Instrumentation and control by notebook PC
- Invitation to HT(HP) – BASIC-

Tominaga, Masaki
Researcher
Advanced Technology Research Group,
National Research Institute for Earth Science and Disaster Prevention

Abstract

Instrumentation and control systems by use of computer are becoming popular. Most of them that we can use in daily experiments are offered in the form of pre-fixed systems. The pre-fixed systems are designed to remove complexity in the setup of the instrumentation and control systems. It is true that the hardware is well combined with the software in those systems, but it is difficult or impossible for us to modify them for our own. Use. On the other hand the published software that have flexibility in use are so well designed that researcher cannot see what is going on in the systems or the programs. In the present paper HT-BASIC is introduced for the use of instrumentation and control by use of notebook PC. The HP-BASIC was a well-acknowledged software to control measurement systems by PC. Researchers could manipulate them for their own use. Since discontinuance of the PC specified for the use of the HP-BASIC the software became unpopular. But new version of the HP-BASIC that name was changed to “HT-BASIC for WINDOWS” is designed to work well in WINDOWS environment. We can make personal systems for instrument and control by the “HP-BASIC for WINDOWS” without much difficulty.

Keywords; Notebook PC, BASIC, Field instrumentation, User oriented instrumentation system

1. Foreword

These days computers have been widespread and owned privately as multipurpose terminals for information retrieval through internet, e-mail exchange and creating documents. Cray 1, a super computer marketed by Cray for 2.7 billion yen in 1976, was with 160 M- FLOPS, 8 MB of memory, on the other hand an ordinary PC today is with 2GHz clock and 500MB of memory with the improvement of function. It could be said that we have more than super computer on the desk. And also the software for instrumentation (VEE, LabView etc.) operating on an universal OS, WINDOWS, have become available. Based on the WINDOWS specific GUI (Graphic User Interface) these software are easy to use, however it is difficult for researchers to see what is going on in the systems or in the programs. It is frustrating for researchers to use these ready-made application, because they need to know every process of the

instrumentation system and give clear instruction for data collection and post-processing [Ref. 5, 6]. C language on the other hand is used because it is highly-portable, however it is far from convenient for setting up an interactive system among operator, measured data, environmental condition and instrumentation & control system.

It is hard to understand the system flow for anyone else than the programmer himself. Even the programmer himself might forget it after a few years, unless he wrote down comments in every modification. Followings are the reason why C language is not suitable for instrumentation and control system – there are too many ways for one operation to see the purpose, there are different algorithms for each programmer to express one operation in function, and programmer forgets the purposes of settings in time lapse. When I see young researchers or students who do not have complete control of instrumentation systems, although they spend much energy to the job, I have no other choice to reevaluate HT(HP)-BASIC which is now overshadowed by the prosperity of graphical programs. This report is to introduce the outline of HT(HP)-BASIC based on the experiences of our Institute for all students and researchers who have faced above problems.(All brand names and product names are trademarks or registered trademarks of their respective companies.)

2. History of HT(HP) – BASIC

In the past 30 years history of instrumentation system, it was still a dream to systematize the instrumentation in the beginning of 70s, except for CAMAC which was already used in only huge projects like nuclear energy. At the time researchers used stand-alone instruments and they were satisfied if they could record data of temporal change by ink recorder or data recorder. However with dynamic study of signal theory in the field of signal processing, accuracy of measurement time was very important as well as raising precision of measured signals for obtaining required information. Digitalization was not common yet, though it was desired to materialize the arbitrary instrumentation by combining multiple instruments. It was only after the operational amplifier and the logic IC of Texas Instruments became widely used with development of IC technology that digitalization progressed rapidly. At the site of manufacturing industry or plant, instrumentation does not exist solely. But controlling process at plant or manufacturing lines based on the measured results always follows it. Also in field of research we need to do some operations to change experimental condition or change input signals based on the measured results. Previously specific designed individual systems were required to drive the actuator for switching or operating valves based on the analog signals. In order to establish a versatile system which meets users' requirement, realization of the instrumentation system fully controlled by computer was desired. It was 1973 Hewlett-Packard realized the first digital control bus system (HPIB, Hewlett-Packard Interface Bus) as a company standard. It would not have been realized unless HP had both know-how as instrumentation manufacturers and desktop computer technology. After that HP applied this as international standard and was approved as IEC Standards in 1975. Since

then this system was officially spread as GPIC (General Purpose Interface Bus, though HP internally called as HPIB).

Even when it was just a company standard of HP, this system included printer, desktop computer with tape cartridge storage drive, A/D converter, scanner, digital clock, D/A converter, signal synthesizer and frequency and time interval counter. With all these equipment, an instrumentation and control system could be realized.

In 1977 HP released 9825A, a historic desktop computer. It was designed for controlling GPIB when GPIB became a global standard. Henceforth the desktop computer as center of the system has been called "Controller". A standard 9825A included only one-line- display with 32 letters, 250kbyte data cartridge by tape, 16-digit thermal printer and 6.8kbyte RAM, in retrospect it was incredibly small since there was no nonvolatile RAM. However it could build up an advanced instrumentation and control system with its excellent operating system and unique language (Hewlett-Packard Language). Thereafter HP-BASIC Language (also called Rocky Mountain Basic) appeared together with the desktop computer with built-in CRT display, which is now-familiar though. This language became a core language for the instrumentation & control computer (Controller) of HP which included not only instruction for external devices and data in- and output but also functions of numeric data processing, graphics output, creating files and data transfer. Since then the concept of Laboratory Automation or Factory Automation became widespread. After that HP-BASIC became a comprehensive language upgraded with HP-BASIC PLUS as frontier of GUI.

In 1986 Toshiba USA released a portable notebook PC, T-3100. This made a big impact and determined the future stream of personal computer. As the result users of HP-BASIC strongly desired portable controller, since with this controller it would be possible to build up an instrumentation system not just for indoor use. As soon as IBM PCs became global standard, Intel CPU chips also became standard. Motorola was also manufacturing CPU chips, however they had to give up its production because their architecture called 68 system was different from Intel's. According to this HP announced stoppage of production of controller which included Motorola's CPU chip in the middle of 1990s. This announcement surprised HP global users and they began to stock up on HP controllers for obtaining spare parts. According to the users' requirement HP had to postpone the production stoppage in a couple of years. And at last in October 1996 the production of HP9000/300 series PC supporting HP-BASIC operating system was stopped. HP-BASIC was terminated with version 6.2. As long as the production stopped and no maintenance was available, there was nothing users could do when existing controller broke up. And so the requirement for HP-BASIC operating on the standard OS for IBM PC, Windows was increased. On the other hand since 1988 TransEra was selling a HP-BASIC called Rocky Mountain Basic which was operative on IBM PC. Then HP was selling OEM from TransEra as HP-BASIC for Windows from the beginning of 1996 through November 2000.

Herewith users did not have to worry about the breakdown of their controller technically, but

it brought confusion because what the controller needed to be became vague. One of the reasons was that user had greater respect for conventional HP-BASIC. In these days many kind of CPUs and its manuals were on the market and people enjoyed hand made computers. It was well known to development engineers that Motorola's 68system CPU had different architecture than Intel's. As both were based on different systems, it was impossible to convert the old program to new one completely. The comparison between old and new program and the detail explanation about what could not be done were written in English manual for HP-BASIC for Windows. It was wholly reasonable for development engineers but it made users feel that it was impossible to use. HP committed the development of VEE and sold HP-BASIC for Windows from TransEra, however HP did not corporate in a positive manner. HP Japan's Japanese manual was only for the participants in their paid seminar, so it was not available for researchers and students who were interested in it. Even if they could get the manuals, they only found the way of programming but no brilliant features of HP-BASIC, which would allow them to control instrumentation easily. Appearance of Windows also affected the case. As at the first stage Windows were advertised as a dream come true that could realize everything, some people still chose the high portable C language for their program. There might be some experienced engineers who spent much time for learning C language or searching study books. A range of study books must show the difficulty of C language. Especially it was difficult to find manual of C language focused on instrumentation and control. But it became possible for Windows to create visual documentation easily, and applications for instrumentation and control like VEE from HP or LabVIEW from National Instruments also appeared on the stage with which visual programs could be created. While NI hold application seminars for students as prospect users and published manuals in several languages, HP was not an eager promoter at all. According to the wide spread of PCs, various manufacturers other than NI and HP began to sell their own interface board with their own application software for instrumentation and control. These Windows based systems were spread because not only students who touched the system for the first time but also users who were familiar to other systems could use them by reading manuals. It is possible for users to collect data by clicking icons according to the manual, however users also feel discontent or anxiety because they can not understand the process how they collect data. In this manner data are collected solely and calculation and making graphs are done by separate applications.

HP-BSIC described in this report is an all in one application, which includes all above-mentioned functions. HP-BASIC changed its name to "HT-BASIC for WINDOWS" by TransEra, is existing hitherto with improvement of editing view.

So far we have seen the stream of instrumentation and control systems since HPIB, the control for instrumentation is still based on the GPIB protocol today and each company's PC cards for GPIB and for GPIO (General Purpose Input and Output) can be controlled by HT(HP)-BASIC. It is because of its low profile why HT(HP)-BASIC is not so popular among

students, though it was a global standard for the engineers and researchers who are now in their late 40s or 50s. At that time measuring instruments and controllers (computers) from HP were so expensive that it was difficult for universities to have them. As a Japanese manual was available, it was used at well-endowed official organizations or private enterprises. However it was difficult for universities to budget it, because one dollar was around 300 yen and the exchange rate was fluctuating at that time. As a result, HP-BASIC did not become popular at research lab of universities and was not handed down among students. If nothing is done, I don't believe it will be spread among students from now, because it has array of English manuals (only on-line manuals were available) and no picture of which is seen. However now is the chance for us to be able to program the instrumentation and control system easily by using notebook PC, as the cost for introduction of the system has become lower. This report is to aim for you to understand the overview of HT(HP)-BASIC and to open your eyes to the possibility of the HT(HP)-BASIC, if you once read it through.

It was March 1974 when the facility of large-scale precipitation experiment was completed in National Research Center for Disaster Prevention (current National Research Institute for Disaster Prevention). At that time instrumentation facility was not established yet. In order to carry out various experiments, we thought it was the best if configuration of the instrumentation and control system centered on computer would be modified freely, however we had no models yet. Around that time HP proposed HPIB as a company standard, and I found an ordinary system could be made through with it. At this stage HPIB was not official standard, but we decided to use it because it was possible to configure a closed instrumentation system only with products of HP. Afterward HPIB (GPIB) became an international standard and so widely spread that there was no instruments without GPIB connector. And many companies also released controllers. Today it has still advantage controlling GPIB by HT(HP)-BASIC from the view of researchers who need to conduct experiments by configuring instrumentation and control system freely. TransEra has been improving HT-BASIC taking over HP-BASIC. Compared with the final version of HP, HT-BASIC succeeded to extend and integrate editing function, mathematical function, subroutine, and visual display, maintaining its high-toned control function. Keeping with the trend, HT-BASIC has become an easy-to-use program based on GUI operated on Windows.

3. Outline of HT-BASIC (Reference 1, 2)

HP-BASIC originally developed by HP was taken over by TransEra, changed its name to HT-BASIC and is being developed. HT-BASIC version 9.2 is currently available. Hereinafter I will call it "HT-BASIC" and will overview of its feature.

HT-BASIC is programming language includes controlling external devices connecting GPIB bus line, controlling digital signal input/output, controlling CRT output display (including LCD), creating files on disc, controlling data input/output and conducting general numeric calculation. Figure 1 shows overall framework of HT-BASIC. As for desktop computers GPIB

boards and GPIO boards for general purpose input/output are produced and as for notebook PC computers PC cards (PCMCIA) for GPIB and for bit input/output are commercially available.

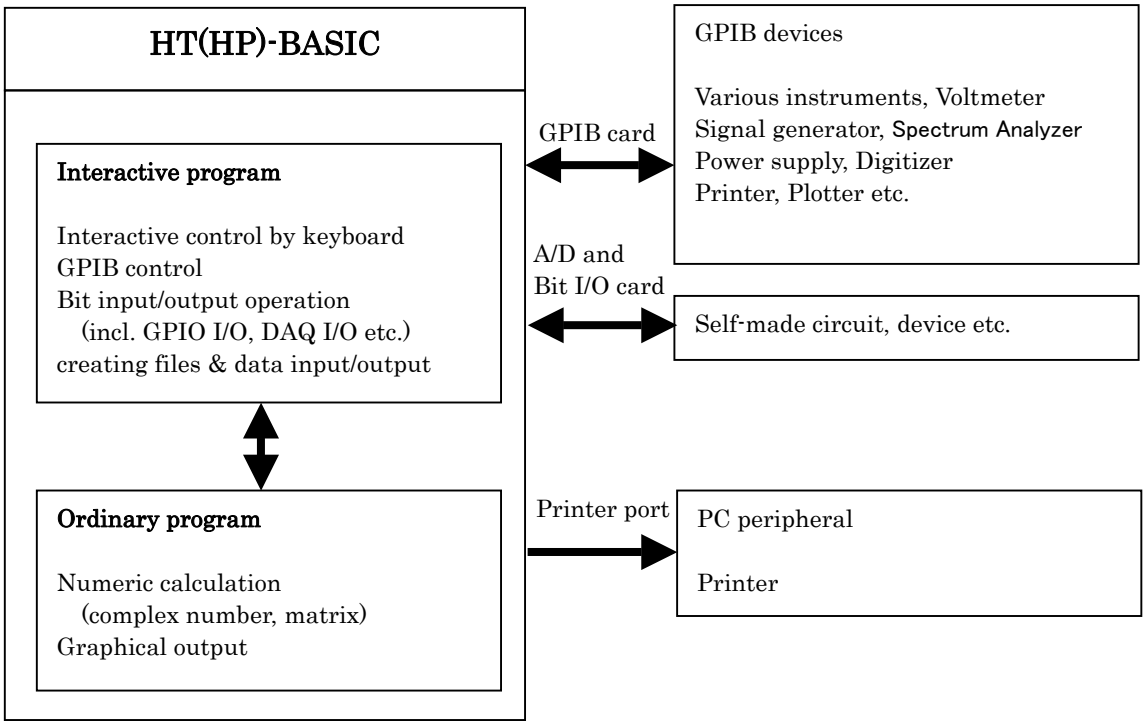


Fig. 1 HT(HP)BASIC and its environment

3.1 Feature of HT-BASIC

(1) Interpreter language :

HT-BASIC is basically an interpreter language, which reads the instruction line one by one and runs the program. It is not appropriate for running large-scale numeric calculation, since it is not a compiler language like C or Fortran.

However interpreter language is useful in case it should run under response of all other devices which work with each speed like numeric calculation with low speed, operator interactive program and devices (e.g. various measuring instruments, plotter, digitizer, digital signal input/output, or self-produced relay circuit etc.) other than peripherals combining with bus line (printer, scanner, hard disc etc.) .

The greatest characteristic of interpreter language, though its name was derived from it, is the nature that it runs immediately when pressing RUN. In program development at laboratories it is the normal method to write the skeletal program at first based on what you want to conduct, then to add various function afterward for making it easier-to-use. At that time if you use C language you have to repeat “save files -> compile -> execute” for every tiny modification, which might make you trouble to confirm a flow of ideas. (Refer to postscript). It often happens at site of experiment or instrumentation that original program needs to be improved. In this case you have to pay close attention not only to variables but also to flow of

program, if you use C language. But if you use BASIC language, it is easy to confirm the flow of the program by following lines. If the original program was created by anyone else, it is impossible to rewrite C language but BASIC. In this manner the program with BASIC will increase its value as property of laboratory.

On reflection, these feature of BASIC are something we don't have to list. It is the mission of program to execute what we want to do. And it is the reason of this report that the mission of program has become vague these days.

(2) Compiler function :

HT-BASIC is basically interpreter language. But compiler function annexed is also available in numerical calculation.

(3) BASIC language :

Like its name, HT-BASIC is a BASIC language and has common structure to other BASIC language and simple logic. If anyone else reads it or programmer himself will read it in a few years, it is understandable and easy to improve. A program language needs to be simple. It must be an intrinsic flaw for C language that even experienced program developer is subject to make fundamental mistakes with it.

(4) Subroutines :

There are not only ordinary subroutines but also library of compiled mathematical function and subroutines, which can be called out and used in the program freely. In HT-BASIC almost all of mathematical function are in CSUB style and it is also possible to create CSUB for oneself with C language.

(5) File Input/Output :

It is possible to create a file with any name and write collected data in the program, which can do execution for long term instrumentation.

(6) GPIB :

HT-BASIC is a simple and strong language and it proves its worth in controlling measurement hardware via GPIB buslines. When using a notebook PC, insert the GPIB card to slot for PC card, and connect the special connector at the other end with GPIB socket of the measuring instrument. There are also USB cables having GPIB connector at another end.

(7) Bit Input/Output :

As with the case of GPIB the bit I/O card is used for controlling bit I/O. Insert it to PC card slot and connect a self-made digital signal converter circuit at another end of the cable. GPIO is the name of HP's I/O device, which controls parallel 16-bit input line and 16-bit output line. Though only GPIO as PCI board for desktop computer are currently sold, NI and ines are selling PC cards called DAQ (Data Acquisition) which can control bit I/O. Among them there is a PC card with built-in A/D converter, which can control 8 to 24-bit input/output as well as measure voltage solely. If the PC card has built-in A/D converter, it is possible to make an instrumentation system on its own for measuring signals by converting into voltage signals.

(8) HT-BASIC Plus :

With an extension language called HT-BASIC Plus, measured data can be shown on a meter or as a graphical display. This is a pioneer function of GUI, which has become a main stream of current personal computer. However many people among HP-BASIC users have not used this function because it is regarded as a sub function for the purpose of instrumentation and control and it needs complicated parameter setting for display and can be substituted by HT-BASIC.

(9) Graphics-Output :

Graphics can be drawn directly on CRT. If a plotter which can read HPGL graphical language is connected to GPIB line, graphics can be output on plotter. HT-BASIC does not have an instruction for saving graphics on CRT as graphics file, but it can be saved by using "Paint" of Windows or other graphical capture software.

3.2 Installation of HT-BASIC

(1) Program :

The latest HP-BASIC is HT BASIC for Windows version 92. from TransEra　You can install according to instruction. Select not legacy version but HT-BASIC and custom install, then install all component. It takes about only 40M byte in total. If you don't specify anything, the directory after installation will be

C:\Program Files\HTBwin

notes; Correctly, although it is HTBwinXY (X:Version and Y:Release), below, it is marked as HTBwin.

(2) GPIB Card :

GPIB cards are sold by various manufacturers, but NI's GPIB+ card will be used in example below. Insert the GPIB card to card slot and install the driver according to the instruction.

(3) Digital I/O Card :

Digital I/O cards are also sold by various manufacturers, but ines's DAQ card, i250　will be used below. Among several cards which ines are selling I select i250 because it has no failure up to now and it is the most popular card. Install the driver according to the instruction. That's it for preparation.

I described general features so far and I will show how to create the program in following Chapter.

4. Programming of HT-BASIC 1 (program for bi-directional, interactive control)

HT-BASIC shows its ability in controlling devices connected with bus line and it also has capability for programming of numerical calculation or graphical output. In Chapter 4 & 5 the procedure for creating an interactive program will be outlined and the procedure for creating a routine program and its advantage will be outlined in Chapter 6 and 7.

4.1 Interactive Program

In this Section some example programs will be shown which represent the feature of

HT-BASIC as interpreter language. Here I would like you to compare it with C language or FORTRAN to realize that HT-BASIC is easy-to-follow, simple and convenient to modify the program. Instruction used in the program statement is called [INSTRUCTION], instruction sequence is called [STATEMENT] further below. Phrase in bracket e.g. <ENTER> shows the name of key on the keyboard and also input by keyboard or menus for selection by keyboard or emphasis so far as not causing confusion. In HT-BASIC [INSTRUCTION] is written capital letters. As most of these [INSTRUCTION] are used both in program statement and input by keyboard, I will not specify so far as not causing confusion. Detail explanation and right examples of use of [INSTRUCTION] are shown in help manual. Further, please refer to the example file where the concrete examples of use of each [INSTRUCTION] are explained. In editor screen of HT-BASIC for Windows Ver. 9.2, [INSTRUCTION], variables, string etc. are shown in different colors so that you can easily identify them. More functions have been added for example; to convert [INSTRUCTION] into capital letters automatically and to indicate errors in grammar. If you find some unknown [INSTRUCTION] in example program, select it by left click and view menus right click by the mouse button, then Help will appear at the bottom lineselect if by left click, you will see the explanation of the [INSTRUCTION].

(1) EDIT Screen :

Execute HTBwin.exe file in the directory

C:\Program Files\HTBwin.

Then black screen will appear. Select File -> New in toolbar, then select Edit -> Edit Mode, it will be changed into the program edit screen with white background. You may also paste the statements created in other text editor to edit screen for editing the program.

(2) Input by keyboard (Interruption of program and change variables, restart from arbitrary line) :

Input program in EDIT screen as below.

```
10  INPUT A                 ! input by keyboard
20  PRINT 3*A               ! output to display
30  PAUSE                   ! interruption of program
40  END                     ! end of program
```

Input program line and press <ENTER>, cursor will move to the next line and assign the line number automatically. Statement after <!> is comment text. If you put <!> at the top of the line, the whole line will become comment text. If you key in

RUN <ENTER>

or select Run on the toolbar (which is right pointed green triangle symbol), then the program will be executed. Then on the left bottom corner of the screen

?

will be displayed. This means it is waiting for key-in. If you key in 5 and press <ENTER>, 15 will be shown on the left top of the screen, then it will stop at line 30. 5 is assigned to variable A. To confirm it, key in

A <ENTER>

then, 5 will be shown at the left bottom corner. As the program remains at rest, key in

A=123 <ENTER>

and again A <ENTER>, 123 will appear at the left bottom. You will see 123 is assigned to variable A. The key in

CONT 20 <ENTER>

The program will restart from line 20 and display the result of calculation of 3*123 ,369 then stop.

If you key in

CONT <ENTER>

without assigning line number, the program will end at END line.

In interpreter language you can stop the program on the way and/or change variable then restart in this way. Even if the program stops with irregular response during measurement, you will be able to check the variable and restart the program without cessation.

To save the program return to EDIT screen and select <File -> Save as> from toolbar and save with adequate name. <SAVE> means to save the program as text. To read out program you can select <OPEN> from toolbar and left click the file name in an ordinary way. You will be able to read it out without thinking that *the text file saved by "SAVE" will be read out by "GET".*

(3) Key in strings by keyboard (change character variable, restart from random line)

Return to EDIT screen and key in below program similar to the program in previous Section and execute.

```
10  INPUT A$                ! input by keyboard
20  PRINT "Abc"&$           ! combination and output of string
30  PAUSE                   ! interruption of program
40  END                     ! end of program
```

A$ in line 10 is a character variable. If you put <$> after any variable identifier, it will be regarded as a string. Then "?" will be shown on the left bottom of the screen, press

def <ENTER>

then a combined string, "Abcdef" will be shown on the left top corner and the program will stop at PAUSE in line 30. The <&> in line 20 is an instruction to combine the strings, with which "Abc" and A$ will be combined.

If you key in

A$ <ENTER>

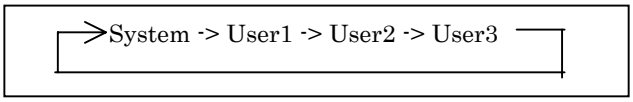
"def" will be shown on the left bottom corner of the screen. The string shall be put parentheses like <" ">, if you key in

AS = "xyz" <ENTER>

to give a new string for A$ and key in

CONT 20 <ENTER>

then a new string “Abcxyz” will be displayed on the left top corner.
As the program is suspended at line 30, key in
CONT <ENTER>
then the program will end at END line.
HT-BASIC is no match for C language in flexibility in operation of strings.
(4) Store and load of program
The program can be reserved in <STORE> and <SAVE> status in HT-BASIC. With <STORE> the program is reserved as immediate workable program. The program reserved by <STORE> can be called by <LOAD> and if line number is added to LOAD instruction, the program can be run soon after loaded. Another way is to reserve only the text of the program by <SAVE> instruction as ASCII file. The file reserved by SAVE will be called by <GET>. The file called by GET will not run immediately, however it is convenient for copying it to another system, as it is reserved in ASCII format. In HT-BASIC the program will be invoked judging the type of files automatically, you don’t have to think about LOAD or GET, if you did not specify any.
(2) Softkey and Suspension, Continuation and Cancellation of the program
Softkey menu is displayed at the bottom of the screen when the program is running.
Display or hide will be switched by pressing function key <F9>.
Every time when pressing <Shift + F9> the softkey menu will be gone round



If you press some softkey or left click by mouse, it has the same effect as executing this instruction by keyboard. When you have <System> softkey displayed and left click <Run> (F3) by mouse, the program will be executed. When you click <Pause> (F4), execution of the program will be suspended. When you click <Continue> (F2), the suspended program will be restarted. When you press <Stop> (F4), the execution of the program will be stopped and the system will be back in the idle state. Contents to be executed are assigned to softkeys under the name of displayed menus. It means that contents assigned include each control character. However it does not always mean that softkey and key in from keyboard have the same effect, if you key in as displayed. Users can assign three user-key-menus as they like. If you want to assign them, it will be of your help to refer to the key definition in default status. The contents assigned to each key can be shown by <list key + Enter>. It can be printed by <LIST KEY #10 + ENTER>. <#10> is the service number for default printer. (See 4.3 (2)).

4.2 External device control

(1) Device control by GPIB (Address assignment, Function settings, Data input)

Fig.2 GPIB Card and Cable (National Instruments Co.)

Insert GPIB card to the slot for PC card and install driver for the card. Connect GPIB connector at the other end to GPIB device. I used NI's GPIB and card and connected to HP's digital multi meter, 3475A as GPIB device in this report. Fig.2 shows NI's PC card and GPIB cable.

Go back to edit screen and key in below program lines to read the data from digital multi meter.

```
10  ASSIGN @hp3457 TO 722                ! open I/O pass
20  OUTPUT @hp3457;"NPLC .0005"          ! data to hp3457
30  OUTPUT @hp3457;"DCV AUTO, .033"      ! data to hp3457
40  WAIT 1                               ! wait one second
50  ENTER @hp3457;B$                     ! read character data from hp3457
60  PRINT B$,VAL(B$)                     ! display character and value
70  CLEAR 722                            ! close I/O pass
80  END
```

<ASSIGN> in line 10 means to open I/O pass to the device with GPIB address #22 through I/O port #7. I/O port #7 is generally GPIB card. #22 is an individual address number for the device connected with GPIB line. #0 to 32 can be assigned to each devices but the same number cannot be used if the devices are connected with the same GPIB line. <@hp3457> in ASSIGN statement is the name of the I/O pass at # 722. Once the name <@hp3457> and pass <722> are corresponded in ASSIGN statement, assignment for pass will be done later only by giving name.

Line 20 and 30 are instruction for sending data to I/O port <722> which was specified by the name <@hp3457>. These have to be character strings. Meaning of symbols and numbers are different among devices and explained in manuals for each devices. For HP3457A, line 20 is to set the integration time of A/D conversion at minimum value and line 30 is to measure the DC voltage with specified precision by auto range. Line 40 is to delay the execution of the program by about one second. This is the time for instrumentation system to be stabilized in case measured object has transient response. It is not always necessary to have delay. Sometimes it takes time for a whole system including measured object to be stabilized even if measuring instrument itself has high performance. It is a matter of course for user to understand what he is going to measure.

Line 50 is to read data from I/O port 722 specified by the name of @hp3457. Since the data is output as string, it is read as string.

Line 60 <VAL (B$)> is an instruction to convert B$ to a numeric value. The value read as a string can not be used as numeric variable directly. The string and the converted value are displayed in line 60.

Line 70 is to close the I/O port, @hp3457.

Control of GPIB busline is done in this procedure for most of devices.

Before executing this program, you have to preload a binary program for controlling GPIB card. As each manufacturers has different binary program for controlling GPIB, HT-BASIC provides different binary programs for each GPIB cards. In case of NI's GPIB card,

```
LOAD BIN "GPIBNI"
```

has to be executed before the program including control for GPIB is executed. <BIN> means binary program.

"GPIBNI" file is in the directory

```
C:\Program Files\HTBwin.
```

If the default directory is different from the above, execute

```
LOAD BIN "C:\Program Files\HTBwin\GIBNI"
```

This can be also executed by keyboard. As <LOAD BIN "GPBNI"> can not be overwritten, it is not possible to add it as a command at the top of each program which includes GPIB control and to read it every time when the program is executed. It has to be loaded only once beforehand. However in case of bit input/output card which will be explained in Chapter 5, a special command for controlling card will be read as CSUB (compiled subroutine) in every program execution. You will never forget it, if you put the command for reading these CSUB at the top of the program. The difference in operation is delivered from the background of origin of HP-BASIC. For HP-BASIC control of GPIB was so basic function as four arithmetic operations. At the time when only HP's GPIB board was available, what you had to do was just to install command for HP's GPIB board it in the system program. However now various companies are selling their own GPIB control card for notebook PCs with Windows, it is not possible anymore to prepare every command corresponding each PC cards. For this reason pre-load before program execution has become necessary. If you don't use any card than NI's GPIB card, you may use auto-start function (4, 2 (3) Section) which I will explain later, and execute <LOAD BIN "GPIBNI">. With this you can just create program and run without thinking about if you use GPIB or not. Table 1 shows the list of GPIB drivers which are supported by HT-BASIC Ver. 9.x and PC cards from various manufacturers. As for corresponding Windows OS for each, please confirm for yourself.

Table 1　GPIB supported by the HT-BASIC 9.x

HT-BASIC 9.2 Name of driver	Manufacturer	PCI bus	PC card	USB bus	ISA/EISA bus
HPIBS.DW6	HP/Agilent(USA)	82350		82357	82341
	ines(Germany)	GPIB-PCI-NT+	GPIB-PCM-NT+		GPIB-ISA-NT+
	TAMS(USA)	61488		63488	
GPIBNI.DW6	NI(USA)	PCI-GPIB	PCMCIA-GPIB	GPIB-USB	
	ines(Germany)	GPIB-PCI-XL	GPIB-PCM-XL		
GPIB900.DW6	TransEra(USA)				HM900

(2) Bit Input/Output (GPIO, Bit manipulation)

Currently most of measuring instruments are equipped GPIB interface, so it is easy to control with GPIB busline, but sometimes it is also necessary to control together with self-made equipment like relay circuit or switching valve. HP's HP9000/300 series controllers had also GPIO interface other than GPIB. GPIO was the interface to control input/output of 16bit completely parallel digital signals and could be controlled by HP-BASIC language freely. Even now GPIO cards mediated PCI bus for notebook PC are being sold commercially and that can be controlled by HT-BASIC. Due to the restriction of PC cards there is no cards which can control completely parallel 16bit digital input/output. However PC cards combined digital input/output and A/D converter are being sold on the market. I believe there will be often measuring only through A/D converter in the future. In this case it is not necessary to connect voltmeter via GPIB and it is much convenient if you can make A/D conversion and control external device by digital input/output with one PC card. I will explain about bit input/output in Chapter 5 in more detail.

(3) Auto start

When you start HT-BASIC, the program will search for the file with the name of <AUTOST> in the same directory. If there is the file with the name, the program will <LOAD> the <AUTOST> file and execute automatically. It was desktop computer 9825A, to which this function added in HP's computer for the first time. This function became to be used together with the function for memorizing of running system status. If power went out during some instrumentation and came back afterwards, the computer could reboot the system based on the system status written in AUTOST file in tape cartridge and carry on the instrumentation. This is not possible for the computer today, since there is no instruction to memorize system status. However AUTOST is now used to read the binary program in advance which is necessary for execution. I will give a specific example as follows.

AUTOST program is in the directory of

C:\Program Files\HTBwin.

Open it on edit screen with "LOAD" instruction. Or whenever HT-BASIC starts　AUTOST is executed, you can view AUTOST program by clicking toolbar <Edit -> Edit Mode> on the

black screen after start-up. In this program the instruction for setting up various binary program and driver are described as comment text. If you follow the instruction (in English) and delete <!> at the top of the comment text to be loaded, it will directly become the statement. Save it by <RE-STORE> instruction in the same directory after selecting the necessary statement. Only in case AUTOST file and CSUB instruction are added at the end of the program, the program has to be loaded by <LOAD> instruction and has to be saved by <RE-STORE> instruction. If the program does not include CSUB, you can use whichever <(LOAD)/RE-STORE> or <(GET)/SAVE>.

If you want to use CSUB in the program, don't instruct it in AUTOST file but use below instruction at the first line of the program, for example

```
LOADSUB ALL FROM "c:\inesDAQ\htbasic\daghtb.csb"
```

and add CSUB at the end of the program and execute. If you want to save it, delete added CSUB then you can save it by "SAVE". In case you execute "SAVE" without deleting CSUB, it will cause error when you call the program by "GET" (<File -> Open>). However if you delete CSUB on edit screen, you can execute it by "RUN" as the program goes back to original state, which includes only text.

4.3 File Handling

In HT-BASIC it is pretty easy to save the measured data as files during experiment or to access the data in created file while programming.

(1) Creating ASCII File, writing data, reading data

In HT-BASIC it is easy to create file during programming and to save data to the created file or to read data. Input following program.

```
10   MASS STORAGE IS "d:\hp\temp"          ! set up directory
20   CREATE "test.txt",0                   ! create file
30   ASSIGN @File TO "test.txt", FORMAT ON ! open I/O pass to file
40   FOR I=1 TO 5                          ! loop
50   NPUT "numeric, characters",A,B$       ! input numeric and character by keyboard
60   PRINT A, B$
70   OUTPUT  @File;A,B$                    ! write to file
80   NEXT I
90   RESET  @File                          ! reset file pointer
100  PRINT  "------"
110  FOR I=1  TO  5
120  ENTER  @File;C,D$                     ! read from file
130  PRINT  C,D$
140  NEXT I
150  ASSIGN  @File TO *                    ! close I/O pass
160  END
```

Line 10 is instruction to specify the directory where the new file is created. Normally if you input from keyboard, you may use the abbreviation of <Mass Storage Is>, <MSI "d :\hp\temp">. The directory once declared with <Mass Storage Is> becomes the implicitly specified directory in that program unless being expressly changed.

Line 20 is to create a file with the name of <text.txt> in the directory specified in line 10. As

there is no extension <.txt> recognized in HP-BASIC, if you write some data to the file <test.txt>, this will not be written as ASCII file automatically. We should recognize this extension is used for descriptive purposes in order to use on Windows application e.g. Excel. The number after comma is to specify the number of the data to be recorded in the file as the number of records. Since there is no limit for the number of files to be stored in a file in Windows, <0> is used here.

Line 30 is to open I/O port to <test.txt> file by the name of <@File>. <FORMAT ON> indicates the data is in ASCII. If it is <FORMAT OFF>, the data is binary data. Since ASCII file is compatible with other OS, it is better to declare <FORMAT ON> whenever using ASSIGN.

From line 40 to 80 make a loop, <FOR – NEXT> for reading data by keyboard, display and writing in a file.

The writing style of line 50 is a bit different from that of INPUT instruction described in Section (2). Line 50 is to indicate the character set within <" "> and to wait for the number and character input. For number and character input, section the data with comma <,> then press <ENTER>.

Line 60 is to display the data and line 70 is to write into file. Operation of writing file will end in line 80.

Lines from 90 are to read data which were written in the file. Line 90 is to reset the file pointer.

Line 100 is to output the string bracket off with <" "> on the screen.

From line 110 to 140 are to read out the written data from the file to the real variable C and to the string variable D$.

Line 150 is to close the I/O port in different style than described in Section (4). Originally ASSIGN is the instruction to open the I/O port, however it is not possible to open multiple ports at once. Therefore instruction to open the I/O port with wild-card <*> pass can not open any port. As the result it closes all I/O port. When you execute this program you will realize that the data are read out in the same sequence as they were written. This file can be read out on MS EXCEL, since it is saved in ASCII characters.

(2) Auto-generation of file names and Array Input/Output

In the previous Section data was saved and read out one by one. It is also possible to save and read out the whole array of data. And you can put any name to the file unless it includes hyphenation character. The following program can be created with an assumption that some accident like blackout during the experiment occurs.

```
10   OPTION BASE                                  ! assign minimum value of array designator
20   DIM P$[11],Q$[8]                             ! define array of string (number) [length]
30   DIM A$(3) [18]m B$(1:5) [20], C$(1:5) [22]   ! (from:end) [length]
40   MASS STRAGE IS "d:\hp\temp/"                          ! assign working directory
50   FOR I=1    TO 3                              ! assign loop iteration, I is array designator
60       LOOP
70          P$=DATE$(TIMEDATE)                   ! dd Mmm yyyy, read the date
80          Q$=TIME$(TIMEDATE)                   ! hh:mm:ss, read the time
90          EXIT IF Q$[8;1]="0"                  ! [from, length], specify sub string
100   END LOOP
110    A$(I)="data("&QS[1,2]&"-"&Q$[4,5]&"-"&Q$[7,8]&").txt"
                                                  ! [from,end], assign sub string
120    CREATE A$(I),0                             ! create file
130    FOR J=1 TO 5
140       B$(J)=P$& "&TIME$(TIMEDATE)
150       WAIT 1                                  ! wait a second
160    NEXT J
170    ASSIGN @File TO A$(I), FORMAT ON           ! open I/O pass
180    OUTPUT @File;B$(*)                         ! output array to file
190  NEXT I
200  CAT "d:/hp/temp"                             ! output file list
210  FOR I=1 TO 3
220     PRINT "------"
230     PRINT A$(I)
240     ASSIGN @File TO A$(I); FORMAT ON
250     ENTER @File;C$(J)                         ! read out array from file
260     FOR J=1 TO 5                              ! loop for display the read array
270         PRINT C$(J)
280     NEXT J
290  NEX I
300  ASSIGN @File TO *                            ! close I/O pass
310   END
```

If you define the array by DIM in HP-BASIC, you can assign lower and upper limit of array designator at random. Line 10 is the instruction to assign the lower limit to 1, unless the range is specified. If <OPTION BASE 1> is not clearly specified, the lower limit becomes <0> (equivalent to the case if <OPTION BASE 0> is specified).

Line 20 is to define the length of character variable. If the character variable is not in array, you can use it in the program freely, and its default length is 18 letters. Actually line 20 is instruction which is not necessary to be given, however here it was defined to show that you can assign any character in P$ and Q$ in line 70 and 80. I would recommend to set it with safety allowance.

<A$(3) [18]> in line 30 is to define the three strings with up to 18 characters. <B$(1:5) [20]> means five arrays with 20 characters each which define the array designator to 1 to 5. In this way you can specify the range of the array designator in <DIM>.

Line 40 is to specify the working directory.

Line 50 to 190 are repeating loop of <FOR-NEXT>.

Line 60 to 100 are loop of repeating operation. Line 90 is to judge the end of the loop. Line 70 is to read the date. And <TIMEDATE> means number of seconds which originated from

the 48th Century B.C.. If you call up <P$=DATE$(TIMEDATE)> you can assign the string divided by space, for example <11 Apr 2004>.

With <Q$=TIME$ (TIMEDATE) in line 80 you can assign 8 characters array which indicates <time : minute : second>, for example <12:54:32>.

Line 90 is to judge to get out of the loop created by line 60 to 100. The formula <Q$[8;1]="0"> means it responds <1> when the first letter beginning from 8th character of string Q$ is "0" and the working procedure gets out from the loop. By Q$ it will get out of the loop every 10 seconds.

Line 110 is to create string and assign it to the array A$(I).

<"data("&Q$[1,2]> at the top of

```
A$(I)="data("&Q$[1,2]&"-"&Q$[4,5]&"-"&Q$[7,8]&").txt
```

is a string combining <"data("> and <Q$[1,2]> with <&>. Q$[1,2] here means the string from the first to the second character of the string Q$. Therefore if line 110 is executed, a time indicating string, <"data(hh-mm-ss).txt"> with 18 characters is eventually assigned to A$(I). In this manner there are two ways for picking up a part of a string, as line 90 and 110. In HT-BASIC the part of string picked up is called sub-string.

Line 120 is to create a file with the name of string A$(I). That means it creates a file and the file name includes time.

Line 130 to 160 are a loop to create data to be written in the file.

Line 140 is to combine TIME$(TIMEDATE) to string P$ by inserting space " ".

Since line 150 is to wait approx. one second, the date of P$ read in line 70 and the date to be read newly are assigned to

```
B$(J)=P$& " "&TIME$(TIMEDATE)
```

as combined string for example <"11 Apr 2004 12:54:33">.

Line 170 is to open the I/O pass with the name of A$(I) and to instruct data output in ASCII format.

Line 180 is to write the data in array B$(*) into the file collectively. (*) is a wild card designator to specify whole arrays.

CAT in line 200 is an instruction to display the file list of directory in order to confirm whether the file is created in a right way.

If you add an option to CAT, you can print out the file list of the directory which is specified by Directory$.

```
CAT Directory$ TO #10
```

In Windows #10 represents default printer. As Windows does not have (but DOS has) the command to print out the file list of directory, this instruction is convenient In case character arrays in print out do not look in order, select non proportional font from the toolbar at the top of edit screen of HT-BASIC.

<Tool->Device Setup->Win-Print->Properties->Select font->MS Mincho> etc.

Line 210 to 290 are the loop to read out and display each data from created three files in

order to confirm the a.m. operation.

Line 220 is to show the rule line to rule off.

Line 230 is to show the file name.

Line 240 is to open I/O pass to the file with the name of A$(I) and to instruct data input in ASCII format.

Line 250 is to read out all data to the character array C$(*) collectively.

Line 260 to 280 are to display readings.

Line 300 is to close I/O port.

If you change the part from line 130 to 160 of the program to the GPIB control program described in Section (4), you will be able to include the data from a measuring instrument. During long-term experiment, it may occur that a series of experiments can not be completed due to unforeseen blackout or failure of additional devices. If you create data files for each measurements and save in a disc, you will be able to read out the data at least until the interruption.

5. Programming of HT-BASIC 2

(Control by multi-purpose PC card (inesDAQ i250)) (Reference 3)

Here I would explain the way of use of Elan's (GB) PC card, AD132. This card has 16- channel A/D converter and 8 bit digital I/O port. The precision of A/D conversion is 12 bit and the maximum speed is 250kHz. The reason why I selected this card is that this card has satisfying specification for ordinary instrumentation and that no error has been reported with Elan's A/D converters. Hardware of this card is manufactured by Elan but driver software is supplied by ines (Germany). Its product number by ines is inesDAQ i250. Fig.3 shows inesDAQ i250 PC card and cable. In following this card (inesDAQ) will be called i250, since you will see the product number i250 in the software program. 16 channel A/D does not mean that it has 16-channel of A/D converter boards but that it can scan 16-channel analog input-/utput signals for one A/D converter board.

5.1 Overview of inesDAQ i250 PC card

As above mentioned precision of the A/D conversion is 12 bit and number of input port is 16 channels and sample rate of the A/D conversion is 305.1Hz to 250kHz. As standard mode it will read out real data, but it can also read out integer as integer mode for high-speed reading. If you use integer mode you need to convert integer data to real data later. If you use 1 channel input, you can assign up to 250kHz sample rate for both real data and integer data mode, but if you use multiple channels, 90kHz sample rate is recommendable (though 250kHz can be assigned). You can assign maximum range of analog input voltage from –10 to +10 volt and 16 ways in polarity, 8 ways in unipolarity. You can assign input channel one by one, however if you use multiple channels only contiguous channels can be assigned. Input mode for single-end grounding is 16-channel, and for differential input with two input ports is 8-channel.

Digital I/O has 8 bits of TTL level port(0V<Low<0.8V, 2V<High<5V). You can assign the I/O by 1, 2, 4 or 8 bits however there is some restriction as described later. The output cable from the card has 37 pole D-sub/plug connector at the end. There is no unified standard for digital I/O, so you can create the external circuit according to the pin number shown in Table 2.

Fig. 3 inesDAQ i250 and cable (ines Co.)

Table 2 Pin assignment of ines DAQ i250

Pin No.	Function	Differential Input	Pin No.	Function
1	A/D input 1	1+	19	DI/O 0
2	A/D input 5	1-	20	DI/O 1
3	A/D input 2	2+	21	DI/O 2
4	A/D input 6	2-	22	DI/O 3
5	A/D input 3	3+	23	DI/O 4
6	A/D input 7	3-	24	DI/O 5
7	A/D input 4	4+	25	DI/O 6
8	A/D input 8	4-	26	DI/O 7
9	A/D input 9	5+	27	+5V
10	A/D input 13	5-	28	+15V
11	A/D input 10	6+	29	-15V
12	A/D input 14	6-	30	grounding
13	A/D input 11	7+	31	trigger input
14	A/D input 15	7-	32	grounding
15	A/D input 12	8+	33	grounding
16	A/D input 16	8-	34	grounding
17	Analog grounding		35	grounding
18	Digital grounding		36	grounding
			37	grounding

5.2 Program for A/D conversion

You will be able to configure the A/D conversion of inesDAQ i250 in detail like selection of channel, conversion speed, number of data and trigger setting etc.. Below you will see an example for converting data one by one and an example for high-speed conversion.

(1) Reading one data by 1 channel

Install the driver for inesDAQ i250 and input following program. Generally it is necessary to load a number of CSUB (compiled subroutine) sequentially when you operate PC cards. Following program shows the framework for operating A/D function of i250.

```
10   LOADSUB ALL FOM   "c:\inesDAQ\htbasic\daghtb.csb"
20   OPTION BASE 1
30   DIM A(1)                                    ! array data
40   INTEGER Dummy, Hadc,Cadc
50   DIM L$[256]                                 ! definestring variable with the length 256
60   CALL Daqhtb_ioctl(Dummy, "( fctn init ) ")  ! initialize PC card
70   CALL Daqhtb_open(Hadc,"i250 ADC", 0)        ! open A/D function by real mode(0)
80   L$="( ioa ( timeout 10000 fsmpl 100000 chan 1 iomode single trigmode off
samples 1 range [ -10 10 ] ) )"                  ! defining statement of A/D mode operation
90   CALL Daqhtb_ioct1(Hadc,L$)                  ! write A/D mode defining statement to PC card
100 CALL Daqhtb_read(Cadc,Hadc,A(*))             ! read data (only one)
110 PRINT A(1)                                   ! print data to be read
120 CALL Daqhtb_close(Hacd)                      ! end PC card operation
130 END
```

Line 10 is to read out CSUB (compiled subroutine) for operating inesDAQ card. When this instruction is executed, CSUB necessary for operating ines DAQ is added at the end of the program.

Line 20 is to instruct the argument of array starts from 1.

Line 30 is to declare array of real for data acquisition. Be aware that element count of array A is one.

Line 40 is definition of integer variable. <Dummy>, (Hadc>, and <Cadc> are the return variables for operating inesDAQ card and you can name them at random.

Line 40 is to definestring variable with the length of 256 characters.

Line 60 is to initialize the PC card.

```
CALL Daqhtb_ioctl(Dummy, "(fctn init)")
```

Dummy in argument is the variable to be added for initializing inesDAQ card and is not used in the program. The second argument, the string<"(fcn init)"> is defining statement for initializing the PC card and it has to be written in this way. You will be able to specify the size of FIFO buffer (1Mbyte by default) and the number of sample(8192 by default, half of the internal buffer). till the IRQ (interrupt request) occurrence

Line 70 is to load the A/D conversion function of the PC card. <"i250 ADC"> in argument means using ADC function (A/D conversion) of i250. The value of the handle (the number for recognizing PC card's function) specified by <"i250 ADC"> returns to integer variable <Hadc>. The variable <Hacd> is always used as argument henceforth until the end of operation every time when CSUB is invoked. <0> indicates that the operation shall be done in real-value. Assign <1>, in case reading out by integer value like high-speed data acquisition. (Which will be described later.)

Line 80 is the string to define the contents of operation of A/D conversion. In the example it is shown in two lines due to the length of it, however you may not insert line feed by pressing <ENTER>.

If you want to show the program clearer in more than two lines for modification later, you may divide it in some strings. You will be able to modify line 80 as below, if you define string

variable <P$[128], Q$[128] in <DIM> in line 50. Be aware that each assignment shall be divided by inserting space.

```
50   DIM L$[256],P$[128],Q$[128]
  . . . . . .
75   P$=( ioa ( timeout 10000 fsmpl 100000 chan 1 iomode single"
76   Q$="trigmode off samples 1 range [ -10 10 ] ) ) "
80   L$=P$&" "&Q$
```

< ioa > indicates the I/O operation of analog signals (A/D conversion in this case) and the operation in detail is specified in < ( ) > following.

```
timeout 10000          ! timeout of operation is 10000msec.
fsmpl 100000           ! sampling frequency is 100kHz
chan 1                 ! A/D conversion of voltage signal at channel #1
iomode single          ! Input mode of voltage signal is single-end grounding
trigmode off           ! no trigger mode used
sample 1               ! read only single data
range   [-10 10]       ! range of input voltage is –10~+10V
```

Insert space between each character and value. You can assign other functions than these, however the value by default will be given where you did not specify. There are two ways for input mode of voltage signal. You can use 16 channels for single end and 8 channels for differential input, which uses two channels at once. You will be able to assign trigger mode in detail. You can assign the range of input voltage both for positive and negative (16 ways) and for only positive (8 ways).

Line 90 is to write the defining statement in line 80 into the function specified by the handle number <Hadc>.

Line 100 is to read out data by the amount specified by <A(*)>. In this example the element is 1 for A(*) to read only one data, however you may put any number if it is 1 or more. Number of data to be read is in <Cadb>.

Line 110 is to print out the data to display.

Line 120 is to end the operation of PC card. If you start to assign a new operation with inesDAQ, you necessarily have to end the previous operation.

Line 130 is to end the program.

Below you will see the procedure from initialization to end by invoking five CSUB with inesDAQ.

1. initialize PC card
2. input A/D conversion or setting up digital I/O function
3. specify the contents of operation in detail
4. input or output of data

4’ convert integer data to real value

5. end operation of PC card

4' is necessary for high-speed data loading which will be explained in Section (4) (ii) Data acquisition in integer mode

(2) Read out data one by one by multiple channels

The program for reading out data one by one by multiple channels is as below. The flow is same as by 1 channel.

```
10   LOADSUB ALL FROM   "c:\inesDAQ\htbasic\daghtb.csb"        ! read CSUB
20   OPTION BASE 1
30   DIM A(2),L$[256]
40   INTEGER Dummy,Hadc,Cadc
50   CALL Daqhtb_ioctl(Dummy,"( fctn init) ")
60   CALL Daqhtb_open(Hadc,"i250 ADC",0)
70   L$="( ioa ( timeout 10000 fsmpl 100000 chan 102 iomode single trimode off samples 2 range
     [ -10 10 ] ) )"
80   CALL Daqhtb_ioctl(Hacd,L$)
90   CALL Daqhtb_read(Cadc, Hadc,A(*))
100  PRINT A(1),A(2)
110  CALL Daqhtb_close(Hadc)
120  END
```

Line 30 is to define A(*) which has two elements and is for reading data.

Line 50 is to initialize the PC card.

Line 60 is to open function of A/D conversion with real number mode <0>.

Line 70 is to specify the operation. In the example program it is shown in two lines due to the length of it, however you may not insert line feeds by pressing <ENTER>.

```
fsmp1 100000          ! sampling frequency is 100kHz
chan 102              ! A/D conversion of signals from channel 1 to channel 2
samples 2             ! two data to be read
trigmode off          ! trigger mode off
```

If you use multiple A/D input channels, you can not select the channel to be measured randomly due to the restriction of hardware of inesDAQ. You have to select successive multiple channels. If the channel number at the end is single-digit, add prefixed zeros as needed to make a two-digit number. It is not recommendable to select trigger mode in data acquisition with multiple channels, since it will be impossible to recognize from which channel it starts with reading data. However if you use the first A/D input channel of multiple channels for trigger exclusively and make it trigger above the range of input voltage, the following data is to input in the second channel. And if you throw out the data in the first channel, multi-channel A/D input with trigger will be realized. The sampling frequency for two channels is 100kHz in total.

Line 80 is to write operation defining statement L$ to the function specified by the PC card number Hadc.

Line 90 is to read out each one data from both channel 1 and channel 2. The data in channel 1

is read out to A(1) and the data in channel 2 is to A(2).

Line 100 is to print out.

Line 110 is to end PC card operation.

If you want to read out data continuously at regular intervals, put the operation described in (1) or (2) into loop (See Chapter 6) and repeat it. If you have the data to be written into file as described (1) and (2) in Section 4.3, almost unlimited amount of data can be loaded.

(3) High-speed data acquisition with 1 channel

Sometimes you may need to acquire data with high-speed when some event occurs and you need to monitor its change, though you usually monitor the data one by one at regular intervals. Below you will see an example program for this. Sampling up to 250 kHz with 1 channel is available.

```
10   LOADSUB ALL FROM "c:\inesDAQ\htbasic\daghtb.csb"
20   OPTION BASE 1
30   DIM A(1024) ,L$[256]
40   INTEGER Dummy, Hadc,Cadc
50   CALL Daqhtb_ioct1(Dummy,"( fctn init) ")
60   CALL Daqhtb_ioeb(Hadc,"i250 ADC",0)
70   L$="( ioa (timeout 10000 fsmpl 100000 chan 1 iomode single trigmode level.gt triglevel –10.0
      samples 1024 range [ -10 10 ] ) )"
80   CALL Daqhtb_ioctl(Hadc,L$)
90   CALL Daqhtb_read(Cacd,Hacd,A(*))          ! 64 (2^6) < A(*)=2^n , 16348(2^14)
100  ----------------------------
110  CALL Daqhtb_close(Hadc)
120  END
```

Line 30 is to define array A(*) which is to be data buffer.

Line 60 is to open function of A/D conversion with real number mode <0>.

Line 70 is to specify the operation. In the example program it is shown in two lines due to the length of it, however you may not insert line feeds by pressing <ENTER>.

```
fsmp1 100000              ! sampling frequency is 100kHz
trigmode level.gt         ! trigger mode "when signal value is more than specified level"
triglevel –10.0           ! trigger level is –10V
pretrig 0                 ! pre-trigger is not to be loaded
samples 1024              ! 1024 data to be loaded
```

Length of loaded data shall be set between 64(2^6) and 16348(2^14) , it shall be set in 2^n. If trigger mode is assigned (except the case <trigmode off>), A/C conversion is constantly working, therefore previous data will be monitored from the moment trigger is on. With <pretrig> you can specify how far the data shall be loaded.

Line 80 to write operation defining statement for A/D conversion mode.

Line 90 is to read to array A(*). Reading will be stopped when the buffer becomes full.

In the example program, monitoring voltage from channel #1 sends out trigger signal.

If you want to set the trigger voltage out of the range assigned in line 70 (less than –10V or

more than +10V), apply trigger signal at trigger input pin #31 shown in Fig. 5.

(4) High-speed data loading with multiple channels

There are two ways for high-speed data loading with multiple channels, one is loading data into real number array the other is loading data into integer number array. In case of loading data into real number array, it takes time to transform the result of A/D conversion to real number. It is possible to do the high-speed data loading into integer number without transformation into real number, however the data in integer number has to be transformed in real number later anyway. Sampling speed up to 90 kHz is recommendable for data loading with multiple channels.

(i) Data acquisition in real number mode

You will see an example program for data input from A/D port by channel 1 and channel 2.

```
10   LOADSBU ALL FROM "c:\inesDAQ\htbasic\daghtb.csb"
20   OPTION BASE 1
30   DIM A(1024),Dat(512,2), L$[256]              ! array definition
40   INTEGER Dummy,Hadc,Cadc
50   CALL Dabhtb_ioctl(Dummy,"fctn init)")
60   CALL Dabhtb_open(Hadc,"i1250 ADC",0)
70   L$="( ioa ( timeout 10000 fsmpl 90000 chan 102 iomode sngle trigmode off samples 1024 ) ) "
80   CALL Daqhtb_loctl(Hadc,L$)
90   CALL Daqhtb_read(Cdac,Hacd,A(*))             ! read out to A(*)
100  CALL Daqhtb_close(Hadc)
110  FOR J=1 TO Cadc/2.                           ! sorting data to each channel
120  Dat(J,1)=A(2*J-1)
130  Dat(J,2)=A(2*J)
140  NEXT J
150  END
```

Line 30 is to declare the array for the data to be read out. Acquire 1024 data into array A(*) through channel 2. After that each 512 data are sorted into array Dat(*) for each channels.

Line 50 is to initialize the PC card.

Line 60 is to open the A/D conversion function of the card with real number mode <0>.

Line 70 is to specify the operation. In the example program it is shown in two lines due to the length of it, however you may not insert line feeds by pressing <ENTER>.

```
fsmp1 90000          ! sampling frequency is 90kHz
chan 102             ! scan A/D input of successive channel 1 to channel 2
iomode single        ! single-end grounding input
triggmode off        ! trigger mode off
samples 1024         ! total 1024 data to be loaded by two channels
```

The channel number to be used for A/D input has to be successive. You can assign 16 channels for single-end grounding input and 8 channels for differential input. You will see the connection of input cable in the Table 1.

Line 80 is to send the contents of operation to PC card.
Line 90 is to read out data. As A/D converter sends the data into arrays switching channels by given sample rate, total 1024 data are read out in A(*) of two channels alternately. Cadc is the number of data to be loaded.
Line 100 is to end the operation of the card.
Line 110 to 140 are loop for sorting data loaded into A(*) into channel 1 and channel 2. <Cadc/2> is the number of data in each channel.

(ii) Data acquisition in integer mode

It takes time to transform the result each time into real number in real number mode measurement. Therefore to perform high-speed measurement, data are collected as binary signal at first, then after the completion of measurement they are to be transformed into real number. As integer data of PC card is 32bit (4byte), while integer data of HT-BASIC is 16bit (2byte) in integer mode, it is necessary to provide the array which has twice as many elements as the data required. Below you will see an example program for this. Here the number of data to be measured is 256 for each channel, but you can assign it with 2^n as you like.

```
10   LOADSUB ALL FROM  "c:\inesDAQ\htbasic\daghtb.csb"
20   OPTION BASE 1
30   INTEGER A(1024)                          ! declare integer number array
40   DIM R(512),Dat(256,2)                    ! declare real number array
50   INTEGER Dummy,Hadc, Cadc                 ! declare return variable
60   DIM L$(256)                              ! declare string variable for assigning operation
70   CALL daqhtb_ioctl(Dummy,"(fctn init)")   ! initialize PC card
80   CALL daqhtb_open(Hacd,"i250 ADC",1)      ! read out in the mode of integer number (1)
90   L$="( ioa ( timeout 10000 fsmp 100000 chan 102 iomode single
     trigmode off sample 1024 ) ) "
100  CALL Daqhtb_ioctl(Hacd,L$)
110  CALL Daqhtb_readi(Cadc,Hadc,A(*))        ! read data into integer number array
120  CALL Daqhtb_demangle(R(*),A(*),L$,0)     ! transform integer number input to real number
130  CALL Daqhtb_close(Hadc)
140  FOR J=1 TO Cadc/2.                       ! sort real number data for each channel
150       Dat(J,1)=R(2*J-1)
160       Dat(J,2)=R(2*J)
170  NEXT J
180  END
```

Line 30 is to declare the array for reading binary data after A/D conversion directly as integer value. The PC card outputs 32bit data as integer value, but HT-BASIC handles it as 2byte (16bit). Therefore two array elements are necessary to read one integer value. A(1024) means array for 512 integer value.
Line 40 is to declare real number array. R(512) is the array to store 1024 data read into A(*) after transformation into real value. Dat(256,2) is the array to collect 256 data to be sorted for each channels.

Line 80 is to assign integer variable <1> for A/D conversion of PC card.
Line 90 is to specify the contents of operation. In the example program it is shown in two lines due to the length of it, however you may not insert line feeds by pressing <ENTER>.

```
fsmp1 100000          ! sampling frequency is 100kHz
chan 102              ! scan A/D input of successive channel 1 to channel 2
iomode single         ! single-end grounding input
triggmode off         ! trigger mode off
samples 1024          ! total 1024 data to be loaded by two channels
```

Line 110 is to read data into A(*), and as mentioned two elements of A are equivalent to one integer data of the PC card. That means 512 integer data are actually read out in array A with samples 1024.
Line 100 is to write the contents of operation to PC card.
Line 110 is to read out data in array A(*). In real number mode the program was <Daqhtb_read>, however in integer mode it becomes <Daqhtb_readi>.
Line 120 is a necessary function in integer mode which is to convert integer data into real number data. The number of data in A(*) is 1024, however the number of elements in R(*) is 512, since 512 data are actually read out.
Line 130 is to end the operation.
The loop from line 140 is to sort the date for each channel.

5.3, Bit Input/Output Program

Digital I/O function of inesDAQ i250 are to specify the function of 8 bit-port (whether input or output), to write data to be output and to read out data to be input. Fig. 6 show how to specify each bit-port. Due to the size constraint of the card, input/output for each port cannot be assigned freely, but it has to be per 1 or 2 or 4 at a time as shown in Table 2. I believe it is practically convenient .
Below you will see the example for bit output and bit input.
(1) Bit data output

```
10   LOADSUB ALL FROM   "c:\inesDAQ\htbasic\daghtb.csb"
20   OPTION BASE 1
30   DIM D(1)
40   INTEGER Dummy, Hdio, Cdio
50  CALL Daqhtb_ioctl(Dummy,"(fctn init)"      ! initialize card
60  CALL Daqhtb_open(Hdio,"i250 DIO",1)        ! specify digital I/O function
70  CALLDaqhtb_ioctl(Hdio,"( iod ( dir 255 ) )") ! specify digital I/O port
80  INPUT P                                    ! key in decimal number
90  D(1)=BINAND(P,255)                         ! leave low-order 8bit in binary-coded form
100 CALL Daqhtb_write(1,Hdio,D(*))
110 CALL Daqhtb_close(Hdio)
120 END
```

Input following program. The order for loading CSUB (compiled subroutine) is the same as in

A/D conversion.

Line 10 is to read out CSUB for controlling PC card collectively.

Line 20 is to assign the designator of array to start from 1 (not from 0).

Line 30 is the array D(*) for giving bit output value which consists of one element.

Line 60 is to open digital I/O function of inesDAQ i250 with integer mode.

Line 70 is to assign 8bits of the digital I/O port for output according to Table 2.

Line 80 is to input numerical number by keyboard.

Line 90 is to pick out lower-order 8bit from input value and assign it to D(1). BINAND(A,B) is the formula for picking up AND per a bit from the value A and B in 16-bit-form.

Line 100 is to write one data.

Line 110 is to end the operation.

(2) Bit data input

This is the same procedure as in bit data output except the line 70 which specifies bit data output.

```
11   LOADSUB ALL FROM   "c:\inesDAQ\htbasic\daghtb.csb"
21   OPTION BASE 1
31   DIM D(1)                                    ! array for reading out bit data
41   INTEGER Dummy, Hdio, Cdio
50  CALL Daqhtb_ioctl(Dummy,"(fctn init)"       ! initialize card
60  CALL Daqhtb_open(Hdio,"i250 DIO",1)         ! call digital I/O function in integer mode
70  CALLDaqhtb_ioctl(Hdio,"( iod ( dir 255 ) )") ! assign 8 bit port for input
80  ----------
90  CALL Daqhtb_read(Cdio,Hdio,D(*))            ! read out data in array D
110 CALL Daqhtb_close(Hdio)                     ! end of operation
120 END
```

Line 70 is to assign whole 8-bit port as input port.

Input for each bit has to be given with the TTL level (0V<Low<0.8V, 2V<High<5V). Fig.7 shows the example of external actuator of GPIO output. With this board through GPIO line driving 16 external relays, input from 2-channel and 16-bit digital counter, driving relays with 100 VAC and controlling random out signals are possible.

Fig.8 is an external test board of inesDAQ i250. With this board 16-channel analog input, 8-bit digital input-output can be tested.

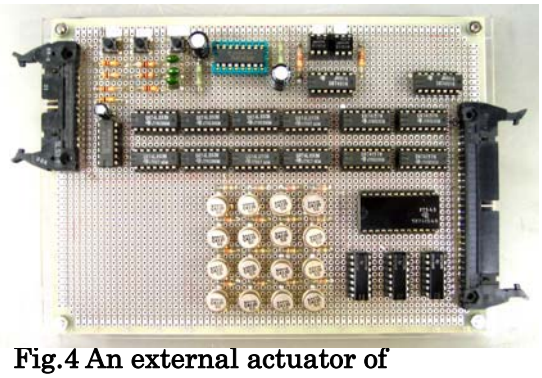


Fig.4 An external actuator of GPIO output

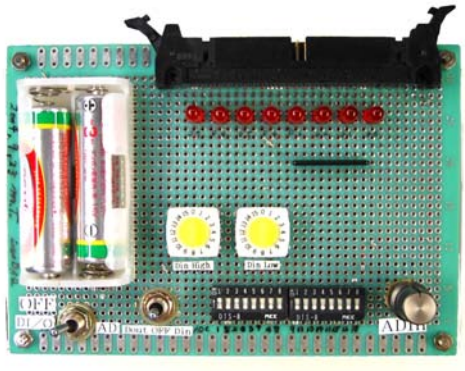


Fig.5 An external test board of inesDAQ i250.

.

Table 3 Assignment code of inesDAQ digital port(I:input, O:output)

hexadecimal format	0x0	0x1	0x2	0x3	0xC	0xD	0xE	0xF	0xF0	0xF1	0xF2	0xF3	0xFC	0xFD	0xFE	0xFF
Decimal format	0	1	2	3	12	13	14	15	240	241	242	243	252	253	254	255
DIO 0	I	O	I	O	I	O	I	O	I	O	I	O	I	O	I	O
DIO 1	I	I	O	O	I	I	O	O	I	I	O	O	I	I	O	O
DIO 2	I	I	I	I	O	O	O	O	I	I	I	I	O	O	O	O
DIO 3	I	I	I	I	O	O	O	O	I	I	I	I	O	O	O	O
DIO 4	I	I	I	I	I	I	I	I	O	O	O	O	O	O	O	O
DIO 5	I	I	I	I	I	I	I	I	O	O	O	O	O	O	O	O
DIO 6	I	I	I	I	I	I	I	I	O	O	O	O	O	O	O	O
DIO 7	I	I	I	I	I	I	I	I	O	O	O	O	O	O	O	O

6. Programming of HT-BASIC 3 (program construction (loop & branch, calculation))

In preceding Section, the feature of interactive program was focused, which can not be realized easily by C language. In this Chapter the way of general programming is overviewed.

(1) Loop and Branch(For~NEXT, IF~THEN, LOOP, WHILE, SELECT~CASE,)

(i) FOR~NEXT

```
10    FOR I=1 TO 1000
20          FOR J=2*PI TO 0 STEP –PI/1100
      ------------------
80          NEXT J
90    NEXT I
```

You can specify increase and/or decrease with STEP from 2pi to 0 by –PI/100 as in line 20. If you don't specify any, it will be regarded as 1.

(ii) IF~THEN (~ELSE~END IF)

In conditional expression or logical expression if the evaluation result is true, it replies <1> (right).

```
10       IF J2=K THEN 1200
       -----
30       IF X=Y THEN Y=Z*Z
       -----
40       IF A<0 THEN
50            GOTO END
60       ELSE
       --------
80    END IF
90    END:END
```

In line 10 if the logical expression <J2=K> is true, the flow of the program branches to line 1200.

If the conditional expression in line 30 line is true, Y=Z*Z is executed.

In line 40 to 80 if A<0 is true, the flow of the program branches to line 100 which has the

name of END. If it is false, the program below ELSE is executed. You can name each line at random. After the name of the line you should mark with colon<:>.

(iii) LOOP~EXIT IF~END LOOP

```
100    LOOP
              ----
170    EXIT IF J=5 OR A$.B$
              ----
190    END LOOP
```

Line 100 to 190 are executed repeatedly, and if the logical expression in line 170 is true, it will get off the LOOP. The conditional judgement is done on the way of loop.

(iv) WHILE~END WHILE

```
100   WHILE x<1000
200   END WHILE
```

If the logical expression in line 100 is true, it repeats the program up to line 200. Conditional judgement is done at the top of the loop.

(v) REPEAT~UNTIL

```
60      REPEAT
    -----
80      UNTIL X=10
```

It returns to REPEAT and repeats the processing until the logical expression in line 80 becomes true. Conditional judgement is done at the end of the loop.

(vi) SELECT~CASE~END SELECT

```
 10    SELECT Options
 20    CASE “B”
 30          A=1
 40    CASE “0” TO “9”,“y”,”n”
 50          A=2
 60    CASE ELSE
 70          A=0
 80    END SELECT
```

The string variable Option$ is selected in line 10 and the contents are compared by CASE statement in line 20 and 40. If the logical expression Option$=”B” is true, it executes line 30 and if it does not correspond to any CASE statement, it executes line 70. If you SELECT numerical variable, you need to use the logical expression with value.

(2) Subprogram and function 1 (SUB, COM, FN)

(i) Without having common variables with main program

```
10    CALL Mysub                ! call subprogram
20    END                       ! end main program
30    SUB Mysub                 ! definesubprogram
40          PRINT “In My SUB”
50      SUBEND
```

Line 10 to 20 are main program and call a subprogram, Mysub. Subprogram is written after <END> statement which indicates the end of main program.

(ii) Having common variable with main program (COM)

```
10    COM /Test/A$(2)[20],P                    ! declare common variable with name
20        A$(1)=”He”
30        A$(2)=”11o”
40        P=7
50    Subit                                    ! call subprogram
60    END                                      ! end main program
70    SUB Subit                                ! definesubprogram
80    COM /Test/C$(*),R                        ! declare common variable
90        PRINT “A$(*)=”C$(1)&C$(2),P=”:R      ! display combined characters
100   SUBEND
```

<COM> statement in line 10 is to define he common variable. </Test/> is the name for the defined variable as a block.

Line 50 is to call subprogram. Subprogram should be normally called by CALL statement like <CALL Subit>, you can leave it off for calling subprogram only.

In line 80, a variable name different from main program is used as an example, however it would be better to use the same name to avoid confusion, if you don’t have any special reason for it.

(iii) With argument

```
10    DIM A$(2)[20]
20    A$(1)=”He”
30    A$(2)=”11o”
40    P=7
50    Subit(A$(*),P,Q)                  ! sending data by argument
60    PRINT Q
70    END
80    SUB Subit(C$(*),R,S)              ! receiving data by argument
90          PRINT “A$(*)=”;C$(1)&C$(2),”P=”;R
100         S=R*R                       ! reply result of calculation by argument
110   SUBEND
```

The sorting order of variables in the line 80 should be the same as in line 50. If you create SUB program, you should add it at the end of the main program line. If you try to define the argument C$(*) in line 80 as in line 10, it causes error.

(3) Subprogram and Function 2 (GOSUB, FN, CSUB)

(i) Subroutine in context (GOSUB)

In HP-BASIC a subroutine can be placed in main program. You have to be careful for the flow of operation in this case, as the flow branches by GOSUB instruction and returns by RETURN statement to the next line after the GOSUB instruction.

```
10        Y=3
20        Z=4
30        GOSUB Calc_x
40        PRINT "X = ";X
50        STOP
60 Calc_x:X=Y*45/Z
70        RETURN
80        END
```

In line 30 it jumps to the subroutine in line 60 and return by the RETURN statement in line 70. If there is no STOP instruction in line 50, it will execute from line 60 to 70 once again. The program is in idle state by <STOP> instruction as by <END> instruction. It can not <CONTINUE>.

(ii) Function (DEF FN)

You can define any function and use it in the program.

```
10    PRINT "5 + 8 =";FNAdd(5,8)          ! quote the function
20    END                                 ! end main program
30    DEF FNADD(A,B)                      ! define function
40    RETURN A+B
50    FNEND
```

Line 30 to 50 are the definition of function. The definition should be added after the main program. The name of function should start with FN. In this case no CALL statement is necessary.

(iii) Compiled subprogram (CSUB)

In HT-BASIC many mathematical function are provided as compiled subprogram. Below is an example for this.

```
10     LOADSUB ALL FROM "C:\Program Files\HTBwin\mathlib/\Mathlib.hts"

100    A=FNErf(x)

300    END
```

X, a function value of argument FNErf(X) is substituted to A in line 100. FNErf(X) is an error function. For using the compiled subprogram, the subprogram file has to be loaded in advance. File list, explanation of function and name of subroutine file which should be loaded for the compiled subprogram of the mathematical function are placed in a file in default directory as below.

C:\Program Files\HTBwin\mathlib\Mathlib.hlp

As especially in the 10th line of the above program

C:\Program files\HTBwin\mathlib\Mathlib.hts

includes all of CSUB program for mathematical function, you can use all the mathematical function to be provided if it is loaded. But if the program has CSUB subroutine added at the end, it can not be saved as ASCII file. It needs to be saved by RE-STORE and call up by LOAD.

(4) Matrix Calculation (MAT)

Matrix operation and vector operation can be easily done with MAT operator. Below you will see a few examples.

```
 10  OPTION BASE 1                  ! assign minimum value of array designator
 20  DIM A(3,3),B(3,3),C(3,3)       ! define 3:3 matrix
 30  DIM V1(3),V2(3)                ! define 1:3 vector
 40  DATA 1,3,2, -1, -2, -1,2,4,3   ! matrix element
 50  RESTORE 40                     ! specify DATA to be read with the next READ statement
 60  READ A(*)                      ! read DATA to A(*)
 70  CALL Subdispl ("A(*) = ",A(*))  ! display element of A(*) by subprogram
 80  MAT B=INV(A)                   ! assign inverse matrix of A(*) to B(*)
 90  Subdispl("INV A(*) = ",B(*))   ! display element of B(*)
100  MAT C=B*A                      ! assign product of B(*) and A(*) to C(*)
110  Subdispl("INV(A)*A = ",C(*))   ! display element of C(*)
120  MAT v1=B(1,*)                  ! assign element of B'1,*) to vector V1(*)
130  Subdispl2("INV(1) = ",V1(*))   ! display element of V1(*) by subprogram
140  MAT V2=V1*A                    ! assign product of vector V1(*) and A(*) to v2(*)
150  Subdisp2("INV(1)*A = ",V2(*))  ! display element of V2(*)
160  END                            ! end main program
170  SUB Subdispl (P$,Q(*))         ! define subprogram 1
180       PRINT " "                 ! display (print) blank line
190       PRINT P$
200       FOR I=1 TO 3
210            PRINT USING "3(10D.DD)";Q(I,1),Q'I,2),Q(I,3)        ! formatted display (print)
220       NEXT 1
230  SUBEND
240  SUB Subdisp2(P$,Q(*))          ! define subprogram 2
250       PRINT " "
260       PRINT P$
270          PRINT USING "3(10D.D)";Q(*)                         ! formatted display (print)
280  SUBEND
```

Line 40 is a DATA statement which will provide data in the program. Data shall be marked with <,>.

Line 50 is to specify the DATA statement to be read by the READ statement in the next line.

Line 60 is to read DATA statement in matrix A(*). The order to be read from DATA statement is appointed from the left toward the end. For example the data are read in the matrix P(2,3,2) as below.

P(1,1,1). P(1,1,2). P(1,2,*). P(1,3,*). P(2,1,*). … P(2,3,*)

Line 70 is to call out the subprogram to display the contents of matrix A(*).
Line 80 is to assign the inverse matrix of A(*) to B(*), and line 100 is to assign the product of B(*) and A(*) to C(*). If you add MAT at the top of a normal assign statement it becomes matrix assign statement.
Line 90 and 110 are to call out the same subprogram as in line 70. If the line is only for call out subprogram, CALL can be omitted.
Line 120 is to assign the element of B(1,*) to vector V1(*) – one dimensional array. If you want to assign one array to another array, you have to be careful that the number of elements should correspond each other.
Line 130 and 150 are to call out subprogram to display the element of vector.
Line 140 is to assign the product of vector V1(*) and matrix A(*) to V2(*). If the one dimensional array, V1(*) is placed left in multiplication, it is processed as row vector, and if it is placed right, it is processed as column vector.

```
MAT V2=V1*A      ! V1(*) is row vector, the result is one dimensional array of row vector
MAT V2=A*V1      ! V1(*) is column vector, the result is one dimensional array of column vector
```

Line 170 to 240 are subprogram to display the matrix and line 240 to 280 are subprogram to display the vector. Both are called out by (P$,Q(*)) with character string and array data as argument.
Line 210 and 270 are formatted display statement.

```
PRINT USING #3(10D.DD)!;Q(I,2),Q(I,3)
```

<3"(10D.DD)"> instructs to repeat displaying <integer part 10 digits, decimal point, decimal part 2 digits> three times. The data to be displayed is placed after <;>. Line 270 is to assign data string at a time by Q(*). You can instruct in detail of data processing in HP-BASIC. With regard to format you will see detailed explanation in the <IMAGE> Section of its reference manual. You will also see some examples for the usage of each instruction in example folder.

(5) Logical expression

In logical expression if it is true it replies 1 and if it is false it replies 0. With this you will be able to write a mathematical formula, for example

```
PRINT (A=B) OR (C>D)*PI
```

If both (A=B) and (C>D) are false, 0 will be output (display or print) and otherwise PI(circle ratio) will be output.

(6) Bit manipulation

Below is the instruction for bit manipulation for the numerical value in 16-bit count.

```
BINAND(A,B)   ! take AND of two 16-bit count values per bit
BINCMP(A)     ! reverse each bit of a 16-bit count value
BINEOR(A,B)   ! take exclusive disjunction of two 16-bit count values per bit
BINIOR(A,B)   ! take OR of two 16-bit count values per bit
BIT(A,5)      ! return specified bit of a 16-bit count value
ROTATE(A,2)   ! rotate the bit sequence of a 16-bit count value in clockwise or counterclockwise direction
```

5 in BIT instruction indicates bit position. You can specify it by 1 to 16. If 2 in ROTATE instruction is positive, it rotates in clockwise direction for the numerical value. Bits running off from the end will go back to another end. If 2 in SHIFT instruction is positive, it shifts to right for the numerical value. Bits running off from the end will disappear and 0 will appear at another end.

(7) Complex number

Complex number is processed as a pair of real numbers (8byte*2=16byte).

```
COMPLEX A,B              ! declaration statement of complex variable
A=CMPLX(2,1)             ! A=2+1
B=CMPLX(5,1/5)*A         ! B=(5+i1/5)*(2+i)=9.8+i5.4
```

7. Programming of HT-BASIC 4 (Graphics)

HP-BASIC has various graphical instructions. Sample programming for graphics will be shown in this Chapter.

7.1 Graphical output

(1) Make a graph of a function

In this Section an example for making graphs of function on the plane of X-Y coordinates will be explained and the procedure of graphical output will be shown.

```
10    GINIT                              ! initialize parameter for graphical output
20    PLOTTER IS CRT,"INTERNAL"          ! direct the output of graphics to display
30    Xlow=1                             ! minimum value of X-axis of the graph to be output
40    Xhigh=1                            ! maximum value of X-axis of the graph to be output
50    Xlength=Xhigh-Xlow                 ! length of value on X-axis of the graph to be output
60    Ylow=-1                            ! minimum value of Y-axis of the graph to be output
70    Yhigh=1                            ! maximum value of Y-axis of the graph to be output
80    Ylength=yhigh-Ylow                 ! length of value on Y- axis of the graph to be output
90    VIEWPORT 40,120,20,90              ! specify range of the graphics on output device
100   WINDOW Xlow, Xhigh,Ylow,Yhigh      ! define user-specified value in range of the graphics
110   FRAME                              ! frame range of the graphics by solid line
120   AXES Xlength/20,Ylength/20,0,0,5,5,2   ! plot the axis of coordinate
130   CLIP OFF                           ! cancel limit of the range of graphics
140   F$="F(X)=X*X*X"                    ! assign character string
150   MOVE 0,Yhigh                       ! move graphical pen to specified axis
160   LORG 4                             ! specify label position corresponding to axis.
170   LABEL F$                           ! output character string to label
180   CLIP ON                            ! set limit of range of graphics
190   PEN 6                              ! select pen
200   FOR X=Xlow TO Xhigh STEP Xlength/20   ! loop for plotting lines
210       PLOT X,X*X*X                   ! plot lines
220   NEXT X                             !
230   CSIZE 3                            ! specify font size
240   LORG 1                             ! specify label position corresponding to axis
250   PEN 2                              ! select pen
260   FOR X=Xlow TO Yhigh STEP .5        ! loop for plotting scale of X-axis
270       MOVE X,0                       ! move graphical pen to specified axis
280       LABEL X                        ! output numerical label
290   NEXT X
300   FOR Y=Ylow TO Yhigh STEP .5        ! loop for plotting scale of Y-axis
310       MOVE 0,Y                       ! move graphical pen to specified axis
320       LABEL USING "DD.D";Y           ! output numerical label
330   NEXT Y
340   END
```

Line 10 is to initialize the parameter for plotting graphics. The parameter includes kind of graphical pens, font size in LABEL, positioning of characters in axis of coordinate and so on.
Line 20 is to direct the graphical output to the display. In place of this, if you use

```
PLOTTER IS 712,"HPGL"
```

The graphical output is directed to the plotter #12 connected with GPIB cable. "HPGL" (Hewlett Packard Graphic Language) is the language for graphics output developed by HP.
Line 30 to 80 are to assign some commands for plotting a graph, like range of X-axis and Y-axis to arbitrary variables.
Line 90 is to specify the range of graphical output in a plotting device, like PC screen. If VIEWPORT is executed, the subsequent output range is limited to the range specified by VIEWPORT. The range is specified setting Y-axis as the maximum length of the screen and labeling Y-axis with 100 and then labeling X-axis with the scale according to the width of the screen. In HP-BASIC this scale is called GDU(Graphic Display Unit). For example normally the screen of a PC is landscape-oriented, 100 for Y-axis and bigger than 100 for X-axis. If you want to confirm it, execute

```
RATIO
```

<RATIO> replies ratio of the length of X-axis to Y-axis for the range available for plotting, e.g. 1.4. In case the ratio by RATIO for the screen of a notebook PC is 1.4, the drawing space is 100 on Y-axis and 140 on X-axis and specified as

```
VIEWPORT laterally leftmost, laterally rightmost, longitudinally lower limit, longitudinally upper limit
```

Thus line 90 assigns to draw in a rectangle with lateral 40~120 and longitudinal 20~90 within the whole screen with the range of X-axis approx.140 and Y-axis 100. You will be able to confirm it by executing RATIO through keyboard beforehand. If you use If RATIO is used in the program as below

```
Xscale=100*RATIO
VIEWPORT 0.3*Xscale,0.7*Xscale,20,80
```

drawing space is always specified from 30%~40% of X-axis and 20~80 on Y-axis.
Line 100 is to define the axis of coordinate to be used for drawing within the range specified by VIEWPORT. According to the variables defined in line 30 to 80,

```
WINDOW Xlow,Xhigh,Ylow,Yhigh
```

it defines left end –1, right end 1 and lower end –1, upper end 1.
Line 110 is to frame the drawing range.
Line 120 is to plot the axis of coordinate. Parameters are set as below.

AXES Xt, Yt, 0x, 0y, Cx, Cy, S
and Xt : distance between scales on X-axis
Yt : distance between scales on Y-axis
0x : value of X-axis when X intersects Y
0y : value of Y-axis when Y intersects X
Cx : distance between key scales on X-axis
Cy : distance between key scales on Y-axis
S : length of mark of key scales by the unit used in VIEWPORT

Line 120 is

AXES Xlength/20,Ylength/20,0,0,5,5,2

which means

＜Xlength/20,Ylength/20,> ! label X-axis equally divided 1/20 of entire range of X-axis direction and label Y-axis in the same way
<0,0,> ! X and Y intersect at coordinate origin.
<5,5,> ! key scale for both X and Y-axis appear at intervals of 5.
<2> ! length of key scale is 2.

Line 130 ~180 are to put names of drawing objects as labels. Up until previous Chapter PRINT instruction has been used in examples for output characters and values. However with PRINT you can only output characters in place along with ruled line. If you want to output characters and values freely according to where the drawing objects are placed, you have to use <LABEL> instruction.

Line 130 is to cancel the limit of drawing range specified by <VIEWPORT> in line 90 by <CLIP OFF> instruction.

Line 140 is to assign a character string.

Line 150 is to move the position of pen to assigned point. In this case X-axis is 0, Y-axis is Yhigh. Yhigh is the upper end of the drawing range specified in line 100. If you are to write any characters around there, you can write within the range specified VIEWPORT, but you can't do it outside of the range. Therefore the drawing range is temporarily cancelled by line 130.

Line 160 is to specify the position of the pen in the character string to be written as label, which has been moved by line 150. The number shall be from 1 through 9 and each has following meanings.

3 : upper-left (of character string)	6 : upper-middle	9 : upper-right
2 : left-middle	5 : center-middle	8 : right-middle
1 : lower-left	4 : lower-middle	7 : lower-right

Therefore <LOG 4> in line 160 is to position the center of the lower end of the character string of whole label with the coordinate value specified by line 150.

Line 170 is to output the label.

Line 180 is to limit the drawing range once again as directed by <VIEWPORT>, since the range has been cancelled in order to writing label.

Line 190 is to specify the color of the pen. The number shall be from 0 through 15 and each has following colors:

0(black), 1(white), 2(red), 3(yellow), 4(green), 5(cyan), 6(blue), 7(magenta), 8(black), 9(olive green), 10(aqua), 11(royal blue), 12(maroon), 13(brick red), 14(orange),

It would be better to confirm the color when you draw .

Line 200 to 220 are the loop for making line chart.

Line 210 is to draw line to indicated coordinate (X, X*X*X).

Line 230 to 290 line are the loop for labeling X-axis.

Line 230 line is to specify the font size with GDU (see the explanation for line 80) which is to be made by LABEL instruction.

<CSIZE> instruction has two parameters as below.

CSIZE height of character, width of character/height of character

Height of the character is specified by GDU and the ratio of width of the character to height can be also given. If only height of the character is specified, the ratio of width shall be automatically 0.6 as default.

As explained at line 150, line 240 is to direct that pen shall move to the coordinate which is bottom left of the character string.

Line 250 is to specify the color of pen as 2 (red).

Line 260 to 290 are to label X-axis with moving pen by 0.5 step.

Line 300 to 330 are to label Y-axis with moving pen by 0.5 step.

As for the formatted output in line 320, please refer to Chapter 6, Section (4), at line 210.

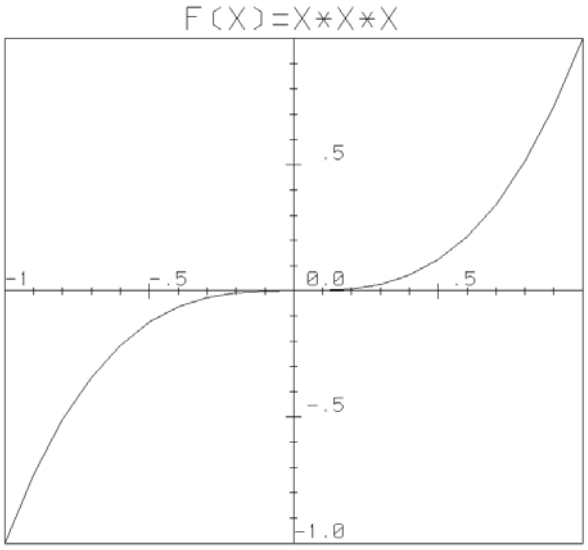


Fig.6 A graphic output

(2) Graphical display of online input data

Input following program after installing the driver for inesDAQ i250. This program includes the program for graphical display of measured data.

```
10    LOADSUB ALL FROM "c:\inesDAQ\htbasic\datahtb.csb"
20    OPTION BASE 1
30    DIN A(1),B(1024) ! Data arrays
40    INTEGER Dummy, Hadc,Cadc
50    DIN L$(256)                                    ! Label for configuring ADC card
60    CALL Daqhtb_ioctl(Dummy,"(fctn init)"
70    CALL Daqhtb_open(Hadc,"i250 ADC",0)
80    L$="(ioa(timeout 10000 fsmpl 100000 chan 1 iomode single trigmode off
samples 1 range [-10 10] ) ) "
90    PEN 3
100   AXES 5,5,0,50,2,2,2
110   PEN 1
120   MOVE 0,50
130   FOR I=I TO 101
140       CALL Daqhtb_ioctl(Hacd,L$)
150       CALL Daqhtb_read(Cadc,Hadc,A(*) )
160       DRAW I,A(1)*4+50
170       B(I)=A(1)
180       WAIT .1
190   NEXT I
200   FOR I=1 TO 101 STEP 10
210         PRINT I,B(I)
220   NEXT I
230   CALLDaqhtb_close(Hadc)
240   END
```

The line 80 is to instruct the operation of data input. In the example program it is shown in two lines due to the length of it, however you may not insert line feeds by pressing <ENTER>.

Line 90 to 120 are to prepare for displaying data like drawing coordinate axis.

Line 130 to 190 are for reading analog data and graphical display by inesDAQ i250.

Line 200 to 220 are numerical display of reading.

7.2 Output and save of graphics

As mentioned in preceding Section there are three ways to save graphics.

(i) The first one is to output directly from a plotter via GPIB interface. If you rewrite the statement <PLOTTER IS CRT,"INTERNAL"> in line 20 of the example program of the preceding Section into

```
PLOTTER IS 712,"HPGL"
```

it will be output by "HPGL" instruction code from the plotter addressed #12 which is connected via GPIB. The "HPGL" is the language developed by HP for graphics and the plotter to be used must have the "HPGL" language function. Plotters of HP had this function, though we seldom use plotters these days. MYPLOT series of Graphtec has this "HPGL" function among the plotters currently on sale.

(ii) The second way is to output from a general purpose printer. In this case the printers to be

used also must have the “HPGL” language function.
(iii) The third way is to import image displayed on the screen as bitmap file. In this case the quality of graphics is lower than output by plotter, however once it is save as a graphical file it is easy to copy and paste on documents on Windows. The easiest way is :

<Alt + Print Screen>
➔ <->Start Menu -> Program -> Accessory -> Paint >
➔ <->New document -> Edit -> Paste>

Hold the <Alt> key down and press the <Print Screen> key which is located upper right of the keyboard, then the screen of the active window will be copied. Next open the <Paint> application of Windows by selecting <->Start Menu -> Program -> Accessory -> Paint > and select <->New document -> Edit -> Paste>. Then new document comes up and the copied active window screen will be pasted on it. This image can be saved as a file it can be processed as JPEG file or pasted on a WORD file. In addition you can also do the same kind of operation with a graphical application by Adobe by selecting < File ->New document> or you can do it with a graphical tool, “Suguremo 3” by Source Next. Since it is not always necessary to hold the <Alt> key to use <Print Screen>, I would recommend you to try by models. I think the third way is the most convenient, because the direct output by plotter and printer can not be pasted to documents.

7.3 HT-BASIC Plus

HT BASIC has several GUI functions, and they are defined by program. Originally they were a program group called “HT-BASIC Plus” which is included in HT BASIC ver .9.2 as a standard function. For example, following functions are available, such as checking value of variable by graph, while running program. Also assigning value to variable by keyboard, adopting an appropriate program from several candidates, ordering a process while error occurs. HT-BASIC can handle the above operations by its original instructions, even though GUI of HT-BASIC Plus is not available.
There are many HT-BASIC users from the time when HT-BASIC Plus was not available, who can manage without HT-BASIC Plus.
Below is a sample program which shows a dialog (inquiry screen) adopting an appropriate operation when an error occurs

```
10   MASS STRAGE IS “C:\Program Files\HTBwin\”
20   LOAD BIN “BPLUS”            ! Binary program of HT-BASIC PLUS
30   DIM S1$ (0:1) [10],P$[20]
40   INTEGER Btn
50   S1$(0)=”Abort”
60   S1$(1)=”Continue”
70   P$=”Error is caused”
80   DIALOG “ERROR”,P$,Btn,SET(“DIALOG BUTTON”:S1$(*)),TIMEOUT 5
90   SELECT Btn
100 CASE=-1                      ! Btn=-1 means TIMEOUT
110       PRINT Btn
120       DISP “Timeout”
130 CASE=0
140       PRINT Btn
150       DISP S1$(Btn)
160 CASE=1
170       PRINT Btn
180       DISP S1$(Btn)
190 END SELECT
200 END
```

If you use HT-BASIC Plus, you have to execute

```
LOAD BIN “BPLUS”
```

as shown in line 10 to 20.

Then binary program has to be loaded.

Line 80 is to set “ERROR” dialog.

```
DIALOG “ERROR”,P$,Btn;SET (“DIALOG BUTTONS”:S1$(*)),TIMEOUT 5
```

“ERROR” shows kinds of dialog in its parameter. P$ is a message, when dialog is called, and it is defined based on a string variable in line 70. Btn is assigned by choosing an integer number of button, like 0, 1, 2, etc. If any value is not assigned, and time out parameter (time limit) is assigned, -1 is assigned automatically after time limit is over.

SET in parentheses ( ) shows a name of button by string array. TIMEOUT 5 means that time out is 5 seconds. When this program runs, 2 buttons, “Abort” and “Continue” are displayed in a dialog screen. When you choose the left button, number 0, 0 will be displayed. Also right button is chosen, number 1 and characters for “button” will be displayed. You can define more than 3 buttons. When no button was chosen and 5 seconds passed, -1 and “Timeout” will be displayed.

HT-BASIC Plus has 10 diagrams (query screen) and 30 widgets (tools for display meter, menu, graph). Programmable instructions in HT-BASIC Plus are described in help documents. Program samples are available in the name of folder of “plus examples”. However, as explained above, widgets have so many set-up parameters that programming by HT-BASIC may be easier than by HT-BASIC Plus basis in some cases, like a graphic chart in Chapter 7-(1).

8. Definition of Variables

(i)Naming rule of Variable
Max 15charactors, includes numeric, alphabets, and <_>.
The first character must be alphabet and capital.
The last character of string variable must be <$>.
(ii)Kinds of Variable
- Numeric Variable :numeric (e.g.: Data, Name)
Real Precision: 8Bytes, +/- effective digit 15 +/- power 308
Maximum and minimum limit is refereed by instruction sets of <MAXREAL>,<MINREAL>.
Integer Precision: 2Bytes, -32768 ~ +32767
- Strings Variable: 1Bytes/1chractor, default setting is Max.18charactors.
Given character value is defined by DIM.
(example) Data$: Default setting is max18 characters.
- Simple Variable: Variable, which is not defined any dimension.
- Array Variable: Variable, which is not defined any dimension of array.
(iii)Rule of Variable Definition
- Numeric Variable
Real number: REAL X,A(5) : Definition of real number (REAL) can be omitted.
(example) INPUT X : X is defined as a real number.
DIM A(5) : 5 dimensions as a real number array
DIM Data(10:14) : 5 dimensions as a real number array , which are called as suffix 10 ~ 14.
Integral number: INTEGER Y,B(5) : integral simple variable, and integral array variable
Complex number is handled as a couple (16Bytes) of real number (8Bytes).
- String Variable:
(example) DIM Name$[20] : string variable and simple variable
INPUT B$: B$ is string variable, MAX 18 characters.
DIM Name$(5) :5pcs of string variable, MAX 18 characters.
DIM Name$(10:14)[20]: 20 characters array , which is called Suffix 10~15.
(iv)Definition of Array
Number of dimension is MAX 6.
Element of each dimension is MAX 32767.
Suffix of array: case of no assignment of suffix dimension,
DIM A(5) : 5 real number array
OPTION 0 : call 0 ~ 4 (default)
OPTION 1 : call 1 ~ 5
(v) Numeric Operator and Priority Sequence
(): bracket
FN: function
^: exponential
*, / , MOD , DIV: product, quotient, modular, division,
+, -: plus, minus
=, <, >, >=, <=, <>: relational operator
NOT: logical operator
AND: logical operator
OR, EXOR: logical operator
+, -, *, MOD, DIV: If both operator object is integer number, operator is integer arithmetic.
Real number of relational calculus:
DROUND(A,12)=DROUND(B,12) : half adjust by digit 12 .
ABS(A-B)<=1E-10 : Absolute value of deference is smaller than range of error.

9. Afterword

1) Today's notebook PC is mainly used for electrical stationery and information terminal. 2) Even calculation, as an original function of notebook PC is operated by ready-made programs. 3) Notebook PC for instrumentation and control use, as mentioned in this report, is completely different from information terminal and calculator. When anyone, who is skilled for an information terminal, tries to use calculator function, he/she is going to face some barriers for understanding technical term and concept, even though hardware itself. Even more if he/she tries to use a notebook PC for instrumentation and control, he/she must encounter many obstacles not only in operating hardware itself, but also in cabling peripherals such as sensors, measuring instruments, actuators and so on. There is a reason, why ready-made AD converter board is designed for easy connection with step by step instruction and data acquisition can be made by ready-made application without touching the complex inner structure by user himself. It seems this trend is growing further. However system availability with complete control by their own idea is mandatory for academic researchers, engineers and students. Only given tools and systems cannot produce any free idea. Outsourcing cannot produce any appropriate system user required, unless the contracted engineers have skills and experiences on the field.

HT-BASIC is an all-in-one type computer language, which can work three types of use each other. (i) Building instrumentation system, and gathering data into files. (ii) Reading data from files, then calculation and plotting a graph by HT-BASIC or another ready-made software. (iii) To leverage the result directly or save graph image out of the result for presentation later. HT-BASIC can operate the above flow, as if writing instructions with a pencil and paper. Of course those flows are also operated by another program languages. However those programs are only handled by special trained persons, or there might be potential by their best efforts. There is a big difference between only trained person can do and ordinary researchers, engineers and students can do. Their daily operations are field study, arrangement of laboratory equipment and literature search. They are neither specialist for computer programming nor specialist for instrumentation system. It is only with HT-BASIC that those people can operate easily by following general logic step by step.

I have been discussing with many people in universities, companies and laboratories, about technique of instrumentation in the past, current, and future. What was common among many arguments is that everybody tries to find and create his own way for instrumentation, since the concept of computer has changed after Windows debut. They are the generation who knows HT-BASIC well. But there is an another opinion; why BASIC again, though C language is major for programming today. Some of elders who have experience and know the history, said that it was convenient using LabVIEW, as they could collect enough data and use accessory software with it, even if they didn't know the contents. On the other hand, younger

generation said that they had been using only ready-made software like LabVIEW, and had no experience in data acquisition by writing command lines even with C language. Among students operations tends to be done by ready-made instrumentation board with accessory software, then free license software from internet library will be used for data processing and reporting. As mentioned in Section 3-1, the freewheeling thinking can be generated through the best use of experimentation in laboratory. For instance you need to have a system which you can easily change parameters and run a program through a trial and error process. Early morning on August 23rd in 1977, Dr. Paul B. MacCready and his fellow succeeded to fly a figure of eight by manpower plane Gossamer Condor, for the first time in our history. It was Dr. Peter Lissaman who designed the main wing by the desktop computer HP9820A. Dr. Lissaman stated the reason why he could win the flight was that the body design of the airplane was easy to modify, and that by HP9820A he could confirm his idea immediately (Reference 4), even though he had old theory and an old computing machine. The documentary film of this flight won the Academy Award documentary section in 1978. Two years later, in 1978, they succeeded in Channel crossing by manpower plane Gossamer Condor for the first time in the human history. I assume that HT-BASIC can not only meet requirements from instrumentation sites but also make a significant contribution in numeric computation based on its compiler function and compiled subroutines and powered up computer performance. It is also possible for students to improve programs developed by older members in university laboratories.

In this report I have described the outline of HT-BASIC, which is going to be forgotten at least in Japan, unless anything is done. HT-BASIC has so many instruction sets such as GPIB system control, input and output control commands and error messages, in addition to the computing function and logical decision that even heavy HT-BASIC users cannot grasp the whole picture of HT-BASIC completely. However program samples in this report must be useful, so far as in ordinary usage of HT-BASIC.

In this report I also explained detail operation of inesDAQ i250. With those explanation you can generate a quite good program for data gathering with this adapter card, without using GPIB card.

As mentioned in the foreword, instruction manual for HT-BASIC is only available as online manual in English currently. Furthermore, those manuals don't include the total picture of HT-BASIC. I am pleased, if this report would be of some help for users to understand HT-BASIC.

Books for Reference

1) TransEra (2002): TransEra HTBasic Installing and Using Manual.

2) TransEra (2002): TrandEra HTBasic Online Manual

3) ines (2001): inesDAQ Online Manual

4) Blair, John (1979): Keyboard, Jul-Aug, pp.14/16, Hewlett-Packard.

5) CQ Publishing (1996): Introduction of Instrumental and Control by PC, Transistor Gijutu special issue, No.53.

6) CQ Publishing (1999): Introduction of Instrumental and Control by Windows PC, Transistor Gijutu special issue, No.68.

7) Hirobumi Fujiwara (1994): C Programming Special Course, Gijutsu-Hyohron Co., Ltd.

8) Iwanami Shoten (1981): Science Journal(special), 51-10