1) Memory dump requirements *24*
   2) Input routine problems *26*
   3) Pattern 1: Programming using INPUT *27*
   4) Pattern 2: Programming using INKEY$ *30*
   5) Comparison between Pattern 1 and Pattern 2 *33*
   6) Program to write machine language in memory using POKE *38*
5. A look at the inside of the FP-200 *40*
6. Basic principles of computer operation *43*
   1) Logic circuit functions *43*
   2) Stored programs and microcomputers *44*
7. Functions of registers and counters *47*
   1) Working register *48*
   2) Program counter *49*
   3) Stack pointer *50*
8. Instruction formats *51*
9. 14 basic machine language instructions *53*
   1) REGISTER TO REGISTER TRANSFER instruction *53*
   2) IMMEDIATE instruction *54*

3) LOAD instruction *54*
4) STORE instruction *55*
5) ADD instruction *56*
6) SUBTRACT instruction *56*
7) INCREMENT instruction *56*
8) DECREMENT instruction *57*
9) UNCONDITIONAL JUMP instruction *57*
10) CONDITIONAL ZERO JUMP instruction *57*
11) SUBROUTINE CALL instruction *58*
12) SUBROUTINE RETURN instruction *58*
13) NO OPERATION instruction *58*
14) HALT instruction *59*

10. Writing programs in machine language *60*
1) Creating machine language programs *60*
2) Subroutines *63*
3) ROM MAP *64*

Part 2. RS-232C Communications

1. **Foreword** *68*
2. **Basic RS-232C information** *69*
1) RS-232C connection *69*
2) RS-232C electrical characteristics *70*
3) Mode of synchronization *72*
4) Data transmission rate *72*
5) Character configuration *72*
6) Start bit *73*

7) Data length *73*
8) Parity bit *74*
9) Stop bit *75*
10) Connection cables *75*
11) Connectors *76*
12) Pin layout *77*
13) Send data SD (TXD) *78*
14) Receive data RD (RXD) *79*
15) Request to send RS (RTS) *79*
16) Clear to send CS (CTS) *79*
17) Data set ready DR (DSR) *80*
18) Data carrier detect CD (DCD) *80*
19) Data terminal ready ER (DTR) *80*

3. **Basic information about communication modes** *83*
4. **Preparation for FP-200 serial data communication** *85*
   1) RS-232C terminal on the FP-200 *85*
   2) Connection cables *86*
   3) Communication protocol *86*
   4) Data transmission/reception timing *86*
   5) Preparation for serial data communication (hardware) *87*
5. **FP-200 serial data communication software** *88*
   1) Transmitting a program – SAVE "COM 0:", A *89*
   2) Receiving a program – LOAD "COM 0:" *90*
   3) Data transmission – OPEN "COM 0:" FOR OUTPUT AS #n *90*
   4) Data reception – OPEN "COM 0:" FOR INPUT AS #n *91*
   5) Summary of serial data communications (software) *92*
6. **Communication between an FP-1000/1100 and an FP-200** *93*
   Connecting the FP-200 to an FP-1000/1100 *94*
7. **Program transfer between an FP-200 and an FP-1000/1100** *98*
   Transferring programs between an FP-200 and an FP-1000/1100 *99*
8. **FP-200 key data communication** *101*
   1) FP-200 key data transmission program *102*
   2) FP-200 key data reception program *103*

3) FP-200 key data transmission/reception program *104*
4) FP-1000/1100 key data transmission program *106*
5) FP-1000/1100 key data reception program *107*
6) FP-1000/1100 key data transmission/reception program *108*
9. **RS – 232C signal interface** *109*
1) Signal interface 1 : Special connector assembly *109*
2) Signal interface 2 : Connector with alligator clips *110*
3) Signal interface 3 : Jumpered universal signal interface *110*

Part 1. A 8085 Machine Language Primer

Introduction

For what purpose did you purchase your FP-200? To perform office work more efficiently? To keep up with the computer age?
Whatever the reason, you ought to be well aware of the fact that without software, a computer is of little use. Unfortunately, not all computers are now used to their full capacity. Many offices, having installed expensive small business computers, depend on outside software firms for their applications because of the lack of staff capable of understanding software.
At the time when you purchased your FP-200, you must have had various applications in mind. At the same time, you might have thought that it would be easy to master the BASIC language, as well as the simple language called CETL available with the FP-200. In reality, however, you may already have encountered problems such as difficulties involved in using BASIC, insufficient memory capacity, slow processing speed, limited capabilities of CETL, and so on.
One approach to solving these problems is to master the FP-200 machine language.
As is well known, BASIC has become the most widely used computer language. Recently, however, users who wish to master machine language in addition to BASIC are increasing in number. Perhaps you are one of those users. Why is BASIC, which is a high-level language, insufficient? The answer lies in its processing speed, as described in detail in later sections.
Machine language is often considered difficult to understand, but is really not so difficult if it is learned step by step.
Part 1 is intended to help the reader to learn the FP-200 machine language and the fundamentals of programming in this language using the FP-200.

The FP-200 is a so-called hand-held computer. Relatively inexpensive and easy to carry, the FP-200 has much greater performance than conventional desk calculators. Given sophisticated software, the FP-200 can be used in a wide variety of applications.
Master the FP-200 machine language and you will be able to make the most of your FP-200.

2

Why learn a machine language now?
(Machine language processing speed)

1 Why use machine language?

Before discussing this question, let us review some of the advantages and disadvantages of machine language from various aspects.

(1) Machine language cannot be used without thorough understanding of both the hardware and software of computers.
(2) Machine languages are difficult to learn and difficult to use.
(3) Machine languages have evolved into assemblers, compilers, interpreters, and simplified languages. Why do we have to use a machine language now?
(4) Machine languages are apt to cause programming errors which are difficult to debug.
(5) Once a program written in a machine language overruns, it becomes unmanageable.
(6) Machine language cannot be used interchangeably among different computer models.
(7) Machine languages offer higher processing speeds and require smaller memory capacity.

2 Machine language processing speed

Judging from the apparent merits and demerits enumerated above, it seems that machine language has more demerits than merits. Then why use machine language? As mentioned earlier, the answer to this question lies in its processing speed.

Is the processing speed of a machine language really high? If so, how fast? As an example, let us compare the processing speed of the FP-200 machine language with that of BASIC by an experiment.

The method of experimentation will be to compare the time required to draw on the entire liquid crystal display of the FP-200 for machine language and BASIC.

The sample programs for the experiment are shown below. Since the FP-200 has multiple programs, the BASIC program is loaded in PROG 0 and the machine language program is loaded in PROG 1 so that the comparison can be made any number of times.

BASIC program (PROG 0), example 1

```
10 INIT(0,0),1,1
20 CLS
30 X=0:Y=0
40 FOR J=0 TO 62
50   FOR I=0 TO 150
60   DRAW(X,Y)
70   X=X+1
80   NEXT I
90 X=0
100 Y=Y+1
110 NEXT J
120 END
```

Machine language program (PROG 1), example 1

```
POKE 40704,205
POKE 40705,142
POKE 40706,2
POKE 40707,5
POKE 40708,194
POKE 40709,0
POKE 40710,159
POKE 40711,6
POKE 40712,19
POKE 40713,13
POKE 40714,194
POKE 40715,0
POKE 40716,159
POKE 40717,201


10 CLEAR,40703
20 CALL 40704,255,0,0,4927
30 END
```

Now, execute the programs. You will find that the BASIC program takes approximately eight minutes to perform the task, while the machine language program takes less than one second.
Assuming that BASIC takes just eight minutes, or 480 seconds, and the machine language takes 0.5 second, the processing speed of the machine language is approximately 1,000 times faster than that of BASIC. Even if the machine language takes one second, it is still 500 times faster.
Let us make the comparison more accurately.
Look at the following sample programs. They use the TIME function (available in FP-200 BASIC) at the beginning and at the end (before the END) so that the starting time and the ending time can be checked.

BASIC program (PROG 0), sample 2

```
BASIC EX-1
PROG 0
 10 INIT(0,0),1,1
 15 T1$=TIME$
      (
      )
110 NEXT J
115 T2$=TIME$
116 PRINT T1$,T2$
120 END
```

Machine language program (PROG 1), sample 2

```
MACHINE,L EX-1
PROG 1
POKE 40704,205


 10 CLEAR,40703
 15 T1$=TIME$
      (
      )
 25 T2$=TIME$
 26 PRINT T1$,T2$
 30 END
```

Now, execute these programs. This time, the times required for execution are displayed: 7 minutes and 46 seconds, or 466 seconds, for BASIC, and 1 second for machine language. We find that the machine language is approximately 500 times faster than BASIC.

Now you see the reason why machine language is attracting increasing attention, despite the fact that it has a number of disadvantages in other respects.

Incidentally, an experiment is only half done unless the results of the experiment are verified. In order to verify the above-mentioned results, some of the questions which may occur to you, and the appropriate answers, are given below.

Q1: I'm very doubtful as to the processing time of one second for the machine language program. Isn't there any method of measuring the processing time more accurately?

A1: The TIME function of the FP-200 counts the time in increments of one second. Hence, in order to measure the time more accurately, it is necessary to use either of the following methods:

(1) Based on the result of analysis of the TIME function internal mechanism, create software which permits setting more minute time slices. Note, however, that this method requires a thorough knowledge of the BASIC internal mechanism and of machine language.

(2) Repeat the same process 10 times, then divide the total time by 10. This method is relatively easy, although some modification to the machine language is required. This is described later in more detail.

Q2: I think that line number 10 INIT (0, 0), 1, 1 in both BASIC programs is unnecessary.

A2: Exactly. It is unnecessary in this case. The INIT statement is conveniently used when determining a scale or setting coordinates in graphic applications.

Q3. Is it possible to replace the (X, Y) in the DRAW statement with I, J in the FOR statement? It will reduce the length of the entire program, and hence the execution time will beocme shorter.

A3: It is possible. It certainly will make the program easier to read. Try the same experimentation using the program in which I, J is substituted for X, Y.

Q4: In the sample programs, the maximum values of I and J are 62 and 150, respectively. Aren't they 63 and 159, respectively?

A4: You are right. The ranges of I and J are 0 to 63 and 0 to 159, respectively, as determined by the display configuration.

Q5: Isn't it possible to improve the processing speed of the BASIC program by using a compound statement?

A5: Generally speaking, the use of a compound statement reduces the memory requirement and improves the processing speed, although it makes the program look complicated. It will be interesting to compare the processing speed between a program with a compound statement and a program without compound statement. In the above experiment, it may be possible to substitute a single compound statement for the entire BASIC program.

Q6: In the machine language program, it is tiresome to repeat direct input of a POKE command as many as 14 times. Isn't there any better method?

A6: Certainly this is a problem with the FP-200 having no monitor for its machine language. One way to make the work easier is to use a DATA statement in the program. This and other possible ways will be discussed in more detail in later sections.

Q7: In our experimentation, the machine language program is called by the CALL statement in BASIC. Taking this into consideration, the actual processing speed of the machine language should be faster than one second.

A7: Exactly. This will also be discussed later.

Q8: Can the processing speed of the BASIC program be increased by rewriting the DRAW statement as (X1, Y) – (X2, Y), where Y is in the range 0 to 63?

A8: Intrinsically, the DRAW statement is more efficient in drawing a line than in plotting dots. Verify this using a modified program.

With the above-mentioned questions and answers in mind, four sample programs are provided for the reader's convenience.

Modified sample programs (PROG 2 to PROG 5)

```
BASIC EX-1A
PROG 2
 10 T1$=TIME$
 20 CLS
 30 FOR J=0 TO 63
 40 FOR I=0 TO 159
 50  DRAW(I,J)
 60 NEXT I,J
 70 T2$=TIME$
 80 PRINT T1$,T2$
 90 END
```

```
BASIC EX-1B
PROG 3
 10 T1$=TIME$:CLS:FOR J=
    0 TO 63:FOR I=0 TO 15
    9:DRAW(I,J):NEXT I,J:
    T2$=TIME$:PRINT T1$,T
    2$:END
```

```
BASIC EX-1C
PROG 4
 10 T1$=TIME$
 20 FOR Y=0 TO 63
 30 X1=0:X2=159
 40 DRAW(X1,Y)-(X2,Y)
 50 NEXT Y
 60 T2$=TIME$
 70 PRINT T1$,T2$
 80 END
```

```
BASIC EX-1D
PROG 5
 10 T1$=TIME$:FOR Y=0 TO
    63:X1=0:X2=159:DRAW(X
    1,Y)-(X2,Y):NEXT Y:T2
    $=TIME$:PRINT T1$,T2$
    :END
```

3

Use of binary, decimal and hexadecimal numbers

Look at the first machine language program used in our previous experimentation. On the first line, you see the numbers '40704, 205.' These express a machine word in decimal notation. In actual practice, hexadecimal notation is often used to represent a machine language. Inside the computer, however, the binary numbers, 0 and 1, are used.
Since the concepts of binary numbers, decimal numbers and hexadecimal numbers will prove extremely important in later sections, let us explain them in detail.

1 Binary numbers and decimal numbers

The computer uses two types of electrical signals, "1" (a pulse of 2.2 to 5V) and "0" (a pulse of 0 to 0.8V),to perform various types of processing. Thus, binary numbers are used inside the computer.
As you already know, the unit of information represented by either 1 or 0 is called a bit. The CPU employed for the FP-200 is an 8085, and the FP-200 is called an 8-bit machine.
By the way, can you immediately tell whether a particular 8-bit binary number, say, 10001000, is a large number or a small number, or calculate the decimal equivalent of that number?
The decimal equivalent of the binary number 10001000 is 136. As is evident, a binary number has more digits than its decimal equivalent, and is somewhat inconvenient for reading by humans. Thus, binary notation has the drawback of being inconvenient to use and apt to cause errors.
Why not use decimal numbers instead of binary numbers? As a matter of fact, computers employing decimal notation were used at one time. After all, we humans perform arithmetic operations using decimal numbers with few exceptions.
This is because we are accustomed to the use of decimal numbers, hence the possibility of making mistakes is minimal. Surprisingly, however, decimal numbers are a nuisance to the computer. Namely, in order for the computer to represent decimal numbers, 4 bits ($2^4 = 16$ different combinations) are required because 3 bits ($2^3 = 8$ different combinations) are insufficient to represent 10 different digits. If 4 bits are used to represent decimal digits, however, six combinations become redun-

dant (because only 10 different combinations are required to represent the decimal digits 0 through 9). This means that the six redundant combinations always remain unused during computer operations, whether they be arithmetic operations, accesses to memory, or data processing. In other words, the use of 4 bits to represent the decimal digits substantially reduces computer efficiency.

2 Hexadecimal numbers

In the case of hexadecimal notation, all the possible combinations of 4 bits ($2^4 = 16$) are used. 10 combinations are used to represent the digits 0–9, and the remaining 6 combinations are used to represent the letters A, B, C, D, E, and F.

This system of notation enables an 8-bit machine to divide eight bits into two 4-bit blocks for maximum efficiency. Actually, hexadecimal numbers are used to represent the FP-200 machine language.

For example, the binary number 10001000 can be represented in hexadecimal notation as 88_{16}. In this case, in order to prevent the hexadecimal number 88 from being confused with the decimal number 88, the hexadecimal number is written as 88H or &H88. This is unnecessary when there is no fear of such confusion.

Table 1-1 shows binary, decimal and hexadecimal equivalent values for your reference.

Table 1-1. Binary, decimal and hexadecimal notations

Binary numbers	Decimal numbers	Hexadecimal numbers
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	8
1 0 0 1	9	9
1 0 1 0	10	A
1 0 1 1	11	B
1 1 0 0	12	C
1 1 0 1	13	D
1 1 1 0	14	E
1 1 1 1	15	F

3 BASIC and decimal numbers

Decimal numbers, although not used for machine languages, are used in BASIC.

BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code. It is a computer language developed by Professors Kemeny and Kurtz at Dartmouth College around 1965 in order to enable even beginners to write programs easily. In order to attain this particular objective, Kemeny and Kurtz had to use decimal numbers.

BASIC lets the computer perform decimal-to-binary conversion for arithmetic operations, etc., and then perform binary to decimal conversion for data output. By so doing, memory efficiency can be slightly improved, although extra work and time are required. In any case,

BASIC was developed with top priority given to ease of programming. This approach proved successful. With the development of microcomputers, BASIC became popular at a rapid pace.
Although BASIC is primarily based on decimal numbers, most versions permit handling hexadecimal numbers by specific methods of representation (e.g., &HABCD, HAB13, 1234H).

Getting accustomed to hexadecimal numbers

If you wish to learn machine language, you will need to get accustomed to hexadecimal numbers. As a matter of fact, calculations using hexadecimal numbers will be needed as we discuss the monitor and disassembler written in BASIC in the next chapter. Note, however, that the FP-200 has no monitor to handle machine language. Therefore, machine language programs can only be entered by means of the BASIC language. The point is, therefore, to learn how to handle hexadecimal numbers in BASIC.

EX-2: Create a program to add two single-digit hexadecimal numbers, A and B, which are keyed in and to display the sum in hexadecimal.

We show four sample programs, each of which will be discussed below. Compare these programs with yours.

Program 2-1

```
' EX-2-1
10 INPUT &HA,&HB
20 &HA=&HA+&HB
30 PRINT &HA
40 END
```

Program 2-2

```
' EX-2-2
10 INPUT A,B
20 A=A+B
30 PRINT A
40 END
```

Input A and B in the form of &H1, ... , &HF.

Program 2-3

```
' EX-2-3
10 INPUT A$,B$
20 A=VAL(A$)+VAL(B
   $)
30 PRINT STR$(A)
40 END
```

Program 2-4

```
' EX-2-4
10 INPUT A,B
20 A=A+B
30 PRINT HEX$(A)
40 END
```

To state the conclusion first, all the four programs shown above result in an error. Although they are apparently correct, a TM error, etc., will occur when they are executed. Let us now check the programs one by one.

Program 2-1: The variable '&HA' cannot be used. The character '&' must be omitted. An SN error will occur when this program is executed.

Program 2-2: Although there is no program error, the '&H' added to indicate that input data is hexadecimal will cause a TM error. A TM error will also occur when the hexadecimal number A or B is keyed in without &H.

Program 2-3: This program will work when only decimal digits are involved. However, when letters indicating hexadecimal digits, say, A and B, are input, the addition in hexadecimal is not performed. Even when hexadecimal digits are input, the addition is not performed correctly if a carry occurs. All this is ascribable to line 20 which performs the addition.

Program2-4: This program does not produce any answer. This is because the HEX$ function is not available in C$_{85}$—BASIC for the FP-200.

If your program did not work, try again paying attention to the comments given above. The following hint will help you create the correct program.

As mentioned above, **program 2-3** will work if only decimal digits are involved and if a carry does not occur. This suggests that the program will be able to be used if it has a proper method of keying in letters representing hexadecimal digits.

Thus, note the following:

1) HEX$ → DEC Hexadecimal letter → Decimal digits
2) Perform the addition in decimal.
3) DEC → HEX$ Decimal digits → Hexadecimal letter

Now, we will show you three valid sample programs, each of which is discussed below.

Program 2-5

```
 5 'EX-2A-5
10 INPUT A$,B$
20 C$=A$:D=0:K=1:L=1
30 IF C$="A"THEN C=10:GOTO 100
40 IF C$="B"THEN C=11:GOTO 100
50 IF C$="C"THEN C=12:GOTO 100
60 IF C$="D"THEN C=13:GOTO 100
70 IF C$="E"THEN C=14:GOTO 100
80 IF C$="F"THEN C=15:GOTO 100
90 C=VAL(C$)
100 D=D+C
```

```
110 ON K GOTO 120,130
120 C$=B$:D=C:K=2:GOTO 30
130 IF D=10 THEN C$="A":GOTO 210
140 IF D=11 THEN C$="B":GOTO 210
150 IF D=12 THEN C$="C":GOTO 210
160 IF D=13 THEN C$="D":GOTO 210
170 IF D=14 THEN C$="E":GOTO 210
180 IF D=15 THEN C$="F":GOTO 210
190 IF D>15 THEN D=D-16:L=2:GOTO 130
200 C$=STR$(D)
210 ON L GOTO 220,230
220 PRINT C$:GOTO240
230 PRINT "1"+C$
240 END
```

Program 2-6

```
  5 'EX-2A-6
 10 CLEAR
 20 DIM E$(5)
 30 E$(0)="A":E$(1)="B":E$(2)="C":E$(3
)="D":E$(4)="E":E$(5)="F":K=0
 40 FOR I=0 TO 1
 50  INPUT A$
 60  FOR J=0 TO 5
 70   IF A$=E$(J)THEN C=J+10:GOTO 100
 80 NEXT J
 90 C=VAL(A$)
100 D=D+C
110 NEXT I
120 IF D>15 THEN D=D-16:K=1
130 FOR I=0 TO 5
140  IF D=I+10 THEN A$=E$(I):GOTO 170
150 NEXT I
160 A$=STR$(D)
170 IF K=1 THEN PRINT"1"+A$ ELSE PRINT
A$
180 END
```

Program 2-7

```
    5 'EX-2A-7
   10 CLEAR
   20 OPTION BASE 1
   30 DIM E$(6)
   40 FOR I=1 TO 6
   50  READ E$(I)
   60 NEXT I
   70 DATA A,B,C,D,E,F
   80 FOR I=1 TO 2
   90 INPUT A$:IF A$>"/"THEN IF A$<":"TH
EN 130 ELSE IF A$>"@"THEN IF A$<"G"THEN
100 ELSE 90
  100 FOR J=1 TO 6
  110  IF A$=E$(J)THEN C=J+9:GOTO 140
  120 NEXT J
  130 C=VAL(A$)
  140 D=D+C
  150 NEXT I
  160 IF D>15 THEN D=D-16:K=1
  170 FOR I=1 TO 6
  180  IF D=I+9 THEN A$=E$(I):GOTO 210
  190 NEXT I
  200 A$=STR$(D)
  210 IF K=1 THEN PRINT"1"+A$ ELSE PRINT
 A$
  220 END
```

First, look at **program 2-5**. Taking into consideration the fact that **program 2-3** failed to convert hexadecimal letters to decimal digits, **program 2-5** uses IF statements for the conversion. Line number 110 ON K GOTO 120, 130 is a unique feature of this program. Note that control is passed to the conversion routine twice for A$ and B$.

In **program 2-6**, a FOR statement instead of an IF statement is used for the same purpose. Note that line 30 is the assignment statement to read data into the array.

In **program 2-7**, the READ statement and the DATA statement are used. Note that line number 20 of this program specifies OPTION BASE 1. You may of course use OPTION BASE 0 rather than OPTION BASE 1. By the way, what will happen if characters other than the decimal digits (0 – 9) and the letters A – F are keyed in?

Try to see what will happen. In the case of programs 2-5 and 2-6, the previously input numbers will remain unchanged. In the case of program 2-7, you will be requested to key in the correct data.

You might think that it cannot be helped if the program produces an incorrect result due to incorrect data input by a human operator. However, since human beings make mistakes from time to time, it is desirable to guard against possible mistakes. In this respect, program 2-7 is superior to the other two programs.

There is one more thing to remember. Although the FP-200 permits storing up to 10 different programs at a time, the variables stored in memory are not erased even when the NEW statement is executed. Therefore, when a DIM statement is used to define an array variable, a double definition error may occur. In order to prevent such trouble, it is advisable to insert a CLEAR statement in the program.

4

Creating a disassembler

In this chapter, let us learn how to create a disassembler for using the machine language. As the first step, let us create a BASIC program to perform a memory dump.

Memory dump requirements

Since no monitor is available with the C85—BASIC for the FP-200, only the PEEK function can be used to perform a memory dump.
First, let us list various requirements for performing a memory dump:

(1) Since the PEEK function is available in BASIC, addresses are input in decimal. In the case of machine language, however, hexadecimal notation is used for the memory map, etc. In this light, it is desirable that hexadecimal notation be used for address input. This means that conversion from hexadecimal to decimal is required.
(2) A maximum of 64K bytes can be addressed by 16-bit addresses. Thus, 4-digit hexadecimal numbers are handled.
(3) It is necessary to guard against input errors.
(4) When the PEEK function is used, data is output in decimal. Hence, decimal-to-hexadecimal conversion is required. In this case, each decimal number is converted to a 2-digit hexadecimal number, since it consists of one byte.
(5) It is necessary to implement the HEX$ function, which is not available with C85—BASIC.
(6) It is desirable to create a mode in which the contents of memory can be automatically checked while incrementing the address by +1.
(7) The availability of a printer is extremely desirable. It is also desirable for the printer to offer at least three styles.
(8) In order to display addresses, it is necessary to convert 4-digit numbers from decimal to hexadecimal.
(9) It is advisable to arrange 8 or 16 data items in each horizontal row during a memory dump in order to facilitate checking addresses at a later time.

Fig.1-1 Creating the disassembler

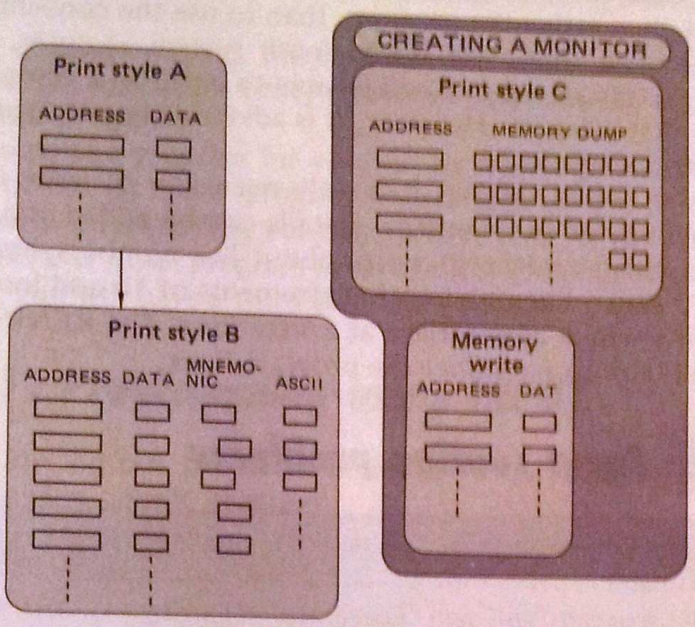
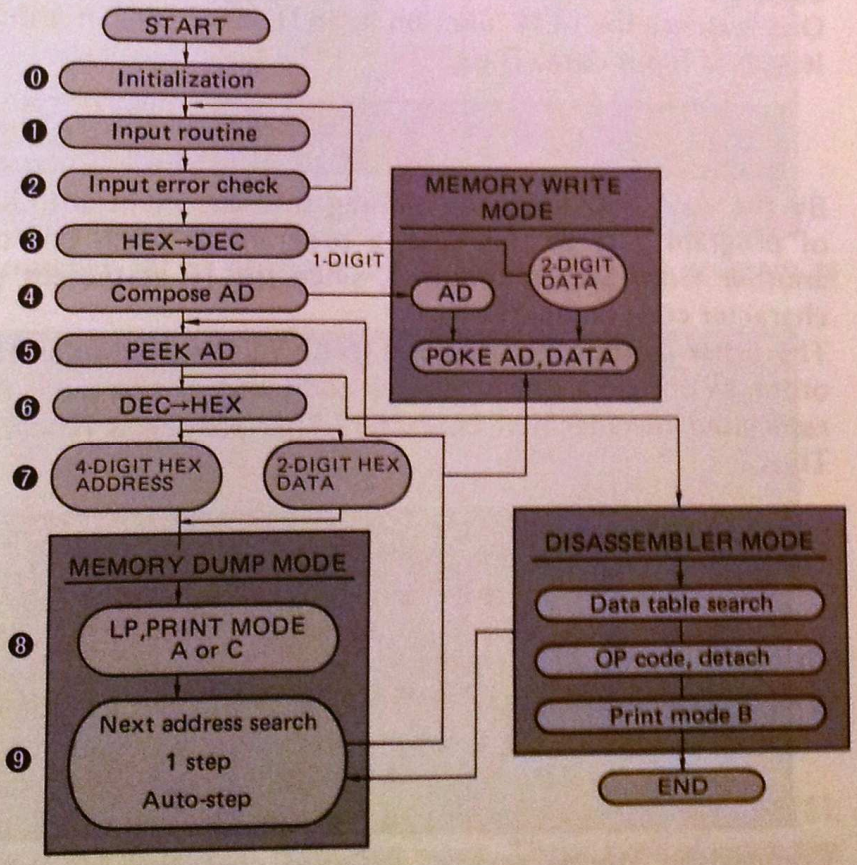


Fig. 1-2 Task flow



The task flow is shown in Fig. 1-2. In preparing a BASIC program, you will find it easier and more efficient to begin with easy-to-write segments, proceed to more difficult segments, combine them into a single program and then refine the program, than to use the conventional programming technique that calls for in-depth system analysis, precise flowcharts, etc. After all, no one will be able to perform a complex job perfectly at his first attempt. However, it is advisable to construct a flowchart at an intermediate stage.
In the initial setting, it is only necessary to write CLEAR, DIM, etc., required for the moment. Details can be added at a later time. At first, it is not necessary to worry about line numbers, etc. For example, you may assign line numbers in increments of 10 and insert additional statements where appropriate at a later time. The RENUM statement can be used later to renumber the program lines.

2 Input routine problems

```
10 INPUT "START ADDRESS=", SA$
```

This program will run 'correctly' regardless of the number of digits of input data. Isn't there any method to ensure that only 4-digit data is accepted? Here are two methods.
One is to use the LEN function in an IF statement in order to check the length of input data. Thus,

```
20 IF LEN(SA$)=4 THEN____ELSE10
```

By the way, how about combining this statement with line number 90 of program 2-7? In this case, a program to fetch one character after another from SA$ is required, since the IF statement performs one-character code comparisons.
The other method is to use the INKEY$ function in an IF statement in order to obtain a one-character code and at the same time check the associated hexadecimal character. This process is repeated four times. Thus,

```
10 FOR I=1 TO 4
20  H$=INKEY$:IF H$=" "THEN 20ELSE
    IF H$>"/" THEN IF H$<":" THEN□
    ELSE IF H$>"@"THEN IF H$<"G"TH
    EN□ELSE 20
30 NEXT I
```

Comparison among several different programs for the same problem will reveal programming defects which would be overlooked if only one

program is created. This will help you to create a refined program.
Now, let us create programs using the two different methods described above: one is using INPUT (pattern 1) and the other is using INKEY$ (pattern 2).
At various stages of programming, we shall test the programs to compare ease of operation, execution time, memory requirements, etc., in order to create better programs.
You are advised to write a program for yourself by referring to the programs presented below. If your program meets appropriate criteria (various requirements, checkpoints, etc.) and proves valid, it will afford you great satisfaction. If your program proves better than our two programs (see below), you will have a third effective program.

5 Pattern 1: Programming using INPUT

First, we shall study various problems involved in our programming work using the following sample program.

```
10 DIM A$(3)
20 INPUT "START ADDRESS=";SA$
30 IF LEN(SA$)=4 THEN 40 ELSE 20
40 FOR I=0 TO 3
50 A$(I)=MID$(SA$, I, 1)
60 IF A$(I)>"/" THEN IF A$(I)<":" THEN
   ELSE IF A$(I)>"@" THEN IF A$(I)<"G"
THEN   ELSE 20
```

We add line number 90 of **program 2-7** at the end of this program as follows:

```
70 FOR J=0 TO 5
80 IF A$(I)=E$(J) THEN AN(I)=J+10:GOTO110
90 NEXT J
100 AN(I)=VAL(A$(I))
110 END
```

Now let us check the problems involved in the program shown above.
(1) It is necessary to use the CLEAR statement at the beginning of the program since the FP-200 variables are common among all programs loaded in different program areas.
(2) It is necessary to include E$(5) and AN(3) in DIM.
(3) In **program 2-7**, character data must be previously assigned to E$(5). This can be done in several different ways. Three typical examples are shown below.

```
10 FOR I=0 TO5
20   READ E$(I)
30   DATA A, B, C, D, E, F
40 NEXT I
```

```
10 READ E$(0), E$(1), E$(2), E$(3),
E$(4), E$(5)
20 DATA A, B, C, D, E, F
```

```
10 E$(0) = "A": E$(1) = "B": E$(2) = "C":
E$(3) = "D": E$(4) = "E": E$(5) = "F"
```

Of these, the last one, using a compound statement, seems the best since it consists of a single line, requiring the smallest memory capacity and offering the highest execution speed.

(4) The FOR statement on line number 40 has no NEXT I statement associated with it to form a loop. Hence, we should insert line number 105 NEXT I.

(5) Here is an elementary question. Is it possible to write 10 INPUT A$, B$, C$, D$ in order to input one character after another?
In practice, this method is not recommendable, since it requires pressing the RETURN key each time a character is input. Besides, if two-character data is input, the program accepts that data, causing unwanted question marks (?) to be displayed.
When this program is executed, the CRT displays as follows:

```
START  SA=?9
?E
?3
?E
```

Then, how about using the following program?

```
10 INPUTA$;B$;C$;D$
```

When this program is executed, an SN error occurs.

(6) As for line number 50 MID$ (SA$, I, 1), an error occurs when I=0. In order to prevent this, I + 1 should be substituted for I. (As described in the Reference Manual, the value of I must not be less than 1.)

Having studied the above-mentioned items, we now have the program shown below. Execute this program; you will find that a blank is inserted between numbers.

Program 3-1

```
 10 CLEAR
 20 DIM E$(5),A$(3),AN(3),D(3),D$(3)
 30 E$(0)="A":E$(1)="B":E$(2)="C":E$(3
)="D":E$(4)="E":E$(5)="F"
 40 INPUT"START ADDRESS=";SA$
 50 IF LEN(SA$)=4 THEN 60 ELSE 40
 60 FOR I=0 TO 3
 70  A$(I)=MID$(SA$,I+1,1)
 80 IF A$(I)>"/"THEN IF A$(I)<":"THEN
120 ELSE IF A$(I)>"@"THEN IF A$(I)<"G" T
HEN 90 ELSE 40
 90 FOR J=0 TO 5
100 IF A$(I)=E$(J)THEN AN(I)=J+10:GOTO
 130
110 NEXT J
120 AN(I)=VAL(A$(I))
130 NEXT I
140 FOR I=0 TO 3
150  AD=AD+AN(I)*16^(3-I)
160 NEXT I
170 D=PEEK(AD)
180 D(1)=INT(D/16):D(0)=D MOD 16
190 FOR I=0 TO 1
200 IF D(I)<10 THEN220
210 J=D(I)-10:D$(I)=E$(J):GOTO 230
220 D$(I)=STR$(D(I))
230 NEXT I
240 PRINT
250 PRINT D$(1)+D$(0)
260 END
```

Pattern 2: Programming using INKEY$

The sample program used for discussion is shown below.

```
10 CLEAR
20 DIM E$(5),H$(3),HN(3)
30 E$(0)="A":E$(1)="B":E$(2)="C":E$(3
   )="D":E$(4)="E":E$(5)="F"
40 FOR I=0 TO 3
50 H$(I)=INKY$:IF H$(I)="" THEN 50
60 IF H$(I)>"/" THEN IF H$(I)<":" THE
   N 100 ELSE IF H$(I)>"@" THEN IF H$
   (I)<"G" THEN 70 ELSE 50
70 FOR J=0 TO 5
80 IF H$(I)=E$(J) THEN HN(I)=J+10:GOT
   O 110
90 NEXT J
100 HN(I)=VAL(H$(I))
110 NEXT I
120 END
```

The problems involved in this sample program are as follows:

(1) This program does not display anything. When used properly, the INKEY$ function is very useful (it does not cause a question mark (?) to be displayed).
However, we should insert a PRINT statement so that the desired data can be displayed. Inserting a PRINT statement in this program is relatively easy, since the data to be displayed is contained in the array.

```
115 PRINT "START ADDRESS=";H$(0);H$
(1);H$(2);H$(3)
```

(2) The PRINT statement shown above is not recommended, since no data is displayed until all data is input. This means that errors which may occur during data input cannot be detected. It is therefore advisable to separate the PRINT statement into two statements, one consisting of the prompt portion of the original PRINT statement and the other within the FOR–NEXT loop. Thus, the program appears as follows:

```
35  PRINT "START  ADDRESS=";
55  PRINT  H$ (I);
```

(3) The routine using an IF statement (after the INKEY$ function on line number 50) to cause the system to wait for key input is unnecessary. Since line number 60 discards unnecessary codes, ELSE 50 causes the system to wait for key input.

(4) Now, let us review how to convert a hexadecimal number to a decimal number. Converting a one-digit hexadecimal number (0–9, A–F) to a decimal number is easy. Then, how do we convert a hexadecimal number of two or more digits to a decimal number?
A decimal number, say, 2534, can be expressed as follows:

```
2534=2*1000+5*100+3*10+4
2534=2*10^3+5*10^2+3*10^1+4*10^0
```

If 2534 is a hexadecimal number, it can be expressed as follows:

```
2534=2*16^3+5*16^2+3*16^1+4*16^0
```

```
115  AD=HN (0) *16^3+HN (1) *16^2+HN
(2) *16+HN (3)
```

(5) Now, a memory dump can be done by D = PEEK (AD).
However, since D is always a decimal number, decimal to hexadecimal conversion is required.
This conversion is performed by the HEX function, which is not available with the FP-200. So, let us create a program which performs the same function as HEX. It is only necessary to reverse the process performed by line number 115 shown above. Since the value of D will not exceed three digits, the associated hexadecimal number will not exceed two digits.

We now have the program shown below. This time, the results of execution are shown at the beginning of the program. As shown, the contents of address 0 are 3 1. As was the case with pattern 1, a space is inserted between the digits 3 1. The results shown in alphabetic characters seem all right.

Program 3-2

```
START ADDRESS=0003
 3E
START ADDRESS=0004
CA
START ADDRESS=0000
 3 1


   10 CLEAR
   20 DIM E$(5),H$(3),HN(3),D(3),D$(3)
   30 E$(0)="A":E$(1)="B":E$(2)="C":E$(3
)="D":E$(4)="E":E$(5)="F"
   40 PRINT"START ADDRESS=";
   50 FOR I=0 TO 3
   60 H$(I)=INKEY$:IF H$(I)=""THEN 60
   70 PRINT H$(I);
   80 IF H$(I)>"/"THEN IF H$(I)<":"THEN
120ELSE IF H$(I)>"@"THEN IF H$(I)<"G"THE
N 90 ELSE 60
   90 FOR J=0 TO 5
  100 IF H$(I)=E$(J)THEN HN(I)=J+10:GOTO
 130
  110 NEXT J
  120 HN(I)=VAL(H$(I))
  130 NEXT I
  140 AD=HN(0)*16^3+HN(1)*16^2+HN(2)*16+
HN(3)
  150 D=PEEK(AD)
  160 D(1)=INT(D/16):D(0)=D MOD 16
  170 FOR I=0 TO 1
  180  IF D(I)<10 THEN 230
  190  FOR J=0 TO 5
  200   IF D(I)=10+J THEN D$(I)=E$(J)
  210 NEXT J
  220 GOTO 240
  230 D$(I)=STR$(D(I))
  240 NEXT I
```

```
250 PRINT
260 PRINT D$(1)+D$(0)
270 END
```

5 Comparison between Pattern 1 and Pattern 2

At this moment, both programs (pattern 1 and pattern 2) are executed too fast to permit any meaningful comparison of their performance. So, let us modify those programs so that they can continuously read data from a given memory area.

This modification will prove useful, since the function it offers is used for memory dump.

As a matter of fact, providing the functions of key input, hexadecimal to decimal conversion, decimal to hexadecimal conversion, etc., as subroutines (software components) facilitates programming. At the same time, you can learn how to create subroutines.

Continuously reading data from a given memory area can be done by varying the value of AD in AD = AD + 1, D = PEEK (AD). For this purpose, two methods are available: one is to set a start address; the other is to count how many times the value of AD is varied. Either method may be used, depending on the amount of data to be read.

Programs 3-3 and 3-4 are modified versions of programs 3-1 (pattern 1) and 3-2 (pattern 2), respectively.

Program 3-3. Pattern 1 program

```
 10 CLEAR
 20 DIM A$(3),A(3),D$(3),D(3),E$(5),H$
(3),H(3),DD(7)
 30 E$(0)="A":E$(1)="B":E$(2)="C":E$(3
)="D":E$(4)="E":E$(5)="F"
 40 T1$=TIME$
 50 INPUT"START ADDRESS=";SA$
 60 HA$=SA$:N=3:M=2:HD=0:K=0
 70 IF LEN(SA$)=4 THEN GOSUB 240 ELSE
50
 80 ON M GOTO 50,90
 90 AD=HD:HD=0
100 INPUT"END ADDRESS=";EA$:HA$=EA$
110 IF LEN(EA$)=4 THEN GOSUB 240 ELSE
100
```

```
  120 ON M GOTO 100,130
  130 ED=HD:P=0
  140 D=PEEK(AD)
  150 N=3:GOSUB 350
  160 PRINT
  170 PRINT D$(0)+D$(1)+D$(2)+D$(3)+" ";
  180 N=1:GOSUB 350
  190 PRINT D$(0)+D$(1)+" ";
  200 P=P+1:DD(K)=D:K=K+1
  210 IF AD=ED THEN T2$=TIME$:PRINT T1$,
T2$:END ELSE AD=AD+1
  220 IF K>7 THEN P=0:K=0:GOTO 140 ELSE
D=PEEK(AD):GOTO 180
  230 'HEX-DEC CONV.SUBRUTIN
  240 FOR I=0 TO N
  250  H$(I)=MID$(HA$,I+1,1)
  260  IF H$(I)>"/"THEN IF H$(I)<":" THE
N 300 ELSE IF H$(I)>"@"THEN IFH$(I)<"G"T
HEN 270 ELSE M=1:RETURN
  270  FOR J=0 TO 5
  280   IF H$(I)=E$(J) THEN H(I)=J+10:GO
TO 310
  290  NEXT J
  300 H(I)=VAL(H$(I))
  310 HD=HD+H(I)*16^(3-I)
  320 NEXT I
  330 RETURN
  340 'DEC-HEX CONV.SUBRUTIN
  350 IF N=1 THEN 370
  360 D(0)=INT(AD/4069):D(1)=(AD/256)MOD
16:D(2)=(AD/16)MOD16:D(3)=AD MOD16:GOTO
380
  370 D(0)=INT(D/16):D(1)=D MOD 16
  380 FOR I=0 TO N
  390  IF D(I)<10 THEN 410
  400 J=D(I)-10:D$(I)=E$(J):GOTO 420
  410 D$(I)=RIGHT$(STR$(D(I)),1)
  420 NEXT I
  430 RETURN
```

Program 3-4. Pattern 2 program

```
    10 CLEAR
    20 OPTION BASE 1
    30 DIM A$(4),AN(4),D$(4),E$(6),DD$(2)
,D(4),DW$(2),DW(2)
    40 E$(1)="A":E$(2)="B":E$(3)="C":E$(4
)="D":E$(5)="E":E$(6)="F"
    50 PRINT"START ADDRESS"
    60 T1$=TIME$
    70 FOR I=1 TO 4
    80  H$=INKEY$:IF H$=""THEN 80 ELSE GO
SUB 260
    90  ON M GOTO 80,100
   100  A$(I)=H$:AN(I)=H
   110  PRINT A$(I);
   120 NEXT I
   130 PRINT:P=0
   140 AD=AN(1)*4096+AN(2)*256+AN(3)*16+A
N(4)
   150 PRINT AD
   160 PRINT A$(1)+A$(2)+A$(3)+A$(4)+" "+
" ";
   170 N=1
   180 D=PEEK(AD) :K=2:GOSUB350
   190 IF P=40 THEN T2$=TIME$:PRINT T1$,T
2$:END
   200 P=P+1
   210 PRINT D$(1)+D$(2)+" ";
   220 AD=AD+1
   230 IF N=8 THEN 240 ELSE N=N+1:GOTO 18
0
   240 D=AD:K=4:GOSUB 340
   250 A$(1)=D$(1):A$(2)=D$(2):A$(3)=D$(3
):A$(4)=D$(4):PRINT:GOTO 160
   260 'HEX-DEC CONVERT
   270 IF H$)"/"THEN IF H$<":"THEN 280 EL
SE IF H$)"@"THEN IF H$<"G"THEN 290 ELSE
```

```
M=1:RETURN
  280 H=VAL(H$):M=2:RETURN
  290 FOR J=1 TO 6
  300  IF H$=E$(J) THEN H=9+J:GOTO 320
  310 NEXT J
  320 M=2:RETURN
  330 'DEC-HEX CONVERT
  340 D(1)=INT(D/4096):D(2)=(D/256)MOD 1
6:D(3)=(D/16)MOD 16:D(4)=D MOD 16:GOTO 3
60
  350 D(1)=INT(D/16):D(2)=D MOD 16
  360 FOR I=1 TO K
  370  IF D(I)>9 THEN 390
  380  D$(I)=RIGHT$(STR$(D(I)),1):GOTO 4
20
  390  FOR J=1 TO 6
  400   IF D(I)=9+J THEN D$(I)=E$(J)
  410  NEXT J
  420 NEXT I
  430 RETURN
```

Characteristic features of the two programs are discussed below.

(1) **Features of pattern 1 program using INPUT**

This program has useful subroutines. For example, the numeric expression on line number 310 permits converting both hexadecimal addresses and hexadecimal data to decimal numbers regardless of the number of hexadecimal digits. Merely by calling this subroutine once, it is possible to obtain the appropriate decimal number.

```
310  HD=HD+H(I)*16^(3-I)
```

Another example is the character conversion subroutine on line number 400. This subroutine employs the table search technique for character conversion.

```
400  J=D(I)-10:D$(I)=E$(J):GOTO420
```

(2) **Features of pattern 2 program using INKEY$**

Simplicity is the major feature of this program. For example, line number 140 performs simple calculations in accordance with basic

rules given in the C8s—BASIC Reference Manual. Despite this, it will offer higher processing speeds, since it has eliminated the need to perform exponential calculations by substituting 4096 for 16^3 and 256 for 16^2.

```
140  AD=AN(1)*4096+AN(2)*256+AN(3)*16
     +AN(4)
```

Another example is line number 340. It also performs simple calculations for decimal to hexadecimal conversion.

```
340  D(1)=INT(D/4096):D(2)=(D/256)MOD16:
     D(3)=(D/16)MOD16:D(4)=DMOD16:GOTO360
```

It seems, however, that there is room for improvement in line number 340.

Now, let us compare programs 3-3 and 3-4.

For purposes of comparison, assume the start address as 0000.

First, let us dump 40 items. Since P=40 in pattern 2, the end address in pattern 1 should be 0028.

• **Result 1**

	1st time	2nd time
Pattern 1	26 seconds	25 seconds
Pattern 2	21 seconds	21 seconds

As shown, the execution time of pattern 2 is faster than that of pattern 1. What happens when the number of items to be dumped is increased? Let us change the value of P on line number 190 in the pattern 2 program to 128 before executing the program again. Accordingly, the end address in the pattern 1 program should be 0080.

• **Result 2**

	1st time	2nd time
Pattern 1	59 seconds	58 seconds
Pattern 2	62 seconds	59 seconds

You will find that this time the execution time of pattern 1 is faster than that of pattern 2, although the difference between them is insignificant the second time.

Let us further increase the number of items to be dumped and try again. Assume the end address in the pattern 1 program is 0200 and the value of P in the pattern 2 program is 512.

• **Result 3**

	1st time	2nd time
Pattern 1	3 min 24 sec	3 min 23 sec
Pattern 2	3 min 50 sec	3 min 49 sec

From the above-mentioned results, it can be seen that the table search for character conversion in the pattern 1 program contributes to the faster execution speed. It is particularly effective when substantial

amounts of data are to be dumped.
The memory requirements for both programs can be easily checked with the SYSTEM statement. The pattern 1 program requires 1143 bytes and pattern 2 program requires 109 bytes. Although the pattern 1 program requires more memory, the difference in memory requirements will become minimal when the pattern 2 program uses the INPUT statement to input the number of items to be dumped.
To conclude, it can be seen that the pattern 1 program is somewhat better than the pattern 2 program. By carefully reviewing the various problems we have discussed thus far, you will find many useful hints for creating better programs.

Program to wrtie machine language in memory using POKE

Using the above-mentioned programs as a base, let us create a program, using the POKE statement, to write machine language in memory.
Since addresses and data handled by the POKE statement are decimal numbers, it is necessary to convert them to hexadecimal numbers before they are input. It is also necessary to provide an error checking facility. In short, the program we are going to create is a monitor which has a memory write function.
Requirements for the program are listed below.

(1) The start address is keyed in as a 4-digit hexadecimal number.
(2) Data to be written is keyed in as a 2-digit hexadecimal number.
(3) After one data item has been written, the next address is displayed automatically and the program waits for the data to be keyed in.
(4) It is desirable, if an error occurs, to be able to return to the associated address by inputting the character "S" in order to permit re-writing the correct data.
(5) A check of key input must be performed. If an error is detected, the correct data must be re-input.

Once the program has been created, it permits writing data to and reading data from memory. At the same time, you can confidently take the first steps to master machine language.

Disassembler

```
10 CLEAR ,40703
20 OPTION BASE 1
30 DIM A$(4),AN(4),D$(4),E$(6),DD$(2)
,D(4),DW$(2),DW(2),AD$(255)
40 E$(1)="A":E$(2)="B":E$(3)="C".E$(4
)="D":E$(5)='E":E$(6)="F"
50 INPUT"MEMORY READ OR WRITE R/W";MR
W$
60 IF MRW$="R"THEN L=1:GOTO 70 ELSE I
F MRW$="W"THEN L=2:GOTO 70 ELSE 50
70 PRINT"START ADDRESS"
80 FOR I=1 TO 4
90 H$=INKEY$:IF H$=""THEN 90 ELSE GO
SUB 390
100  ON M GOTO 90,110
110  A$(I)=H$:AN(I)=H
120  PRINT A$(I);
130 NEXT I
140 PRINT
150 AD=AN(1)*4096+AN(2)*256+AN(3)*16+A
N(4)
160 PRINT AD
170 PRINT A$(1)+A$(2)+A$(3)+A$(4)+" "+
" ";
180 LPRINT A$(1)+A$(2)+A$(3)+A$(4)+" "
+" ";
190 N=1
200 ON L GOTO 210,310
210 D=PEEK(AD) :K=2:GOSUB480
220 PRINT AD,D
230 PRINT D$(1)+D$(2)
240 LPRINT D$(1)+D$(2)+" ";:GOTO 260
250 H$=INKEY$:IF H$=" "THEN 260 ELSE I
F H$="S"THEN 280 ELSE 250
260 AD=AD+1:ON L GOTO 270,310
270 IF N=8 THEN 290 ELSE N=N+1:GOTO 21
0
280 AD=AD-1
290 D=AD:K=4:GOSUB 470
300 A$(1)=D$(1):A$(2)=D$(2):A$(3)=D$(3
):A$(4)=D$(4):LPRINT:GOTO 170
310 FOR I=1 TO 2
320  H$=INKEY$:IF H$=""THEN 320 ELSE G
OSUB 390
330 ON M GOTO 320,340
340 D$(I)=H$:DW(I)=H
350 NEXT I
360 DW=DW(1)*16+DW(2)
370 POKE AD,DW
380 GOTO 230
390 'HEX-DEC CONVERT
400 IF H$>"/"THEN IF H$<":"THEN 410 EL
SE IF H$>"@"THEN IF H$<"G"THEN 420 ELSE
M=1:RETURN
410 H=VAL(H$):M=2:RETURN
420 FOR J=1 TO 6
430  IF H$=E$(J) THEN H=9+J:GOTO 450
440 NEXT J
450 M=2:RETURN
460 'DEC-HEX CONVERT
470 D(1)=INT(D/4096):D(2)=(D/256)MOD 1
6:D(3)=(D/16)MOD 16:D(4)=D MOD 16:GOTO 4
90
480 D(1)=INT(D/16):D(2)=D MOD 16
490 FOR I=1 TO K
500  IF D(I)>9 THEN 520
510  D$(I)=RIGHT$(STR$(D(I)),1):GOTO 5
50
520  FOR J=1 TO 6
530   IF D(I)=9+J THEN D$(I)=E$(J)
540  NEXT J
550 NEXT I
560 RETURN
```

5

A look at the inside of the FP-200

In this chapter we shall touch upon the hardware of the FP-200. Let us look inside the FP-200.
Be careful: rough handling may destroy your expensive machine.
First, turn off the power. You are advised to make it a habit to turn off the power whenever handling the hardware of the FP-200.
Loosen the four screws at the bottom of the FP-200 and push the case and the case will come off.
Figs. 1-3 and 1-4 show the printed circuit board around the CPU. The part about 1 cm square located in the center is the CPU (MSM80C85). Located to the right of the CPU are custom ICs such as the μPD65010G012 I/O, key decoder, etc.
The RAM can be seen in the bottom left corner of Fig. 1-4. As shown in Fig. 1-5, the RAM consists of four TC5518BPs.
Fig. 1-6 shows the printed circuit boards for the cassette tape controller buffer, printer controller bus buffer, etc. They are different from the main printed circuit board.
Figs. 1-7 and 1-8 show the liquid crystal display and the back of the keyboard, respectively.
The IC to drive the liquid crystal display can be seen at the bottom right corner of Fig. 1-8. Fig. 1-9 shows an enlarged view of the IC. Fig. 1-10 shows the power supply components.
The CPU on the main circuit board is housed in a 44 pin flat package for industrial use, and is very compact in size. Normally, CPUs are packaged in 40 pin dual inline packages which are somewhat larger. (Both types are available for the 80C85CPU.)
The code "MSM" is used to indicate that the product is manufactured by Oki Electric Co., Ltd.
The characteristic features of the FP-200 hardware are the following.

(1) The MSM80C85A is a silicon-gate CMOSC80. It is an 8-bit 1-chip parallel processing CPU, compatible with the Intel 8085A. Although the MSM80C85A is based on the 8085A developed by Intel and offers the same throughput as the 8085A, it consumes much less power (approximately one tenth). The MSM80C85A can operate on a battery.

Fig. 1.3 Inside of FP-200

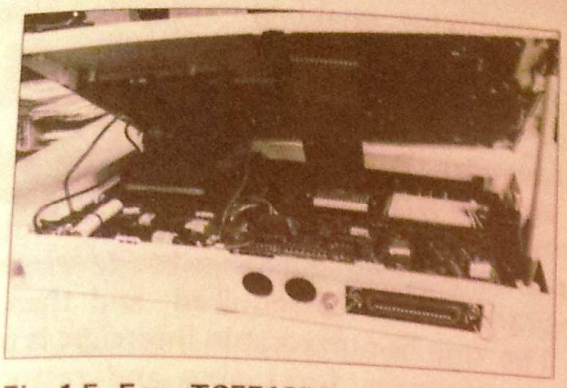


Fig. 1.4 CPU and printed circuit board

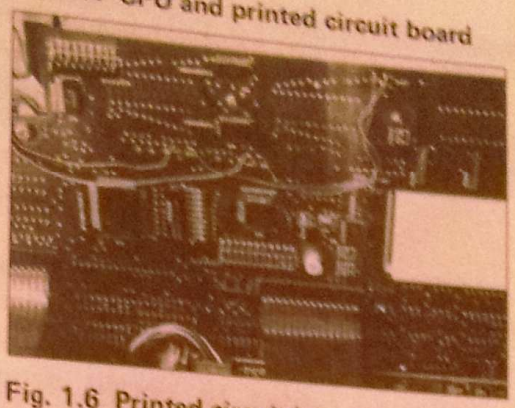


Fig. 1.5 Four TC5518BPs

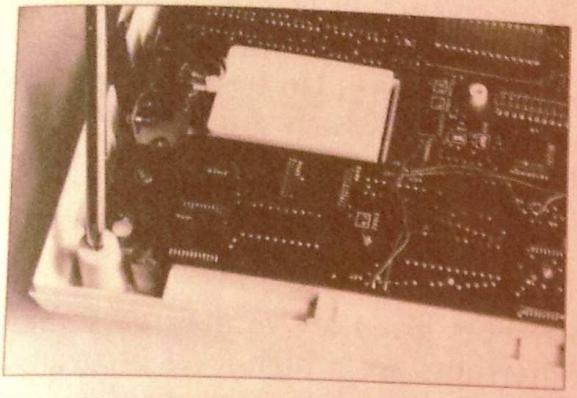


Fig. 1.6 Printed circuit board for buffer

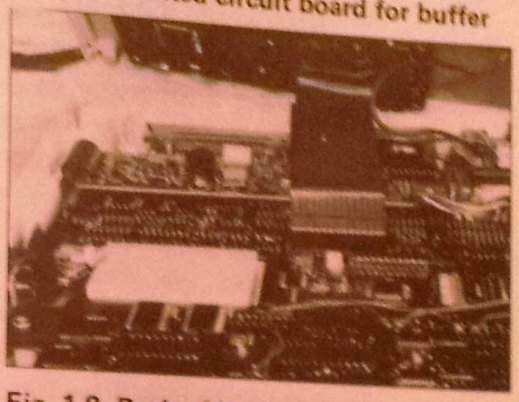


Fig. 1.7 Liquid crystal display

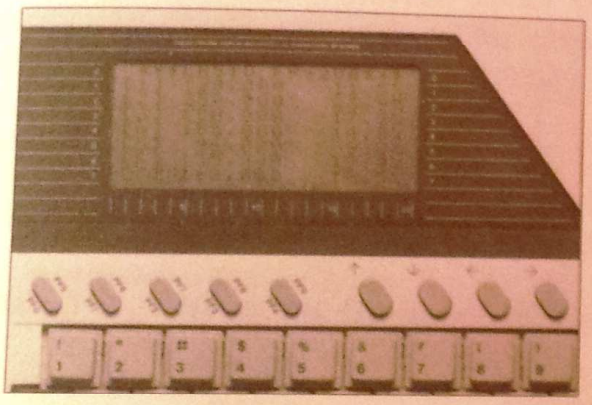


Fig. 1.8 Back of keyboard

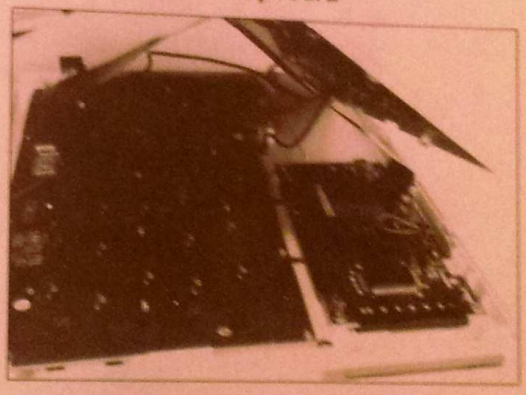


Fig. 1.9 Detail of Fig. 1.8.

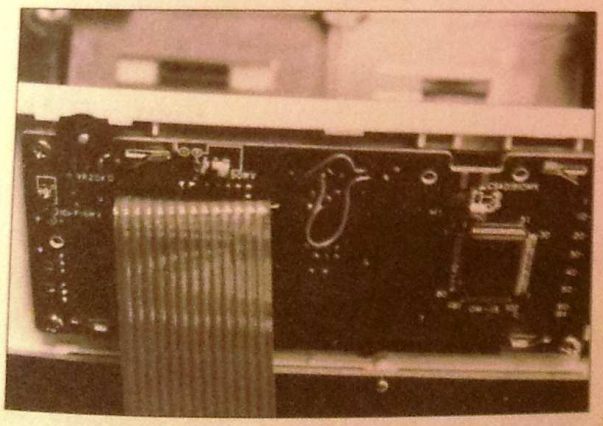
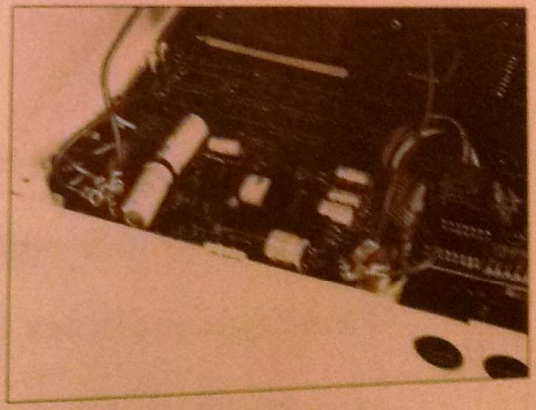


Fig. 1.10 Power supply components



(2) The FP-200 uses a 3 to 6 V single power supply. The instruction cycle is 1.3 μs when a 5 V power supply is used.
(3) The FP-200 has a built-in clock oscillator, enabling to provide a clock using an external crystal oscillator or an external CR circuit.
(4) Four vectored interrupts are available (one non-maskable) for jumping to a particular memory address for interrupt processing.
When the CPU receives an interrupt request signal, it suspends the current job being processed and then performs interrupt processing at a particular address. By previously providing an appropriate interrupt processing program for that address, it is possible to execute the interrupt processing program whenever required, and then to resume the interrupted job. The one non-maskable interrupt is provided to permit processing a program for saving and restoring important data, etc., in case of emergency or power failure.
(5) Built-in serial I/O ports (one for input and one for output).
(6) Direct addressing of 64K bytes of memory. Up to 64K bytes can be addressed using 16 address lines, 8 of which are shared by data. Addresses are not confused with data, since different timings are established for them.

The description of vectored interrupts given above may be somewhat difficult to understand. These interrupts are a high-level programming technique and will be discussed in more detail in a later chapter.

Basic principles of computer operation

In this chapter, we shall study some of the basic principles of computer operation, which will help understand the machine language.

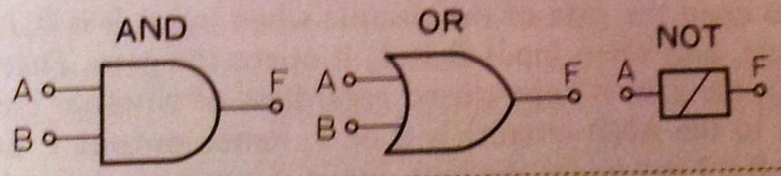
Logic circuit functions

We have already learned that the computer operates on the binary system using the digits 0 and 1. How is this binary system used in actual computer operation? How does the computer perform, say, an addition, using the digits 0 and 1?

Perhaps you have heard of the term 'logical circuit'. An AND circuit, an OR circuit, a NOT circuit, etc., are logical circuits. These logical circuits can be combined to form an adder circuit. (As a matter of fact, it is possible to create not only adder circuits, but also subtracter circuits, multiplier circuits, and divider circuits, by using the three types of logical circuits in various combinations.)

The symbols used to represent AND, OR, and NOT circuits are shown in Figure 1-11. Although each of these logical circuits may be represented in different ways, its basic operation is the same.

Figure 1-11. AND circuit, OR circuit, NOT circuit



(1) AND circuit
A circuit whose output F is 1 only when both A and B are 1.

(2) OR circuit
A circuit whose output F is 1 only when either A or B is 1.

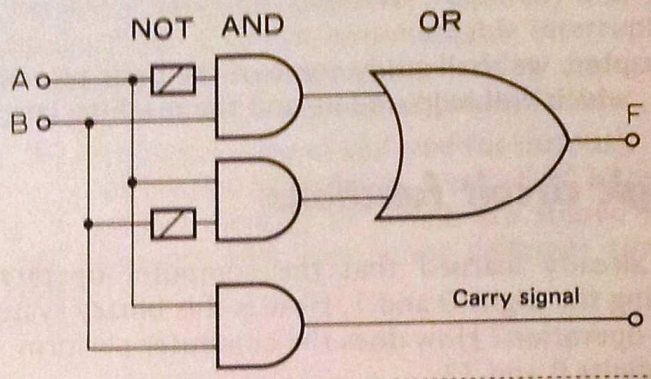
(3) NOT circuit
A circuit whose output F is 0 when A is 1 and whose output F is 1 when A is 0.

These circuits are combined to form an adder circuit. By the way, assuming that A and B are input for a binary addition, there are only four possible results. Thus,

A = 0 and B = 0 . . . . . F = 0
A = 1 and B = 0 . . . . . F = 1
A = 0 and B = 1 . . . . . F = 1
A = 1 and B = 1 . . . . . F = 0 ——— Carry 1

Any circuit which produces output F from inputs A and B is called an adder circuit. Figure 1-12. shows an adder circuit consisting of three AND circuits, one OR circuit, and two NOT circuits. Check output F by inputting 1 as A and 0 as B.

Figure 1-12. Combination of 3 AND circuits, 1 OR circuit, and 2 NOT circuits



Actual adder circuits are more complex. An 8-bit CPU has eight operational circuits such as shown above. The CPU processes eight bits in those circuits at a time. Such processing is called parallel processing. Formerly, when transistors were very costly, there were computers which processed eight bits using a single operational circuit eight times. Such processing is called serial processing. Today, most microcomputers employ a parallel processing system because of its higher processing speed.

AND and OR circuits are otherwise called AND and OR gates. They are so called because they perform the function of a gate. For example, assume that input B — one of two inputs to an AND circuit — is used as a control to open the gate of the circuit: when input B is 0, it does not open the gate, and when input B is 1, it opens the gate. Then, as far as input B is 0, the gate is kept closed regardless of whether input A (the other input to the AND circuit) is 0 or 1, hence output F becomes 0. When input B is 1 and opens the gate, input A can pass through the gate, hence output F becomes the same as input A (whether it be 0 or 1). Thus, the AND circuit can be used as a control gate.

Stored programs and microcomputers

Incidentally, about 20 years ago, the authors and several colleagues paid a visit to a factory of IBM. In those days, IBM was engaged primarily in the manufacture of accounting machines. An IBM man who had been guiding us around the factory suddenly stopped in front of a machine.

He started inserting numerous cords into holes in a honeycomb-like board, shaking his head from time to time. Some time later, he installed the wired board to the side of the machine and switched on the machine. To our surprise, a stack of cards set in a stacker were sorted out in a few seconds. In retrospect, that machine must have been sorter using a patchboard, rather than a stored program as widely used today.

Figure 1-13. Patchboard system (1)

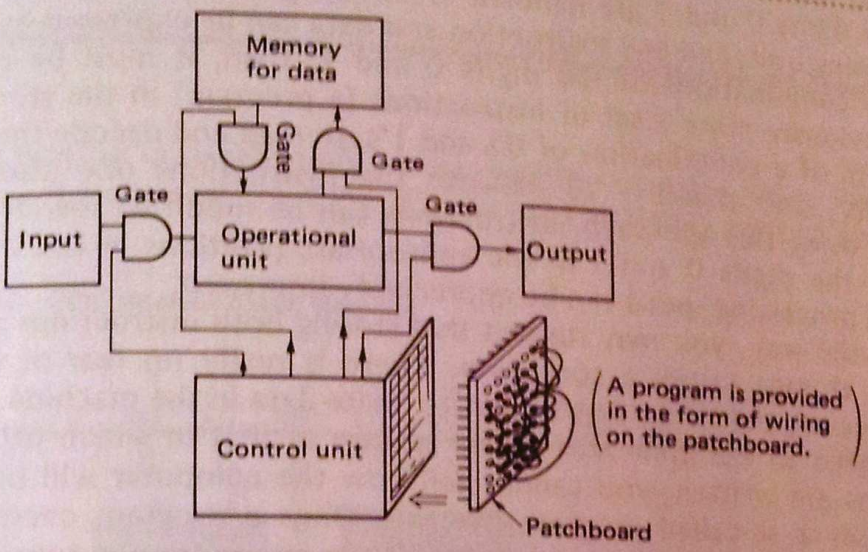
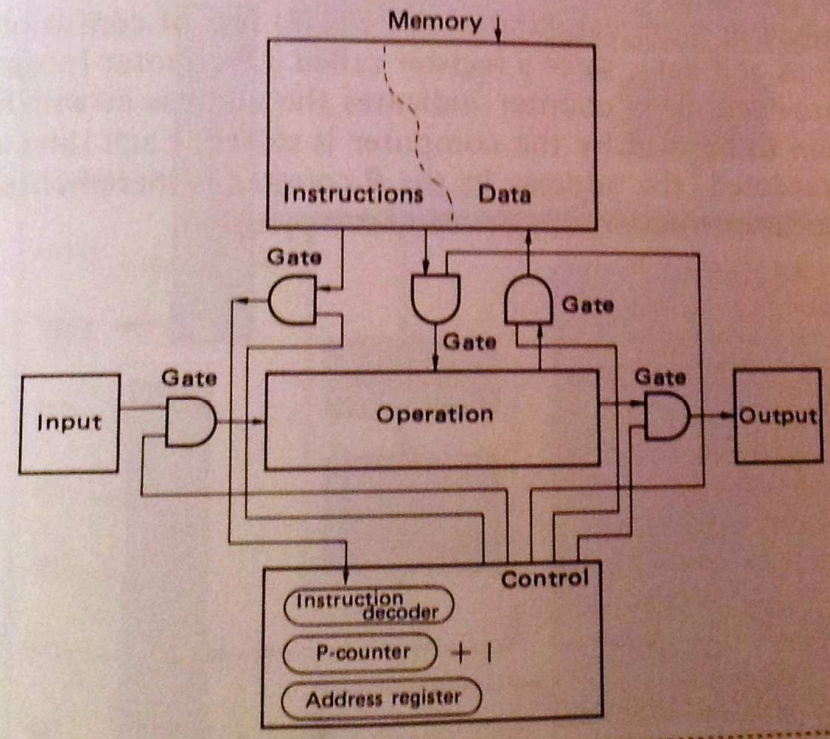


Figure 1-14. Patchboard system (2)



Eventually, this patchboard system was replaced by the stored program system because of difficulties involved in programming and program modifications.

The stored program system, otherwise called the Von Neumann system (Von Neumann is the developer of this system), is employed by almost all computers of today.

The idea underlying this system is this: with a digital computer, only the digits 0 and 1 are handled in its storage unit, control unit, and processing unit; and any instruction and data can be expressed by appropriate combinations of the digits 0 and 1; then, it must be possible to previously store a set of instructions (a program) in the storage in the form of a combination of 0's and 1's, to read and decode them sequentially, and thereby, to execute the instructions one after another. By using this approach, instructions can be modified merely by changing the digits 0 and 1 in the appropriate positions in the storage, and the processing speed can be appreciably improved.

By the way, you may suspect that storing both instructions and data in the storage causes a confusion. There is really no fear of that. Note, however, that if an instruction to write data in the machine language is written to the same area in the storage as that in which other instructions are written, you cannot tell how the computer will behave. This causes a so-called program overrun. Once a program overrun occurs, there is no alternative but to turn off the power supply, turn it on again, and press the RESET button.

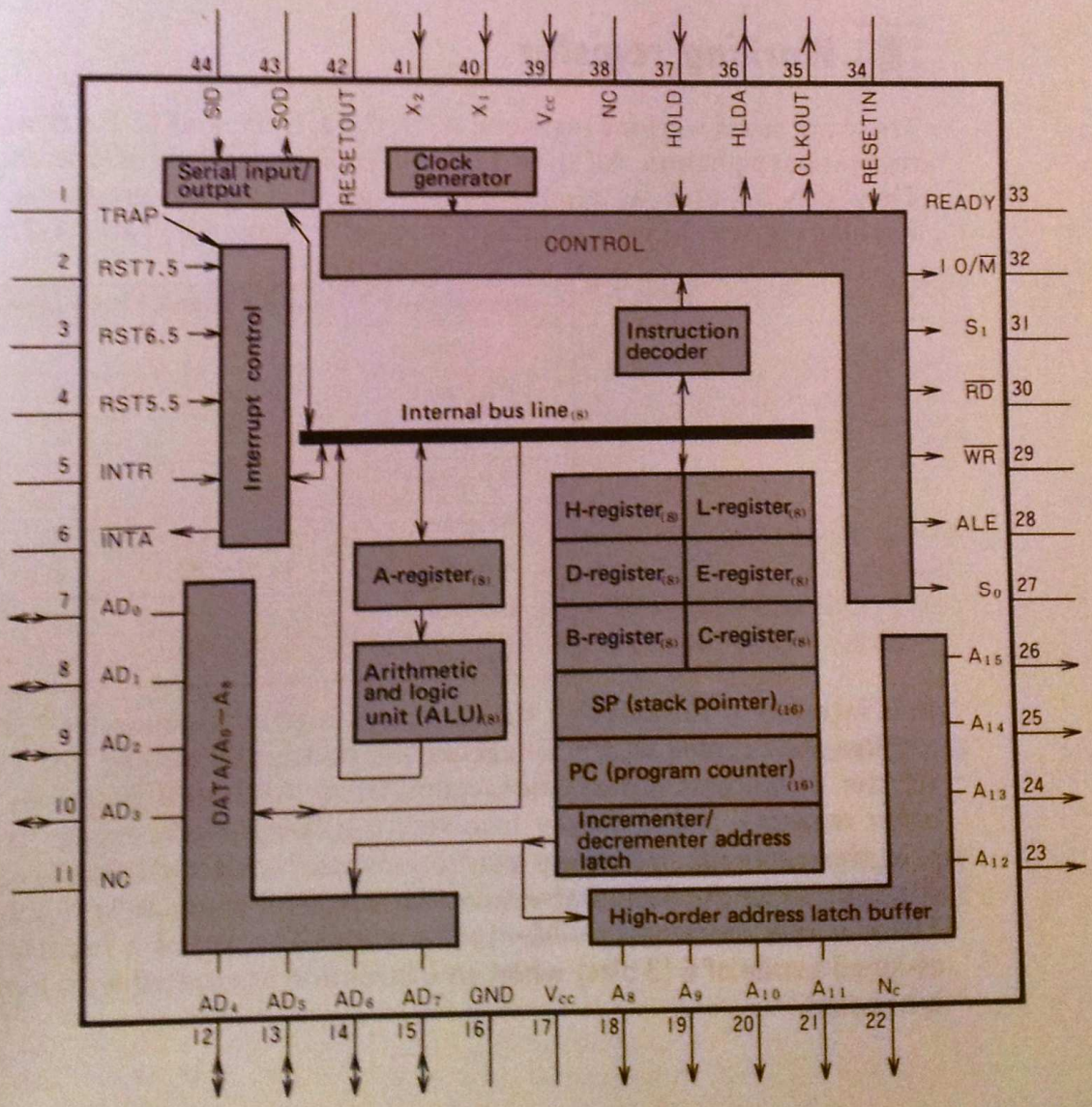
As described in more detail later, there is no fear of confusion between instructions and data, since a register called a P-counter (program counter) is provided. This counter indicates the address in which the next instruction to be read by the computer is stored. Each time an instruction is executed, the address in the P-counter is incremented by 1 so that the next instruction can be read for execution.

7

Functions of registers and counters

Before starting an in-depth study of the machine language, let us review the internal structure of the CPU. Figure 1-15 is a CPU internal block diagram.

Figure 1-15. CPU internal block diagram



As you see, there are various counters and registers. From the software viewpoint, the 16-bit program counter (PC), 16-bit stack pointer (SP), 8-bit flag register (F), and 8-bit working registers (A, B, C, D, E, H, L) are particularly important.
Other registers such as registers W and Z and temporary registers are provided for the CPU; they cannot be manipulated by the user (or software). Computer processing is, after all, the transfer of data between two registers, between a register and memory, between a register and an I/O device, etc., involving the ALU. For the purpose of computer processing, various instructions are provided. These instructions can be combined to let the computer perform various types of processing automatically.
The term 'register' may be interpreted as a recorder or a device to hold numeric data temporarily.
Now, let us see the functions of individual registers in more detail.

Working register

There are seven working registers: A, B, C, D, E, H, and L. Each working register has eight bits. All these registers are independent of one another. They can be used as temporary storage devices by programs. Each working register is given an integer number as shown in Table 1-2.

Table 1-2

Register name	Binary notation	Integer notation
A-register	1 1 1	7
B-register	0 0 0	0
C-register	0 0 1	1
D-register	0 1 0	2
E-register	0 1 1	3
H-register	1 0 0	4
L-register	1 0 1	5
M-memory	1 1 0	6

It is interesting to note that the A-register is given the number 7, the B-register the number 0, the C-register the number 2, and so on. The A-register (otherwise called the accumulator) has more functions than other registers, and is deeply involved in arithmetic processing. In addition, there are relatively many instructions associated with the A-register. All this may be the reason why only the A-register is uniquely numbered. There is one more entry —M— in the table. This is not a register. It is assigned a code of 6 (3 bits) when an instruction associated with memory is handled.

Two working registers (B and C, D and E, H and L, A and F) may be coupled to form a 16-bit register. Again, the A-register is assigned a special function. The A-F pair is named the PSW (Program Status Word), which can conveniently used for interrupt processing and the processing necessary for subroutine calls.
The other register pairs permit indirect addressing. Indirect addressing uses 16 bits of data in two concatenated registers, say, B and C, to indicate a memory address, by which an access to memory is performed. Although indirect addressing is a somewhat complex technique, it is extremely useful. While direct addressing specifies a memory address directly, indirect addressing specifies a memory address by data in a register pair. Indirect addressing will be described in more detail using illustrations.
There is another register called a scratchpad register. This is used to retain information from programs for execution.

Program counter

Any computer employing a stored-program system has a program counter. Since most computers today adopt this system, program counters are very widely used.
The program counter is a register indicating the address in which the next instruction to be executed by the CPU is stored. Since the content of this register is incremented by 1, 2, or 3 each time an instruction is executed, the name 'program counter' rather than 'program register' is given to this register.
Normally, the content of the program counter (PC) is incremented each time the CPU executes an instruction. In the following four cases, however, the content of PC can be changed to specify a particular instruction to be executed.

(1) **Input a RESET signal**
Pressing the RESET switch sets the program counter to 0. In this case, the CPU executes instructions sequentially from memory address 0. When the power supply is turned off, the POWER ON/RESET button also causes the CPU to execute instructions from address 0.

(2) **Execute a JUMP instruction.**

(3) **Execute a CALL instruction.**
The jump address (calling address) enters the program counter, and the CPU executes the instruction stored in that address.

(4) **Execute a PCHL instruction.**
A memory address specified by the content of the H-L register pair (this address is an indirect address) enters the program counter, and the CPU executes instructions from that address.

Stack pointer

The stack pointer indicates an area to which data is to be saved before the processing of a subroutine call or an interrupt is started.
The data thus saved is fetched on a last-in, first-out basis.
It is interesting to note that the content (address) of the stack pointer is decremented each time a data item is fetched. This is the reverse of the program counter whose content is incremented each time an instruction is executed.

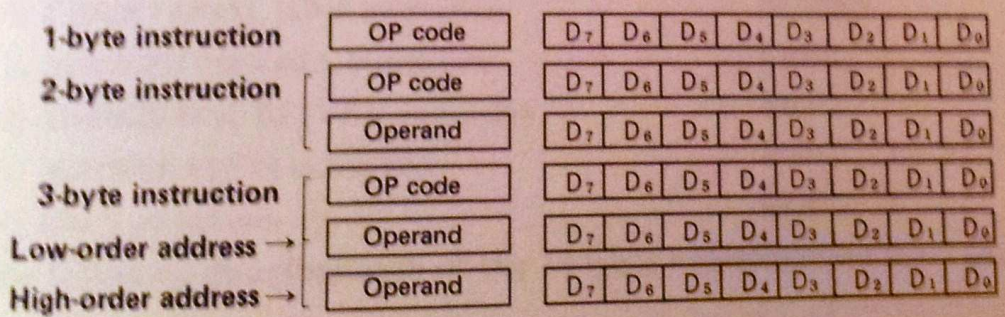
8

Instruction formats

In this chapter, we shall discuss instruction formats. Various instruction formats are devised by many CPU designers. Each instruction format has its own merits and demerits.

In the case of an 8-bit machine, it is possible to provide 256 (2^8) different instructions if all the eight bits are used for this purpose. Although 256 instructions are more than enough for computer operation, dedicating all the eight bits to the instructions is inefficient when performing data transfer between memory and the CPU, since memory addresses should be specified in the instructions.

In the case of 80C85A, approximately 64K (2^{16}) addresses can be directly specified. In this case, each machine instruction requires 3-byte, or 24-bit. Using 24-bit for each instruction is wasteful, since a register-to-register transfer instruction, for example, requires only eight bits. Besides, a 24-bit instruction requires accessing memory three times. In order to solve this problem, the 80C85A permits varying the length of instructions (1-byte, 2-byte, 3-byte) as required, thereby minimizing wasteful use of memory. In this case, it is prerequisite that 2-byte and 3-byte instructions are stored in consecutive memory addresses.



You may wonder why the program counter can positively indicate the next instruction to be executed, without confusing instructions with data. In this respect, the computer is designed quite deliberately. Let us illustrate this.

There is a machine language operation code (OP code) called 3E. Now, assume that a 3E is stored in a certain memory address. If the CPU fetches the 3E for OP code from the memory, the program counter is

automatically advanced by 1 and indicates the byte following the OP code (3E). The 3E thus fetched is subjected to the instruction decoder in the CPU in order to decode the operation code 3E. The CPU identifies this operation code as the IMMEDIATE instruction (a 2-byte instruction to transfer data in a memory directly to a specified register) and reads the contents of the memory address indicated by the program counter. After reading the contents (data), the CPU increments the program counter by 1 to prepare for the next instruction. Now that the CPU is ready to execute the 3E instruction, it immediately executes the instruction. As a result, the 1 byte data following the 3E is transferred to the A-register in the CPU.

By the way, what happens if the 3E instruction is immediately followed by an OP code '76' (HALT instruction)? In this case, the CPU, after encoding the 3E instruction, transfers the OP code 76 as data to the A-register. Thus, the CPU places the HALT instruction (OP code 76) in the A-register as data and proceeds to the next instruction.

This should be kept in mind when creating a program. You may think that the HALT instruction always stops the program when it is executed. However, if the HALT instruction is placed in a wrong address described above, it is completely ignored by the CPU. It may even cause a program overrun.

It must be remembered that instructions in a program or memory have no special mark to discriminate them from data. Special attention should be paid to the JUMP instruction. Specifying an incorrect address may cause the CPU to interpret an instruction as data, or vice versa.

14 basic machine language instructions

The 14 machine language instructions shown below are the most basic ones that permit writing machine language programs of a few steps for the FP-200.
For the moment, you are advised to master the use of these instructions.

Instruction	Symbol	Example	OP code
1) REGISTER-TO-REGISTER TRANSFER instruction	MOV r1, r2	(A)→(B)	47
2) IMMEDIATE instruction	MVI r, B2	(A)←⟨B2⟩	3E
3) LOAD instruction	MOVr, M	(A)←(M)	7E
4) STORE instruction	MOVM, r	(A)→(M)	77
5) ADD instruction	ADD r	(A)+(B)	80
6) SUBTRACT instruction	SUB r	(A)−(B)	90
7) INCREMENT instruction	INR r	(A)+1	3C
8) DECREMENT instruction	DCR r	(A)−1	3D
9) UNCONDITIONAL JUMP instruction	JMP		C3
10) CONDITIONAL JUMP instruction	JZ, JNZ		CA, C2
11) SUBROUTINE CALL instruction	CALL		CD
12) SUBROUTINE RETURN instruction	RET		C9
13) NO-OPERATION instruction	NOP		00
14) HALT instruction	HLT		76

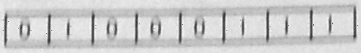
The following describes each of the 14 basic instructions in more detail.

1 REGISTER TO REGISTER TRANSFER instruction

The operation code is as follows:

0	1	D	D	D	S	S	S

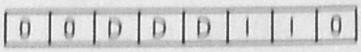
DDD represents the code of a destination register, and SSS represents the code of a source register. For the register codes, see **Table 1-2.**
For example, when transferring data from A to B, DDD is the B-register and SSS is the A-register. Since the B-register code is 000 and the A-register code is 111, the operation code becomes as follows:



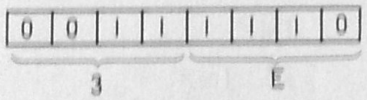
By dividing this operation code into two 4-bit parts, it is possible to represent it in hexadecimal notation. In this particular example, the operation code becomes [47]. In the same manner, it is possible to obtain the operation code for a particular register-to-register instruction (A to C, A to D, B to A, etc.).
The REGISTER-TO-REGISTER TRANSFER instruction is a one-byte instruction and requires four clocks for execution.

IMMEDIATE instruction

This instruction fetches a numeral directly from memory into a CPU register. The contents of the byte following the OP code stored in a specified register.
This OP code is expressed as follows:



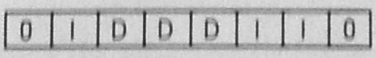
DDD represents the code of the destination register, and the code 110 indicates the memory operation. Assume, for example, that data is to be stored in the A-register. Since DDD (the A-register code) is 111, the operation code becomes [3E] as shown below.



Similarly, the operation code for the instruction to store data to the B-register becomes [06], since DDD (the B-register code) is 000. Also, when the destination register is the C-register, the operation code becomes [0E] (because DDD is 001).
The IMMEDIATE instruction is a 2-byte instruction. Seven clocks are required to execute this instruction.

LOAD instruction

This instruction transfer the contents of a certain memory address into a specified register. This instruction is expressed as follows:



The operation code becomes [7E] for the A-register, since DDD is 111 and [46] for the B-register because DDD is 000.

Wait a minute! Where is a memory address to be specified? Yes, we remember that 16-bit are required when accessing memory. In this case, the registers H and L are used to specify a memory address through indirect addressing. Thus, before executing this instruction, it is necessary to store the higher 8-bit of a memory address in the H-register and the lower 8-bit in the L-register. This can be done by using the IMMEDIATE instruction or some other appropriate method.

The LOAD instruction is a 1-byte instruction. Seven clocks are required to execute this instruction. Note that this instruction really requires about five bytes, since data should be stored in the registers H and L when executing this instruction.

LOAD instruction in this manner, it is possible to fetch data in succession starting from a certain address.

The LOAD instruction is a one-byte instruction. Seven clocks are required to execute this instruction. Note that this instruction really requires about five bytes, since data should be stored in the registers H and L when executing this instruction.

STORE instruction

The STORE instruction transfers the contents of a specified register into a certain memory address.

This instruction is expressed as follows:

0	1	1	1	0	S	S	S

The operation code becomes 77 because the A-register code is 111.

0	1	1	1	0	1	1	1
	7				7		

Here, the problem is the term 'a certain memory address.' How is it specified? As in the LOAD instruction, it is specified by data in the registers H and L, that is, through indirect addressing.

The STORE instruction is a one-byte instruction. Seven clocks are required to execute this instruction. As is the case with the LOAD instruction, the STORE instruction requires two to three bytes taking into consideration the need to enter data in the registers H and L.

In general, after a data transfer instruction is executed, the contents of the destination memory or register changes, while the contents of the source memory or register remains unchanged. Also, it is important to note that the flag does not change after a transfer instruction is executed.

ADD instruction

Normally, the ADD instruction and other arithmetic instructions handle data using the A-register. The OP code is expressed as follows:

I	0	0	0	0	S	S	S

In the case of A+B, SSS is the B-register. Hence,

I	0	0	0	0	0	0	0

The operation code is 80 in hexadecimal notation. The result of A+B is stored in the A-register. Similarly, the operation code for A+C→A is 81 and that for A+D→A is 82.
The result of an operation may cause an overflow, may become zero, etc. The occurrence of such a condition is indicated by a flag in the F-register.

SUBTRACT instruction

The SUBTRACT instruction is expressed as follows:

I	0	0	I	0	S	S	S

In the case of A—B, SSS is the B-register code. Hence,

I	0	0	I	0	0	0	0

Therefore, the operation code is 90.
The result of execution of this instruction is stored in the A-register, and the flag changes accordingly.
As is the case with the ADD instruction, it takes four clocks to execute this instruction.

INCREMENT instruction

The INCREMENT instruction may be considered as a special form of ADD instruction to add 1 to the contents of a specified register. This instruction is used in programs very frequently.
The INCREMENT instruction is expressed as follows:

0	0	D	D	D	I	0	0

In the case of A+1, DDD is the A-register code. Hence,

0	0	I	I	I	I	0	0
		3		C			

The operation code is 3C. The result of A+1 is stored in the A-register. The flag changes according to the result obtained. Four clocks are required to execute this instruction.

DECREMENT instruction

The DECREMENT instruction may be considered as a special form of SUBTRACT instruction to subtract 1 from the contents of a specified register. This instruction is expressed as follows:

0	0	D	D	D	1	0	1

The operation code is 3D for A−1→A, 05 for B−1→B, 0D for C−1→C, and 15 for D−1→D. In contrast with the INCREMENT instruction, the result of execution of the DECREMENT instruction is stored in the appropriate register. Four clocks are required to execute this instruction. The flag varies according to the result obtained.

UNCONDITIONAL JUMP instruction

This instruction is expressed as follows:

1	1	0	0	0	0	1	1

Hence, the operation code is C3.
This instruction requires a jump address, which indicates a memory address to which a jump is to be made. This address is specified by the 2-byte following this OP code.
It should be noted that the first byte inidcates the low-order part of the memory address and the second byte indicates the high-order part of the memory address. For example, if a jump is to be made to memory address 8000, you should specify C3, 00, 80. Note that you should not specify C3, 80, 00, which causes a jump to memory address 0080.
The UNCONDITIONAL JUMP is a 3-byte instruction. Ten clocks are required to execute this instruction.

CONDITIONAL ZERO JUMP instruction

This instruction is expressed as follows:

1	1	0	0	1	0	1	0

The operation code is CA.
This is a 3-byte instruction. As is the case with the UNCONDITIONAL JUMP instruction, of the 2-byte following this OP code, the first byte specifies the low-order part of the jump address and the second byte specifies the high-order part of the jump address.
In the case of the CONDITIONAL ZERO JUMP instruction, however, a condition called ZERO jump is attached. Namely, if the ZERO flag in the F-register is 1, a jump is made to the specified address based on the recognition that the result of the preceding operation is 0; and if the ZERO flag is 0, a jump is not made and the next instruction is executed, since the result of the previous operation is not 0. Thus, this instruction is used as a conditional branch instruction.

The execution time is 10 clocks when a jump is made, and 7 clocks when the next instruction is executed (a jump is not made).
There are several other conditional branch instructions (e.g., the JNZ instruction — OP code C2 — which causes a jump when the result of the preceding operation is not 0). For the moment, try to master the use of this CONDITIONAL ZERO JUMP instruction.

11 SUBROUTINE CALL instruction

When using the machine language in BASIC on the FP-200, it is necessary to use various subroutines stored in the FP-200 ROM for input of data from the keyboard, for display of data on the screen, etc. The SUBROUTINE CALL instruction may be used to call a desired subroutine from the ROM.
The operation code of this instruction is CD. This is a 3-byte instruction, and the second byte specifies the low-order part of the address storing the desired subroutine, and the third byte specifies the high-order part of that address.
As a matter of fact, a subroutine cannot be used if its entry address (start address) is unknown, since the entry address should be specified in the 2-byte following the operation code of the SUBROUTINE CALL instruction.

12 SUBROUTINE RETURN instruction

When using the machine language in BASIC on the FP-200, the BASIC CALL statement is used to call a machine language instruction. The SUBROUTINE RETURN instruction is used to return control from the machine language instruction to BASIC.
The operation code of this instruction is C9.
When a subroutine stored in the BASIC ROM is used, the user need not worry about executing the SUBROUTINE RETURN instruction, since a SUBROUTINE RETURN instruction must be included in that subroutine.
It takes 10 clocks to execute this instruction.

13 NO OPERATION instruction

This is a 'do-nothing' instruction. You might think that such an instruction is unnecessary. Nevertheless, the NO-OPERATION instruction is really useful.
In mathematics, we use the number 0 to indicate nothing. Without the concept of 0, modern mathematics cannot be complete. There is, however, a controversy as to whether or not a no-operation is absolutely required in the same sense as applied to 0 in mathematics.

As a matter of fact, the NO-OPERATION instruction is different in nature from the other instructions discussed in this chapter. As we have already learned, the CPU decodes an OP code by the instruction decoder when fetching it. When the CPU fetches a NO-OPERATION OP code, it increments the program counter and proceeds to the next instruction. It takes four clocks for the CPU to complete this operation. This time interval can be used to take timings of processing, etc. by executing several number of NO-OP instructions. The NO-OP instruction can also be used for debugging a machine language program.
For example, in a machine language program, three to four NO-OP instructions are inserted every 10 other instructions. By doing so, if a bug (i.e., program error) is detected indicating that an instruction need be added, that instruction can be written in the place of a NO-OP instruction. This will eliminate the need to rewrite JUMP instructions, etc. In such a case, the usefulness of the NO-OP instruction makes itself felt.
The operation code of this instruction is [00]. The flag does not change when this instruction is executed.

HALT instruction

This instruction is expressed as follows:

0	1	1	1	0	1	1	0

The operation code is [76].
The HALT instruction is used to stop the program. The CPU itself cannot be released from a HALT state except when an interrupt is made or the RESET button is pressed.
Five clocks are required to execute the HALT instruction. The flag remains unchanged when this instruction is executed.

We have thus far studied the basic machine language instructions. In the next chapter, we shall learn how to use these instructions in actual programming.

10

Writing programs in machine language

In this chapter, we shall study in detail how to create machine language programs, how to use subroutines, and how the machine language is associated with the BASIC language, with the aid of the sample programs used in the early chapters to compare the execution speeds between BASIC and the machine language.

Creating machine language programs

In chapter 2, we used sample programs to draw the entire liquid crystal display and then clear the display in order to compare the execution speed between BASIC and the machine language. We used the POKE statement to write the machine language program in decimal notation. In this chapter, let us write a machine language program in hexadecimal notation.

ADDRESS	OP		ADDRESS	OP	
9F00	1E	Setting number of times	9F0C	9F	
01	0A	10 times	0D	0D	C − 1
02	0E	C←3F	0E	C2	JNZ
03	3F		0F	04	
04	06	B←13	10	9F	
05	13		11	2F	A ← $\bar{A}$
06	CD	Subroutine	12	1D	E − 1
07	8E		13	C2	JNZ
08	02		14	02	
09	05	B − 1	15	9F	
0A	C2	JNZ	16	C9	····· RETURN
0B	06				

As for the BASIC program, a TIME$ function may be placed after the CLEAR statement and before the END statement for the measurement of the execution speed.

```
10  CLEAR, 40703
20  CALL  40704, 255
30  END
```

Let us first look at the machine language program.
The OP code [1E] is stored in the address 9F00. This is a 2-byte instruction to enter data directly into the E-register. In the next address, the code 0A is stored. This represents the decimal number 10, which is transferred in the E-register.
The same applies to [0E] [3F] in the address 9F02 and [06] [13] in the address 9F04. Namely, 3F (decimal 63) is stored in the C-register, and 13 (decimal 19) is stored in the B-register. These values represent the ordinate and abscissa of the liquid crystal display. They are required by a subroutine described below.
The 3-byte from the address 9F06 are a SUBROUTINE CALL instruction, [CD] [8E] [02]. Pay attention to the address specification. The subroutine to draw the liquid crystal display is stored in the address 028E. (8E indicates the low-order part of the address and 02 indicates the high-order part of the address.)
The instruction in the address 9F09 subtracts 1 from the content of the B-register. The Z-flag turns 1 when the result of the subtraction is 0, and it turns 0 when the result is not 0.
The JNZ instruction in the address 9F0A determines the jump address according to the status of the Z-flag after the preceding operation (subtraction). Namely, when the result of the subtraction is not 0, a jump is made to the address 9F06; and when the result of the subtraction is 0, the next instruction in the address 9F0D is executed.
The instruction [2F] in the address 9F11 reverses the contents of A-register. (This instruction was not included in the 14 basic instructions). The RETURN instruction [C9] in the address 9F16 which causes a return to BASIC.
By the way, the JNZ instruction (C2, 06, 9F) in the addresses starting from 9F0A was not included in the 14 basic instructions, either. This JNZ instruction can be replaced by a JZ (CA) instruction. It should be noted, however, that using a JZ instruction requires three additional bytes for the jump address. Hence, the use of a JZ requires changing the addresses for the subsequent JUMP instruction.
In creating a machine language program, address calculations for JUMP instructions often involve some complexity: addresses will have to be changed when instructions are added or deleted.
Here, the usefulness of the NO-OP instruction makes itself felt. When deleting a particular instruction, it is only necessary to replace that instruction with a NO-OP instruction.
In this respect, it is advisable to insert an appropriate number of NO-OP instructions at some appropriate positions in a program. By doing so,

correcting a program can be greatly facilitated. It is also advisable to collect similar functional blocks in groups. This appreciably facilitates program corrections and debugging. It also permits creating easy-to-read programs.

Initialization

AD	OP	COMMENT	AD	OP	COMMENT	AD	OP	COMMENT
9F00	1E	E←0A	9F10	CD		9F20	0D	C−1
01	0A		11	8E		21	CA	JZ
02	0E	C←3F	12	02		22	27	
03	3F		13	05	B−1	23	9F	
04	06	B←13	14	CA	JZ	24	C3	J
05	13		15	1A		25	04	
06	00		16	9F		26	9F	
07	00		17	C3	J	27	1D	E−1
08	00		18	10		28	CA	
09	00		19	9F		29	33	
0A	00		1A	00		2A	9F	
0B	00		1B	00		2B	3C	A+1
0C	00		1C	00		2C	CA	
0D	00		1D	00		2D	02	
0E	00		1E	00		2E	9F	
0F	00		1F	00		2F	3D	A−1

9F30	C3	J
31	02	
32	9F	
33	C9	

Now, we have a program using only basic instructions. The provision of NO-OP instructions in the program shown above will enable even substantial modifications. If you prefer, you may write the instruction C3, 10, 9F from the address 9F06. In this case, the NO-OP instructions after the address 9F09 are skipped.

By the way, what is the function of the instruction 3C, A+1 in the address 9F2B? This is somewhat technical. A decimal number 255 (hexadecimal FF is stored in the A-register by the BASIC CALL statement. Since all the eight bits are 1, if 1 is added to the contents of the A-register, a carry is made, causing all the bits to become 0. As a result, the ZERO flag becomes 1. Thus, the next time the program routine is executed, the contents of the A-register become FF as a result of 0−1.

2 Subroutines

Shown below are subroutines which are frequently used in the Y-ROM.

(1) Key processing subroutines

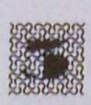
Subroutine name	Entry address	Contents of processing
KRNGC	0D29H	• Resets the key buffer pointer to the beginning. • Conditions: (RINGP) = RINGT (RINGG) = RINGT
INTK	003CH	• Performs key software scanning. If a key is pressed, transfers the associated ASCII code to the key buffer. • Condition: If a key is pressed, the associated key code is transferred to the key buffer and RINGP is updated. RINGP: 833FH RINGG: 8341H RINGT: 8328H
GETRG	0F5CH	• If the key buffer is not empty, transfers a key code from an address indicated by RINGG, to the A-register, and updates RINGG. • Conditions: Z-flag = 1 . . . No key code. Z-flag = 0 . . . A-register = key code; RINGG is updated.

(2) Liquid crystal display subroutines

Subroutine name	Entry address	Contents of processing
BITOU	028EH	• Displays the contents of the A-register. • Conditions: B-register = character X-coordinate (0—19) C-register = graphic Y-coordinate (0—63)
BITIN	0278H	• Reads eight horizontal bits of display data. • Conditions: Same as for BITOU.
XCLS	02F7H	• Clears the display.
OUTAC	0BFAH	• Outputs the ASCII code character in the A-resister to the display controller. (The cursor position is updated.)
PRLCD	02AAH	• Outputs the ASCII code character in the A-register to a specified position. The cursor position remains unchanged. • Conditions: B-register = character X-coordinate (0—19) C-register = character Y-coordinate (0—7)

(3) Printer subroutine

Subroutine name	Entry address	Contents of processing
PRLPT	0216H	Outputs data in the A-register to the printer.



ROM MAP

As you get accustomed to programming using basic machine language instructions, you will wish to use many other machine language instructions and to know more about BASIC subroutines. For your reference, the map of the FP-200 ROM MAP is shown below.

• ROM MAP •

0000H – Initialization
0148H – IOCS No.1 (control of keys, Centronics interfaces, etc.)
036EH – Key samples
047CH – Key code table
061EH – FP-200 system main routines
1051H – Error messages
123AH – Opening message
1282H – Subroutines for initialization
14F6H – CETL commands
1B5CH – CETL command messages
1C5DH – CETL command subroutines
29F7H – BASIC commands
3888H – BASIC commands for numeric expression processing, function processing, etc.
5F0CH – IOCS No.2 (control of FDDs, MT, RS-232C, etc.)
7508H – BASIC graphics commands
773CH – Service routines
77C3H – Object code table
7C17H – PRINT command control
7C57H – Error message table
7CBDH – FP-200 BASIC command table
7CFEH – CETL commands
7DAAH – Function processing
7E0FH – Numeric functions
7E10H – Literals

Much can be learned by examining the contents of ROM. For example, the contents of the addresses starting from 0000H are initialization routine. When the RESET button is pressed or the POWER switch is turned on, the CPU starts the program from 0000H.

AD	OP	
0000	31	SP←8310 Sets 8310 in the stack pointer.
1	10	
2	83	
3	3E	A ← CA
4	CA	
5	C3	JP → To address 0081
6	81	
7	00	
	⋮	

You will see many instructions not included in the 14 basic instructions. Once you have mastered the basic instructions, you will be able to get a rough idea about the operations of other instructions by referring to the 80C85A instruction code table.

Part 2. RS-232C Communications

1

Foreword

Like other personal computers, the FP-200 is provided with an RS-232C interface as a standard data communication facility. Therefore, the FP-200 permits communicating data with a personal computer or hand-held computer over a communication line.

In the following chapter, we shall discuss data communications between the FP-200 and other personal computer via a direct connection cable.

We shall discuss the methods of exchanging programs and data, handling data on external storage devices (cassette tape and floppy disks), and accessing data on external storage.

At the same time, we shall explain the RS-232C protocol and connections between the FP-200 and personal computers in order to provide the reader with accurate knowledge of RS-232C. You may for the moment skip over topics difficult to understand. You may concentrate on the subjects which are of interest to you. In any case, you will be able to have better understanding of the FP-200 and RS-232C by reading the following chapter carefully.

2

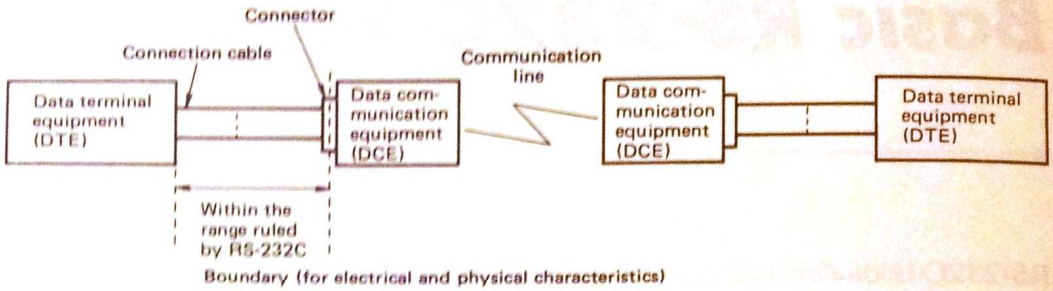
Basic RS-232C information

The RS-232C is one of the protocols established by the Electronic Industries Association (EIA). RS-232C specifies one of the protocols for serial data transmissions. RS-422 and RS-423 provide other protocols.
Of these protocols, the one specified by RS-232C has the longest history, and hence, has been widely applied to serial data transmissions not only by computers, but also by many other devices. In Japan, JIS C 6361 has been established based on EIA's RS-232C. Since the FP-200 employs the RS-232C protocol, once you master the FP-200 RS-232C, you will be able to apply the knowledge to other hand-held computers and personal computers.
In the following sections, we shall discuss the RS-232C protocol as reflected in JIS C 6361 compatible with EIA's RS-232C.
(JIS C 6361 is titled 'Interfaces (25-pin) between Data Communication Equipment (DCE) and Data Terminal Equipment (DTE)' and based on CCITT Recommendations V.24 and V.28 that are compatible with EIA's RS-232C.)

RS-232C connection

The RS-232C protocol is applicable to the scope shown in Figure 1. In the figure, the data terminal equipment (DTE) represents any device connected by an RS-232C interface (e.g., the FP-200, any other hand-held computer, personal computer, printer). The DTE must be provided with a data transmission or reception facility, as well as a communication control facility.

Figure 1. Scope of application of RS-232C



The DCE (data communication equipment) represents modems and acoustic couplers. It must be provided with the function that permits code or signal conversion. As the definitions of the DTE and DCE imply, the RS-232C protocol is concerned with serial data communications over a telephone line. This means that data must be transmitted serially. When we connect the FP-200 directly with another hand-held computer or personal computer, we are connecting two DTEs together. Therefore, we have to change the line connection. We shall discuss the methods of line connections in detail later.

There are various methods of line connections. However, when determining the line connection method, the electrical characteristics of RS-232C interface and other applicable conditions must be taken into consideration. The RS-232C electrical characteristics are shown in Figure 2, and the applicable conditions are shown in Figure 3.

2 RS-232C electrical characteristics

The voltage used for signal transmissions by RS-232C is in the range +5 to +15V or –5 to –15V. Data signals are 0 (low level) on the positive side and 1 (high level) on the negative side. Timing and control signals are on (high level) on the positive side and off (low level) on the negative side. Thus, the on/off levels of data signals are reverse to those of timing and control signals.

The overall effective capacity of the receiver-side impedance is 2,500 pF or less, including the capacity of the connection cable used. Since the connection cable length affects the overall effective capacity, care should be exercised when determining the distance between the DTE and DCE.

The cable capacity varies according to the cable type (cable size, shielded or non-shielded) and the signal rate. As a rule of thumb, 10 to 15m is recommended. If the data transmission speed is low, the cable capacity is not so critical factor as impeding normal data transmission.

Figure 2. RS-232C electrical characteristics

Receiver impedance (impression voltage: positive or negative 3 to 5V)	DC resistance	Min. 3kΩ, Max. 7kΩ	
	Overall effective capacitance	Max. 2500pF	
	Reactance component	Non-inductive	
Signal generator: open circuit voltage		Max. 25V	
Signal voltage (for receiver open circuit voltage 0 and load resistance of 3kΩ to 7kΩ.)		5V – 15V (positive or negative)	
Receiver: open circuit voltage		Max. 2V	
Signal identification	Signal voltage	Min. +3V	Max. –3V
	Data signal	0	1
	Timing and control signals	ON	OFF

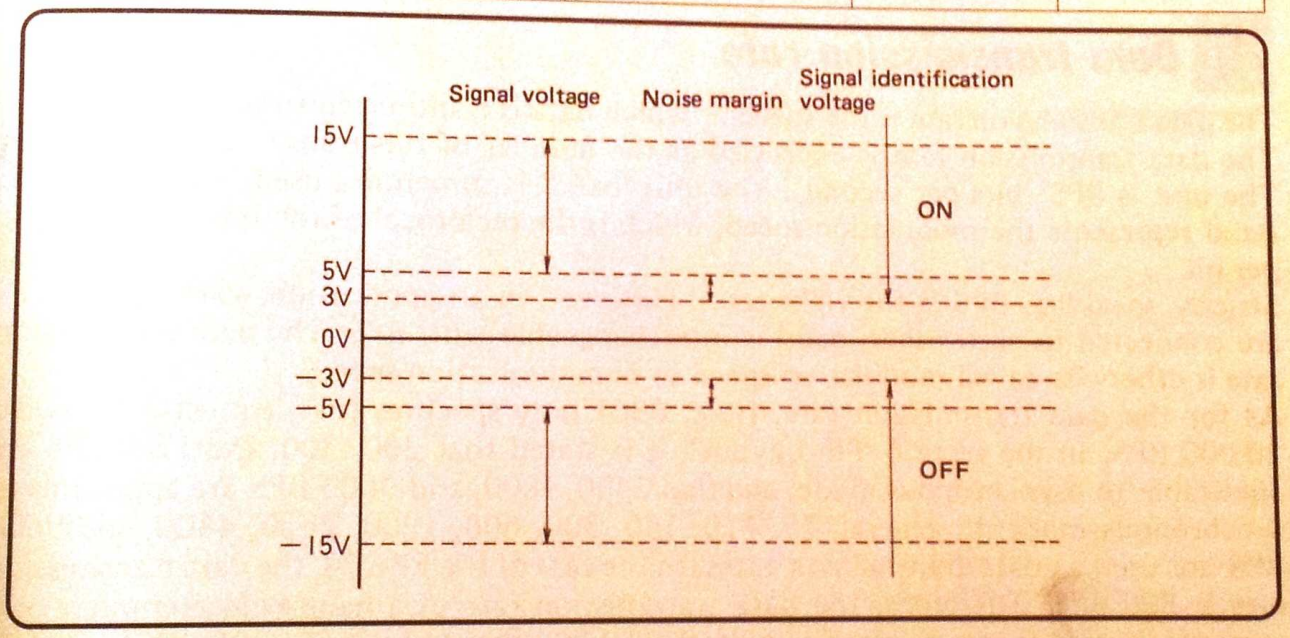


Figure 3. RS-232C applicable conditions

Mode of synchronization	Synchronous or asynchronous
Line type	Leased, switched line
Connection cable	Provided on the DTE side
Data signal rate	Max. 20,000 bps

3 Mode of synchronization

One of the applicable conditions is the mode of synchronization: synchronous or asynchronous. The mode of synchronization specifies how the timing is taken to identify the beginning and end of a data signal (both the sending side and the receiving side must interpret a particular character in the same manner).
In synchronous mode, a synchronization signal is transmitted before data is transmitted. The data is transmitted after the synchronization signal is transmitted. The receiving side uses the synchronization signal to take the timing and starts receiving the data.
In asynchronous mode, a start bit and a stop bit are provided for each character at the sending side. These bits are eliminated at the receiving side. When a parity bit is also provided, it is eliminated as well. Thus, the bits with a start bit, a stop bit, and a parity bit, are assumed as the data.
Like many other hand-held computers and personal computers, the FP-200 employs asynchronous mode regardless of whether it is used as a sending machine or a receiving machine.

4 Data transmission rate

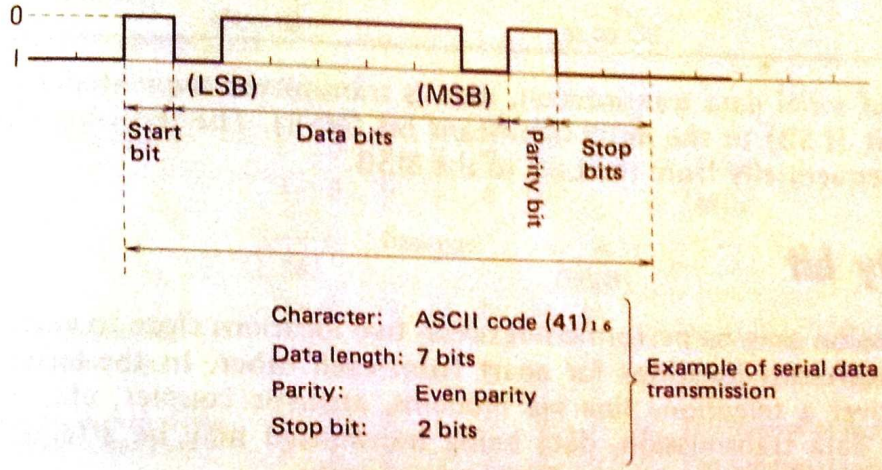
The data transmission rate is the speed at which data is transmitted to or from the DTE. The data transmission rate is expressed as the number of bits transmitted in a second. The unit is BPS (bits per second). The unit 'baud' is sometimes used in place of BPS. Baud represents the modulation speed, which is the reciprocal of the transmission time per bit.
Strictly speaking, BPS is the right term. However, in an application where two DTEs are connected to each other, baud is interchangeable with BPS. The data transmission rate is otherwise called modulation speed or communication speed.
As for the data transmission rate, JIS C 6361 only specifies that it must not exceed 20,000 BPS. In the section 'Pin Layout', it is stated that 200, 300, and 1200 BPS are applicable to asynchronous mode, and that 2400, 4800, and 9600 BPS are applicable to synchronous mode. In general, 75, 110, 150, 300, 600, 1200, 2400, 4800, and 9600 BPS are used as data transmission rates. In the case of the FP-200, the data transmission rate is 300 BPS. Therefore, the data transmission rate of a hand-held computer or a personal computer communicating with the FP-200 should be set to 300 BPS.

5 Character configuration

In performing data communications, it is necessary to make clear the configuration of data characters. The character configuration varies according to the mode of synchronization. The following describes the character configuration for the FP-200 and other hand-held computers and personal computers operating in asynchronous mode.
In asynchronous mode, each character consists of a start bit, data bits, a parity bit, and stop bits. The start bit is a low-level (0) signal. The data bits are a 5- to 8-bit character code expressed by a combination of 0 and 1 (high-level) signals. The parity bit is either 0 or 1 depending on which of the even parity and the odd parity is used and on the data bit configuration.

Some hand-held computers and personal computers need not attach a parity bit (parity checking is not performed). The stop bit may be 1, 1.5, or 2 bits in length. The logic level of the stop bit is 1 (see Figure 4).

Figure 4. Character configuration



Character: ASCII code $(41)_{16}$
Data length: 7 bits
Parity: Even parity
Stop bit: 2 bits

Example of serial data transmission

6 Start bit

In asynchronous mode, a synchronization signal is transmitted each time a data character is transmitted. This signal is the start bit. The data is sent following the start bit.

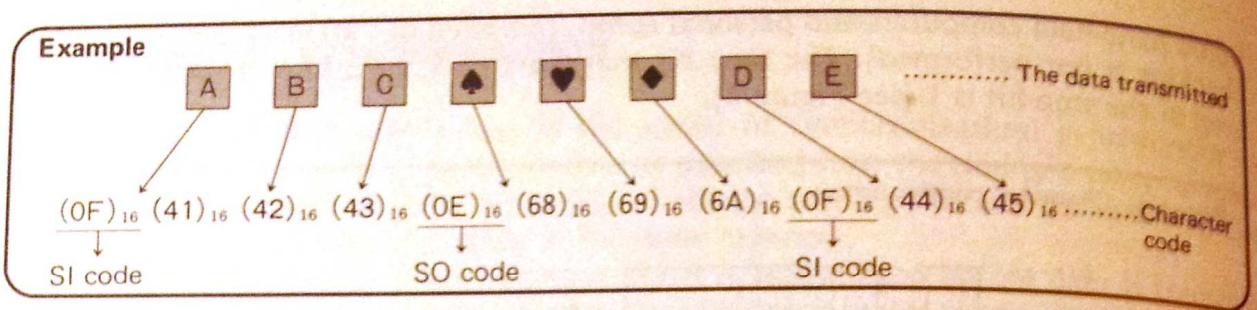
7 Data length

The data bit length varies according to the type of data to be transmitted. If the DTE and DCE communicating with each other use a different data length, the data received differs from the data transmitted. The FP-200 uses 7-bit code characters. Therefore, the data length of the hand-held computer or personal computer used to communicate with the FP-200 should be set to 7 bits.

When the data length is 7 bits, only alphanumeric characters can be transmitted. If a special character (character code $(80)_{16}$ or higher) is transmitted, the most significant bit is dropped (interpreted as 0). As a result, the special character changes to an alphanumeric character (character code 0 to $(7F)_{16}$).

In order to ensure that a special character is received as such, a signal is given indicating that the most significant bit (MSB) of the received data should be interpreted as 1. As a signal serving this purpose, $(OE)_{16}$ is used. This is called the SO (shift-out) code. When it becomes no longer necessary to interpret the MSB as 1, the character code $(OF)_{16}$, called the SI (shift-in) code, is transmitted. Thereafter, the MSB is interpreted as 0 and any special character code changes to an alphanumeric code.

Although the FP-200 can receive special characters through use of SO and SI codes, it cannot receive special characters as such if the counterpart computer does not correctly send the SO and SI codes.

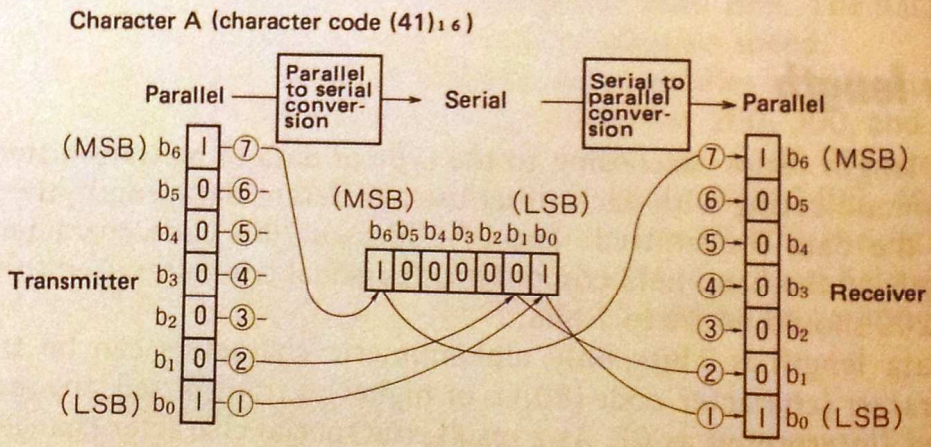


In the case of serial data transmission, data is transmitted sequentially from the least significant bit (LSB) to the most significant bit (MSB). The receiving side also builds up the data sequentially from the LSB to the MSB.

Parity bit

Data transmission may be performed between two locations close to each other (1m or less) or between two locations far apart from each other. In the latter case, data is transmitted over a telephone line via modems, acoustic coupler, etc. In the case of long-distance data transmission, data being transmitted may be affected by external noise, causing a 0 bit to change to a 1 bit, or vice versa.
Assume, for example, that while the character A (7 bits) is being transmitted, bit 3 changes from 0 to 1 due to external noise. In this case, the receiver recognizes this character as I having a character code of $(49)_{16}$. A parity bit is used to detect the occurrence of such a change in bits being transmitted.

Figure 5. Serial data transmission



The bits are sequentially transmitted from the least significant bit (LSB) to the most significant bit (MSB). They are then accumulated serially from the LSB to MSB.

Parity checking is performed in two different ways: one is even parity and another is odd parity. In the case of even parity, the total of logic level 1's is even; and in the case of odd parity, the total number of logic level 1's is odd (see Figure 6).
A parity bit of logic 0 or 1 is attached by the transmitter. Thus, the receiver must use the same parity as the transmitter for parity checking. The FP-200 uses even parity

regardless of whether it is used to transmit or receive data. Hence, the counterpart of data communications should use even parity.
Note that all errors in parity cannot be detected. For example, if an even number of bits changed at the same time or if the parity bit itself dropped, the error arising therefrom cannot be detected.

Figure 6. Parity check

Character A (character code $(41)_{16}$)

	(MSB)						(LSB)		
	b_6	b_5	b_4	b_3	b_2	b_1	b_0		
(1) If bit 3 of the data bits changes	1	0	0	0	0	0	1	$(41)_{16}$	
	1	0	0	1	0	0	1	$(49)_{16}$	
	←	Data bits					→		
	(LSB)						(MSB)		
	b_0	b_1	b_2	b_3	b_4	b_5	b_6		
(2) In the case of even parity	1	0	0	0	0	0	1	0	The number of logic 1's is 2 (even).
In the case of odd parity	1	0	0	0	0	0	1	1	The number of logic 1's is 3 (odd).
(3) If bit 3 changes in even parity	1	0	0	1	0	0	1	0	The number of logic 1's is 3 (odd).
If bit 3 changes in odd parity	1	0	0	1	0	0	1	1	The number of logic 1's is 4 (even).

9 Stop bit

A stop bit is used to indicate the end of a character. The stop bit is 1, 1.5, or 2 bits in length. The FP-200 uses two bits for the stop bit. The stop bit length must be the same between two computers communicating with each other.
The voltage of the line has a logic level of 1 when no data is being transmitted over the line. In asynchronous mode, a start bit is transmitted before the data bits. This start bit has a logic level of 0. Then, the data bits are transmitted from the LSB to the MSB.
Then, if parity checking is performed, one bit of logic level 0 or 1 is transmitted according to the method of parity checking and the number of logic level 1's in the data bits.
The stop bit is sent at the end of each character. The receiver identifies the end of a character when it receives a stop bit.

10 Connection cables

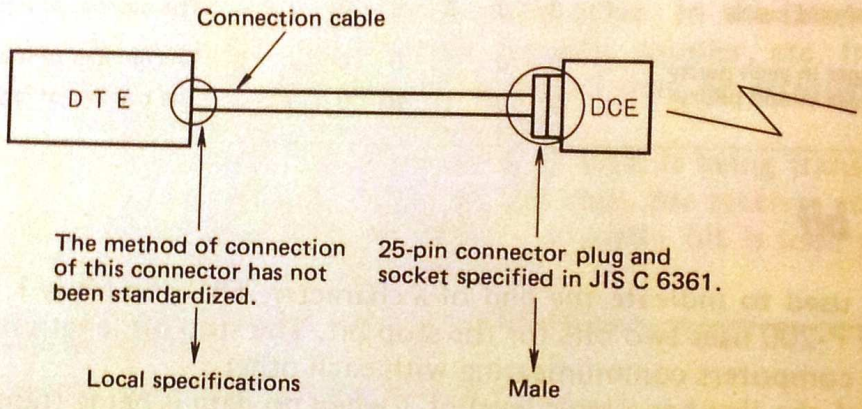
The cable connecting the DTE and the DCE is provided on the DTE side (JIS C 6361). Type DB-25 25-pin connector, abbreviated the D-sub, is used for connection with a DCE. The fact that the RS-232C connector shape and pins provided on personal computers and hand-held computers have not been standardized is ascribable to the absence of standard DTE connectors. As for the pins, only the circuit applications have been standardized. The voltage (logic level), etc., have not been specified.
This is due to the fact that the RS-232C standard was established after RS-232C connectors had been disseminated in the private sector. Makers of computers and peripheral devices use pins of their own choice for their connectors. The pin connections, circuit voltage, etc., established by individual computer makers are called local specifications.

The fact that the circuit voltage is not standardized is a bottleneck to the use of personal computers and hand-held computers for serial data transmission in the future.

Connectors

A 25-pin connector is provided at one end of the RS-232C connection cable. This is a standard connector used to connect a DTE and a DCE. This connector is commonly known as the D-sub.
There are various types of connectors (soldered or crimped to the connection cable, etc.). For example, the socket DB-25S and the plug DB-25P are available on the market.
Connectors are shown in Figure 7, and the shapes and dimensions are shown in Figure 8.

Figure 7. Connection cables and 25-pin connectors



FP-200 RS-232C connection cable

RS-232C connection cable using flat cable

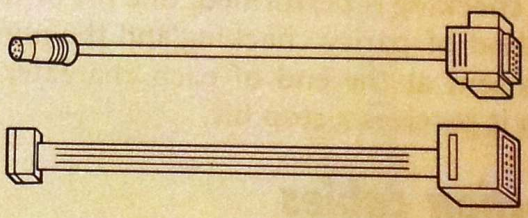
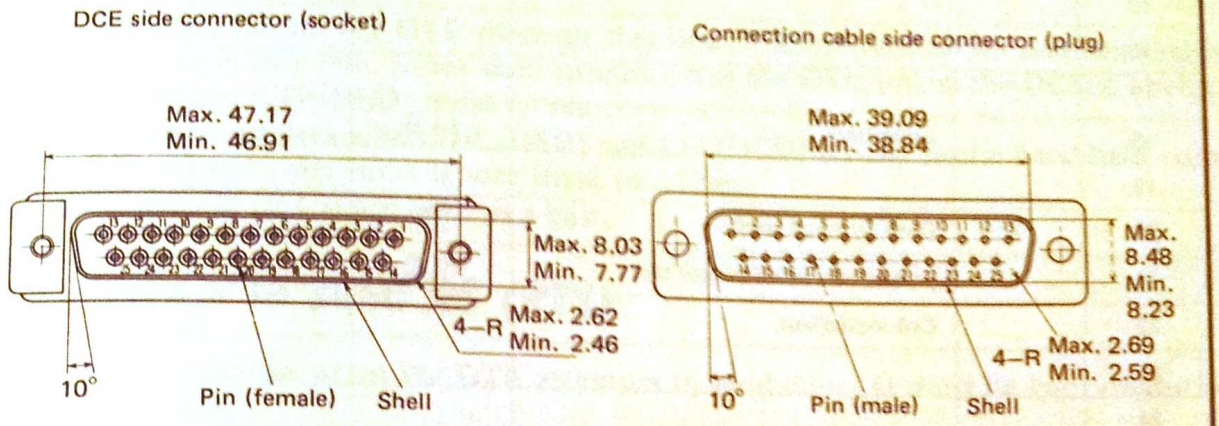


Figure 8. 25-pin connector shapes (plug and socket)



12 Pin layout

The pin layout varies in the meanings and the uses of pins according to the mode of synchronization and the data transmission rate. Since the FP-200 operates at 300 BPS in asynchronous mode, we shall explain the pin layout for the data transmission rate of 300 BPS and asynchronous mode operation. Figure 9 shows the pin numbers, pin names, and signal directions. The pin names are those specified in JIS C 6361. JIS abbreviations and popular ones are shown. The signal direction indicates the direction of data flow between DTE and DCE.

Figure 9. Connector numbers, names, abbreviations, and signal directions

Connector number	FP-200 DIN 8-pin	Name	Abbreviation		Signal direction	
			JIS	Common	DTE	DCE
1		Shield (frame) ground		FG		
2	3	Send data	SD	TXD	○	→
3	4	Receive data	RD	RXD	←	○
4	8	Request to send	RS	RTS	○	→
5	6	Clear to send	CS	CTS	←	○
6	5	Data set ready	DR	DSR	←	○
7	2	Signal ground or common carrier	SG	GND		
8	7	Data carrier detect	CD	DCD	←	○
9						
10						
11		Select send frequency	SSF		○	→
12						
13						

14					
15					
16					
17					
18		Loop-back	LLB		○→
19					
20	1	Data terminal ready	ER		○→
21		Loop-back/maintenance test	RLB		○→
22		Call indication	CI		←○
23					
24					
25		Test indication	TI		←○

In Figure 9, a blank in the pin number column indicates that the associated pin is unavailable with the FP-200. A blank in the name column indicates that the associated pin is not used for 300 BPS, asynchronous mode operation. A blank in the abbreviation column (common) indicates that the associated abbreviation is the same as the JIS abbreviation or there is no abbreviation.
The signal direction indicates the direction of signal flow from DTE or DCE (marked with a circle). Pin numbers 1 and 7 are provided to equalize signal reference potentials, hence they have nothing to do with the signal direction. There is no JIS abbreviation for pin number 1.

The following describes the pins used by the FP-200. The abbreviations are shown in parentheses.

13 Send data SD (TXD)

This line sends data in data transmission. Data is sent from this line only when the following four lines are on. These four lines can be used for handshaking (i.e., to send or stop data according to the status of the counterpart).
In this case, all the lines must not always be turned on: only one line can be used for handshaking. Alternatively, the four lines are sequentially turned on and when all the lines are on, data can be output to the SD (TXD) line.

25-pin number	FP-200 pin number	Name	Abbreviation
4	8	Request to send	RS (RTS)
5	6	Clear to send	CS (CTS)
6	5	Data set ready	DR (DSR)
20	1	Data terminal ready	ER (DTR)

14 Receive data RD (RXD)

Data transmitted enters the DTE through this line. The convention for handshaking is not available with this line. Since data cannot enter the DTE unless the DCE is operating, DR (DSR) and CD (DCD) must be on.
The FP-200 receives data when DR (DSR) and CD (DCD) are on. Some hand-held computers and personal computers ignore these two lines.
SD (TXD) and RD (RTS) are used as a pair.

15 Request to send RS (RTS)

This line is turned on when the DTE attempts to send data. It must be kept on during data transmission.
The FP-200 outputs an off voltage during data reception, and an on voltage during data transmission. This line is used primarily to control the DCE. When it is turned on, the DCE becomes ready to send. The fact that the DCE becomes ready to send can be checked by the CS (CTS) line.

16 Clear to send CS (CTS)

This line indicates that the DCE is ready to send. When RS is turned on, the DCE turns on CS (CTS). When data transmission is completed, RS (RTS) is turned off and the DCE turns off CS (CTS).
RS (RTS) and CS (CTS) are used as a pair.

Figure 10. SD (TXD) and RD (RXD)

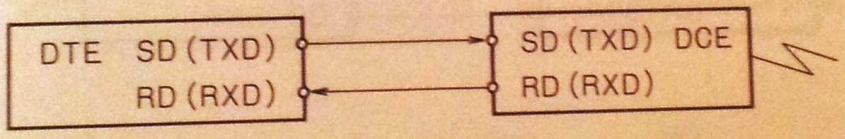
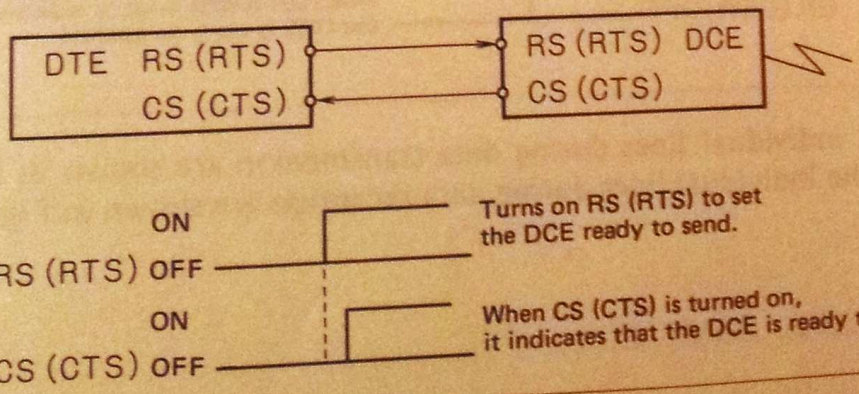


Figure 11. RS (RTS) and CS (CTS)



17 Data set ready DR (DSR)

This line indicates that the DCE is ready to operate. This line is turned on by the DCE. When this line is on, it indicates that data can be sent or received via the DCE. After receiving the status of the DR (DSR) line, RS (RTS) is turned on when data transmission is to be performed. With the FP-200, DR (DSR) must be on during data transmission or data reception.

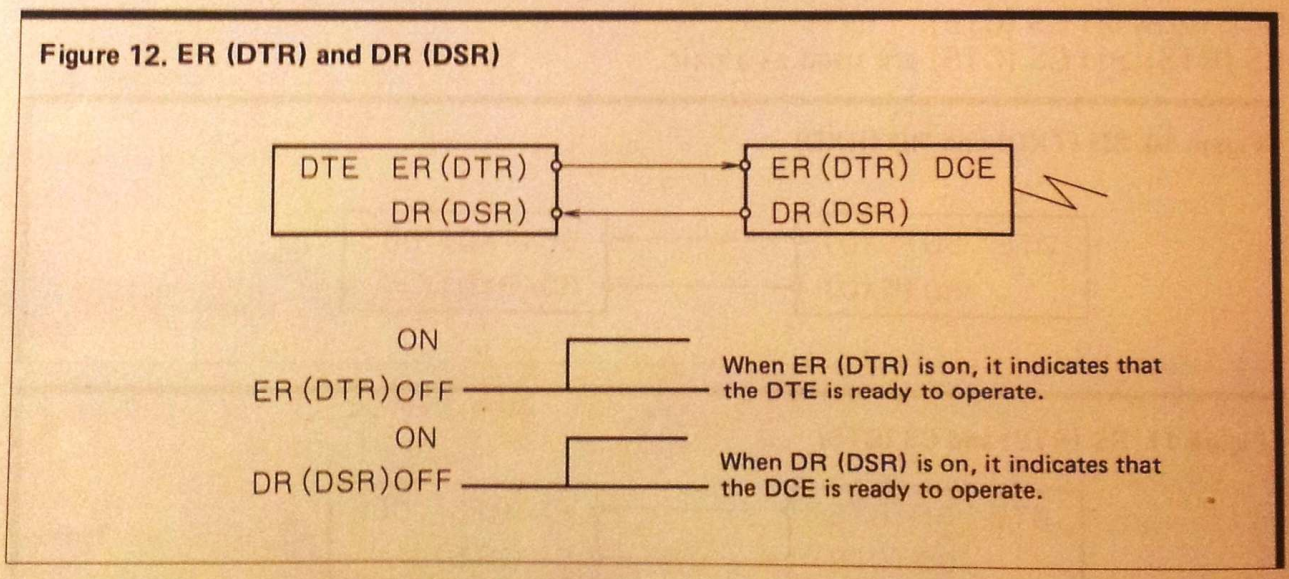
18 Data carrier detect CD (DCD)

When the DCE receives data, it turns on this line, informing the DTE that the DCE is receiving data. The DTE starts receiving data when the CD (DCD) line turns on.
The FP-200 cannot receive data unless both DR (DSR) and CD (DCD) are on.

19 Data terminal ready ER (DTR)

ER (DTR) indicates that the DTE is ready to operate. When the DTE is going to send or receive data, ER (DTR) must be on.
With the FP-200, ER (DTR) turns on the line voltage when the power supply is turned on. This condition is retained until the power supply is turned off.
ER (DTR) and DR (DSR) are used as a pair.

Figure 12. ER (DTR) and DR (DSR)



The timings of individual lines during data transmission are shown in Figure 13, and the timings of the individual lines during data reception are shown in Figure 14.

Figure 13. Data transmission

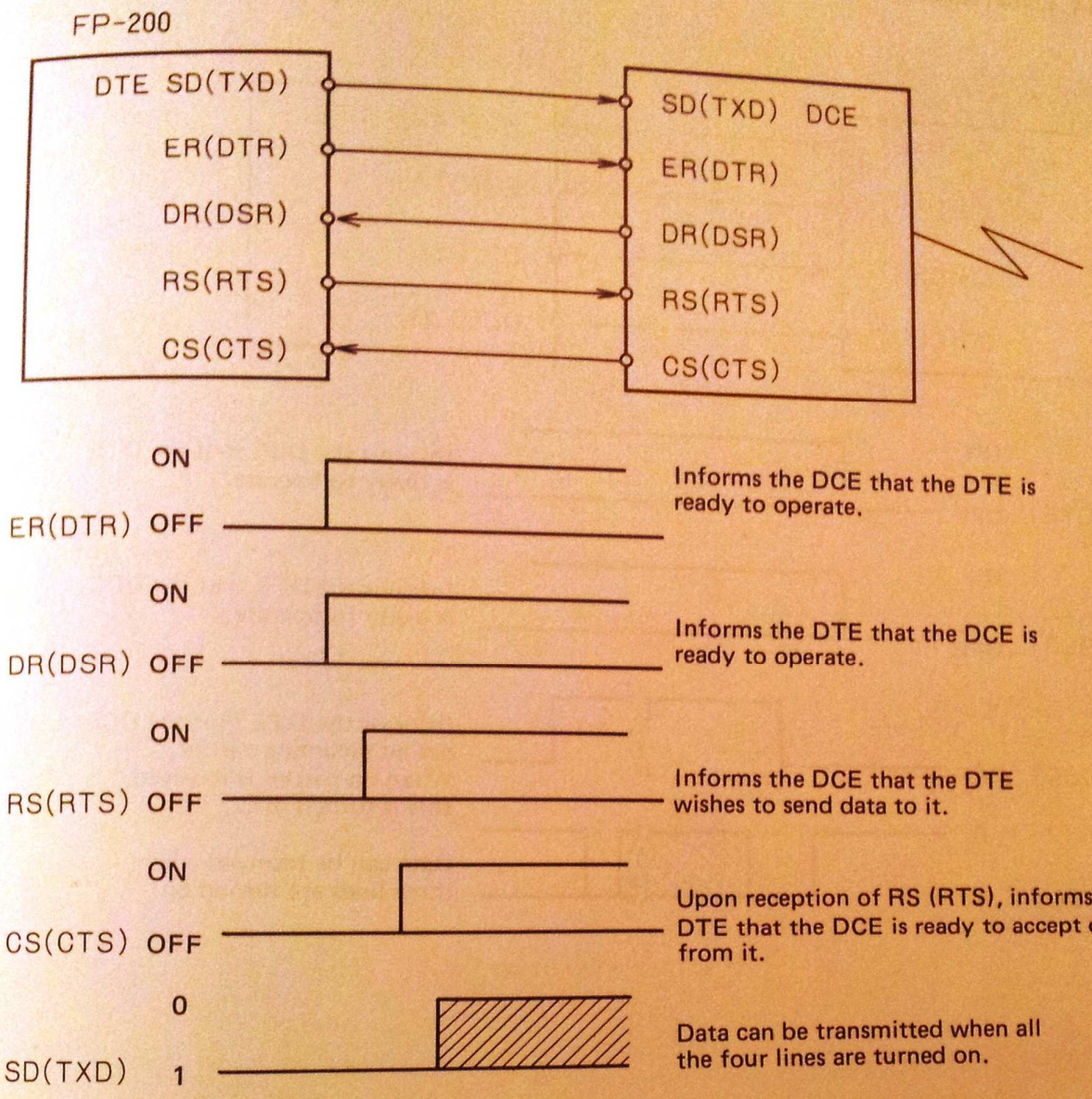
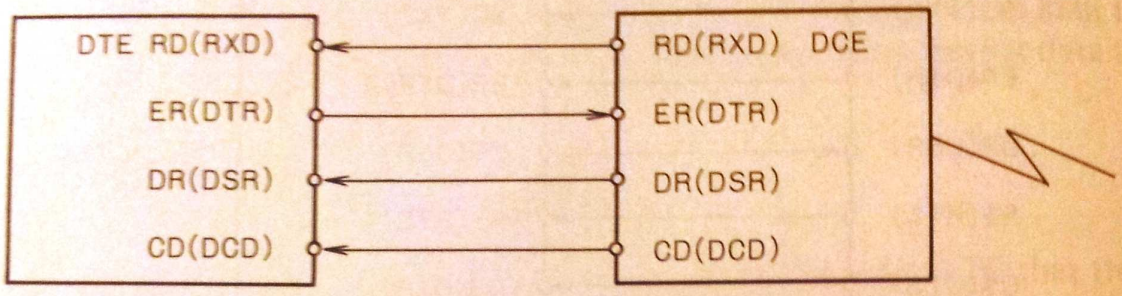
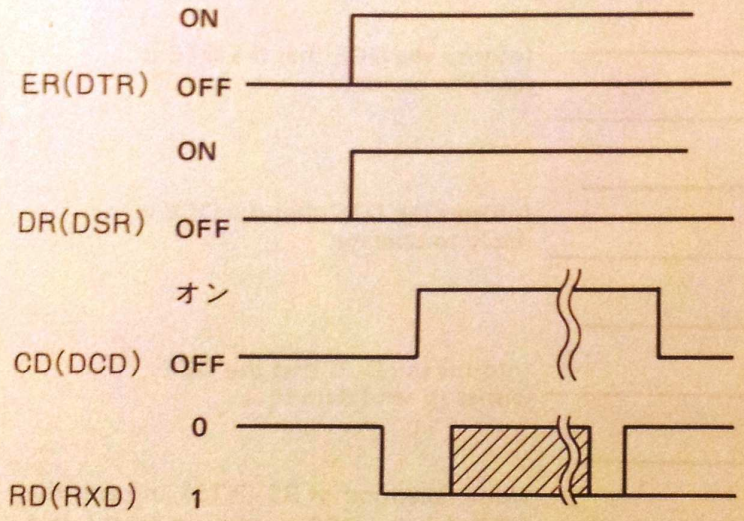


Figure 14. Data reception





Informs the DCE that the DTE is ready to operate.

Informs the DTE that the DCE is ready to operate.

Informs the DTE that the DCE has an incoming carrier. When no carrier is received, this is turned off.

Data can be received when three lines are turned on.

3

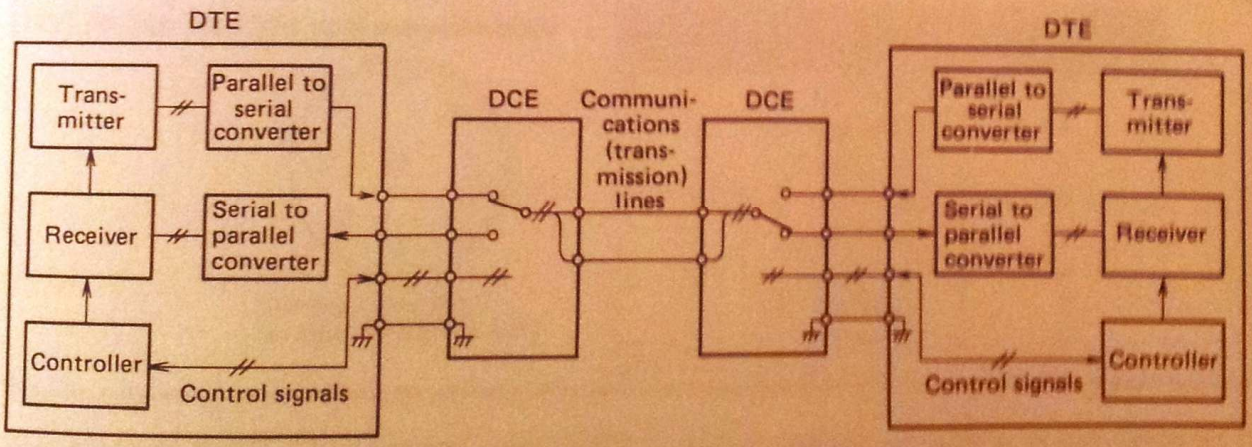
Basic information about communication modes

Data communications are performed in either half- or fullduplex communication mode. The FP-200 uses the half-duplex mode which alternates transmission and reception; that is, FP-200 receives while the other computer transmits and the FP-200 transmits while the other receives. Half-duplex communications can be accomplished by two transmission lines. See Fig. 15.

Full-duplex communications require four independent lines. Each of the two computers requires two lines; one each for transmission and reception. Thus, transmission and reception can occur at the same time. See Fig. 16.

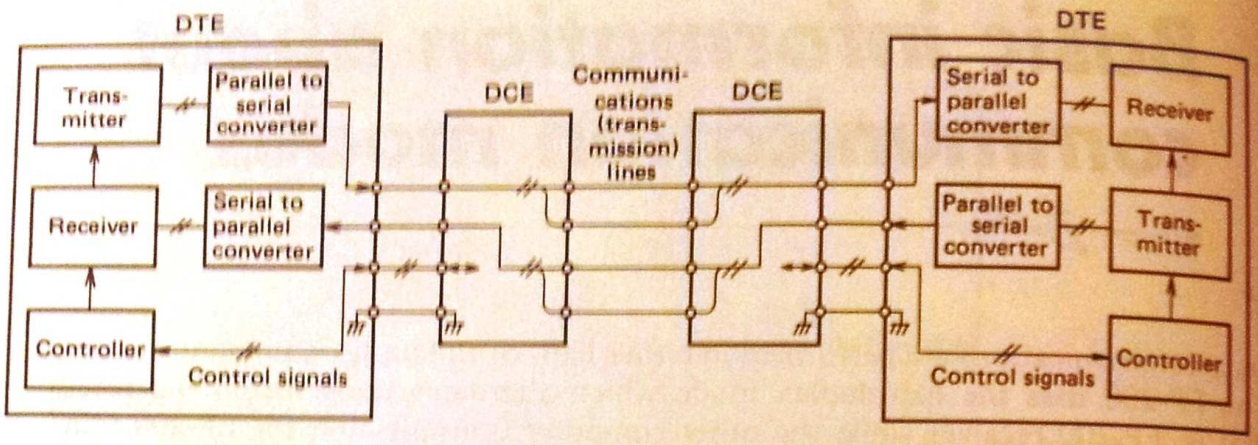
The above discussion stated that a full-duplex communication requires four independent lines. However, it may be accomplished via only two lines by using different DCE modulation frequencies for transmission and reception.

(15) Conceptual half-duplex communication block diagram



The DCEs are switched for either transmission or reception by a control signal from the controller.

(16) Conceptual full-duplex communication block diagram



4

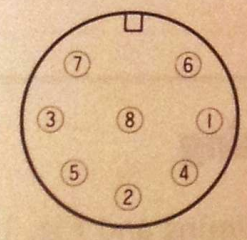
Preparation for FP-200 serial data communication

As you have learned through previous discussions, the FP-200 RS-232C interface conforms to the JIS-C6361 standard. Detailed descriptions have been focussed on the RS-232C protocol. Its protocol defines everything required to use it.
You are now fully prepared for the following discussions.

RS-232C terminal on the FP-200

The hardware connections, including the signal cable connectors, etc., are summarized here.
Fig. 17 shows the FP-200 RS-232C interface signal terminal connector.
It should be noted here that a special AC adapter (model AD-4180) is required when using the RS-232C interface or a printer or floppy disk drive (FDD) because the internal battery supply is insufficient.

(17) FP-200 RS-232C terminal connector



DIN 8-pin connector
(as viewed from the rear)

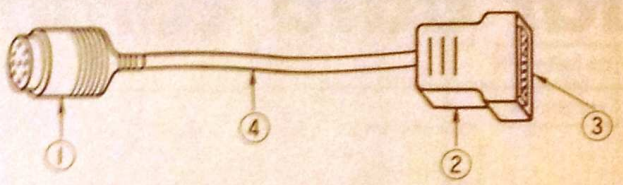
Pin number	Signal name	25-pin connector pin number
1	ER(DTR)	20
2	SG(GND)	7
3	SD(TXD)	2
4	RD(RXD)	3
5	DR(DSR)	6
6	CS(CTS)	5
7	CD(DCD)	8
8	RS(RTS)	4

The FP-200 RS-232C terminal uses DIN 8 pin a connector which is the same as the adjacent cassette tape recorder (CMT) terminal connector. Be sure to use the correct one. If you connect the CMT cable to the RS-232C terminal, the CMT will not be damaged. If the RS-232C cable is connected to the CMT terminal, however, the FP-200 may be damaged by unexpected voltages which can be fed through this cable from the connected external device.

2 Connection cables

The FP-200 RS-232C signal cable is available as model FP-280RSC. It is highly reliable. If you want to assemble the cable by yourself, however, the following parts are necessary.

(18) Parts for RS-232C signal cable assembly



	Parts	Remarks
①	DIN 8-pin plug	Select correct size model
②	D25 connector cover	
③	DB25P connector	
④	8-wire shielded cable	1 to 2 m

The assembly requires soldering at both the DIN 8-pin and DB25P connectors.

3 Communication protocol

Communication		Half duplex
Synchronization		Asynchronous
Transmission speed		300BPS
Transmission character configuration	Start bits	1 bit
	Data character length	7 bits
	Parity check	Even parity
	Stop bits	2 bits

4 Data transmission/reception timing

Refer to Fig. 13 for the FP-200 data transmission timing and Fig. 14 for the reception timing.

When transmitting, the FP-200 monitors the state of the DR (DSR) and CS (CTS) signal lines each time it transmit a character. These signals, respectively, inform the FP-200 that the DCE is operating and that the FP-200 may transmit the next data character. If the receiving computer wants to suspend character transmission from the FP-200, therefore, either signal should be deactivated (turned off).

When your FP-200 "shakes hands" with another computer for communications, one or both of the signals are used. CS (CTS) is essential.

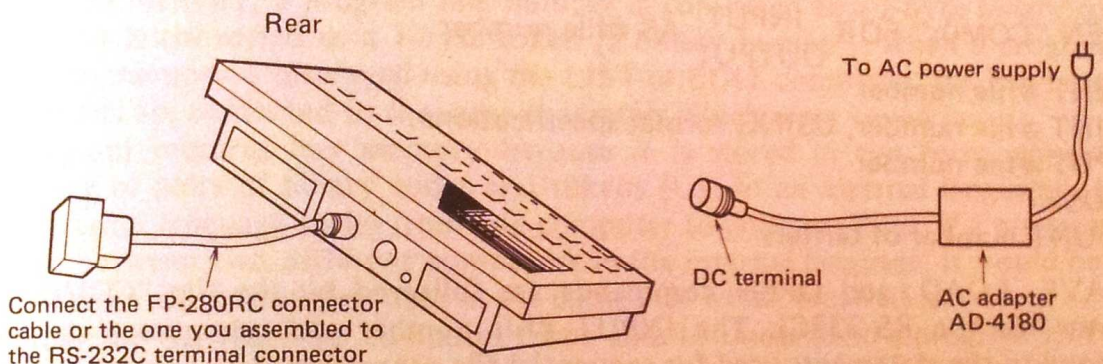
When a transmission from the FP-200 is completed, the RS (RTS) signal turns off; this signal may be used to inform the other computer of the end of the program or data transmission.

Preparation for serial data communication (hardware)

Fig. 19 illustrates the components required to use the FP-200 for serial data communications.

Insert the DIN 8-pin plug of the signal cable (the FP-280RSC or the one you assembled) into the RS-232C terminal connector receptacle at the rear of the FP-200. Make a practice of turning off all devices, including the other computer, before connecting them to the FP-200. This can prevent damage which may be caused by a wrong connection, and prevents incorrect operation due to faulty contacts between pins and jacks. Then connect the DC plug of the special AC adapter (AD-4180) to the DC terminal receptacle at the right side of the FP-200 and plug the AC plug into an outlet. The FP-200 is now prepared for serial data communications.

(19) **Necessary parts for serial data communications.**



5

FP-200 serial data communication software

The FP-200 has the following program transmission and reception commands and statements which are used to provide communications capabilities via the RS-232C interface:

```
SAVE "COM0:", A
LOAD "COM0:"
OPEN "COM0:" FOR INPUT / OUTPUT AS #file number
PRINT #file number
PRINT #file number, USING format specifications;
INPUT #file number
CLOSE
MOUNT number of buffers
```

The SAVE, LOAD, and OPEN commands are followed by the file "COM0:" (the descriptor for the RS-232C). The INPUT #file number and subsequent statements correspond to the statements used for sequential file processing.
This means that the RS-232C, as well as cassette tape, floppy disk, and printer, deals with sequential data files. That is, you can accomplish serial data communications using the RS-232C exactly the same as reading from or writing to a sequential file created on a floppy disk or cassette tape. Most problems which prevent successful communications via RS-232C are not attributable to software; there may be some error or errors in the hardware connections.
No interrupts are allowed during data reception because no statement is available which provides this capability.

Transmitting a program — SAVE "COM 0:", A

The SAVE "COM0:",A command allows the FP-200 user to transmit a program to another personal computer via the RS-232C interface in the same way as when recording a program on a cassette tape or floppy disk. This command directs the program which has been stored in the specified program area to the RS-232C terminal, character by character in the ASCII format.

When recording a program on a cassette tape, you use the command SAVE"CAS0: filename", A which directs the program to the CMT terminal. That is, the only differences between saving a program as a cassette file and program transmission via RS-232C are whether a filename follows the command word SAVE or not, and whether the program is directed to the CMT or to the RS-232C terminal.

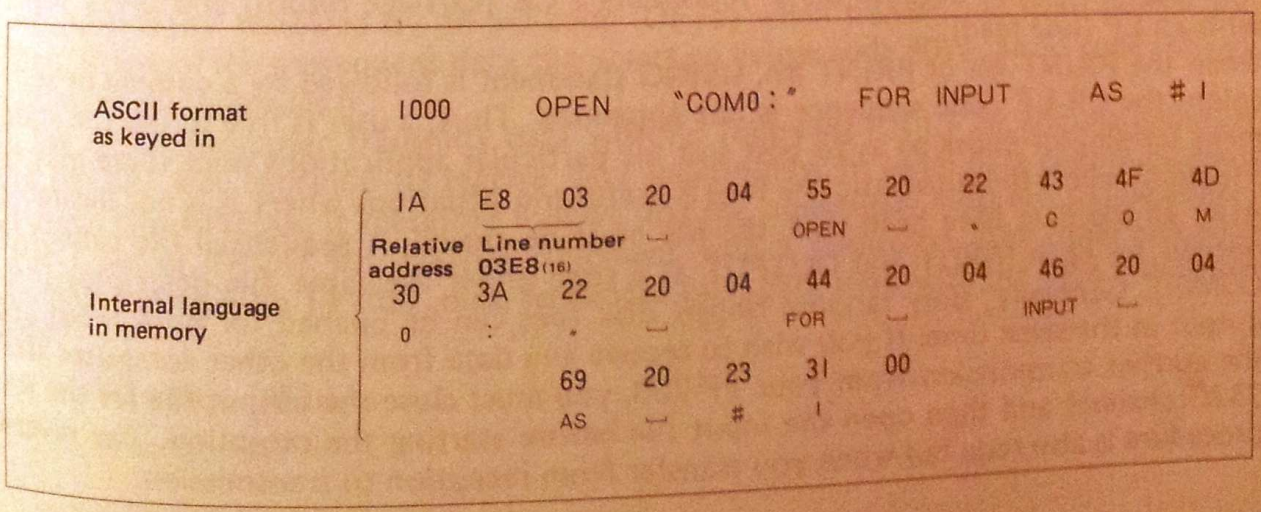
It should be remembered that this command always needs to be followed by a ", A" which instructs the FP-200 to send the program in the ASCII format.

When entering a program from the keyboard, you key in line numbers and statements exactly as they would appear on the listing which is output to the printer. When stored in FP-200 memory, a program line number is converted to a 2-byte binary code and a statement is converted to a 1-byte token (a binary number). When a program already stored in memory is displayed using the LIST or EDIT command, the line numbers and statements are converted to the same characters which were keyed in.

A program requires less memory because it is stored in this form, consisting of a sequence of pairs of binary codes and tokens (i.e., in an internal language). However, this internal language varies from one computer to another. Thus, if a program is transmitted between two different computers in the internal language, it would be no more than a block of meaningless data in the computer which received it.

In order for the receiving computer to be able to understand a program sent from the other computer, whatever the computer may be, it should be transmitted as ASCII data which can be converted to the internal language of the receiving computer in the same procedure as if it had been keyed in.

A SAVE "COM0:" command followed by ", A" causes the internal language program to be directed to the RS-232C terminal after reconverting it to an ASCII string. If the SAVE "COM0:" command were not followed by ", A", the program would be transmitted in the original internal language.



2 Receiving a program — LOAD "COM 0:"

Programs can be received as well as transmitted. This provides various capabilities, such as debugging a program to be run on another computer by using your FP-200, etc., as well as simply reading an FP-200 program entered at another computer via the RS-232C. Programs to be received by the FP-200 must be in ASCII. When receiving an ASCII program, the FP-200 stores it in memory by converting each line number to a binary code and the following statement to a 1-byte token.

During this process, the FP-200 may drop parts (i.e., ASCII characters) of the program because it does more processing than the transmitting computer. This may cause the next program line to arrive before the current line has been processed.

Our measures to cope with this problem are to wait an appropriate interval between program line transmissions at the transmitting computer, because there is no way for the receiving computer to report any character dropouts during reception once it has started. Actual programming techniques are explained later (from page 98 on).

The point here is that you can receive a program from another computer using this LOAD "COM0:" command.

3 Data transmission — OPEN "COM 0:" FOR OUTPUT AS # n

When transmitting a physical data record (a line or logical record), from the FP-200, you first declare, using an OPEN statement, that you are going to use the serial data communications channel (i.e., the RS-232C). Executing the OPEN statement causes the data in the specified sequential file to be accessed and, at the same time, the RS (RTS) signal to be activated (or turned on) if you have specified OUTPUT (i.e., transmission) in this statement. The FP-200 becomes ready for the transmission when all of the signal ER (DTR), DR (DSR), and CS (CTS) turn on in addition to RS (RTS). These signals control RS-232C line transmission and reception, as discussed before. A PRINT #n or PRINT #n, USING statement directs the data specified in it to the RS-232C terminal. If either statement is not followed by a semicolon or comma, a sequence of two characters ($0D_{16}$ and $0A_{16}$) is automatically appended to the data. This causes one line to be fed (i.e., the cursor is moved to the beginning of the new line) at the receiving computer when the data is received, with no extra actions required at the receiving computer, because $0D_{16}$ represents a CR (carriage return) and $0A_{16}$ represents an LF (line feed).

When the PRINT #n or PRINT #n, USING statement is followed by a comma or semicolon, the CR-LF code sequence is not appended. Thus, a user is free to use the statement terminator codes so as to best suit his particular applications since there may be situations where the automatic line feed is useful and situations where it is not desirable.

After all the data have been sent, the file must be closed. A sequential file cannot be opened for both INPUT and OUTPUT at a time under the same file descriptor (e.g., COM0:). Generally, only random floppy disk files can be opened for both input and output at the same time. If you wish to receive any data from the other computer after the current transmission from your FP-200, you must close the output file for the RS-232C channel and then open the input file before starting the reception. The reverse procedure is also required when you transfer from reception to transmission.

Data reception — OPEN "COM 0:" FOR INPUT AS #n

When receiving a physical data record (a line or logical record) from another computer, you declare using an OPEN command, similarly to initiating data transmission, that you are going to use the serial data communications channel (RS-232C). Executing this statement causes the data to be available as a physical record in the specified sequential file for access via the RS-232C, and the reception is initiated when the three signal lines ER (DTR), DR (DSR), and CD (DCD) are activated (turned on).
The received data can be accessed by your program by using a INPUT #n, variable name statement. If the data is followed by CR-LF code sequence or a comma, it is treated as the end of the data. The data is stored in the specified variable, with the data delimiter discarded. Your program should receive the next physical data record after the processing defined by the subsequent statements is completed. This is a typical sequence of events which is used to receive a number of data records.
Some processing must usually be done when each data record is received but before receiving the next record. If the next record arrives during in this processing, it cannot be successfully received (i.e., the record is dropped). Thus, the next record transmission must be withheld until the processing is completed. There are several methods for accomplishing this, such as letting the transmitting computer withhold the next transmission for the time required for the processing, issuing a signal from your FP-200 when the processing is completed to let the transmitting computer transmit the next data after receiving the signal, letting the computers alternate transmission and reception, and so forth.
The second and third methods are reliable but have the drawbacks of placing a heavier load on both of the computers. Furthermore, they are not suitable for transmitting or receiving a larger amount of data. The first method is the simplest; only the transmitting computer has to adjust its wait time. However, the time needs to be varied depending on the transmitted data length and the FP-200 program.
The FP-200 cannot process all arriving data. This is because of a restriction imposed by the INPUT #n statement; the data characters from 0 to $1F_{16}$ followed by the delimiter "," (i.e., a CR-LF code sequence) and data characters from 80_{16} to EF_{16} which are not preceded by an SO code cannot be assigned to the specified variable.
The character codes 80_{16} and above are processed as character codes from 0 to $7F_{16}$ when they are not preceded by an SO code. Any character code of 80_{16} or above is unconditionally followed by an SO code (OF_{16}) when transmitted from the FP-200. Then, if any of the subsequent data characters go below code 80_{16}, an SI code ($0E_{16}$) automatically precedes it.
If the other computer transmits data with no following CR-LF or comma delimiter, the FP-200 will continue to receive data indefinitely. In serial data communications between two FP-200s, it therefore follows that the transmitter must not follow either a PRINT #n or PRINT #n, USING statement with a semicolon. Any other computer communicating with FP-200 must terminate each data item with the CR-LF delimiter code.

SO code $(0E)_{16}$
SI code $(0F)_{16}$

When the last data is received, close the RS-232C receive file using a CLOSE statement. If you want to transmit data after the reception, CLOSE the file and OPEN it again as the sending file.

Summary of serial data communications (software)

The FP-200 serial data communications commands and statements are summarized in table 20. It should be noted that FP-200 uses a 7-bit data character length, and whether received data are followed by the delimiter CR-LF or not. The other computer must after transmitting each physical data record, withhold the next transmission to the FP-200 until the latter completes all post-reception processing, and must follow each data record with a CR-LF delimiter or a comma.

(20) Summary of FP-200 serial data communications software

	FP-200	Origin/Destination — or Direction of transmission	Other computer
Program transmission	SAVE "COM0:", A	⟶	Program reception
Program reception	LOAD "COM0:"	⟵	Program transmission (Withhold next transmission after transmitting each line.)
Data transmission	OPEN "COM0:" FOR OUTPUT AS #n ↓ PRINT #n, data ↓ CLOSE #n	⟶	Data reception
Data reception	OPEN "COM0:" FOR INPUT AS #n ↓ INPUT #n, variable name ↓ CLOSE #n	⟵	Data transmission (Follow each data record with a delimiter and withhold next transmission after transmitting each record.)

6

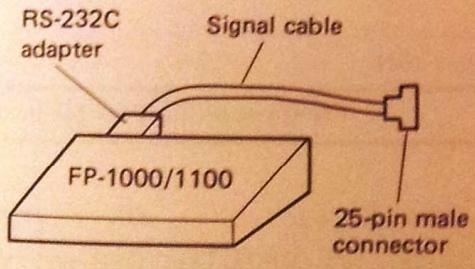
Communication between an FP-1000/1100 and an FP-200

Having completed a basic discussion of FP-200 serial data communications, try actual program and data transmission and reception by connecting your FP-200 to an FP-1000/1100 using its signal cables.

Fig. 21 below shows the hardware components required to connect an personal computer FP-1000/1100 to the FP-200. For the information on the FP-200, see fig. 19 on page 87.

After the FP-1000/1100 prepared, proceed with their connection to the FP-200 after examining how to control the signal lines. Because you are going to directly connect two DTEs without using a DCE or acoustic coupler,etc., a little measure must be used.

(21) Components for connecting the FP-1000/1100 to the FP-200 via RS-232C



Computer	FP-1000/1100
RS-232C adapter	FP-1035RS
Signal cable	RS-232C cable

Connecting the FP-200 to an FP-1000/1100

Table 22 summarizes the FP-1000/1100 communication protocol, and Fig. 23 shows the signals for interfacing the FP-1000/1100 with the FP-200 and their connector pin assignments.

(22) FP-1000/1100 communication protocol

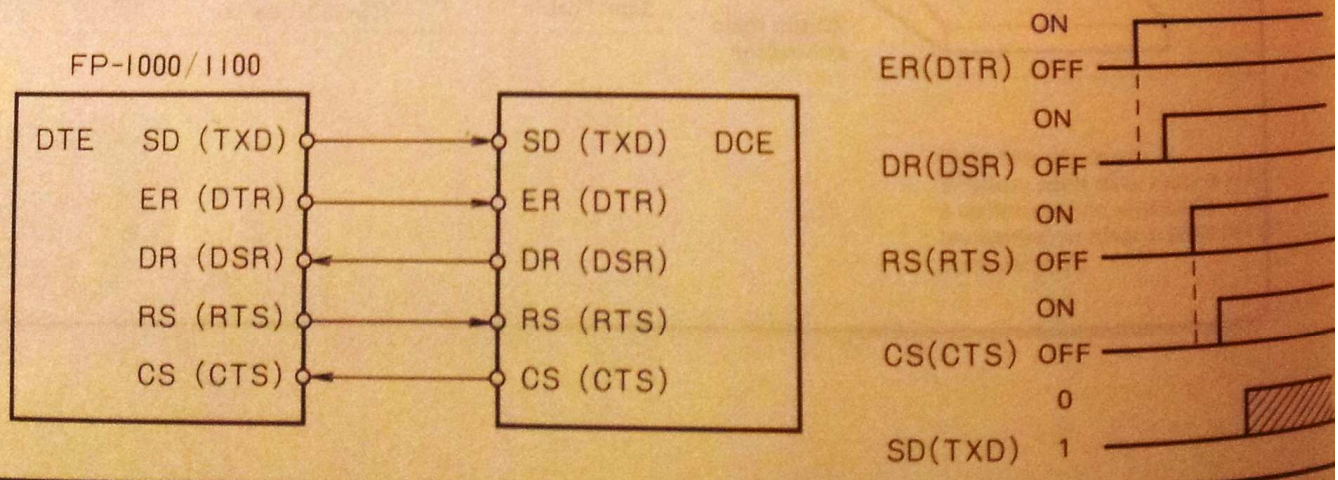
Communication mode		Half or full duplex
Synchronization		Asynchronous or start-stop
Transmission speed		37.5, 75, 150, 300, 600, 1200, 2400, 4800, 9600BPS
Transmission character configuration	Start bits	1 bit
	Data character length	7 or 8 bits
	Parity check	Even, odd or no parity
	Stop bits	1, 1.5, or 2 bits

(23) Interface signals

Assigned connector pins	Signal names	Signal direction	State		
		FP-1100 DCE	When not used	During reception	During transmission
1	(FG)				
2	SD(TXD)	○→	1	1	1/0
3	RD(RXD)	←○			
4	RS(RTS)	○→	OFF	OFF	ON
5	CS(CTS)	←○			
6	DR(DSR)	←○			
7	SG(GND)				
8	CD(DCD)	←○			
24	ER(DTR)	○→	ON	ON	ON

The states are only for the used signals.

(25) FP-1000/1100 data transmission



To use an RS-232C with the FP-1000/1100, an optional RS-232C pack (model FP-1035RS) must be inserted in the universal slot at the rear of the main unit or in the expansion box.

The data transmission speed is selected by a combination of the DIP switch assembly on the RS-232C pack and the TERM command or OPEN statement (see Fig. 24). Because a transmission speed of 300 bps is used in communications with the FP-200, it can be seen from the figure that you should specify F (Fast) for the TERM command or OPEN statement with DIP switch settings 1=OFF, 2=ON, and 3=ON, or specify S (Slow) with switch settings of 1=OFF, 2=OFF, and 3=ON.

■ FP-1000/1100 DTE and DTS

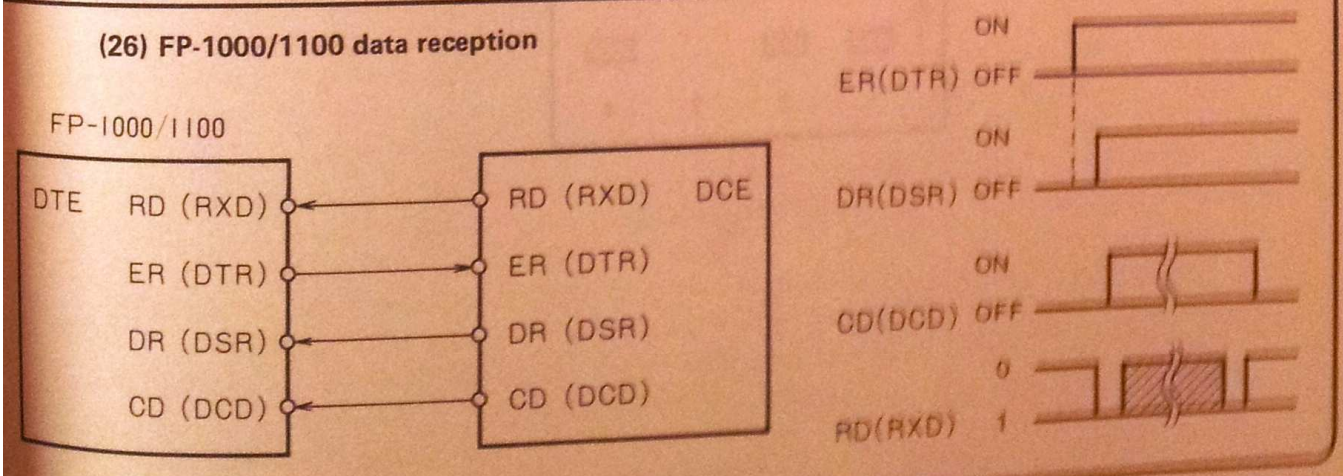
The FP-1000/1100 uses interface signals that the FP-200 does not use. However, this presents no problem in interfacing with the FP-200.

In table 23, only the states of the signals issued from the FP-1000/1100 are listed. In FP-1000/1100 data transmission or reception, the states of the signals are the same as those of the FP-200. The FP-1000/1100 becomes ready for reception when DR (DSR) and CD (DCD) from the DCE turn on after ER (DTR) has been turned on.

(24) FP-1000/1100 transmission speed selection on the RS-232C pack

Pack DIP switch settings			Transmission speed (bps)	
3	2	1	F (Fast)	S (Slow)
ON	ON	ON	150	37.5
ON	ON	OFF	300	75
ON	OFF	ON	600	150
ON	OFF	OFF	1200	300
OFF	ON	ON	2400	600
OFF	ON	OFF	4800	1200
OFF	OFF	ON	9600	2400
OFF	OFF	OFF	External clock	

(26) FP-1000/1100 data reception

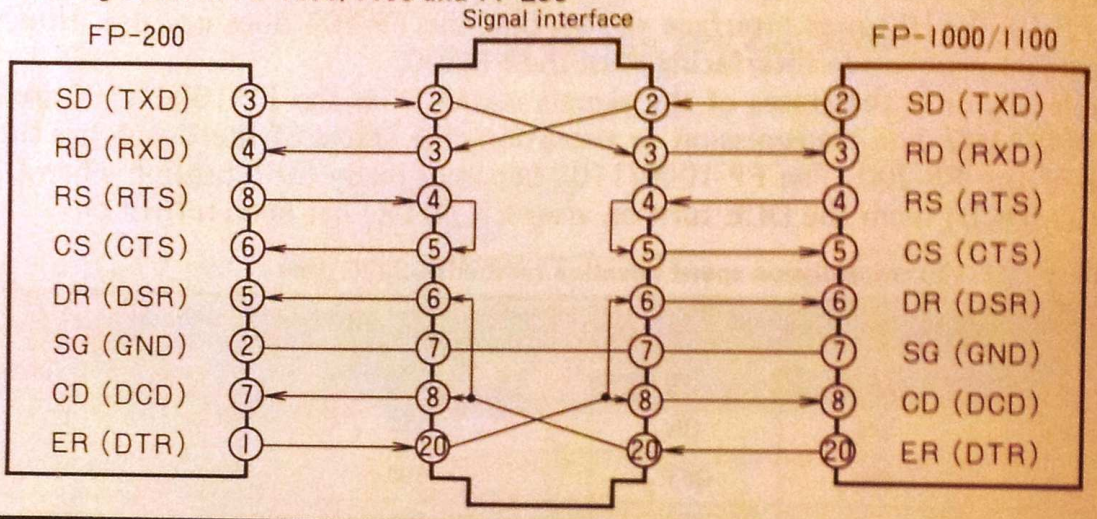


The FP-1000/1100 becomes ready for transmission when DR (DSR) and CS (CTS) from the DCE turn on after ER (DTR) and RS (RTS) have been turned on.
When you specify a 7-bit data character length, FP-1000/1100 alphanumeric and special characters can be transmitted or received using the SO (character code $0E_{16}$) and SI (character code $0F_{16}$) codes. The procedure is the same as with the FP-200.

■ **Interfacing between the FP-1000/1100 and the FP-200**

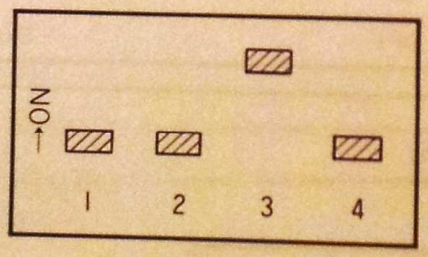
Fig. 27 shows the interface wiring for connections between the FP-1000/1100 and the FP-200.

(27) Interfacing between FP-1000/1100 and FP-200



All the subsequent example programs assume that only the RS-232C pack DIP switch 3 is turned on. Turn the switch on here.

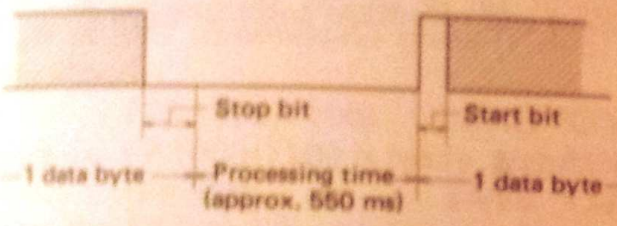
(39) RS-232C pack DIP switch settings



** FP-200 transmission/reception timing

Whenever transmitting programs or data to the FP-200, the computer must adjust the interval between transmissions in order to prevent data dropout or RW errors at the FP-200. We, using an oscilloscope, have determined the minimum safe interval to be around 550 ms. The interval is obtained by a FOR – NEXT loop in other personal computers. The FOR – NEXT loop execution time varies in a wide range depending on the personal computer and programming. The reader should determine the optimum time by repeated trials referring to the values used in the sample programs found in this book.

Transmission wait time ensuring prevention of data dropouts and RW errors during reception at the FP-200 (LOAD "COM0:" or INPUT #n execution timing)



FP-200 transmission (SAVE "COM0:", A or PRINT #n execution) timing

7

Program transfer between an FP-200 and an FP-1000/1100

Let's begin with program transmission and reception between the FP-200 and the FP-1000/1100. Exchanging programs between different computers may not seem to be of any use, but it actually can provide many advantages such as sharing a printer, unified program management, improvement in program development, keyboard data entry, program debugging, etc.
FP-200 uses the following program transmission and reception commands as described earlier:

Program transmission: SAVE "COM0:", A
Program reception: LOAD "COM0:"

When you use 'SAVE "COM0:",A', to transmit the following program over a communications line for example:

Sample program:

```
1000  OPEN "COM0:" FOR INPUT AS #1
1010  INPUT#1, A$
1020  PRINT A$;
1030  GOTO 1010
```

It appears on the line as follows:

$(0F)_{16}$ 1000␣OPEN␣"COM0:"␣FOR␣INPUT␣AS␣#1 $(0D)_{16}$ $(0A)_{16}$
1010␣INPUT#1, A$ $(0D)_{16}$ $(0A)_{16}$ 1020␣PRINT␣A$; $(0D)_{16}$
$(0A)_{16}$ 1030␣GOTO␣1010 $(0D)_{16}$ $(0A)_{16}$ $(0D)_{16}$ $(0A)_{16}$

That is, the program begins with an SI code ($0F_{16}$) and a sequence of $0D_{16}$ and $0A_{16}$ (a CR-LF code) is inserted at the end of each program line, and the entire program is followed by another CR-LF code; two successive CR-LFs represent the end of a program.

Transferring programs between an FP-200 and an FP-1000/1100

The FP-1000/1100 has a program transmission command SAVE "COMn: –" and a program reception command LOAD "COMn: –".
Programs can be successfully transmitted from the FP-200 to the FP-1000/1100 by using the proper commands. When transmitting a program from the FP-1000/1100 to the FP-200, however, the FP-200 LOAD command requires too much time, and RW errors or data dropouts may occur.
Thus, there is no way but transmitting a program from a cassette tape or floppy disk, on which the program has been recorded in the ASCII format, line by line as measuring the communication timing.

• FP-200 to FP-1000/1100 program transmission

First try to transmit a program from the FP-200 to the FP-1000/1100. Assume that only the RS-232C pack DIP switch is turned on (the rest are off) on the FP-1000/1100. The FP-1000/1100 becomes ready for program reception when you press the RETURN key after keying in the following command, giving the parameters in the order of data transmission speed (F or S), data character length, parity, and number of stop bits:

(FP-1000/1100) LOAD "COM0: (S7E)" RETURN

After the FP-1000/1100 is ready, transmit the program from the FP-200:

(FP-200) SAVE "COM0: ", A RETURN

After the FP-200 transmits the program, both the FP-200 and the FP-1000/1100 wait for a command entry.

• FP-1000/1100 to FP-200 program transmission

If a program is transmitted using the FP-1000/1100 SAVE command, the FP-200 cannot successfully receive it. Thus, you first have to record the program on a cassette tape or floppy disk and then transmit it line by line. The author has written a program (filename 232CPR.BAS) to accomplish this transmission.
This program assumes that the program to be transmitted is recorded in the ASCII format. In addition, it provides a transmission wait interval to allow the FP-200 to complete post-reception processing after it transmits each program line.
The entire program transmission procedure using the program is as follows:

(1) Store in the FP-1000/1100 memory the program to be transmitted to the FP-200.
(2) Record the program on a cassette tape or floppy disk in ASCII format:
SAVE "file descriptor: filename", A
(3) Load the transmission program in an unused program area:
LOAD "file descriptor: RS232CPR.BAS"
(4) Key in MOUNT 2 RETURN.
(5) Execute the program:
RUN
(6) Key in the file descriptor for the program to be transmitted:
Device No. – (file descriptor keyed in at step (2))
Key in the filename:
File name – (filename keyed in at (2))
(7) Execute the FP-200 program reception command:
LOAD "COM0:"

(8) Initiate program transmission from the FP-1000/1100:
Ready ? _ (Press RETURN.)
(9) Display the line number when each line is transmitted.
(10) When the entire program has been transmitted, indicate whether there is another program to be sent or not:
Transfer end (Press RETURN or key in Y if there is another program;
Next file (Y/N) key in N otherwise.)
(11) When Y (or RETURN) is keyed in at step (10), the program returns to step (6). Otherwise, the program terminates.

(LIST 1)

FP-1000/1100 program transmission program

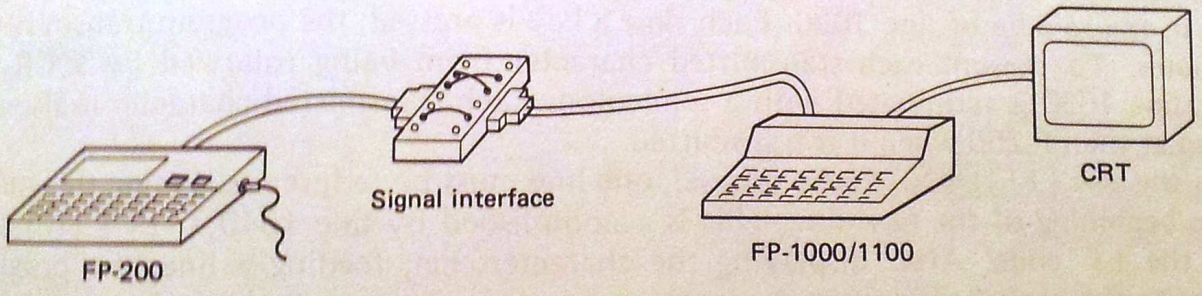
```
100 '*********************************************
110 '  RS-232C Port ASCII file Send
120 '          FP-1000/1100 to FP-200
130 '
140 '       File name   : 232CPR.BAS
150 '       Computer    : FP-1000/1100
160 '       Data signalling rate : 300bps
170 '       Word length          : 7bit
180 '       Parity               : Even
190 '       Stop bit             : 2
200 '*********************************************
1000 '
1010 ON ERROR GOTO 1290
1020 WIDTH 80
1030 PRINT "**  ascii file transfer  **"
1040 INPUT "Device    ",D$
1050 INPUT "File name ",F$
1060 '
1070 OPEN "COM0:(S7E)" FOR OUTPUT AS #2
1080 LINE INPUT "Ready ? ";Z$
1090 OPEN D$+":"+F$ FOR INPUT AS #1
1100 CU=0
1110 '
1120 LINE INPUT #1,P$
1130 IF EOF(1) THEN 1200
1140 CU=CU+1
1150 LOCATE 0,9:PRINT CU;
1160 PRINT #2,P$
1170 FOR I=1 TO 100:NEXT I
1180 GOTO 1120
1190 '
1200 PRINT #2,CHR$(0):CLOSE
1210 PRINT:PRINT "Transfer end":BEEP
1220 PRINT "Next file (Y/N) ";
1230 Z$=INKEY$
1240 IF Z$=CHR$(13) THEN RUN
1250 IF Z$="Y" OR Z$="y" THEN RUN
1260 IF Z$="N" OR Z$="n" THEN CLOSE:PRINT Z$:END
1270 GOTO 1230
1280 '
1290 CLOSE:RESUME 1000
```

8

FP-200 key data communication

This chapter discusses data communications using the keyboard to receive and display at one computer data which has been keyed in at another computer.
Key data communications are accomplished using the same computer hardware configuration and interface signal connections as with program transmission and reception. Refer to the FP-1000/1100 hardware configuration on page 93 and the signal cable connection on page 96. Only a display device (CRT display or built-in LCD), keyboard, connection cable, and signal interface are required to try data communication.

(29) Hardware configuration for key data communication



The FP-1000/1100 can be used in terminal mode, and can provide minimal data reception capabilities so that you need not bother to write programs at a very elementary trial stage. "Minimal" means that the terminal mode provides no line feed capability, and cannot solve FP-200 problems such as data dropout and RW errors which may also occur in program transmission and reception as discussed earlier.
At a more advanced stage, three basic programs are required for each of the FP-1000/1100 and the FP-200. The first program should input data from the keyboard and transmit the data to the FP-1000/1100 (this is the "Key Data Transmission Program"). The second program receives and displays data sent from the FP-1000/1100 (this is the "Key Data Reception Program").

The third program is a key data transmission and reception program which provides real data communications capabilities (this is the "Key Data Transmission/Reception Program").
Discussions begin with the FP-200 programs. The equivalent programs for the FP-1000/1100 and their RS-232C-related statements are explained in subsequent sections.

FP-200 key data transmission program

This program inputs data from the keyboard and transmits the data to the FP-1000/1100.
The conditions for data transmission are as follows:
(1) Each time a character is entered from the keyboard, the character is transmitted.
(2) Transmitted data records are not followed by CR-LF codes
Thus, the receiving computer has to append a CR-LF code to each received physical record.
A sample program for the FP-200 is shown in LIST 2). It is a very short program, consisting of only six lines, but it can accomplish the functions that satisfy the above conditions. Key in this program in a program area so that data transmission can be tried immediately after you put the appropriate data reception program in the FP-1000/1100.
The program can be summarized as follows: The OPEN statement in line 1010 first declares the use of the RS-232C. Data to be transmitted is input using the INKEY $ statement in line 1020. The keyboard keys are actually scanned so fast that the pressed key might be input twice or more during a single keystroke unless a means were provided to avoid this. In this example, this scan speed is effectively slowed down by the loop in the middle of line 1020. Each time a key is pressed, the program transmits the character. To prevent each transmitted character from being followed by a CR-LF code, line 1030 is terminated with a semicolon. Each transmitted character is also displayed at the FP-200 when it is transmitted.
When the ↵ (RETURN) key is pressed, one line must be fed, repositioning the cursor at the beginning of the new line. This is accomplished by line 1040; CHR $ (10) contains the LF code. After displaying the character; i.e., feeding a line, the program returns to the keyboard input statement.

(LIST 2)

FP-200 key data transmission program

```
100 '************************************
110 '  RS-232C Port Key in data send
120 '          FP-200 TO FP-1000/1100
130 '          (C) Copyright 1983
140 '                    A.Kakizono
150 '       File name : 232C1T.BAS
160 '       Computer  : FP-200
170 '       Data signalling rate : 300bps
180 '       Word length          : 7bit
```

```
190 '        Parity                    : Even
200 '        Stop bit                  : 2
210 '*************************************
1000 '
1010 OPEN "COM0:" FOR OUTPUT AS #1
1020 D$=INKEY$:FOR I=1 TO 50:NEXT I:IF D$="" THEN 1020
1030 PRINT #1,D$;
1040 IF D$=CHR$(13) THEN D$=D$+CHR$(10)
1050 PRINT D$;
1060 GOTO 1020
```

2 FP-200 key data reception program

This section discusses a program for receiving data sent from the FP-1000/1100. A sample program is shown in LIST 3) which is also very short. Key in this program in another program area of the FP-200 so that you can try data reception from the FP-1000/1100.

The required reception conditions can be summarized as follows:

(1) Each time a character is received, the character is displayed.

(2) The CR-LF code processing is done at the receiving computer.

The FP-200 has no statement that inputs a single character from the RS-232C receive data buffer (e.g., INPUT$). The INPUT# statement would continue to input the data until a CR-LF code was encountered and the above condition (1) could not be satisfied. This problem is solved by following each transmitted character with a CR-LF code at the transmitting computer. This allows the FP-200 to display character by character.

Condition (2) implies that the FP-200 has to make the preparation for a line feed and the subsequent display from the beginning of the new line when a delimiter such as a CR code is received.

The program is LIST 3) looks similar to the LIST 2) program. It first puts the RS-232C in receive status and processes the appended delimiter when each data character is input, and displays the character. This sequence of events continues until the received character count reaches 0. At this time, the program outputs a CR-LF code to actually feed a line.

(LIST 3)

FP-200 key data reception program

```
100 '*************************************
110 '  RS-232C Port data receive
120 '          FP-200 from FP-1000/1100
130 '          (C) Copyriht 1983
140 '                    A.Kakizono
150 '       File name : 232C1R.BAS
160 '       Computer  : FP-200
170 '       Data sinalling rate : 300bps
180 '       Word lenth          : 7bit
190 '       Parity              : Even
200 '       Stop bit            : 2
210 '*************************************
```

```
1000 '
1010 OPEN "COM0:" FOR INPUT AS #1
1020 INPUT #1,D$
1030 IF LEN(D$)=0 THEN D$=CHR$(13)+CHR$(10)
1040 PRINT D$;
1050 GOTO 1020
```

As you can see, both the programs in LIST 2) and 3) have no end. One of the most popular methods for terminating such an application program is to follow the transmitted data with a character such as "$" which is agreed upon between the two computers. The transmission program terminates when it transmits this character, and the reception program terminates when it receives the character. Try to modify the LIST 2) and 3) programs so that the above program termination can be used.

This modification is rather simple. You are probably astonished to find that you can control the two computers with this simple technique. This example uses only a single agreed-upon character to terminate the reception program at the mated computer. This concept can be easily expanded, for example, to use the first appearance of the question mark "?" as the indicator for redirecting subsequent data characters to the printer, and the second appearance as the indicator for redirecting the subsequent data to the display. The character "T", for example, could be used to direct the reception program to display the time (TIME$) and date (DATE$).

In this way, you can exert a wide variety of control over the remote computer. This concept can be applied in the field of home automation in such ways as turning on automatic cooking appliance timers and air conditioners, etc., at your home by using the phone while you are out.

FP-200 key data transmission/reception program

A program that alternates transmission and reception of data input from the keyboard is discussed here. You can, at last, accomplish communication between two computers. The sample program used for the following discussions is shown in LIST 4).

The program is made up of three modules. The first module signals the initiation of a data transmission, the second module receives incoming data, and the third module transmits data. The second and third modules are, respectively, slightly modified variations of the previous programs LIST 2) and 3). Enter this program in another program area for later testing.

(LIST 4)

FP-200 key data transmission/reception program

```
100 '************************************
110 '  RS-232C Port Key to Key
120 '                    communication
130 '          FP-200 <=> FP-1000/1100
140 '          (C) copyright 1983
150 '                    A.Kakizono
160 '       File name : 232CKY.BAS
```

```
170 '        Computer   : FP-200
180 '        Data signalling rate : 300bps
190 '        Word length          : 7bit
200 '        Parity               : Even
210 '        Stop bit             : 2
220 '*********************************
1000 ' ====================  initialize
1010 OPEN "COM0:" FOR OUTPUT AS #1
1020 D$="Ready ? "+CHR$(13)
1030 PRINT #1,D$;
1040 PRINT "<Ready ?> send,wait!"
1050 CLOSE
1060 ' ====================  data receive
1070 OPEN "COM0:" FOR INPUT AS #2
1080 PRINT "receive > ";
1090 INPUT #2,D$
1100 PRINT D$;
1110 IF LEN(D$)<>0 THEN 1090
1120 PRINT:CLOSE
1130 ' ====================  data send
1140 OPEN "COM0:" FOR OUTPUT AS #1
1150 PRINT "send    > ";
1160 D$=INKEY$:FOR I=1 TO 60:NEXT I:IF D$="" THEN 1160
1170 PRINT #1,D$;
1180 PRINT D$;
1190 IF D$=CHR$(13) THEN PRINT CHR$(10);:CLOSE:GOTO 1060
1200 GOTO 1160
```

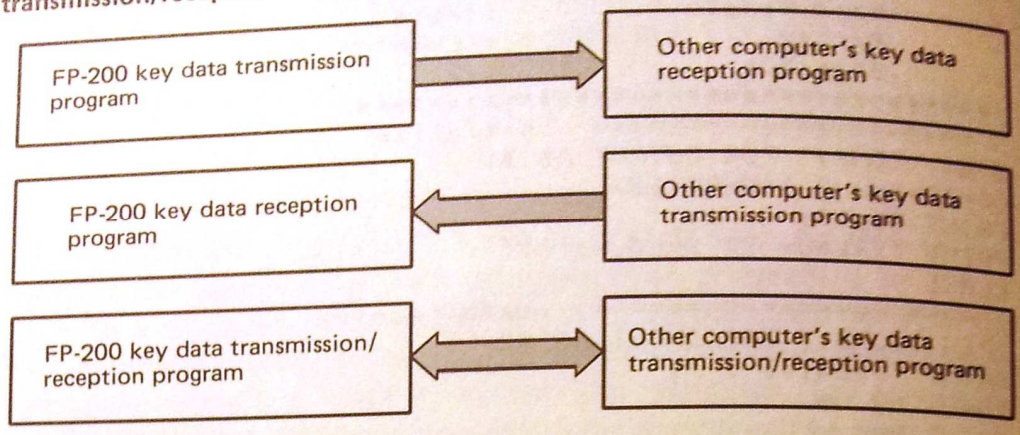
The first module of the above program consisting of lines 1000 through 1050 transmits the message "Ready ?" to check to find whether the FP-1000/1100 is ready for communication. When this is acknowledged (i.e., when an acknowlege code is sent back), communication is initiated. The RS-232C file is closed after the first message is transmitted because it cannot be opened for both input and output at the same time in the FP-200.

The second module, from line 1060 to 1120, receives incoming data. While in receive mode (i.e., while control is in this module), the message "send >" is displayed. When the delimiter or a semicolon indicating the end of a data block (a line or physical record) is received, the module closes the receive file.

When reception has been completed, the last module (i.e., transmission module) issues a line feed and displays the message "send >" at the beginning of the new line. Data is transmitted, character by character, each time a key is pressed. Data characters are input from the keyboard and transmitted in the same way as with the transmission program (LIST 2), except that the cursor is repositioned at the beginning of the new line and the send file is closed when the ↵ key is pressed.

You can try the three programs in the following combinations:

Key data transmission/reception test methods



FP-1000/1100 key data transmission program

This program sends characters keyed in at the FP-1000/1100 to the FP-200 character by character each time a key is pressed.
First enter the key data transmission program, shown in list 5, into the FP-1000/1100, then run the key data reception program previously loaded in the FP-200. Finally, execute the FP-1000/1100 key data transmission program just loaded above.
Now both the computers are ready for one-way communication from the FP-1000/1100 to the FP-200. Press a key on the FP-1000/1100 keyboard. The keyed in character should be displayed on the FP-200 LCD. Press several keys and check to see that all the keyed-in characters are displayed correctly. Then, send various messages.
If the first test fails, check the RS-232C wiring. Is only switch 3 turned on in the FP-1000/1100 RS-232C interface cartridge DIP switch assembly? Are both of the programs correct? Check them one by one.
In this one-way communication from the FP-1000/1100 to the FP-200, you can send alphanumeric characters. Character graphics can also be transmitted, with some exceptions. You can feed one line by pressing RETURN key.

(LIST 5)

FP-1000/1100 key data transmission program

```
100 '*********************************************
110 '  RS-232C Port Key in data Send
120 '          FP-1000/1100 to any computer
130 '
140 '       File name   : 232C1T.BAS
150 '       Computer    : FP-1000/1100
160 '       Data signalling rate : 300bps
170 '       Word length          : 7bit
180 '       Parity               : Even
190 '       Stop bit             : 2
200 '*********************************************
1000 '
1010 WIDTH 80
1020 OPEN "COM0:(S7E)" FOR OUTPUT AS #1
```

```
1030 D$=INKEY$:IF D$="" THEN 1030
1040 PRINT #1,D$
1050 IF D$=CHR$(13) THEN D$=D$+CHR$(10)
1060 PRINT D$;
1070 FOR I=1 TO 100:NEXT I
1080 GOTO 1030
```

5 FP-1000/1100 key data reception program

Enter the key data reception program, shown in list 6, into the FP-1000/1100. Run the FP-200 key data transmission program and the FP-1000/1100 program just loaded above. Now both the computers are ready for one-way communication from the FP-200 to the FP-1000/1100.

Press an FP-200 key. The character should be displayed on the FP-1000/1100 CRT screen.

This program first puts the RS-232C in the receive mode and waits for arriving data. When the data is received, it is displayed character by character. When a CR code (CHR$(13), entered by pressing the RETURN key) is encountered in the received data, a line feed takes place.

If any of these tests fails, find the error by checking the interface wiring and the programs in the same way as with the transmission program.

Both one-way communications have now been achieved. Let's try two-way communications next.

(LIST 6)

FP-1000/1100 key data reception program

```
100 '************************************************
110 '  RS-232C Port Key in data Receive
120 '           FP-1000/1100 from any computer
130 '
140 '       File name   : 232C1R.BAS
150 '       Computer    : FP-1000/1100
160 '       Data signalling rate : 300bps
170 '       Word length          : 7bit
180 '       Parity               : Even
190 '       Stop bit             : 2
200 '************************************************
1000 '
1010 WIDTH 80
1020 OPEN "COM0:(S7E)" FOR INPUT AS #1
1030 D$=INPUT$(1,#1)
1040 IF D$="" THEN 1030
1050 IF D$=CHR$(13) THEN D$=D$+CHR$(10)
1060 PRINT D$;
1070 GOTO 1030
```

FP-1000/1100 key data transmission/reception program

This program accomplishes two-way communications between the FP-1000/1100 and the FP-200, allowing message exchanges between two different computers.
Enter the program from list 7 into the FP-1000/1100 and also load its counterpart into the FP-200.
First run the FP-1000/1100 program and then the FP-200 program. Initiating the FP-200 program should cause the message "Ready ?" to be transmitted, signaling the start of the communication. The "Ready ?" message appearing on the FP-1000/1100 CRT screen indicates that the message transmitted from the FP-200 was successfully received at FP-1000/1100. Then, a request-to-send message "send >" is displayed at the FP-1000/1100. Now send a message from the FP-1000/1100 to the FP-200. The characters are also displayed at the FP-200. The message transmission is terminated by keying in a character with a character code of $1F_{16}$ or below, or a comma. Usually, the RETURN key should be used.
When the transmission is terminated, the message "receive >" is displayed at the FP-1000/1100, indicating that the FP-1000/1100 is now in the receive state. Press the FP-200 keys this time. The characters should appear on the FP-1000/1100 screen. The RETURN key at the FP-200 signals the end of message transmission to the FP-1000/1100. "send >" is displayed again at the FP-1000/1100, indicating that the receive and send states are reversed at the two computers. Thus, the messages "send " and "receive " indicate whether the computer is in the send or receive state.
Try this two-way communication using various messages.

(LIST 7)

FP-1000/1100 key data transmission/reception program

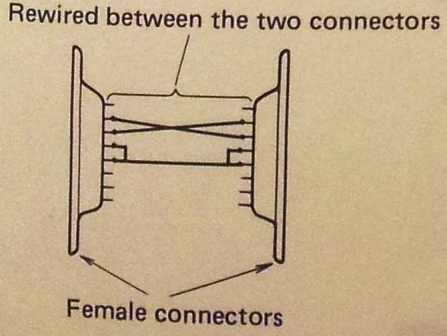
```
 100 '*********************************************
 110 '  RS-232C Port Key to Key
 120 '           FP-1000/1100 <=> FP-200
 130 '
 140 '       File name  : 232CKY.BAS
 150 '       Computer   : FP-1000/1100
 160 '       Data signalling rate : 300bps
 170 '       Word length          : 7bit
 180 '       Parity               : Even
 190 '       Stop bit             : 2
 200 '*********************************************
1000 '
1010 WIDTH 80
1020 OPEN "COM0:(S7E)" AS #1
1030 '
1040 D$=INPUT$(1,#1)
1060 PRINT D$;
1070 IF D$=CHR$(13) THEN PRINT CHR$(10);"send    > "; ELSE 1040
1080
1090 D$=INKEY$:IF D$="" THEN 1090
1100 PRINT #1,D$
1110 IF D$<" " OR D$="," THEN PRINT:PRINT "receive > ";:GOTO 1040
1120 PRINT D$;
1130 FOR I=1 TO 100:NEXT I
1140 GOTO 1090
```

9 RS-232C signal interface

There are often cases where, as discussed earlier, the signal cables of the two computers must be directly connected in applications using the RS-232C. The direction of each RS-232C signal is determined by how the signal functions on the DCE. Therefore, the cross, return, or open, of a signal line performed by DCE must be somehow implemented in order to connect two DTEs (personal computers).
It is not desirable to rewire within the 25-pin connector at one end of the signal cable because it deprives the cable assembly of compatibility and makes future maintenance difficult. Such disadvantages outbalance such advantages of this method as its simplicity and no additional parts requirement. This chapter explains three types of signal interfaces which you can make easily and which are useful for signal rewiring. Once you build one of them, you will never have to resolder the connector for each different computer and can also avoid wiring errors such as disconnection, etc. This will be soon a must for any user who uses the RS-232C.
Such components are not on the market yet.

1 Signal interface 1 : Special connector assembly

This is a fixed combination of two female connectors which can serve two specific computers only. Signal lines are rewired between the two connectors. Though lowest in cost and simplest to build, it is not universally applicable.



Parts

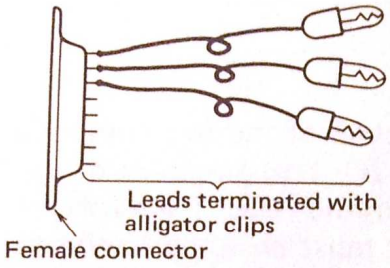
25-pin female connector (DB-25S)	2 PCS
Wire (for wiring between connectors)	1 m

Tools
Soldering iron, solder, nippers, knife, radio pliers, etc.

Problems such as disconnection can be minimized by putting the connectors in a protective case. If the case is used, the computer models should be labeled on it to prevent misuse.

2 Signal interface 2 : Connector with alligator clips

Prepare two female connectors with leads which have an alligator clip at the other end soldered to their pins, each housed within a protective case. This method considerably increases universality because signal lines can be rewired as desired by simply changing clip connections. Leads may be easily identified by assigning different colors to the leads or putting numbered labels on the clips.



Parts

25-pin female connector (DB-25S)	2 PCS
Connector case	2 PCS
Lead wire	15 cm/each color
Alligator clip	As many as required

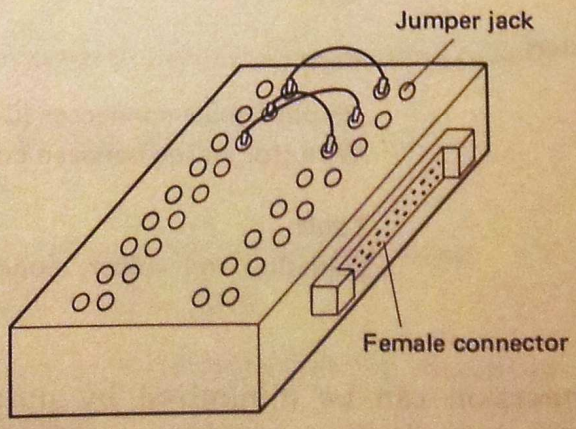
Tools

Soldering iron, solder, nippers, knife, radio pliers, and screwdriver

No more than 14 lead wires are required to connect any two personal computers. For asynchronous communications, the eight lead wires required for the FP-200 are enough for any computers.

3 Signal interface 3 : Jumpered universal signal interface

This method mounts two female connectors on two sides of an aluminum chassis which has jumper jacks on the top. Signal lines can be rewired using jumper lead wires which have plugs at both ends. Inside the chassis, the connector pins are wired to the jacks. The chassis serves as a strong protective case and this interface is highly reliable as well as most universal. However, it requires the most work and time to build because the connector mounting holes and many jumper holes need to be made.
The following is a sample structure:



Parts

25-pin female connecotr (DB-25S)	2 PCS
Pad	4 PCS
Jumper plug	As many as required
Jumper jack	As many as required
Aluminum chassis	1 PC.
Wire (for wiring inside chassis)	Approx. 2 m
Wire (for jumper leads)	As many as required, 15 cm each,

14 jumper lead wires (i.e., 14 pairs of jumper jacks) are enough for communications between any personal computers. If this interface is used for asynchronous communications only, the eight jumper wires used for the FP-200 are enough for any computers. FG (Frame Ground, pin 1) and SG (Signal Ground, pin 7) may be directly wired between the connectors.

The above example has two jumper jacks for each signal line on each side. It also has SG line jumper jacks in order to provide ground points for an oscilloscope, etc.

CASIO®