# MPF-I

## BASIC

## Operation Manual

# BASIC-MPF

# OPERATION MANUAL


**Multitech**
**INDUSTRIAL CORP.**

OFFICE/ 9FL. 266 SUNG CHIANG ROAD, TAIPEI 104 TAIWAN R.O.C.
TEL:(02)551-1101 TELEX:"19162 MULTIIC" FAX:(02) 542-2805
FACTORY/ 1 INDUSTRIAL E. RD., III HSINCHU SCIENCE - BASED
INDUSTRIAL PARK, HSINCHU, TAIWAN, R.O.C.

# TABLE OF CONTENTS

# I.Introduction

BASIC is the most popular high level computer language used by many of today's computers. BASIC stands for Beginners All-purpose Symbolic Instruction Code. However, it should not be assumed that the BASIC language is only used for beginners, and has no application in the "outside world". On the contrary, BASIC is of major importance in the world of business, and a large number of business systems programs are written in this language. The major advantage is that a programmer can produce a highly complex program in a fraction of the time taken to develop the same program in many other languages.

The Micro-Professor with powerful expansion capabilities has been developed by Multitech Industrial Corp. to execute the instruction of Z-80 and 8085 microprocessors. To help the Micro-Professor user to learn BASIC programming, Multitech presents the Micro-Professor BASIC Interpreter.

***Specifications:***

(1) The language used on the MPF-I is composed of several different types of words. The word types are:
* Statements
* Commands
* Variables

(2) A 2K Byte BASIC Interpreter.

(3) A 34 Key BASIC Name Plate overlays the original MPF-I keyboard.

(4) Summary of BASIC commands and statements
(a) Commands
RUN - Execute the current program.
LIST - Print out the current program to display.
LOAD - Load a program from the cassette tape.
SAVE - Save a program on the cassette tape.
CONTINUE - Continue after STOP or PRINT.
NEW - Clear memory to allow a new program to be written.

(b) Statements
PRINT - Output information to display.
INPUT - Allow user input.
LET - Assign a numerical value to a variable.
CALL - Call a machine-code routine.
GOTO - Jump to a given line.
GOSUB - Jump to subroutine.
RETURN - Return to main program from a subroutine.
IF...THEN - Allow statement to be executed conditionally.
FOR...TO - Set the parameters for loop execution.
NEXT - End a FOR...TO statement.
STOP - Force return to the command mode.

(5) Variables

(a) Numeric Variables are the only type used in MPF-1 BASIC.
(b) Variables can only hold " integer " or whole number values such as 235 or -3451.
(c) Limit:
the limit for MPF-1 BASIC is from -32767 to 32767. (But error massage will be occurred if one or both of the numbers to be multiplied are negative.)
(d) Variables Name:
The length is one or two characters. If the length is one character, it must be any of A-F. If the length is 2 characters , the first must be any of alphanumeric letter A-F, the second must be any of numeric digit Ø-9.

(6) Memory Variable
M: The format is M decimal
where decimal is a hexadecimal memory address.
[line no.] LET M decimal = value
To write a byte into the specified memory location.
[line no.] LET Variable = M decimal
Assign Variable with the contents of the specified memory location.

(7) I/O Port Variable
P: The format is P decimal
where decimal is a hexadecimal I/O port address.
[line no.] LET P decimal = value
Output a byte to the specified I/O port.
[line no.] LET Variable = P decimal
Assign variable with the contents of the specified I/O port.

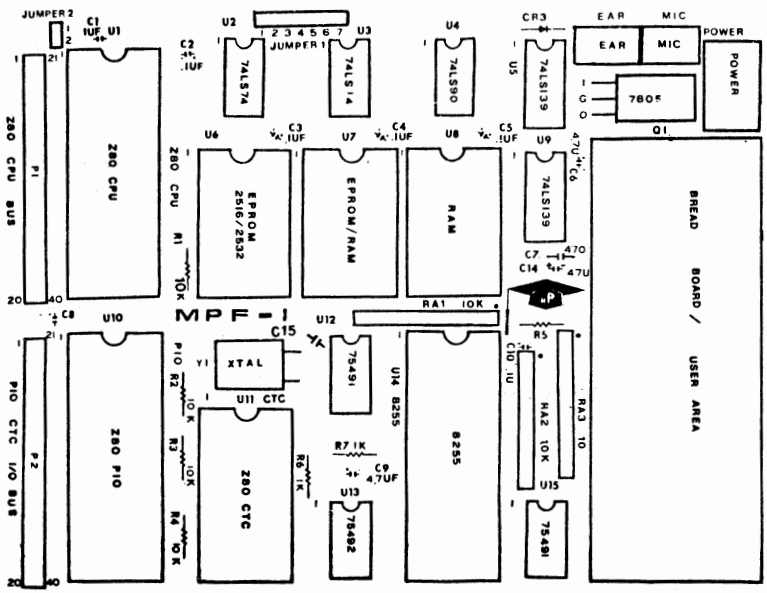
# Ⅱ.MPF-I BASIC Interpreter

This chapter must be read very carefully, including how to install BASIC Interpreter, to make use of keyboard and display, to edit program, and to use a cassette for storing or retrieving programs.

## *2.1 How to Install the BASIC Interpreter*

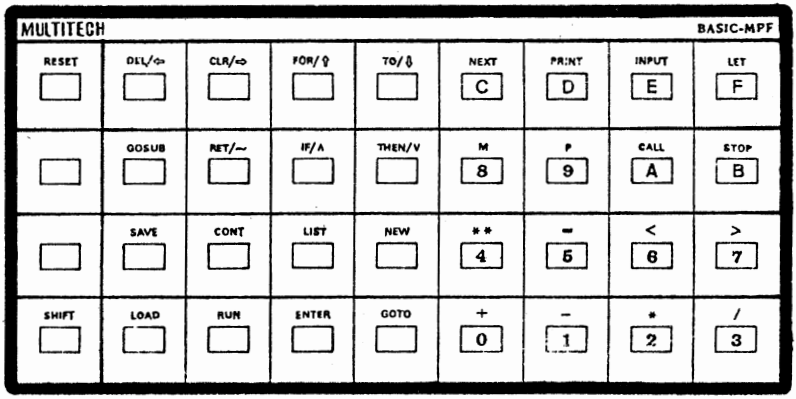
The MPF-I BASIC Interpreter with a total length 2K bytes, is stored in an EPROM either a TMS2516 or an Intel 2716. Another model is the MPF-I monitor with built-in BASIC Interpreter (MPF-IB) on a 4K byte IC storing in TMS2532. There is a spare IC location for memory expansion, marked by U7 in MPF-I. The address range of U7 is from 2000H to 27FFH (refer to the diagram below). Before inserting this EPROM in the socket at U7, insure that the index mark of EPROM is towards the top of the board and the MPF-I is turned off.

NOTE: Care should be taken when inserting EPROM as the pins are fragile and will bend easily if not properly aligned.



Overlay the MPF-I keyboard with the BASIC name plate. Next key in [GOTO] [2] [0] [0] [0] [RUN], but key in [GOTO] [0] [8] [0] [0] [RUN] if users operate a 4K byte monitor. The LED display will show [bASIC .], this stands for BASIC Ready. The rightmost digit is a decimal point, which is the CURSOR. Now the MPF-I is under the control of BASIC Interpreter.

### *2.2 Keyboard Use and Display*




The diagram above is the special Name Plate for MPF-I BASIC. There are 34 keys, but some keys have two different meanings. We can divide these 34 keys into the following groups:

(1) RESET Key: RESET MPF-I and return to the control of MPF-I monitor.
(2) SHIFT Key: To select cursor addressing key and operator key.
(3) Alphabetic Keys: A-F
(4) Numeric Keys: 0-9
(5) Operator Keys: +, -, *, /, **, =, <, >, ~(NOT), ∧(AND), V(OR). These keys are used with SHIFT key.
(6) Cursor Addressing Key: →, ←, ↑, ↓, are used with SHIFT key.
(7) Command Keys: SAVE, CONT, LIST, LOAD, RUN, ENTER, NEW.

(8) Statement Keys: IF, THEN, GOSUB, RET, FOR, TO, NEXT, GOTO, LET, PRINT, INPUT, CALL, STOP.

(9) Delete Keys: DEL, used to delete the letter or keyboard at the Cursor. CLR, used to delete the full line just entered.

(10) Special Variable Keys: M and P used with SHIFT key

The keyword concept is utilized in the BASIC program. Some keys stand for complete words, these keys are convenient to use and save typing time. There is a total of 19 keywords. When the MPF-I is under the control of the BASIC Interpreter, the LED display will show [bASIC.] . After the [GOTO] key is pressed, the LED display will show [IC Got.] . When [DEL] is pressed, the LED display will show [bASIC.] .

The 19 keywords will constitute the majority of the "sentences" written in your computer programs, and each sentence will begin on a new line, with a unique line number to identify it. Try to type

10 PRINT A

The first two characters "10" are the line number "ten". The order of keys are [1] [0] [PRINT] [A] [ENTER]. The corresponding display will show as follows:

| | LED Display |
|---|---|
| Before any key is pressed | [bASIC .] |
| Key in [1] | [ASIC 1.] |
| Key in [0] | [SIC 10.] |
| Key in [PRINT] | [ 10prt.] |
| Key in [A] | [10prtA.] |
| Key in [ENTER] | [bASIC .] |

During the statement entry, the syntax will be checked. Try typing

5 LET A=1

The word syntax is used in programming and relates to the order of your keyboard entries. In English syntax means the pattern of sentences and phrases that are constructed from words. In BASIC the syntax is checked to see if you constructed each BASIC statement correctly. The MPF-I BASIC checks for syntax in two ways. First preliminary checking is done as each part of a BASIC statement is entered. Secondly after the [ENTER] key is pressed, the entire line is checked for correct syntax.

If the [2] key is pressed instead of the [A] key, the display will show [E2 .] after [ENTER] key is entered (2 is the Error Code). By means of pressing [→] , [DEL] , and [←] keys, the statement can be corrected as shown below.

| | Display |
|---|---|
| | [E2 .] |
| Key in [→] | [ 0.] |
| Key in [→] | [ 00.] |
| Key in [→] | [ 005.] |
| Key in [→] | [005LEt.] |
| Key in [→] | [05LEt2.] |
| Key in [DEL] | [05LEt=.] |
| Key in [←] | [005LEt.] |
| Key in [A] | [05LEtA.] |
| Key in [ENTER] | [bASIC .] |

There is now a program stored in the program memory

```
5 LET A=1
10 PRINT A
```

Suppose we want to execute this program, we should press [RUN] and [ENTER]. But, we just need to press [RUN] [5] [0] [ENTER] when we intend to run from the 50 line number of the statement not to go back to the first line. The corresponding display is shown below.

| | LED Display |
|---|---|
| Before key is pressed | [bASIC .] |
| Key in [RUN] | [IC run.] |
| Key in [ENTER] | [ 00001] |
| Key in [CONT] | [bASIC .] |

Following is the key processing and display format.

DISPLAY FORMAT ON MPF-I

| DISPLAY FORMAT ON BASIC – MPF | | | | | | | | | | | | MULTITECH |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| KEY | TO | THEN | LET | PRINT | INPUT | NEXT | GOSUB | RET | GOTO | FOR | IF | CALL |
| DISPLAY | to | then | LEt | Prt | InP | nEt | GoS | rEt | Got | For | IF | CAL |
| KEY | STOP | RUN | LIST | SAVE | LOAD | | | | | | | |
| DISPLAY | StoP | run | LSt | SA | Ld | | | | | | | |
| KEY | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B |
| DISPLAY | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | b |
| KEY | C | D | E | F | ~ | ∧ | V | + | − | * | / | ** |
| DISPLAY | C | d | E | F | ¬ | ∩ | U | + | - | * | / | ^ |
| KEY | = | < | > | M | P | | | | | | | |
| DISPLAY | = | ≤ | ≥ | M | P | | | | | | | |

The following conceptions should be thoughtfully noted when users operate the BASIC Interpreter.

1. After users overlay the MPF-I keyboard with the BASIC name plate and key in [GOTO] [2] [0] [0] [0] [RUN] (0800 for 4K monitor), the BASIC Interpreter will clear the RAM buffer(1800H-1FADH i.e. set the contents of RAM buffer to be zero. Secondly, set the value of memory address 18E7 to be FF as the delimeter. Thus, if users set the starting address to be 2017 (0817 for 4K monitor) and press [RUN] key, the contents of user's program in MPF-I would not be destroyed.

2. The storage areas of user's program begins at the memory address - 18E8H and terminates by a delimeter - FF. For example, if users key in two statements as follows.

```
010  LET  A5 = 180
020  PRINT A5
```

   The BASIC Interpreter will accept the keyword and convert the keyword into the corresponding BASIC internal code (as to the corresponding BASIC internal code, please refer to the page 24). The below instructions occupy 18 bytes of memeory.

```
18E7   FF
18E8   00
18E9   01
18EA   00
18EB   1F
18EC   0A
18ED   05
18EE   18
18EF   00
18F0   00
18F1   01
18F2   08
18F3   80
18F4   00
18F5   02
18F6   00
18F7   20
18F8   0A
18F9   85
18FA   FF
```

The BASIC Interpreter sets Bit 7 of the last byte of each statements as the delimeter with a view to distinguish from another statements. From the above example, we know that 18F3 00 will be shown as this statement had been terminated, but 18F3 80 will be displayed when this statement will be went on with other statements.

3. The stack pointer of the BASIC-MPF is 1F9FH because the USERSTK of MPF-I is 1F9FH. Thus, if the user's programs are so large that in excess of the memory address - 1F9FH, the programs storing in the BASIC Interpreter will be disordered.

### *2.3 Program Edit*

Before editing program, we can check any statement by LIST command. For example, if we want to check statement 10 of the above program, pressing [LIST] [1] [0] [ENTER]. The LED display will show statement 10 from right to left. And then the display will show [A .]. We leave LIST mode by pressing [ENTER] key, LED display will show [bASIC .] to indicate that BASIC is ready. As the data on the display is being shifted from right to left pressing any key (except RESET) will stop the shifting. Shifting may be resumed by pressing any key (except RESET). By repeated key presses (for example the [LIST] key) you may start and stop the shifting action of a display several times. Because each BASIC statement has a unique line number to identity it, we edit the request statement by replacing it with a new statement. For example, if we want to change statement 10 to

```
10 LET C = A
```

The keys we should press are [1][0] [LET] [C] [=] [A] [ENTER]. After [ENTER] key is pressed, the statement 10 PRINT A will be removed from program and 10 LET C = A will be stored into it. The program stored in program memory now becomes

```
 5 LET A = 1
10 LET C = A
```

If we want to insert a statement 7 PRINT A between statement 5 and 10, we can press following keys: [7] [PRINT] [A] [ENTER]. After the [ENTER] key is pressed, the statement 10 LET C = A will move backword from the original memory location to leave space for statement 7 PRINT A. Statement 7 will be inserted between statement 5 and statement 10. Now the program stored in the program memory becomes

```
 5 LET A = 1
 7 PRINT A
10 LET C = A
```

We can delete any statement from the program memory by pressing its line number and then [ENTER] key. For example, we want to delete statement 10 from the above program, press the keys as follow:

[1] [0] [ENTER]

Now the program stored in program memory is

```
5  LET A = 1
7 PRINT A
```

### *2.4 Cassette Storage*

(1) Saving Program onto Tape
We can save the program in the memory on the tape by pressing [SAVE] <filename> [ENTER]. Where:
(a) Filename is a decimal number from 0 to 255 and.
(b) Before pressing [ENTER], you must connect the microphone of the recorder to MIC jack of MPF-I and press PLAY and REC to start recording.

The display is blank during transfer, the TONE-OUT LED is on and the tone sounds. After recording, the display will show [bASIC .].

(2) Loading Program from Type
We can load the program from type into memory by pressing [LOAD] <filename> [ENTER]. Where:
(a) Filename is a decimal number from 0 to 255 and.
(b) Turn the volume of recorder to Maximum.
(c) Press [ENTER], and finally start the recorder by pressing PLAY.

At the beginning, the display is [. . . . . .]. When the requested file is found, the display becomes [- - - - - -]. When the program was saved on the tape an additional value called a checksum was recorded. This value is produced by adding up all the characters (bytes) in the program. As the program is read back into the MPF-I memory a new checksum is calculated. If the newly calculated checksum doesn't match the checksum from the tape, an error has occured and the display will show [-Err]. Control will now be passed to the monitor. If the checksum generated when the program is read from the tape matches the checksum on the tape, then the last input byte will be displayed.

### *2.5 Immediate Instruction*

A statement entered without a statement number will executed immediately after the [ENTER] key is pressed. "Immediate" instructions allow you to use the Micro-Professor as a calculator. The procedure to compute 3 x 8 is shown below.

[PRINT] [3] [*] [8] [ENTER]

The display will show [00024]. To return to [bASIC .] press [CONT].

# Ⅲ.Summary of BASIC

### *3-1. Variables*

(a) Numeric variables are the only variable type used in MPF-I BASIC.
(b) Variables can only hold "integer" or whole number values.
(c) Limit: The limit for MPF-I BASIC integers is from -32767 to 32767.
(d) Variable Name: The length is one or two characters. If the length is one character, it must be any of alphabetic letter A-F. If the length is 2 characters, the first must be an alphabetic letter A-F. The second must be any one of the numeric digits 0-9.

### *3-2. Operators*

| Symbol | Function |
|---|---|
| = | Assignment or equality test |
| - | Negation or Subtraction |
| + | Addition |
| * | Multiplication |
| / | Division (Integer) |
| ** | Raising to a power |
| = | Equal |
| < | Less than |
| > | Greater than |
| ~ (NOT) | 1'S Complement |
| ∧ (AND) | Bitwise AND |
| ∨ (OR) | Bitwise OR |

### *3-3. MPF BASIC Syntax Grammar Table*

```
**********************************************************************
*                                                                    *
*            MPF  BASIC  Syntax  Grammar  Table                      *
*                                                                    *
**********************************************************************

digit    ::= 0 | 1 | ... | 9

letter   ::= A | B | ... | F

operator::= ~ | ^ | v | + | - | * | / | **

unary    ::= + | - | ~

relation::= > | = | <

decimal  ::= <digit> | + <digit> | - <digit> | <decimal> <digit>
         ( Range : -32767 to +32767 inculsivly )

filename::= <decimal>
         ( Range : 0 to 255 inclusivly )

portaddr::= <decimal>
         ( Range : 0 to 255 inclusivly )

variable::= <letter> | <letter> <digit>

memory   ::= M <decimal>

port     ::= P <portaddr>

line#    ::= <digit> | <line#> <digit>
         ( Range : 1 to 3 digit )
```

```
value    ::= <decimal> | <variable>

VMP_val  ::= <value> | <memory> | <port>

VMP_var  ::= <variable> | <memory> | <port>
express  ::= [unary] <value> | [unary] <value> <operator>
             [unary] <value>
VMP_exp  ::= [unary] <VMP_val> | [unary] <VMP_val> <operator>
             [unary] <VMP_val>
*** CLAUSE ***
LET_clause       ::= [<line#>] LET <VMP_var> = <VMP_exp>

PRINT_clause     ::= [<line#>] PRINT <VMP_exp>

INPUT_clause     ::= [<line#>] INPUT <variable>

NEXT_clause      ::= [<line#>] NEXT <variable>

GOSUB_clause     ::= [<line#>] GOSUB <line#>

RETURN_clause    ::= [<line#>] RET

GOTO_clause      ::= [<line#>] GOTO <line#>

FOR_clause       ::= [<line#>] FOR <variable> = <value> TO <value>

IF_clause        ::= [<line#>] IF <VMP_exp> <relation> <VMP_exp>
                     THEN <line#>
CALL_clause      ::= [<line#>] CALL <decimal>

STOP_clause      ::= [<line#>] STOP
```

```
*** COMMAND ***

RUN_command       ::= RUN [<line#>]

LIST_command      ::= LIST [<line#>]

SAVE_command      ::= SAVE <filename>

LOAD_command      ::= LOAD <filename>

NEW_command       ::= NEW

CONTINUE_command::= CONT

DELETE_command    ::= DEL

CLEAR_command     ::= CLR
```

**MPF BASIC Internal Code Table**

| Code (hex) | Mean | Code (hex) | Mean | Code (hex) | Mean | Code (hex) | Mean |
|---|---|---|---|---|---|---|---|
| 00 | 0 | 10 | NOT | 20 | PRINT | 30 | delete |
| 01 | 1 | 11 | AND | 21 | INPUT | 31 | clear |
| 02 | 2 | 12 | OR | 22 | NEXT | 32 | enter |
| 03 | 3 | 13 | + | 23 | GOSUB | 33 | (left) |
| 04 | 4 | 14 | - | 24 | RET | 34 | (right) |
| 05 | 5 | 15 | * | 25 | GOTO | 35 | (down) |
| 06 | 6 | 16 | / | 26 | FOR | 36 | (up) |
| 07 | 7 | 17 | ** | 27 | IF | | |
| 08 | 8 | 18 | = | 28 | CALL | | |
| 09 | 9 | 19 | < | 29 | STOP | | |
| 0A | A | 1A | > | 2A | run | | |
| 0B | B | 1B | M | 2B | list | | |
| 0C | C | 1C | P | 2C | save | | |
| 0D | D | 1D | TO | 2D | load | | |
| 0E | E | 1E | THEN | 2E | new | | |
| 0F | F | 1F | LET | 2F | cont | | |

## 3.4 System RAM Areas of BASIC-MPF

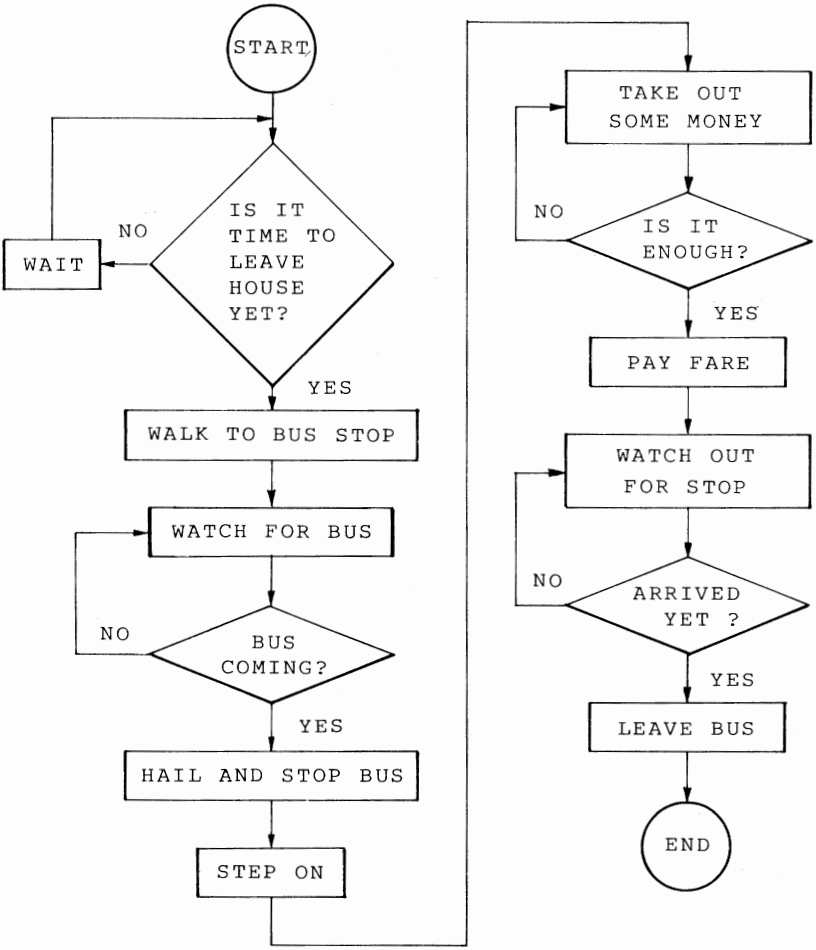
```
1800            ORG     1800H
1800  VARBUF:   DEFS    2* (6*10H+6)   ;variables buffer
18CC            DEFS    1
18CD  LINBUF:   DEFS    21             ;line buffer
18E2  FLAG:     DEFS    1
18E3  STOP:     DEFS    2
18E5  SP__:     DEFS    2
18E7            DEFS    1              ;delimeter FF
      PRGBOT:
                END
```

# Ⅳ.Introduction to programming

### *4-1. Flow Chart*

A computer executes commands in a precisely logical and sequential fashion. This is how you must learn to think while programming. Imagine how you perform the activity of going to the library to select a book. To borrow a book from the library, you must first travel to the library; but before that you have to leave the house, and before that you make a decision to go, and perhaps collect a load of books for return. In the sentence you have just read, the activities are stated in completely the wrong order. However, you are perfectly capable of sorting them out and executing them correctly. To do this, you use a type of intuition born of long experience. The computer you have just bought is not that intelligent. It must be fed with instructions and presented in precisely the correct order, or it will make a mistake.

To help in this process, a diagrammatic "flow-chart" method will be presented--the visual sense being the most receptive at taking in information. You could break down anything you want your computer to do into small logical steps and arrange them in the correct order. Think of the activity of catching a bus. You know what time the bus is supposed to come, so you make a decision to leave the house at some definite time. When you arrive at the bus stop, you must watch for the bus and put your hand out to hail it when it comes. On boarding the bus, you might ask for the price of ride, and then look for the right amount of money. When you have paid, it is essential to keep your eyes open for the right stop and leave the bus when it comes along. The following flow-chart puts some of these ideas into a diagram for illustration. A few different shapes of boxes are used for different purposes, they are explained below.


START
IS IT TIME TO LEAVE HOUSE YET?
NO
WAIT
YES
WALK TO BUS STOP
WATCH FOR BUS
NO
BUS COMING?
YES
HAIL AND STOP BUS
STEP ON
TAKE OUT SOME MONEY
NO
IS IT ENOUGH?
YES
PAY FARE
WATCH OUT FOR STOP
NO
ARRIVED YET ?
YES
LEAVE BUS
END

Circles are used to indicate the beginning and end of a process. The symbol a circle contains the word start. Sometimes it will contain the name of the process. Rectangles are used for statements where some sort of command is to be executed. The major different is in the diamond shaped "decision" boxes. A computer is perfectly capable of making decisions as to whether something is to be performed or not, and the decision which has to be made is written in a diamond as shown. The process of catching the bus is shown here stage by stage. The first decision which has to be made is whether it is time to leave the house yet. If not, you must wait and keep on checking the time until it is time to leave. This is indicated by a "loop" on the chart. That is, a closed path which is executed around and around until some condition is met. Here the condition is the time. When the time comes, you walk to the bus stop and enter another looping process while waiting for the bus. The process should be almost self-explanatory from a flow-chart, and in many ways this form is one of the best for explaining new ideas to people including more computer applications.

### *4-2. Problem with Solutions*

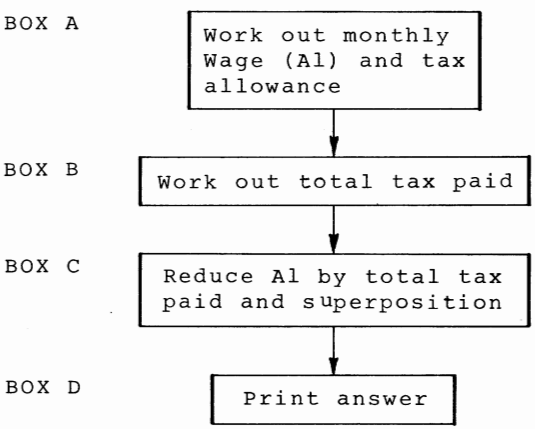
Let us consider the problem of working out one's monthly income after tax. You should have some idea of how the system works already, and it is an easy problem to compute. It also illustrates how you think in a logical manner. The steps are as follows.

(1) Imagine what steps you would use to calculate the answer.
(2) Indentify the output - i.e. what answer do you want.
Having done this, the problem is defined. Once you have set out the information available (input), and what you expect from the computer (output), you can concentrate on the processing of input data to produce the required output.

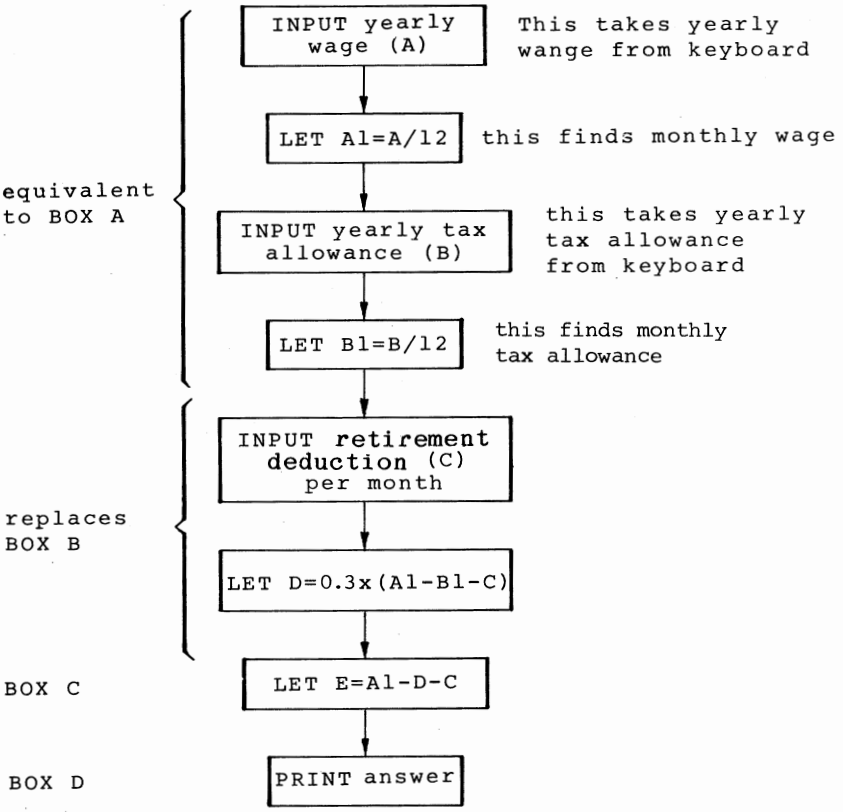
(3) Imagine what steps you would use to calculate the answer.
(4) Write down the steps found in (3) in a "flow-chart", or logical sequence.
(5) Convert into computer language.
(6) Type in and RUN the program.
(7) Debug (i.e. correct programming errors) and return until correct.
(8) Store on tape for future use.

This given information is yearly salary, yearly tax allowance, monthly retirement deduction and tax rate.

The flow-chart as follows:



Breaking down the above chart, a flow-chart of the following from is produced.




We can now write down a program from the last flow-chart which will give a good estimate for most people's monthly wage, given that tax is payable at 30% per year.

(a) All lines must start a line-number. Choose incremens of 10 between line-numbers to allow plenty of room for inserting lines into the program wherever you wish.

(b) Each line contains just one "statement" which must start with a keyword such as PRINT or LET etc.
Check that you can match each of the boxes in the flow-chart with a statement below

Statement below

```
 10 INPUT      A
 20 LET        A1 = A/12
 30 INPUT      B
 40 LET        B1 = B/12
 50 INPUT      C
 60 LET        C1=A1-B1
 70 LET        C2 = C1 - C
 80 LET        C3 = 3*C2
 90 LET        D = C3 / 10
100 LET        E1 = A1 - D
110 LET        E = E1 - C
120 PRINT      E
130 STOP
```

Suppose we run this program after its entering, just key in [RUN] and [ENTER]. The display of MPF-I will show [1 n P.]. After you enter value A and followed by [ENTER], the program will continue execution until statement 30. At this time you should key in value B, and so on until you enter all A, B, C values. The program continues to execute. After statement 120 is executed, the display will show the value E. Press [CONT] key and the display will show 130 . This means that program execution stops at statement 130, press [CONT] to return to BASIC READY. Now key in [RESET], the statement will return to the MPF-I monitor but the original data stored in the memory is still in there. After pressing reset if you enter BASIC by keying in [GOTO] [2] [0] [1] [7] [RUN] ([GOTO] [0] [8] [1] [7] [RUN] for 4K monitor), all of original statements and values represent by the variables will be retained. All variables and statements are lost if you enter BASIC by key in [GOTO] [2] [0] [0] [0] [RUN] (0800 for 4K monitor).

### *4-3. Error Codes*

```
E0  Expression Overflow
E1  Illegal decimal Value
E2  Illegal Variable Name
E5  Illegal Relation  Operator
E9  Syntax Error
EA  Illegal Value
EC  FOR......NEXT  Not Compatible
ED  GOSUB......RET  Not Compatible
EF  Line Not Found
```

## *4-4. Examples*

1. Use FOR...NEXT loop to add from 1 to a certain number.

```
10 INPUT  C
20 LET    A = 0
30 FOR    B = 1  TO C
40 LET    A = A + B
50 NEXT   B
60 PRINT  A
70 STOP
```

[Description]: After executing the statement, key in [RUN] [ENTER], and the LED display will show [Inp .]. At this moment, key in the value C and then key in [ENTER], the display will show the result. For example, the result will be 55 when the value C is 10 (1+2+3+4+5+6+7+8+9+10=55). In a FOR...TO statement the initial value (the number or variable between the equals sign and TO) and the limiting value (the number or variable to the right of TO) must be of the same sign.

2. Account the series digits amount

```
10 INPUT  C
20 INPUT  D
30 LET    A = 0
40 FOR    B = C  TO  D
50 LET    A = A + B
60 NEXT   B
70 PRINT  A
80 STOP
```

[Description]: C is the initial value and D is ending value. If you want to add from 11 to 15, press [RUN], [ENTER] the display will show [INP .], key in an 11 for the variable C. Key in the value 15 and press [ENTER], the display will show 65.

3. In the program below we can use the statement "GOSUB....RETURN" to execute the different subroutines depending on the value C. If the value C is larger than 7, the display will show "1+.. ..+C". When C is equal to or less than 7 then 1+...+C will be computed.

```
 10  LET     A = 0
 20  INPUT   C
 30  IF C > 7 THEN 70
 40  GOSUB   200
 50  PRINT   A
 60  STOP
 70  GOSUB   100
 80  PRINT   A
 90  STOP
100  FOR     B = 1 TO C
110  LET     A = A + B
120  NEXT    B
130  RETURN
200  LET     A = A + 1
210  FOR     B = 1 TO C
220  LET     A = A * B
230  NEXT    B
240  RETURN
```

4. Apply the CALL statement to call the "TONE" subroutine of MPF-I.

```
10  INPUT  C
20  FOR    A = 1  TO  C
30  CALL   1508
40  FOR    B = 1  TO  1000
50  NEXT   B
60  NEXT   A
70  STOP
```

[Description]: Entering the monitor at hexadecimal address 05E2 will produce a 2KhZ tone on the speaker. 05E2 in hexadecimal equals 1506 in decimal. 05E2 (BASE 16) = 1506 (BASE). The value entered into C will determine the number of sound intervals. To

obtain a 1KhZ tone enter the monitor at 05DE (BASE 16) = 1502 (BASE 10). You may write a program in assembly language to produce sounds of different frequencies and duration. Enter the assembly language shown below.

```
1800 0E80    LD  C,80H
1802 21C000  LD  HL,0C0H
1805 C3E405  JP  05E4H
```

change line 30 in the BASIC program to

```
30 CALL  6144
```

note 6144 (BASE 10) = 1800 (BASE 16). Now execute the BASIC program. Use a value of 5 when input is requested.

5. The operations of AND, OR, and NOT

```
10  INPUT  A
20  INPUT  B
30  LET    C = A ∧ B
40  PRINT  C
50  LET    C = A ∨ B
60  PRINT  C
70  LET    C = ~C
80  PRINT  C
90  STOP
```

[Description]: Suppose the input value A is 0 (the corresponding hexadecimal value 0), and B is 255 (the corresponding hexadecimal value is FF). After execution, the LED display will show 0. This representd 00H and 0FFH ANDed together. After key in [CONT], the LED display will show 255. This represents 00H and 0FFH ORed together. Again key in [CONT], the display will show -32512 which represents the NOT of 0FFH.

6. Storing data in RAM

```
10  INPUT   A
20  LET     M 6144 = A
30  LET     B = M 6144
40  PRINT   B
50  STOP
```

[Description]: The input value can be stored in the position of 1800H and 1801H. For example, if the value of A is 13, then 0DH is stored in 1800H and 00H is stored in 1801H. Thus, the value of B is 13 which equals the value of 1801H and 1800H in the memory.

7. Output 055H from the port A of PIO

```
10  LET   P130 = 15
20  LET   P128 = 85
30  STOP
```

[Description]: The control port position of MPF-I channel A is 82H. (The corresponding decimal value is 130.) The PIO channel A is selected to be the output port, so its control word is 0FH. (The corresponding decimal value is 15.) And the position of data port A is 80H. (The corresponding decimal value is 128.) The output data is 55H (the corresponding decimal value is 15).

8. The division operation

```
10   INPUT   A
20   LET     C = A/3
30   PRINT   C
40   STOP
```

[Description]: BASIC variable in the MPF-I are only integers. Suppose the input A is 11, the value C is 3 after executing i.e. C is a quotient.

9. How to check the program stored in the memory.

Use example 8 to demonstrate the list actions shown below.

(1) Key in [LIST] [1] [0] [ENTER], statement 10 will be displayed from right to left (the contents will be shift across the screen).

(2) Key in [↓], the contents of the next statement (statement 20 in this case) will be displayed. Press the [↓] key again - now statement 30 will be displayed.

(3) Key in [↑] the previous statement will be displayed (20 in this example).

(4) Key in [ENTER] if you intend to leave the List mode.

(5) Use [↑], [↓], [←] in the List mode, you don't need to press the SHIFT Key.

(6) You just need only to press the [←] key when you want to review the execution in LIST mode.

(7) Just key in [LIST] [ENTER] directly, if you intend to get the first line of the progam in LIST mode.

10. This example is to instruct users how to write the program storing in RAM buffer to EPROM and how to utilize it. Before the users store the program in the EPROM, having to add a "FF" at the beginning and ending of this program. Thus, it is just needed to run the beginning of this program as the execution on the BASIC Interpreter. We can explain it from the following example.

```
    010     LET   A = 15
    020     PRINT A

Key in      18FF  FF
            1900  00
            1901  01
            1902  00
            1903  1F
            1904  0A
            1905  18
            1906  00
            1907  00
            1908  00
            1909  01
            190A  85
            190B  00
            190C  02
            190D  00
            190E  20
            190F  8A
            1910  FF
```

[RUN] M 6400 [ENTER] ($6400_{10}$ =$1900_{16}$)

[Description]: The users first have to check whether the value of the memory address 2225 or 0A25 (for 4K monitor) is 48H or not. As the contents of this memory is 4AH, the user ought to use the EPB-MPF or another method to change the value to be 48H.

PART NO.: 49.00710.001