PICBASIC DATABOOK

VOL.10

COMFILE Technology, Inc.
30-1, Shingae-dong, Yongsang-ku,
Seoul, 140-190, Korea
Tel.: +82-2-711-2592
Fax: +82-2-711-2593
E-mail: Comfile@comfile.co.kr
Internet: www.comfile.co.kr

UPGRADE INFORMATION

All resources related to PICBASIC such as firmware, IDE and manual etc may be upgraded irregularly for improvement without notice. The information about that is released at COMFILE Technology web site, www.comfile.co.kr, and its details are as follows.

IDE

IDE (integrated Development Environment) for PICBASIC, PICBASIC studio and PICBASIC-LAB are able to be downloaded at www.comfil.co.kr free of charge.

PICBASIC FIRMWARE

Upgrade information for firmware inside of PICBASIC module will be announced at COMFILE Technology web site, www.comfile.co.kr.

Upgrading firmware of PB-3X and PBM-RX is free of charge while that of PB-1X and PB-2X cost.

TRADEMARK

NOTICE

TABLE OF CONTENTS

CHAPTER 1. WHAT IS PICBASIC?

CHAPTER 2. PICBASIC INSTRUCTIONS

CHAPTER 3. PICBASIC PROGRAMMING

Chapter 1.

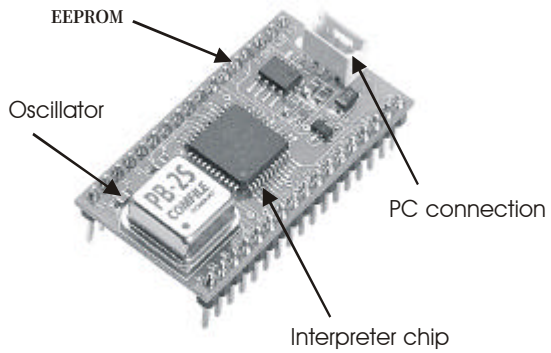
What is PICBASIC ?

What is PICBASIC...

In a word, PICBASIC is micro single board computer controlled by the PICBASIC language. The PICBASIC language, based on BASIC language, is developed by Comfile Technology and includes various single instructions such as ADIN, PWM, and SEROUT etc to be used in automation control field. You can simply make instant use of PICBASIC by connecting to printer port of PC and that there is no need of extra MDS. It could be applicable widely in automation control field such as automated machine, thermal controller, inspection ZIG, ROBOT controller, data acquaintance device etc.

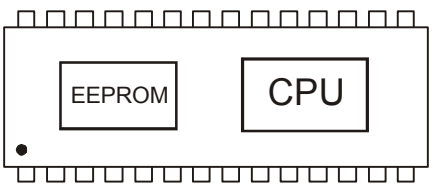
HARDWARE OF PICBASIC

In case of PB-2S, it consists of two key components, PICmicro chip and EEPROM. A program programmed by PICBASIC language is converted into intermediate code and stored at EEPROM. (EEPROM is FLASH type memory conserving its contents without power supplying) PICmicro chip in PICBASIC plays role of a interpreter that translates and executes the program stored at EEPROM. PICBASIC contains interpreter program on board. Exterior of PB-1S is similar as SIP (Single Inline Package) and its lower pins are power source pin, RESET pin and I/O pins.

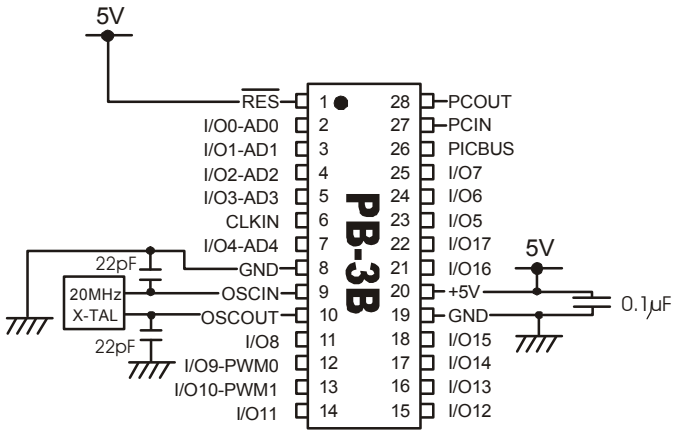


<PB-2S>

In case of PB-3X family, it has different composition. There is no need of extra EEPROM in PB-3X family because that the PICmicro chip used in PB-3X family has built-in EEPROM.

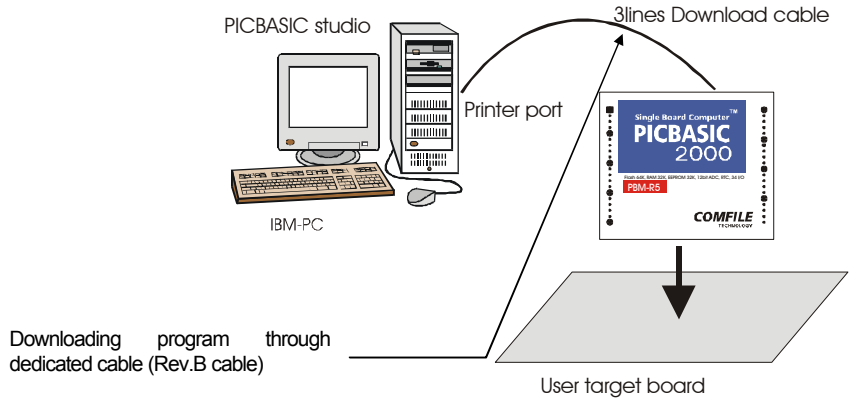


However, basic elements such as outer oscillation and bypass condenser etc should be composed by users.



PICBASIC PROGRAMMING...

First you need to connect a PICBASIC module to PC with the PICBASIC cable. After connection, you can program with PICBASIC studio on PC. The PICBASIC studio is Integrated Development Environment for PICBASIC and is released free of charge by Comfile Technology. You can download it at www.comfile.co.kr. The program takes course of compiling and downloading and running by clicking RUN button in PICBASIC studio. When occurring errors in process of compiling, a error message will appear.

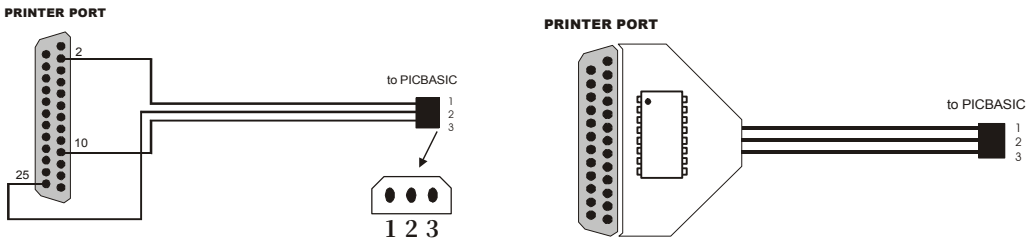


PROGRAM MAINTENANCE

When you need to revise/modify a program, you just simply re-connect the PICBASIC cable to PC, and revise/modify the program on PICBASIC studio. The program is stored at EEPROM, which keeps its data even at power-off state.

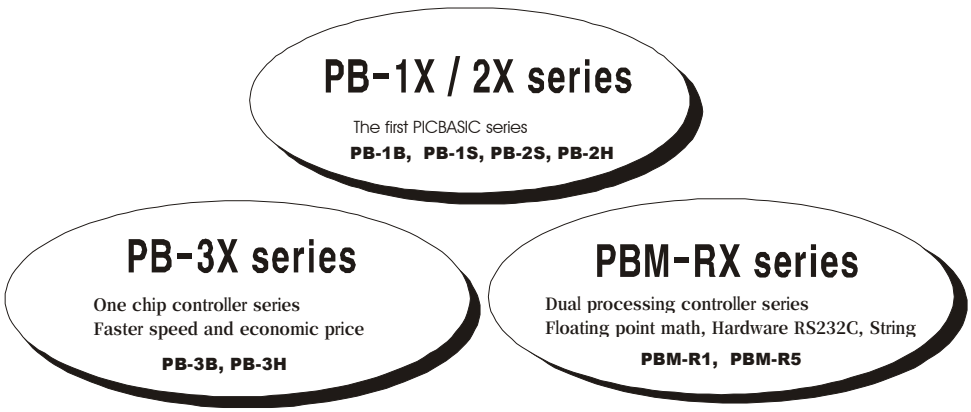
PICBASIC CABLE

Download cable for PB series uses three lines to connect with printer port. Download cable for PBM series named Rev.B cable includes a microprocessor performing error detection and speed control between PC and PICBASIC. It insures to prevent various problems in downloading.



PICBASIC MODULES

PICBASIC modules are classified as three types of series.



"PB series" in this document is means modules of which name stars with PB, such as PB-1B, PB-1s,PB-2S, PB-2H, PB-3B, PB-3H while "PBM series" means PBM-R1 and PBM-R5.

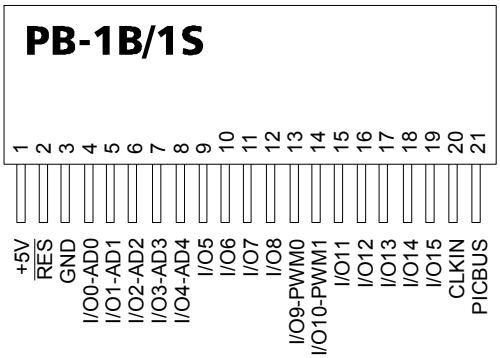
SPECIFICATION

	PB-1B	PB-1S	PB-2S	PB-2H	PB-3B	PB-3H	PBM-R1	PBM-R5
Main Memory	2K byte	4K byte	8K byte	16K byte	4K byte	4K byte	32K byte	64K byte
Data Memory	96 byte	96 byte	96 byte	96 byte	80 byte	80 byte	8 K byte	32K byte
I/O ports	16	16	27	27	21	29	34	34
Main processor	PIC16C73	PIC16C73	PIC16C74	PIC16C74	PIC16F876	PIC16F877	PIC16F877	PIC16F877
Oscillation frequency	4.19MHz	4.19MHz	4.19MHz	20MHz	20MHz	20MHz	20MHz	20MHz
Execution rate comparison*	13.1times	13.9times	13.9times	3.1times	1times	1times	1.4times	1.4times
Memory Access*	serial	serial	serial	serial	serial	serial	serial	Serial
EEPROM for data							8K byte	32K byte
No. of Pin	21	21	34	34	28	40	40	40
A/D channel (Resolution)	5 (8 bit)	5 (8 bit)	8 (8 bit)	8 (8 bit)	5 (10 bit)	8 (10 bit)	8 (10 bit)	8 (10 bit)
12 bit A/D								2channel
PWM channel (Resolution)	2 (8bit)	2 (8bit)	2 (8bit)	2 (8bit)	2 (8bit)	2 (8bit)	2 (10bit)	2 (10bit)
String	N/A	N/A	N/A	N/A	N/A	N/A	Available	Available
32bit integer, real number	N/A	N/A	N/A	N/A	N/A	N/A	Available	Available
RS232 buffering method	N/A	N/A	N/A	N/A	N/A	N/A	Available	Available

- Execution rate comparison is execution speed of each PICBASIC module compared with the fastest module, PB-3X. i.e., PB-1B is slower than PB-3X as much as 13.1 times. The larger execution rate is, the slower execution speed is.
- Memory access is a method of accessing the program memory at CPU. Parallel access is faster than Serial access. Because a PB-3X module uses a built-in BUS, it is much faster than PBM-Rx modules.

PINOUT & CIRCUIT

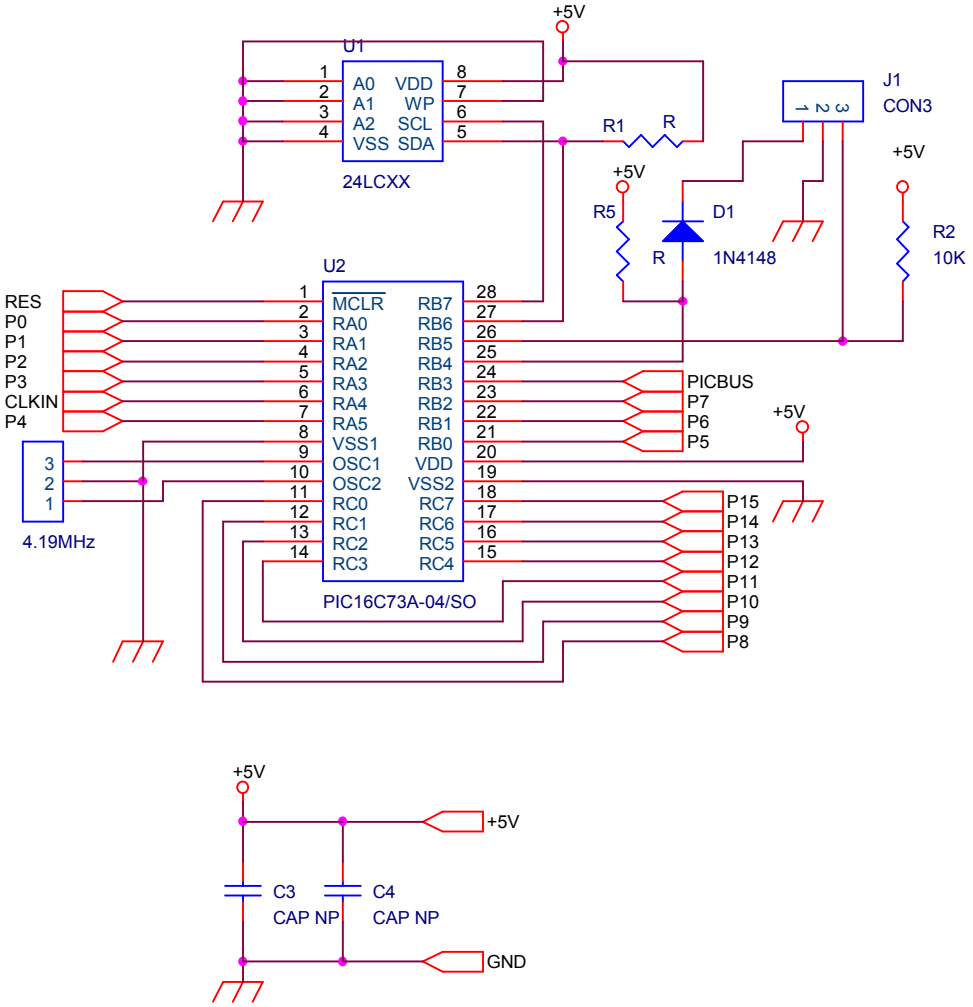
PB-1B /1S module



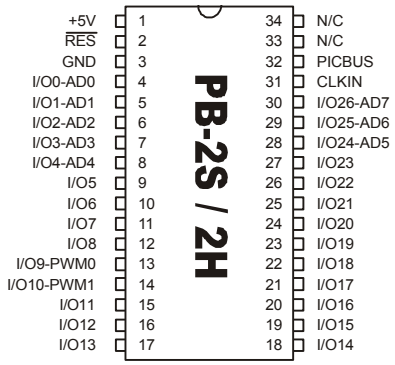
Pin no.	Description		Port block	Input	Function
1	+5V	Power, 5V			
2	/RES	Reset, 5V			
3	GND	Ground			
4	I/O0/ AD0	Port 0	Block 0	TTL	AD input
5	I/O 1/ AD1	Port 1	Block 0	TTL	AD input
6	I/O 2/ AD2	Port 2	Block 0	TTL	AD input
7	I/O 3/ AD3	Port 3	Block 0	TTL	AD input
8	I/O 4/ AD4	Port 4	Block 0	TTL	AD input
9	I/O 5	Port 5	Block 0	TTL	
10	I/O 6	Port 6	Block 0	TTL	
11	I/O 7	Port 7	Block 0	TTL	
12	I/O 8	Port 8	Block 1	ST	
13	I/O 9/ PWM0	Port 9	Block 1	ST	PWM port
14	I/O 10/ PWM1	Port 10	Block 1	ST	PWM port
15	I/O 11	Port 11	Block 1	ST	
16	I/O 12	Port 12	Block 1	ST	
17	I/O 13	Port 13	Block 1	ST	
18	I/O 14	Port 14	Block 1	ST	
19	I/O 15	Port 15	Block 1	ST	
20	CLKIN	Counter input		ST	
21	PICBUS	LCD port			

- ST = Schmitt trigger
- TTL = TTL Level input

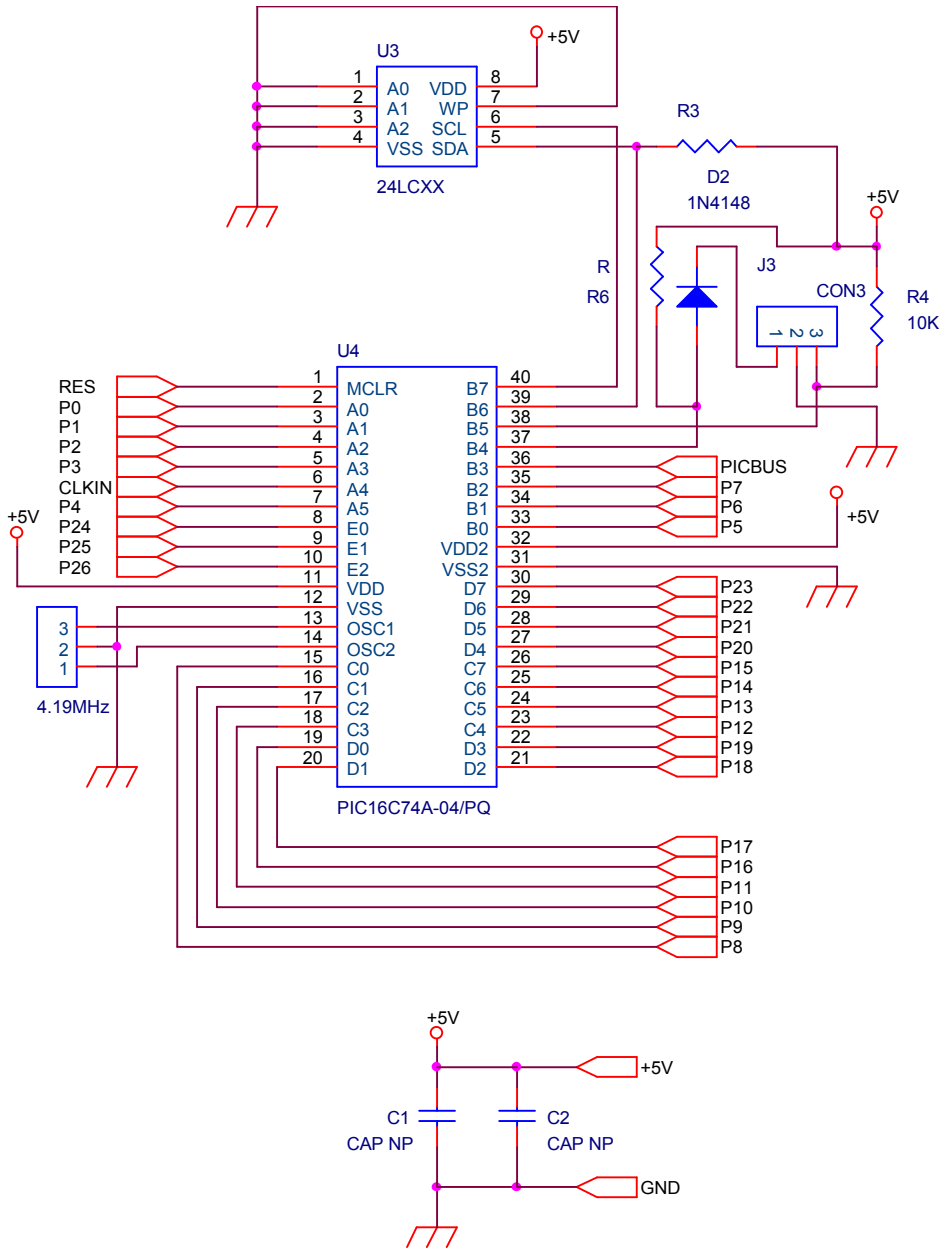
TTL Level input port recognizes the state of a pin as HIGH above 1.4V while as LOW below 1.3V. Schmitt trigger input port recognizes the state of a pin as HIGH above 3.4V while as LOW below 3.3V.


+5V
U1
A0
A1
A2
VSS
VDD
WP
SCL
SDA
24LCXX
R1
R
J1
CON3
+5V
R5
D1
R
1N4148
R2
10K
U2
MCLR
RA0
RA1
RA2
RA3
RA4
RA5
VSS1
OSC1
OSC2
RC0
RC1
RC2
RC3
RB7
RB6
RB5
RB4
RB3
RB2
RB1
RB0
VDD
VSS2
RC7
RC6
RC5
RC4
PIC16C73A-04/SO
RES
P0
P1
P2
P3
CLKIN
P4
4.19MHz
PICBUS
P7
P6
P5
P15
P14
P13
P12
P11
P10
P9
P8
C3
CAP NP
C4
CAP NP
GND

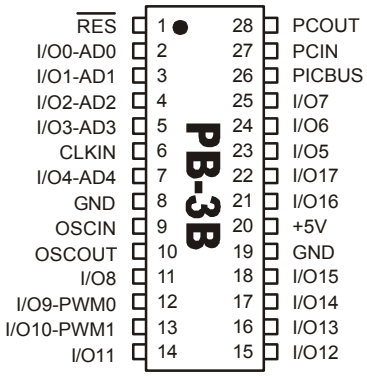
PB-2S / PB-2H



Pin no.	Description		Port block	Input	Function
1	+5V	Power, 5V			
2	/RES	Reset, 5V			
3	GND	Ground			
4	I/O0/AD0	Port 0	Block0	TTL	AD input
5	I/O1/AD1	Port 1	Block0	TTL	AD input
6	I/O2/AD2	Port 2	Block0	TTL	AD input
7	I/O3/AD3	Port 3	Block0	TTL	AD input
8	I/O4/AD4	Port 4	Block0	TTL	AD input
9	I/O5	Port 5	Block0	TTL	
10	I/O6	Port 6	Block0	TTL	
11	I/O7	Port 7	Block0	TTL	
12	I/O8	Port 8	Block1	ST	
13	I/O9/PWM0	Port 9	Block1	ST	PWM port
14	I/O10/PWM1	Port 10	Block1	ST	PWM port
15	I/O11	Port 11	Block1	ST	
16	I/O12	Port 12	Block1	ST	
17	I/O13	Port 13	Block1	ST	
18	I/O14	Port 14	Block1	ST	
19	I/O15	Port 15	Block1	ST	
20	I/O16	Port 16	Block2	ST	
21	I/O17	Port 17	Block2	ST	
22	I/O18	Port 18	Block2	ST	
23	I/O19	Port 19	Block2	ST	
24	I/O20	Port 20	Block2	ST	
25	I/O21	Port 21	Block2	ST	
26	I/O22	Port 22	Block2	ST	
27	I/O23	Port 23	Block2	ST	
28	I/O24/AD5	Port 24	Block3	ST	AD input
29	I/O25/AD6	Port 25	Block3	ST	AD input
30	I/O26/AD7	Port 26	Block3	ST	AD input
31	CLKIN	Count input		ST	
32	PICBUS	LCD port			
33, 34	N/C	No Connection			


U3
A0
A1
A2
VSS
VDD
WP
SCL
SDA
24LCXX
+5V
R3
D2
1N4148
R
R6
J3
CON3
R4
10K
U4
MCLR
A0
A1
A2
A3
A4
A5
E0
E1
E2
VDD
VSS
OSC1
OSC2
C0
C1
C2
C3
D0
D1
B7
B6
B5
B4
B3
B2
B1
B0
VDD2
VSS2
D7
D6
D5
D4
C7
C6
C5
C4
D3
D2
PIC16C74A-04/PQ
RES
P0
P1
P2
P3
CLKIN
P4
P24
P25
P26
PICBUS
P7
P6
P5
P23
P22
P21
P20
P15
P14
P13
P12
P19
P18
P17
P16
P11
P10
P9
P8
4.19MHz
C1
CAP NP
C2
CAP NP
GND

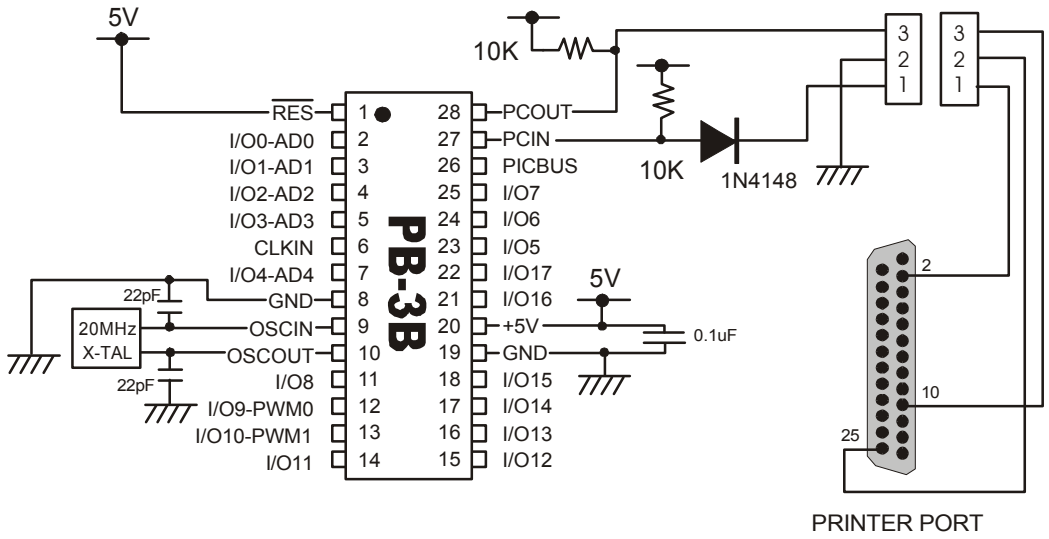
PB-3B



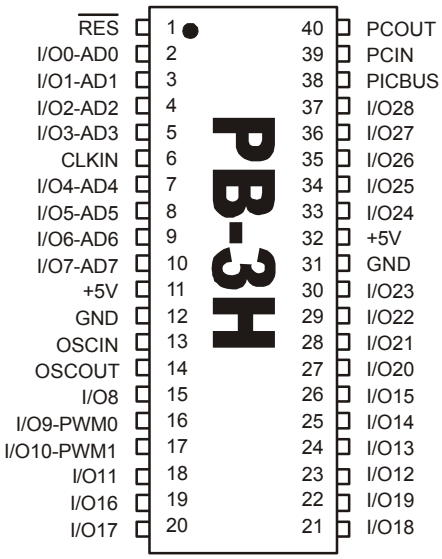
Pin no.	Description		Port block	Input	Function
1	/RES	Reset			
2	I/O0-AD0	Port 0		TTL	A/D input
3	I/O1-AD1	Port 1		TTL	A/D input
4	I/O2-AD2	Port 2		TTL	A/D input
5	I/O3-AD3	Port 3		TTL	A/D input
6	CLKIN	Count input			
7	I/O4-AD4	Port 4		TTL	A/D input
8	GND	Ground			
9	OSCIN	Crystal connection			
10	OSCOUT	Crystal connection			
11	I/O8	Port 8	Block1	ST	
12	I/O9-PWM0	Port 9	Block1	ST	PWM port
13	I/O10-PWM1	Port 10	Block1	ST	PWM port
14	I/O11	Port 11	Block1	ST	
15	I/O12	Port 12	Block1	ST	
16	I/O13	Port 13	Block1	ST	
17	I/O14	Port 14	Block1	ST	
18	I/O15	Port 15	Block1	ST	
19	GND	Ground			
20	+5V	Power, 5V			
21	I/O16	Port 16		ST	Edge interrupt
22	I/O17	Port 17		ST	
23	I/O5	Port 5		ST	
24	I/O6	Port 6		ST	
25	I/O7	Port 7		ST	
26	PICBUS	LCD port			
27	PCIN	PC connection pin, IN			
28	PCOUT	PC connection pin, OUT			

- All pins except V_{DD}, V_{SS} and RESET pin is used for I/O.
- I/O ports are arranged in order of their number and are blocked by eight ports. The blocked eight ports is called Port block. (Port block is essential to process by byte.)
- Generally, RES pin is connected with V_{DD} (Internal RESET delay circuit)
- Because CLKIN pin can detect Rising edge, even 20KHz of high speed pulse can be counted.
- PB-3B module is same as PICmicro PIC16C73 in PIN layout.
- PB-3B module has one port block. Addressing at port block 0 or 2 is ignored by instructions.
- OSCIN, OSCOUT must be connected to 20MHz crystal.
- /RES pin should be connected to 5V without additional circuit.

The picture shown below is basic circuit for operating PB-3B module.



PB-3H

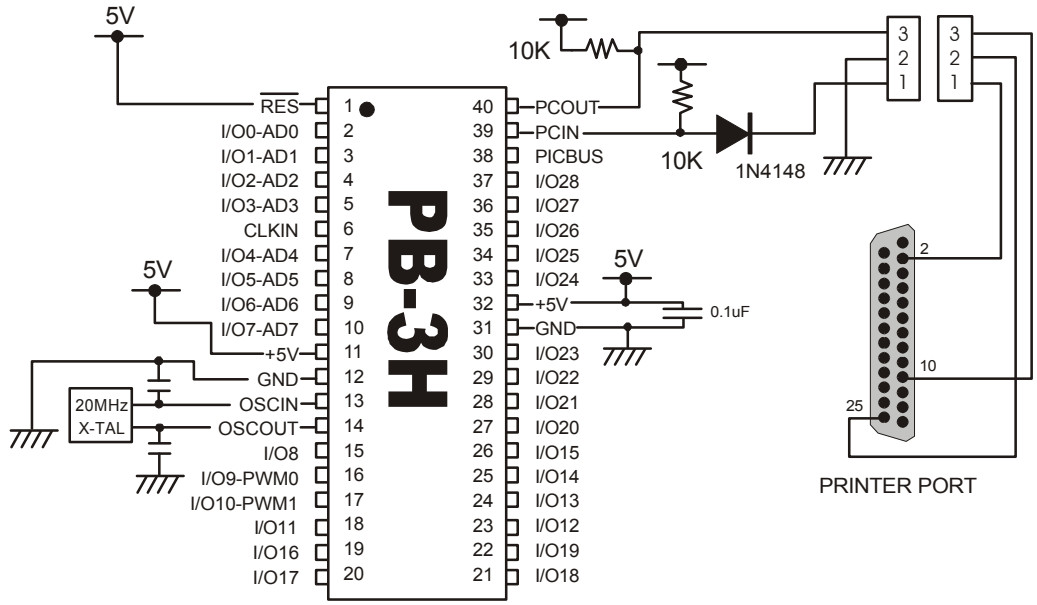


Pin no.	Description		Port block	Input	Function
1	/RES	Reset			
2	I/O0-AD0	Port 0	Block 0	TTL	A/D input
3	I/O1-AD1	Port 1	Block 0	TTL	A/D input
4	I/O2-AD2	Port 2	Block 0	TTL	A/D input
5	I/O3-AD3	Port 3	Block 0	TTL	A/D input
6	CLKIN	Count input			
7	I/O4-AD4	Port 4	Block 0	TTL	A/D input
8	I/O5-AD5	Port 5	Block 0	TTL	A/D input
9	I/O6-AD6	Port 6	Block 0	TTL	A/D input
10	I/O7-AD7	Port 7	Block 0	TTL	A/D input
11	+5V	Power, 5V			
12	GND	Ground			
13	OSCIN	Crystal connection			
14	OSCOUT	Crystal connection			
15	I/O8	Port 8	Block 1	ST	
16	I/O9-PWM0	Port 9	Block 1	ST	PWM port
17	I/O10-PWM1	Port 10	Block 1	ST	PWM port
18	I/O11	Port 11	Block 1	ST	
19	I/O16	Port 16	Block 2	ST	
20	I/O17	Port 17	Block 2	ST	
21	I/O18	Port 18	Block 2	ST	
22	I/O19	Port 19	Block 2	ST	
23	I/O12	Port 12	Block 1	ST	
24	I/O13	Port 13	Block 1	ST	
25	I/O14	Port 14	Block 1	ST	
26	I/O15	Port 15	Block 1	ST	

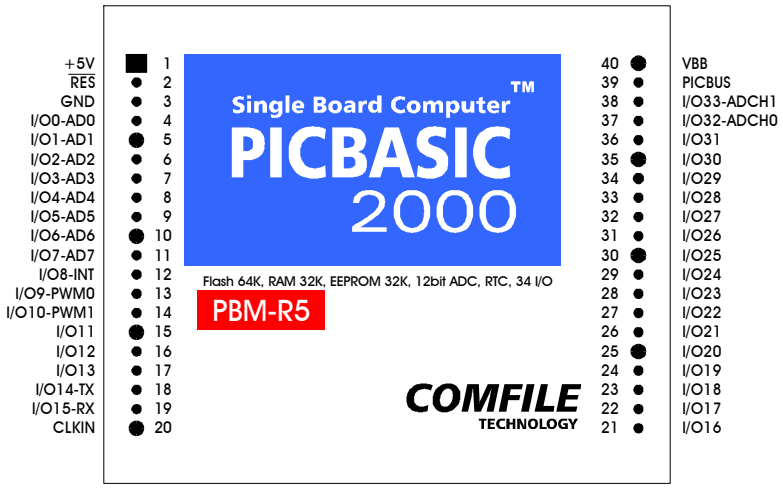
27	I/O20	Port 20	Block 2	ST	
28	I/O21	Port 21	Block 2	ST	
29	I/O22	Port 22	Block 2	ST	
30	I/O23	Port 23	Block 2	ST	
31	GND	Ground			
32	+5V	Power, 5V			
33	I/O24	Port 24		ST	Edge interrupt
34	I/O25	Port 25		ST	
35	I/O26	Port 26		ST	
36	I/O27	Port 27		ST	
37	I/O28	Port 28		ST	
38	PICBUS	LCD port			
39	PCIN	PC connection pin, IN			
40	PCOUT	PC connection pin, OUT			

- PB-3H module is same as PIC16c74, PICmicro in pin layout.

The following picture is a basic circuit to operate PB-3H module.



PBM-R1/ R5



Pin no.	Description		Port block	Input	Function
1	+5V	Power, 5V			
2	/RES	Reset, 5V			
3	GND	Ground			
4	I/O0/AD0	Port 0	Block0	TTL	10bits AD input
5	I/O1/AD1	Port 1	Block0	TTL	10bits AD input
6	I/O2/AD2	Port 2	Block0	TTL	10bits AD input
7	I/O3/AD3	Port 3	Block0	TTL	10bits AD input
8	I/O4/AD4	Port 4	Block0	TTL	10bits AD input
9	I/O5/AD5	Port 5	Block0	TTL	10bits AD input
10	I/O6/AD6	Port 6	Block0	TTL	10bits AD input
11	I/O7/AD7	Port 7	Block0	TTL	10bits AD input
12	I/O8 / INT	Port 8	Block 1	ST	Edge interrupt
13	I/O9 / PWM0	Port 9	Block 1	ST	10bits PWM port
14	I/O10 / PWM1	Port 10	Block 1	ST	10bits PWM port
15	I/O11	Port 11	Block 1	ST	
16	I/O12	Port 12	Block 1	ST	
17	I/O13	Port 13	Block 1	ST	
18	I/O14 / TX	Port 14	Block 1	ST	RS232C transmitting
19	I/O15 / RX	Port 15	Block 1	ST	RS232C receiving
20	CLKIN	Count input		ST	
21	I/O16	Port 16	Block 2	ST	
22	I/O17	Port 17	Block 2	ST	
23	I/O18	Port 18	Block 2	ST	
24	I/O19	Port 19	Block 2	ST	
25	I/O20	Port 20	Block 2	ST	
26	I/O21	Port 21	Block 2	ST	

27	I/O22	Port 22	Block 2	ST	
28	I/O23	Port 23	Block 2	ST	
29	I/O24	Port 24	Block 3	ST	
30	I/O25	Port 25	Block 3	ST	
31	I/O26	Port 26	Block 3	ST	
32	I/O27	Port 27	Block 3	ST	
33	I/O28	Port 28	Block 3	ST	
34	I/O29	Port 29	Block 3	ST	
35	I/O30	Port 30	Block 3	ST	
36	I/O31	Port 31	Block 3	ST	
37	I/O32/ADCH0	Port 32	AD input only		12bits, AD input
38	I/O33/ADCH1	Port 33	AD input only		12bits, AD input
39	PICBUS	LCD port			
40	VBB	Real time clock	Battery	Terminal	

- All pins except $V_{DD,}$ V_{SS} and RESET pin is used for I/O.
- I/O ports are arranged in order of their number and are blocked by eight ports. The blocked eight ports is called Port block. (Port block is essential to process by byte.)
- Generally, RES pin is connected with V_{DD} (Internal RESET delay circuit)
- I/O port 32/33 are dedicated to AD input. I.e. they have no other I/O function except AD input.
- Pin no. 40 (V_{BB}) is backup power for Real-time clock. PICBASIC modules contain 0.1F super condenser. It is charged while the module is supplied electric power. When power source of the module is off, it starts working as battery. If you keep the PICBASIC module off during over 6months, you should need to supply extra electric power to the PICBASIC module through Pin no. 40 so that internal clock keeps working.
- Because CLKIN port can detect Rising edge, even 20KHz of high speed pulse can be counted.

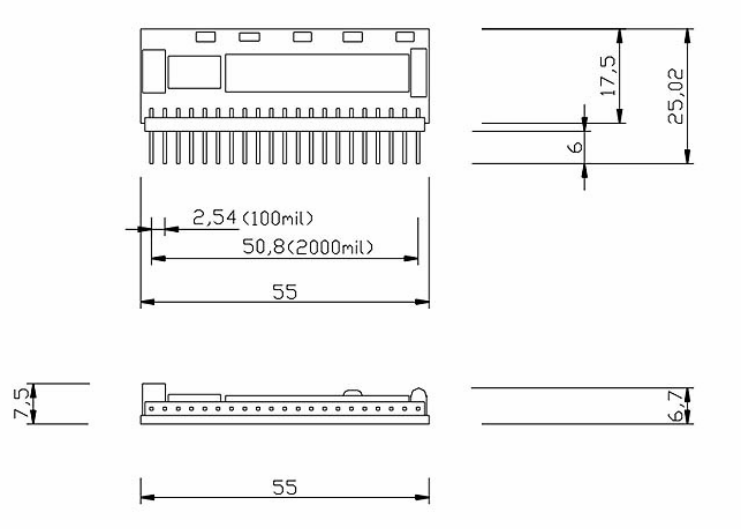
* PBM-R1 is same as PBM-R5 in function except 12bit A/D function and real-time clock function.

ELECTRIC CHARACTERISTIC OF PBM-RX

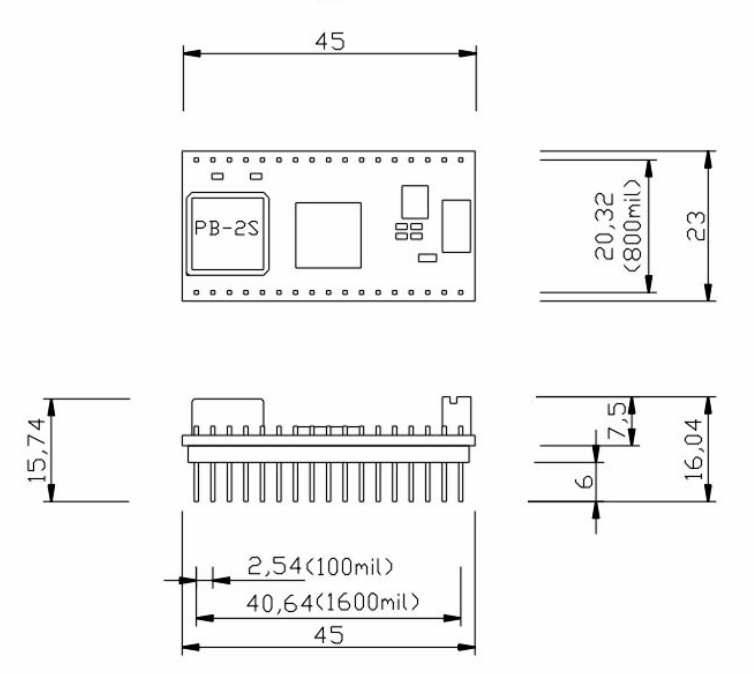
V_{DD} Voltage	: 4.5V ~ 5.5V
Current consumption	: PB-1B, 1S, 2S : 7mA
	: PB-2H : 15mA
	: PB-3B, 3H : 6mA
	: PBM-R1, R5 : 50mA
Storage temperature	: -40° ~ 125°
Operating temperature	: 0° ~ 75°
Source current of port	: 25mA
Sink current of port	: 25mA
Maximum allowable current of V_{SS}	: 300mA
Maximum allowable current of V_{DD}	: 250mA

DIMENSION

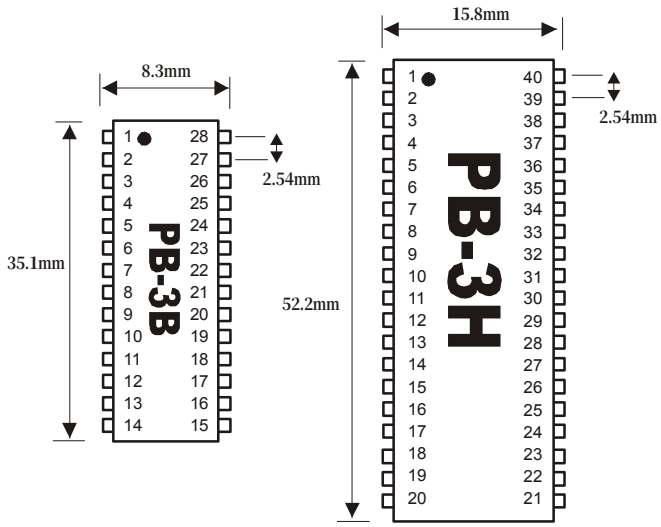
PB-1B / PB-1S



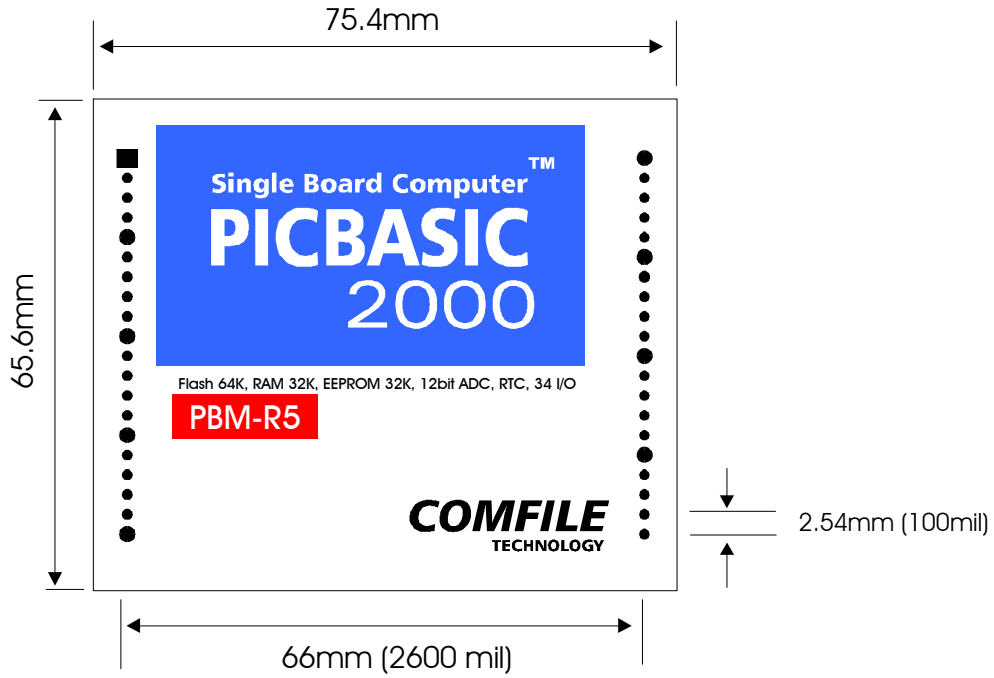
PB-2S / PB-2H



PB-3X

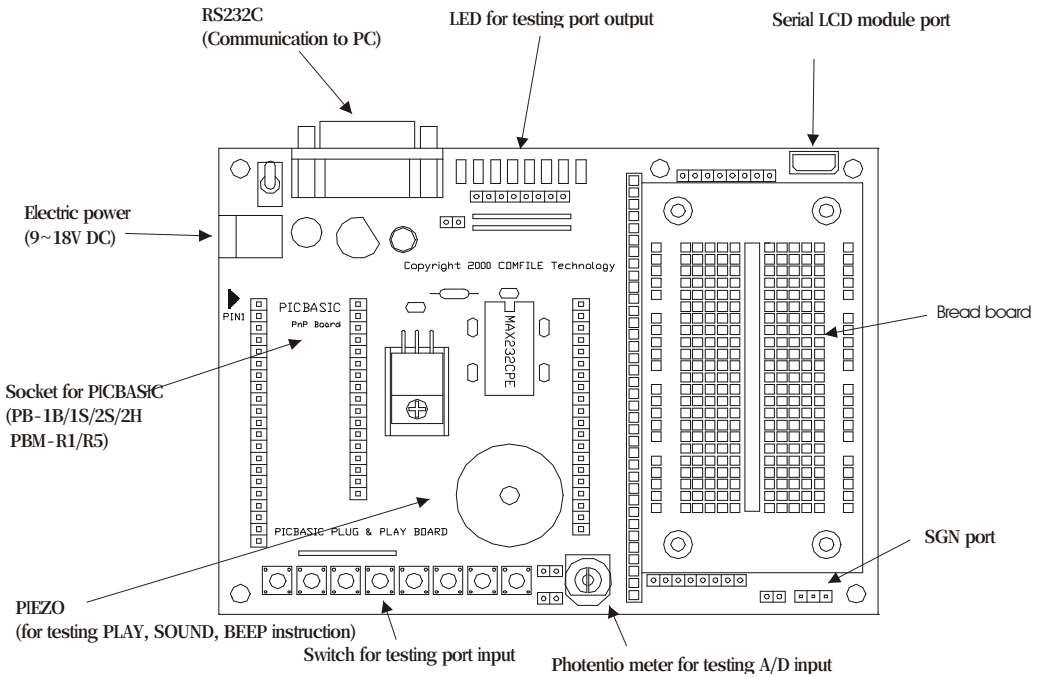

8.3mm
35.1mm
2.54mm
PB-3B
15.8mm
52.2mm
2.54mm
PB-3H

PBM-Rx

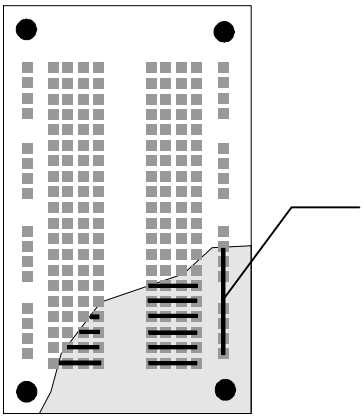

75.4mm
65.6mm
Single Board Computer™
PICBASIC
2000
Flash 64K, RAM 32K, EEPROM 32K, 12bit ADC, RTC, 34 I/O
PBM-R5
COMFILE
TECHNOLOGY
2.54mm (100mil)
66mm (2600 mil)

PICBASIC PNP BOARD

PnP board is an educational board for testing/training PICBASIC without soldering. There are two types of PnP boards, PnP board for PB-1X/PB-2X/PBM-RX modules and PnP board3 for PB-3X modules. PnP board contains a basic power source circuit, S/W input circuits, LED output circuits, a RS232C level conversion circuit and a breadboard etc. The following picture shows each parts of PnP board.

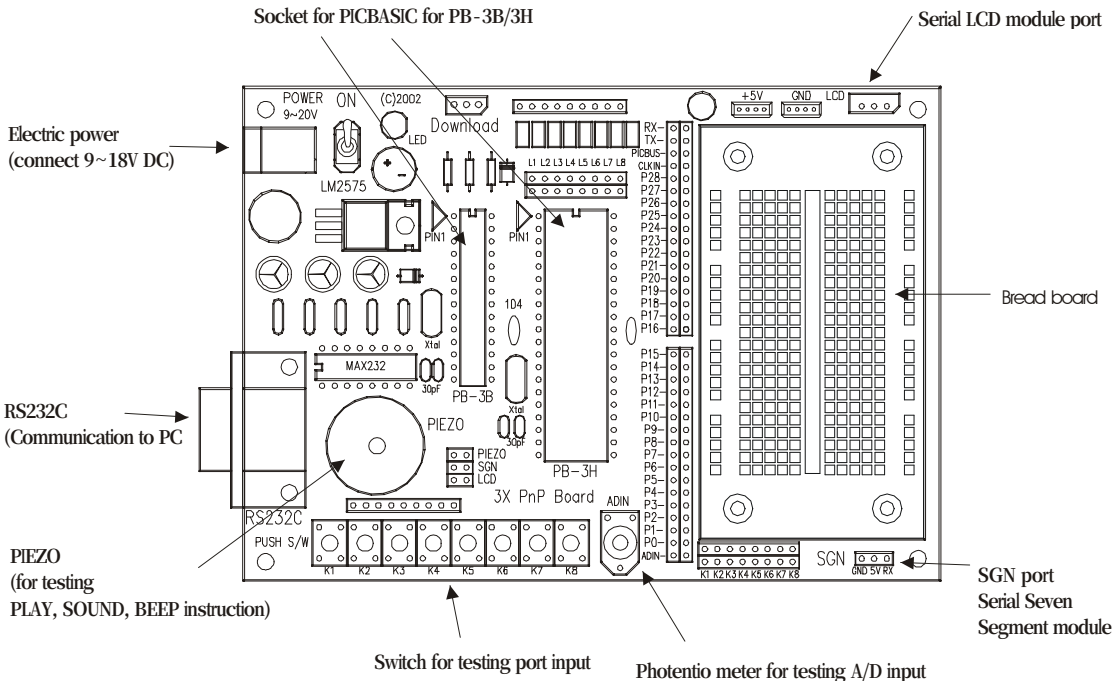


Because PnP board also contains a breadboard, you can add components that you want to use for testing.

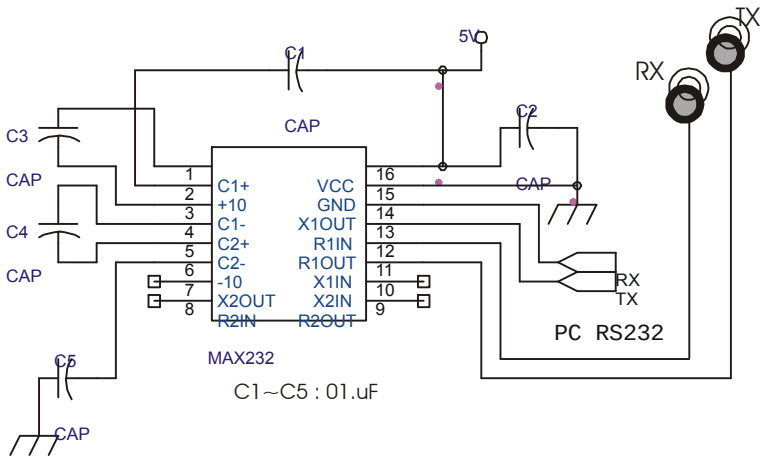


There is unexposed wire connection inside of bread board as shown picture.

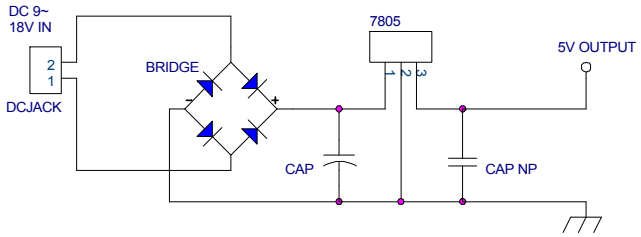
The following picture describes each part of PnP board3.



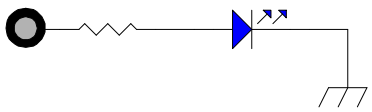
The following picture is RS232C circuit of PnP board.



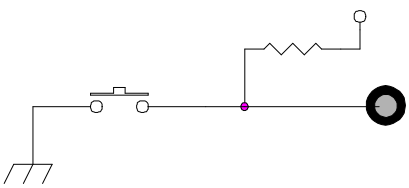
The following picture is power source circuit of PICBASIC PnP board. It output 5V with 9V~20V input. (The rated current of the circuit is less than 200mA. When current more than 200mA is needed, you should use an additional power supply. (PnP board3 for PB-3X has different power source circuit)



The following diagram is LED circuit. (When HIGH, LED turns on)



The following diagram is PUSH S/W circuit. (When a switch is pushed, it turns LOW state.)

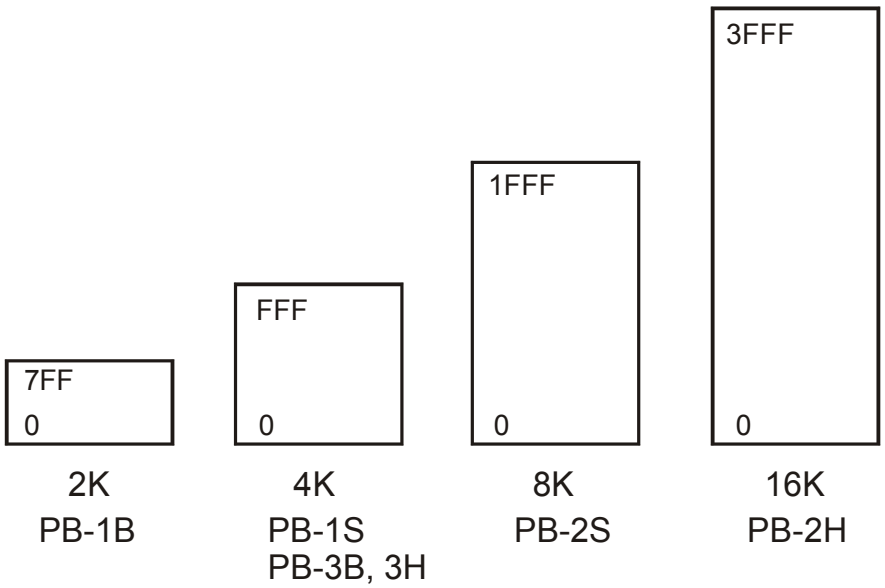


Various application boards can be used with PnP board, such as STEP motor driving module, DC motor driving module, RTC board and so on. For more information on application boards, refer to COMFILE Technology catalog or visit www.comfile.co.kr.

MEMORY

PB MODULES

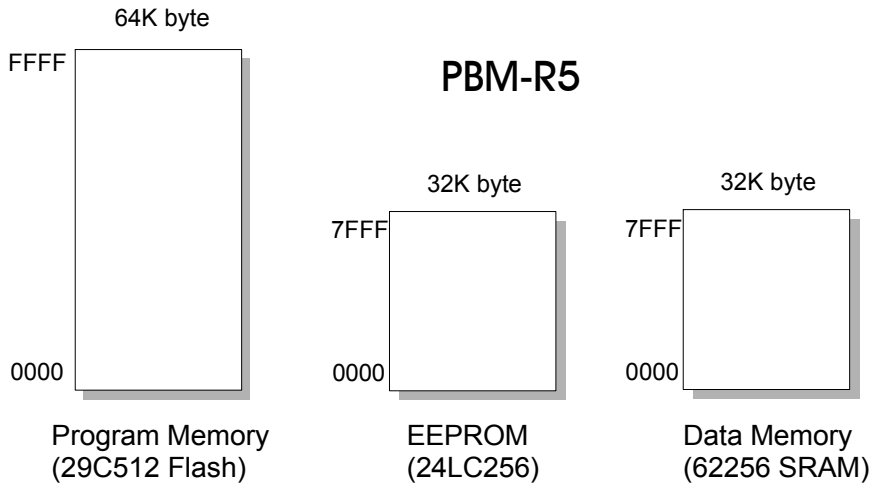
PB modules have 2K~16K of EEPROM, which is used for storing program code or data. The EEPROM is connected to a main microprocessor of a PB module through I^2C protocol, 2 wires serial communication standard. In case of PB-3X, it is connected to its main processor directly. Program code is stored from address 0 and unused area after storing program code can be used for storing data. (EEPROM can keep its content in power-off state.)



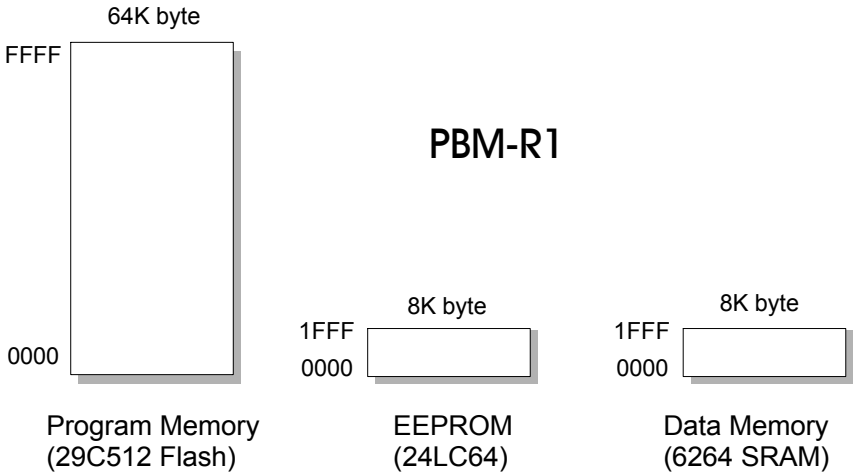
The other memory is 96bytes of SRAM. In case of PB-3X, it is 88bytes. The SRAM is inside of microprocessor and used for storing value of variables. (Data in SRAM is destroyed when power source is off and is set as 0 when power source is on.)

PBM MODULES

A PBM-R5 module has a FLASH ROM 29C512 (64Kbyte) as its main memory and has a SRAM 62256 (32Kbyte) and a EEPROM 24LC256 (32Kbyte) as its data memory.



A PBM-R1 module has a FLASH ROM 29C512 (64Kbyte) of main memory and has a SRAM 6264 (8Kbyte) and a EEPROM 24LC64 (8Kbyte) of data memory. (The differences between PBM-R1 and PBM-R5 are 12bit A/D convert function and real-time clock function)



Unlike PB modules, PBM series modules do not clear content of RAM on RESET. Therefore, user software should clear content of RAM if needed.

PICBASIC Studio

PICBASIC studio is Integrated Development Environment for PICBASIC. It includes Compiler, Editor, Debugger and etc and has similar usage as a word processor and is compatible with Microsoft Windows 98/ME/NT/2000/XP.
You can get it from COMFILE program CD provided with PICBASIC products or download from download section of www.comfile.co.kr/english.

INSTALLATION

To program PICBASIC modules, it is essential to install PICBASIC studio and to make connection between a PICBASIC module and your PC with a PICBASIC cable.

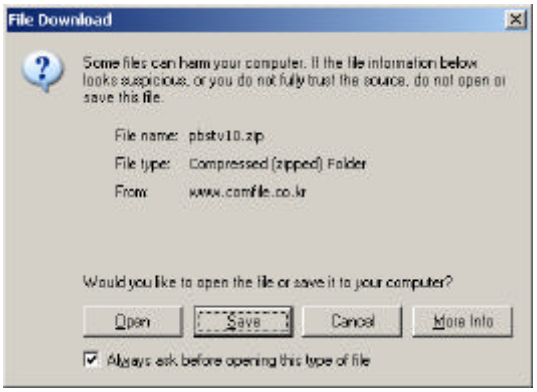
To install PICBASIC studio into your PC, please follow below steps.

Step 1. Insert COMFILE program CD into CD drive of your PC. In a moment, the following screen will appear on your PC. (The screen may be different in according to CD version.)

Step 2. On the screen, click **ENGLISH,** then you will see pages written in English. In the page, select download Tab.

Step 3. In the download page, click download button for PICBASIC studio.

Step 4. In a moment, you will see the message box asking if you open/save the pbstv10.zip or not. Click Open button. Then, installation will be proceeding.



Step 5. While Setup is proceeding, you can set installation option such as destination folder, program folder, etc. Please keep your eyes on screen and follow the message appearing at each stage of installation.

GETTING LATEST VERSION OF PICBASIC STUDIO

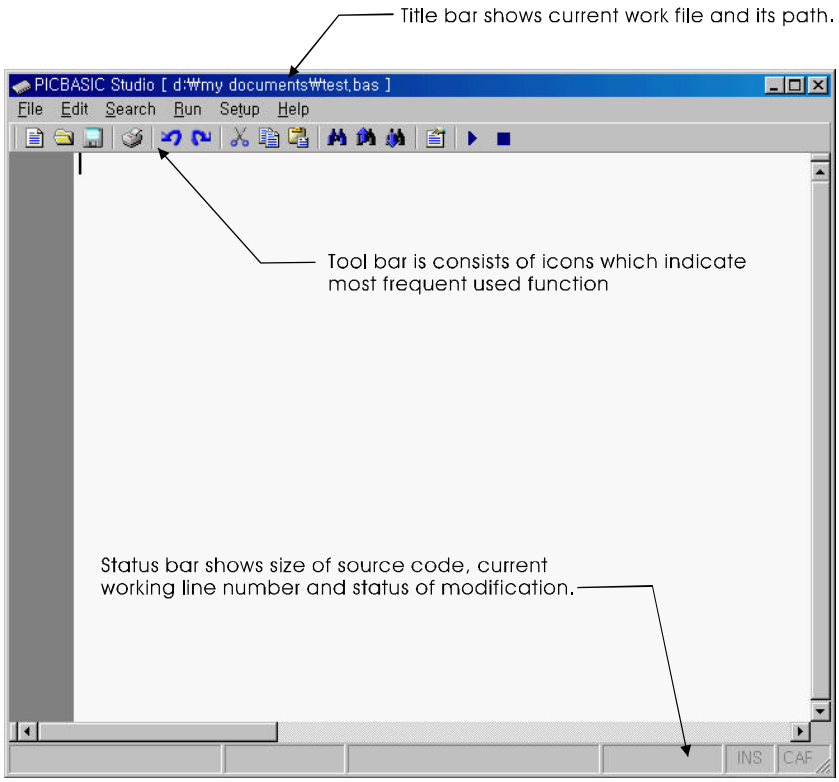
We, Comfile Technology, upgrade our software almost every month irregularly. Therefore, to get latest version of PICBASIC studio, we recommend you to visit our web site for software upgrade regularly at least every month. The download page for international users is **www.comfile.co.kr/english/download.html**.

SPECIAL NOTICE FOR USING REV.B CABLE ON PICBASIC-STUDIO

Rev.B cable is a intelligent cable containing a microprocessor and is dedicated for PICBASIC modules, PB series and PBM series.
If you are using MS Windows XP, you should allocate the LPT1 to HP Laser jet 4. Because that the driver of HP Laser jet4 is in MS Window XP driver package, you don't have to install it separately.
If you don't allocate LPT1 to HP Laser jet 4, you will encounter the error message as 'PICBASIC module not found".

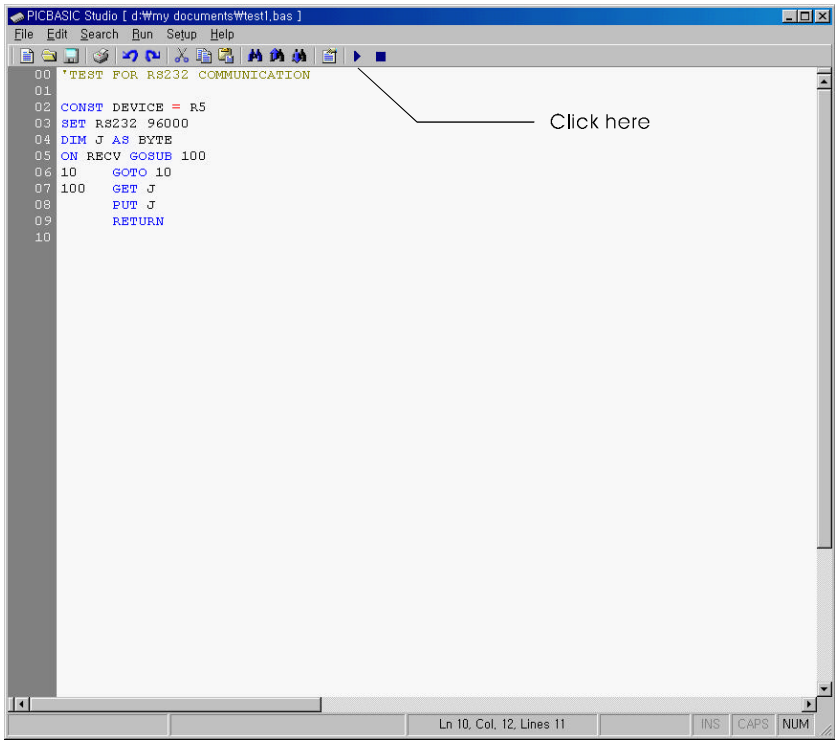
GETTING START WITH PICBASIC STUDIO

The following screen is initial state of PICBASIC studio.

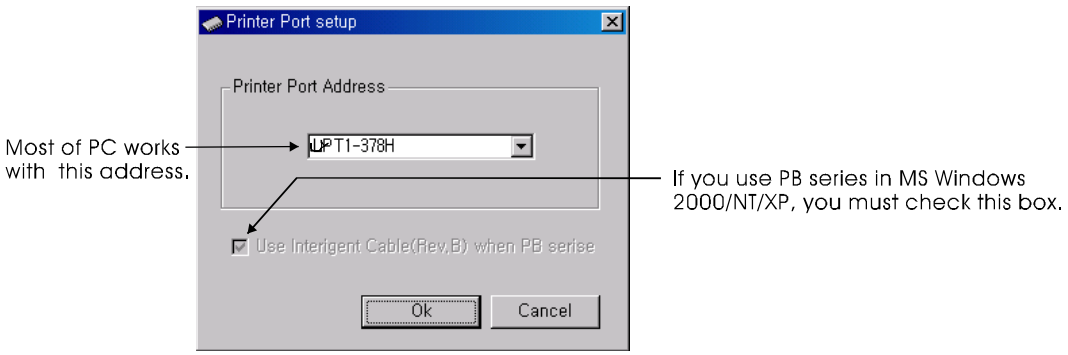


You can program your source just as a typical word process/editor. Do not forget to save your source after programming it.

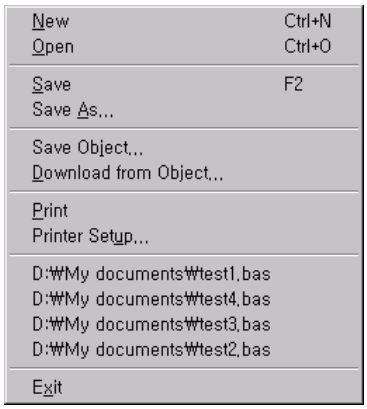
When you finish programming/saving your source, click Run button as below in order to execute the source. Run will check Syntax error in your source and, if no error is found, download the source into PICBASIC module. While those two processes, if there occurs no error, the PICBASIC module is ready to execute the source.



If you encounter the error message, “PICBASIC Module Not Found! Check Cable or Power” during download process, you should look into hardware component and their connection such as cable, electric power and etc. Electric power must be supplied from Target board.
After checking all components and their connection, if you figure out nothing wrong. You should adjust address of Printer port. You can find the Printer Port set box in Setup menu in the Tool bar. You can select one of 378, 278 and 3BC as address of the printer port.



FILE MENU

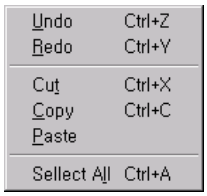


-New : make new file
-Open : open existing file
-Save : save current working file
-Save As : save current working file in different name
-Save Object : save only Object code from current working file
-Download from Object : download only Object code

SAVE/DOWNLOAD OF OBJECT CODE

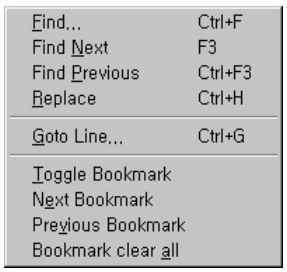
Generally, you can save/keep your source file to save it in the forms of BAS file. If you need to protect your source code, you should save only Object code from your source file. Because none can read your source file from the Object code file, you can keep your source file safe.

EDIT MENU



-Undo : cancel previous work
-Redo : execute previous cancellation
-Cut : cut marked text and place in clipboard
-Copy : copy marked text to clipboard
-Paste : paste clipboard text at cursor
-Select All : select all

SEARCH MENU



-Find : find text
-Find Next : find next occurring text
-Find Previous : find previous occurring text
-Replace : replace marked text
-Goto line : move cursor to specific line
-Toggle Bookmark : mark bookmark
-Next Bookmark : move to next bookmark
-Previous Bookmark : move to previous bookmark
-Bookmark clear all : clear all bookmarks

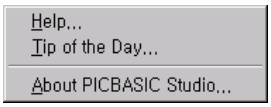
RUN MENU

-Run : execute SyntaxCheck, compile and download in turn
-Break : stop execution and start Debug mode
-Verify : compare source code
-Syntax Check : check syntax error
-View Object code : display Object code
-View Label List : display label list
-View PICBASIC Flash Memory: display content of PICBASIC module
-Check Variable : check value in specific variable
-Modify Variable : modify value in specific variable
-Clear Flash Memory : clear content of PICBASIC module

SETUP MENU

-Use Korean Menu : change language into Korean
-Editor Environment : show Editor setup
-Printer Port Setup : change address of printer port and set Rev.B cable

HELP MENU



-Help : Open PICBASIC manual
-Tip of the Day :
-About PICBASIC Studio : display version

CONST DEVICE

When you work in PICBASIC studio. You must let PICBASIC studio what kind of device you work with by declaring in the first part of your program with CONST DEVICE as follows.

CONST DEVICE = R5

Please refer to Chapter 2 Instructions for details about CONST DEVICE.

EDITOR FUNCTION KEYS

The following function keys can be available in Editor state.

UP arrow	: move cursor up one line
DOWN arrow	: move cursor down one line
LEFT arrow	: move cursor left one line
RIGHT arrow	: move cursor right one line
Ctrl + UP arrow	: move cursor up
Ctrl + DOWN arrow	: move cursor down
Ctrl + LEFT arrow	: move cursor left
Ctrl + RIGHT arrow	: move cursor right
Home	: move cursor at beginning of line
End	: move cursor at end of the line
Ctrl + Home	: move cursor at beginning of program
Ctrl + End	: move cursor at end of program
PgUp	: move cursor up one page
PgDn	: move cursor down one page
Ctrl + F	: Find
Ctrl + Z	: Undo

BLOCK CONTROL

SHIFT + arrow	: highlight one character to direction of arrow
Ctrl + X	: cut marked text and place in clipboard
Ctrl + C	: copy marked text to clipboard
Ctrl + V	: paste clipboard text at cursor
Ctrl + A	: highlight all

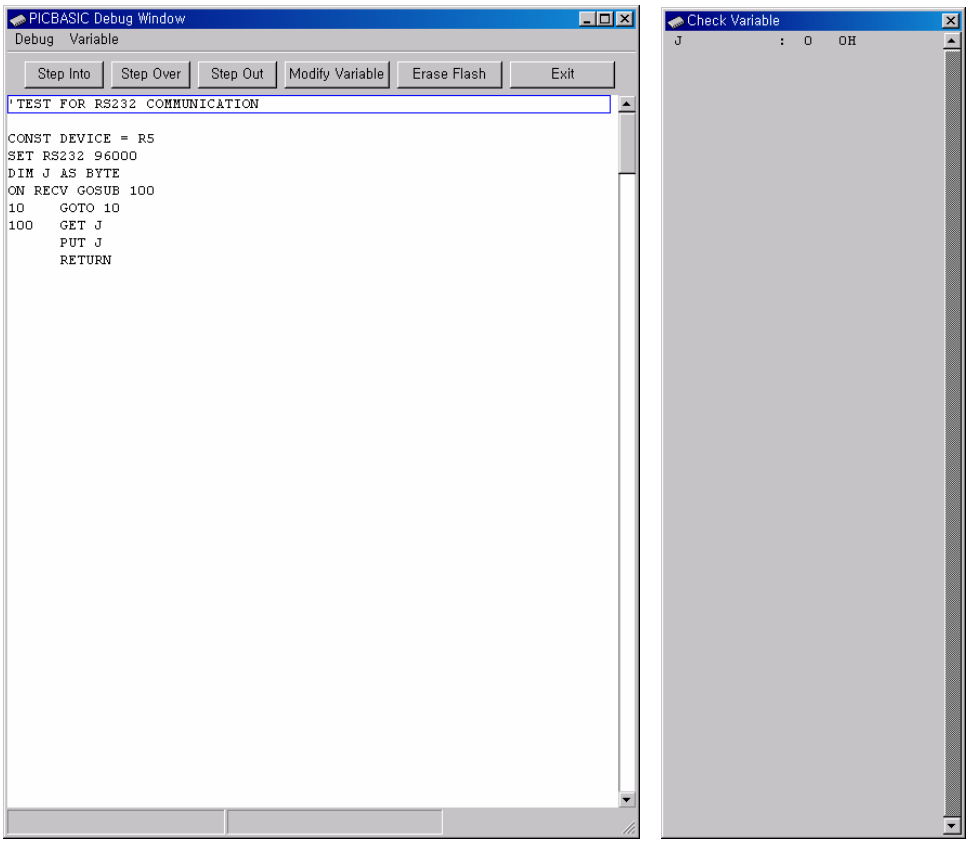
FILE CONTROL

Ctrl + X	: Open another file
F2	: Save current working file

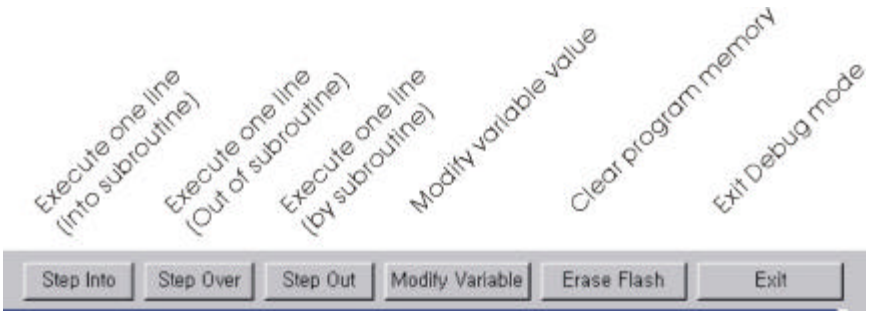
DEBUGGING

BREAK & CHECK STEP OPERATION

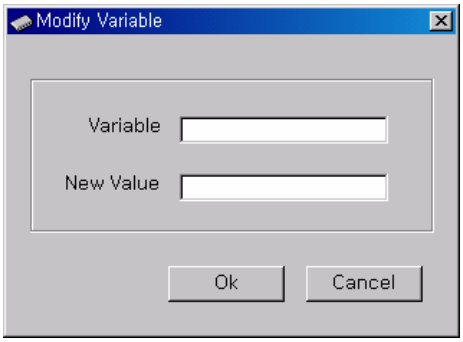
While a PICBASIC module is connected with your PC, you can execute BREAK at any time. Once, you execute BREAK, PICBASIC studio goes into DEBUG mode and shows DEBUG window as follow. The DEBUG window shows location where the execution stop and variable value etc.



In the state of Debug mode, you can look into variation of variable, state of I/O, etc by executing Step Into. In order to restart program, click Exit button. (Only Seep Into mode is available in PB series.)

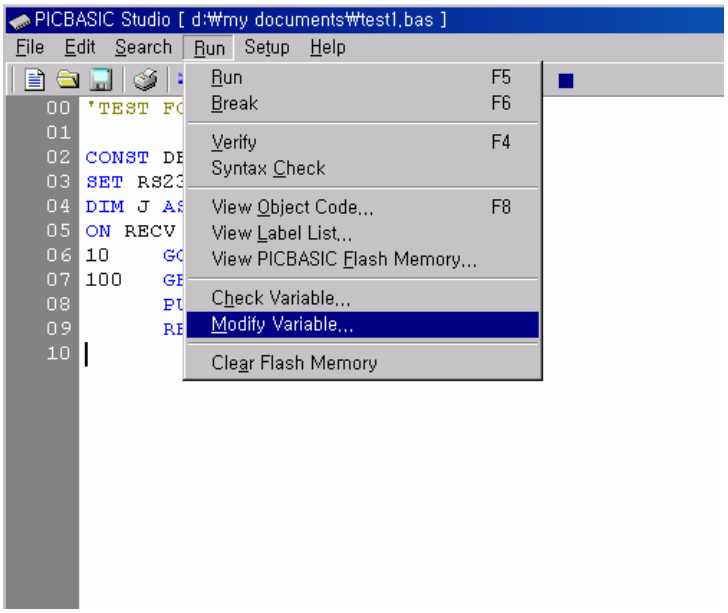


You can modify value of specific variable in the Modify Variable box shown as below.



CHECKING VARIABLE IN OPERATION

This function is available in only PBM series. When you use PB series, it is not activated. This function allows you to check or modify value of specific variable. When you click Check Variable in the Run menu, current values of all variables are displayed on window.



Chapter 2.

Instructions

General rule of instructions

Rule of instruction in PICBASIC language is very similar with BASIC language. Instructions can be classified as commands and functions. A function is used as part of a expression and has round bracket at a instruction's tail.

- Command: PRINT, GOTO, RETURN
- Function: ADIN(0), EEREAD(0) <= Functions have round brackets at the end.

It is different to express constants in PICBASIC language according to scale notation.

- Decimal: 10, 20, 32, 1234
- Hexadecimal: &HA, &H1234, &HABCD
- Binary: &B10001010, &B10101

You can use lower-case letter in PICBASIC. However, make certain that the compiler of PICBASIC language cannot discern lower-case letters. i.e., PICBASIC compiler recognizes lower-case letters as capital letters. If you use lower-case letters and capital letters together, the compiler recognizes all of them as capital letter only. For a instance, "LoopCNT" is recognized as LOOPCNT.

Name of variables must start with a English character and must have less size than 255 characters. You cannot use name of command and function as name of variable.

- Valid variable name: A, B0, I, J, TH, BF1
- Invalid variable name: 23, 3A, INPUT, GOTO

Declaration of variable must be placed at the first part of program just as followings.

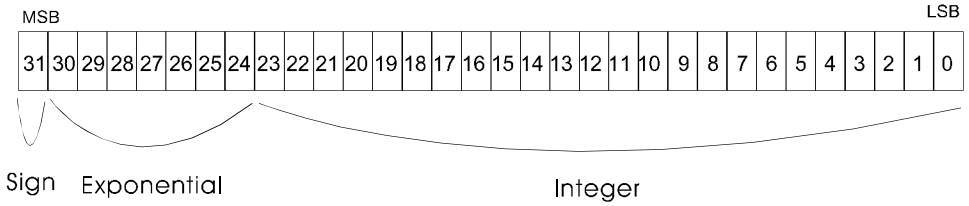
```
- DIM  I  AS  BYTE                    ' Delimiting I as BYTE type
- DIM  J  AS  INTEGER                 ' Delimiting J as INTEGER type
- DIM  I  AS  BYTE, J  AS  BYTE
```

Variable type

PICBASIC language uses five types of variable. PB series can use only two types of variable, BYTE type and INTEGER type, while PBM series use all five types of variable.

- ***Byte*** : 8bit number without sign (0~255)
- ***Integer*** : 16bit number without sign (0~65535)
- ***Long*** : 32bit number with sign (-2,147,483,648~2,147,483,647)
- ***Single*** : 32bit floating point number with sign
- ***String*** : 8bit character (extended up to 92 byte at most)

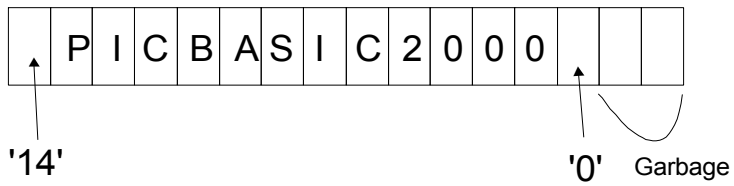
Byte variable is a type of 8bit number without sign and reserves 8bits (1byte) of memory space. Therefore, its expression range is 0~225.
Integer variable is a type of 16bit number without sign and reserves 16bits (2bytes) of memory space. So, its expression range is 0~65535.
Long variable is a type of 32bit number with sign and reserves 32bits (4bytes) of memory space. So, its expression range is -2,147,483,648~2,147,483,647.
Single variable is a type of 32bit of floating point number with sign. In accordance with recommendation of IEEE 754, it consists of seven bits of exponential part, twenty three bits of point part and one sign bit.



String variable is available in PBM series only. It can be 93byte at most and must be used with value indicating its maximum bytes for reserving memory space. Refer to the following example.

```
DIM ST AS STRING * 14       'Declare ST as string variable with 14byte of maximum bytes
```

The following picture describes how string data is stored in memory when maximum bytes of string variable is 14 bytes. If the maximum byte is 14, 2byte of additional memory is reserved. One byte is for the value indicating the maximum byte. The other byte is for NULL value indicating end of string.



In the above picture, **PICBASIC2000,** 12bytes of string data, is stored in 14bytes of memory space. In this case, the rest, 2bytes, is filled with unknown garbage value.

```
DIM ST AS STRING * 16                ' Declare ST as string variable with 16byte of
                                     ' maximum bytes
ST = "COMFILE TECHNOLOGY"            'Because data, COMFILE TECHNOLOGY, is out of
                                     'Reserved range (16bytes), last two characters, GY, will
                                     'Not be stored.
```

As above example shows, because string variable cannot contain data exceeding maximum bytes, you should be prudently determine the maximum bytes.

Using "+," you can bind strings. When you bind strings, you make certain that the result string must have larger bytes than sum bytes of combined strings.

```
SG = SG + ST                    'Bind strings
```

CAUTION

All types of variable of PBM series will not be initialized to specific value on RESET while that of PB series is initialized. Therefore, when you work with PBM series, you had better definite initial value of variables clearly before using them.

Defining constant

Sometimes, it is convenient for maintaining a program to express numbers in characters. Therefore, PICBASIC language provides an instruction, CONST, for that.

```
CONST   RELAYPORT = 5       ' Hereafter, RELAYPORT is substituted for "5"
OUT RELAYPORT, 1            ' You can understand at once that RELAY turns ON.
```

You can express Single data in characters as well as Integer data.

```
CONST   PIE = 3.14          ' Define Single data as constant
DIM K AS SINGLE
K = PIE
```

Array

You can also define Byte type one-dimensional array up to 65535 elements in PICBASIC.

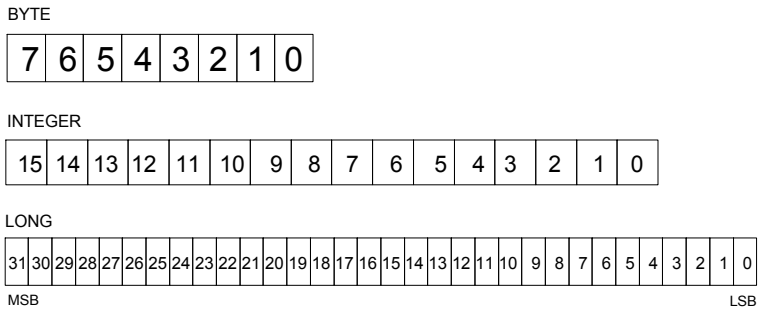
```
DIM     A(20)   AS   BYTE              ' Define 20 of A array
DIM     B(200)  AS   INTEGER           ' Integer array
DIM     C(200)  AS   LONG      ' Long array
DIM     D(200)  AS   SINGLE            ' Single array
```

Parameter of array starts from 0. Therefore, when 20 elements are defined, you can use from 0 to 19. Because that PB modules have limited size of data memory, maximum definition range of array is same as size of data memory. i.e., PB-1S has 96bytes of data memory. Therefore, it is possible to use array up to 96bytes. ***You cannot use array as parameter of array as follows.***

```
I = ARRAY1 (K(J))
```

Bit designator

It is possible to access specific bit of variable by bit pointer. (Bit pointer is available from 0 to 31.) Bit pointer of each variable type is as follow. PB series use dot (.) as bit pointer while PBM series use colon (:). Bit pointer is not available with Single variable.



PBM series DIM I AS BYTE, A AS BYTE

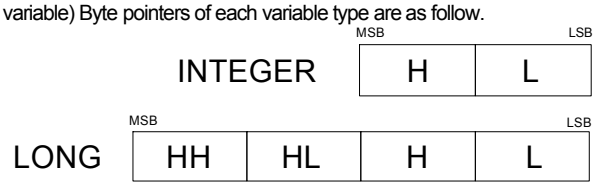
```
I:7 = 0          ' Input 0 into 7th bit of Byte variable I
I:7 = A:2        ' Transmit 2nd bit of variable A to 7th bit of variable I
```

PB series DIM I AS BYTE, A AS BYTE

```
I.7 = 0          ' Input 0 into 7th bit of Byte variable I
I.7 = A.2        'Transmit 2nd bit of variable A to 7th bit of variable I
```

Byte designator

It is possible to access specific lower byte/higher byte in Integer/Long variable by using byte pointer. (Byte pointer is not available with Byte/Single variable) Byte pointers of each variable type are as follow.



Because Integer variable is 2 bytes, it can be divided into H and L while Long variable can be divided into HH, HL, H and L due to 4 bytes.

```
DIM I AS INTEGER
DIM K AS LONG
I:H = 0          'Input 0 into higher byte of I, Integer variable
I:L = 0          'Input 0 into lower byte of I, Integer variable
K:HH = 0         'Input 0 into higher-high byte of K, Long variable
K:HL = 0         'Input 0 into higher-low byte of K, Long variable
K:H = 0          'Input 0 into lower-high byte of K, Long variable
K:L = 0          'Input 0 into lower-low byte of K, Long variable
```

PB series also use dot(.) as byte pointer while PBM series use colon(:).

```
I.H = 0
I.L = 0
M = I.H
```

Constant array

Constant is invariant value while a program operates. PICBASIC provides a function allowing user to definite constants like array. This function is very useful to handling a lot of data. You can use data defined as constant array as array in program. The following example shows how to definite/use constant array.

```
CONST BYTE DATA1 = (31, 25, 102, 34, 1, 0, 0, 0, 0, 0, 65, 64, 34)
I = 0
A = DATA1 (I)          'Return 31
I = I + 1
A = DATA1 (I)          'Return 25
```

Other data type of constant array is also available.

```
CONST INTEGER DATA1 = (6000, 3000, 65500, 0, 3200)
CONST LONG DATA2 = (12345678, 356789, 165500, 0, 0)
CONST SINGLE DATA3 = (3.14, 0.12345, 1.5443, 0.0, 32.0)
```

You can extend data used for constant array in several lines.

```
CONST BYTE DATA1 = (31, 25, 102, 34, 1, 0, 0, 0, 0, 0, 65, 64, 34,
                    12, 123, 94, 200, 0, 123, 44, 39, 120, 239,
                    132, 13, 34, 20, 101, 123, 44, 39, 12, 39)
```

You can use string data as constant array.

```
CONST BYTE DATA1 = ("I LOVE PICBASIC 2000", 13, 10, 0)
```

The difference between general array and constant array is that **data defined as constant array shall be recorded in program memory when it is downloaded into a PICBASIC module.** General array is recorded in data memory (SRAM) and is not conserved after RESET. But, constant array remains in program memory even after RESET.

Content	Array	Constant array
Storage	Data memory (SRAM)	Program memory(FLASH)
Record	In operation	In downloading
Modification during operation	Available	N/A
Used for	Storing variant data	Storing invariant data such as constant
After RESET	Initialization	Conserved

Expression

Expression in PICBASIC forms as follows.

```
I = 0                    ' Input 0 to variable I
I = I + 1                ' Increase variable I by 1
I = J * 12 / K           ' Multiply variable J by 12 and, divide by variable K
```

PBM series can execute Integer number and floating point number operation. Operation with each types of variable must be done with corresponding variables and constants. For instance, Integer operation must consist of Integer variables and constants. If you want to use an Integer variable in floating point number operation, you should convert the Integer variable into a Single variable by CSNG instruction.

```
DIM   S1 AS SINGLE
DIM   I AS LONG
S1 = S1 + 1              'Valid expression
S1 = S1 + 1.0            'Invalid expression.
                         'All constant in floating point expression must have point.
S1 = S1 + I              'Invalid expression.
                         'Used Integer variable and Single variable together
S1 = S1 + CSNG(I)        'Valid expression
                         'Used CSNG instruction for converting variable type
```

All Integer operations proceed internally with Long type data. The result of an operation is converted into the form of destination variable and, is stored at a destination variable. For instance, if a destination variable is Byte variable, the result of an operation is stored in the form of Byte type even though the result exceeds 255. (Make sure that value exceeding 8bits is cut.)

```
DIM   I AS LONG
DIM   J AS BYTE
I = 255
J = I + 1                'Result become 0 because variable J is Byte type.
```

The following table shows operators available in PICBASIC.

Arithmetical operation (Available in all variable types)		Logical operation and shift (Available in Integer variable only)	
+	Addition	AND	Logical operation AND
-	Subtraction	OR	Logical operation OR
*	Multiplication	XOR	Logical operation XOR
/	Division	<<	Left shift
		>>	Right shift
		MOD	remainder of division

Operation priority is order of multiplication, division, bit-operation, addition and subtraction.

```
I = J + 12 * K           'Multiply 12 by K and, add J
I = J + I AND &HF        'Execute AND operation with I and &HF and, add J
```

Complicated operation has possibility of occurring errors during compiling. In that case, it is recommended to divide the operation into several simple operations.

```
I = (J * K) + L / 4        ⇨        I = J * K
                                    I = I + L / 4
```

In case that an operation makes no temporary data in performing operation, it does not matter how long the expression is.

```
I = J * K * L / 4 + 100
```

CAUTION

PICBASIC does not allow using a function inside of other function.

```
PRINT DEC(ASC(ST))              ' This instruction must be modified as below.

I = ASC(ST)
PRINT DEC(I)
```

In case of PBM series, you can divide a line of expression into more than one line by using "_". (This is not available in PB series)

```
I = TABLE(J,192, 12, 13, 142, 123, 0, 0, 0, 1, 2, 3,_
          234, 192, 14, 90, 100, 200, 0, 0, 0, 1)
```

In addition, it is inhibited to use operating expression in the middle of command.

```
IF  A+2 = 0 THEN K =0           ' Improper usage of operation

  B = A + 2
IF B=0 THEN K = 0
```

Floating point operation in PB series

To use floating point number in PB series, you need some trick. For instance, when you multiply 200 by 3.14, you cannot the operation in PB series. In this case, if you use Scaling method, you can obtain same (or approximated) result as that of Floating point operation. Refer to the following sample program.

```
        '
        '200 * 3.14 (Desired result is 628)
        '
        SET PICBUS HIGH
        LCDINIT
        DIM I AS INTEGER
        I = 200
        I = 200 * 3
        I = 200 * 1 / 10 + I
        I = 200 * 4 / 100 + I
        LOCATE 0,0
        PRINT DEC(I)            ' LCD displays 628.
10      GOTO 10
```

When you want to convert a value (0~255) of a A/D port into other value (0~1000),

1000 / 256 = 3.90625

You multiply the value of A/D port by 3.90625. The program for this is as follows;

```
        SET PICBUS HIGH
        LCDINIT
        DIM I AS INTEGER
        DIM J AS INTEGER
10      I = ADIN(0)
        J = I * 3
        J = I * 9 / 10 + J
        LOCATE 0,0
        PRINT DEC(I)                ' Value of A/D is on the upper line.
        LOCATE 0,1
        PRINT DEC(J)                ' Result of conversion is on the lower line.
        GOTO 10
```

Because only 3.9 is used for above multiplying, there is some error in the result of conversion. Though the maximum value is 994, it is still useful trick in PB series.

Instructions

PBM : PBM series only
PB : PB series only

Declaration instructions

	DIM	Declare variable and array
	CONST	Declare constant
	CONST BYTE	Declare Byte constant array
	CONST INTEGER	Declare Integer constant array
PBM	CONST LONG	Declare Long (32bit) constant array
PBM	CONST SINGLE	Declare Single constant array

Flow control instructions

	IF...THEN	Evaluate a condition and, if it is true, go to appointed line
	FOR...NEXT	Repeat execution of instructions
	GOTO	Branch to specific line
	GOSUB..RETURN	Call a subroutine and return

Digital I/O instructions

	IN()	Read status of port
	BYTEIN()	Read eight ports simultaneously
	OUT	Set status of port HIGH or LOW
	BYTEOUT	Change eight ports simultaneously
	OUTSTAT()	Read value being outputted from a port
	TOGGLE	Reverse current value being outputted
	PULSE	Reverse output state during specific period (Pulse output)

RS232C instructions

	SERIN	Software instruction for receiving RS232C
	SEROUT	Software instruction for transmitting RS232C
PBM	GET	Hardware instruction for receiving RS232C (Input port is 14^{th})
PBM	PUT	Hardware instruction for transmitting RS232C (Output is 15^{th})
PBM	SET RS232C	Hardware instruction for initializing RS232C
PBM	BLCR	Initialize receiving buffer of RS232C
PBM	BLEN ( )	Return number of data received at receiving buffer of RS232C

Shift I/O instructions

	SHIFTIN	Shift input (for I^2C, SPI, communication)
	SHIFTOUT	Shift output (for I^2C, SPI, communication)

Analog I/O instructions

	ADIN()	Read AD conversion value
	PWM	Output PWM wave
	PWMOFF	Stop output of PWM wave
PBM	DACOUT	Output PWM wave for D/A conversion during specific period (8bit)

Sound instructions

SOUND	Generate semplice sound
BEEP	Generate key-touch sound
PLAY	Play music

LCD control instructions

LCDINIT	Initialize a LCD
CLS	Clear LCD screen
LOCATE	Appoint location of display
PRINT	Display capital or digit
CSRON	Turn a cursor ON
CSROFF	Turn a cursor OFF
BUSOUT	Transmit specific code to PICBUS
SET PICBUS	Change transmission speed of PICBUS to 19200 or 4800 baud rate

EEPROM access instructions

EEWRITE	Write data at specific location of EEPROM
EEREAD	Read data from specific location of EEPROM

Interrupt instructions

	ON TIMER GOSUB	Execute specific routine periodically
	ON INT GOSUB	Execute specific routine by generating edge at port 8th
PBM	ON RECV GOSUB	Execute specific routine by interrupt of hardware RS232C
PBM	SET ONTIMER ON/OFF	Timer interrupt ON/OFF
PBM	SET ONINT ON/OFF	Edge interrupt of port 8th ON/OFF
PBM	SET ONRECV ON/OFF	RS232C receiving interrupt ON/OFF

Key input instructions

	ADKEYIN()	Receive key-input by A/D conversion
	PADIN()	Read keypad (4*4 key matrix)
PBM	EPADIN()	Read keypad (8*8 key matrix)
	KEYIN()	Read status of port
PB	KEYDELAY()	Adjust value of delay and repeat on key-input.

Real time instructions

PBM	TIME()	Read real-time data
PBM	TIMESET	Set time

Location control instructions

PB	FREQOUT	Generate random frequency (Using PWM port)
PB	CAPTURE	Measure cycle of outer wave
PB	STEPOUT	Output pulse for controlling STEP motor
PB	SERVO	Control position of RC SERVO motor

Instructions relating to string

PBM	DEC()	Convert constant/variable to decimal string
PBM	HEX()	Convert constant/variable to hexadecimal string
PBM	FLOAT()	Convert real type constant/variable to real type string
PBM	LEFT()	Extract characters from left part of string
PBM	RIGHT()	Extract characters right part of string
PBM	MID()	Extract characters center part of string
PBM	ASC()	Return ASCII code of first character in string
PBM	CHR()	Convert constant/variable to character
PBM	VAL()	Convert digit character of string to value
PBM	VANSNG()	Convert real digit character of string to value

Instructions relating to mathematical function

PBM	SIN()	Return value of SIN
PBM	COS()	Return value of COS
PBM	ABS()	Return absolute value
PBM	EXP ()	Return exponential function e^X
PBM	LOG()	Return value of natural logarithm
PBM	LOG10()	Return 10 based natural logarithm
PBM	SQR()	Return value of square root
PBM	POW()	Compute X^Y

Conversion instructions

PBM	CINT()	Convert into Integer type
PBM	CLNG()	Convert into Long type
PBM	CSNG()	Convert into Single type

Other instructions

	RND()	Generate random value
	TABLE()	Transform a table by parameter
	ON..GOTO	branch by parameter
PBM	ON..GOSUB	Execute subroutine by parameter
	BREAK	Generate break
	COUNT()	Read value of count from CLKIN terminal
	PEEK()	Read content of specific address(RAM) on main microprocessor
	POKE	Change content of specific address (RAM) on main microprocessor
	RESET	Reset
	DELAY	Set 1~65535 mS of delay
	CAPTURE	Measure cycle of outer wave

ABS()

PBM

*Single variable = ABS (*value*)

Absolute value

Value is Single constant/variable.

EXPLANATION

This instruction returns the absolute value of an input value. The input value must be a Single constant/variable. If input value is not Single type, you should convert the value into Single type by CSNG instruction before using this instruction.

EXAMPLE

```
DIM F1 AS SINGLE
DIM F2 AS SINGLE
F2 = -1234.5678
F1 = ABS(F2)                ‘1234.5678 is saved at F1
```

Because that Byte/Integer variable is not singed value, there is no need of ABS instruction on those two types of variable. However, because Long variable is singed variable, it should be used as the following example.

```
F1 = CSNG(L1)               ‘Execute CSNG instruction first
F1 = ABS(F1)                ‘Execute ABS instruction
L1 = CLNG(F1)               ‘Execute CLNG instruction
```

“*” means the result of this instruction becomes Single type.

ADIN()

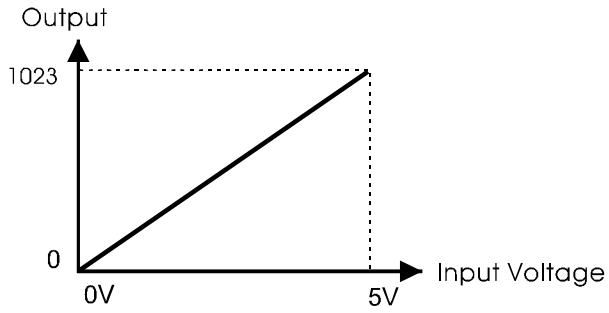
*Integer variable = ADIN (*port*)

A/D conversion

Port is an Integer variable/constant (0-33) acceptable A/D input.

EXPLANATION

This instruction, which is for receiving input of A/D conversion, converts analog value inputted at appointed ***port*** into 8~10bits of digital value (0~255 or 0~1023), and return the digital value. (Converting 0V~5Vof voltage inputted into digital value). For instance, in case of 10bits conversion, it will be 0V to 0, 5V to 1023 and 2.5 to 512.



EXAMPLE

```
I = ADIN(0)              'Execute A/D conversion at port 0 and store the result at variable I
SEROUT 8,93,0,0,[I]      'Transmit the result to RS232C
```

ADDITIONAL INFORMATION

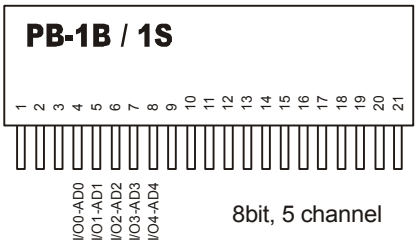
PBM-R5 has extra 12bit A/D input port. When you use ADIN instruction with port 32 and port 33, 12bits A/D conversion is executed. (0~ 4095)

```
DIM K AS INTEGER         'Define K as Integer variable (16bits)
K = ADIN(32)             ' Execute 12bits A/D conversion at port 32, and store the result at K
```

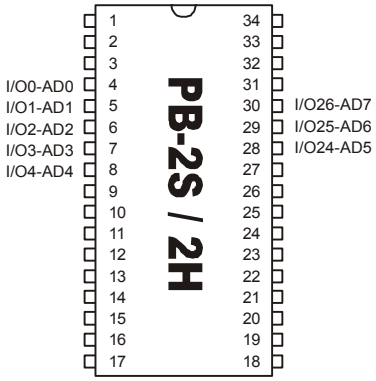
Port 32 and Port 33 cannot be used as general I/O. they are used for only 12bits A/D input.

"*" means the result of this instruction becomes Integer type.

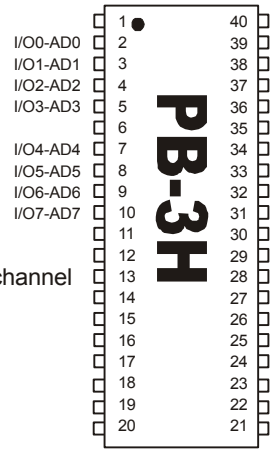
The following pictures show A/D input port of each PICBASIC modules.



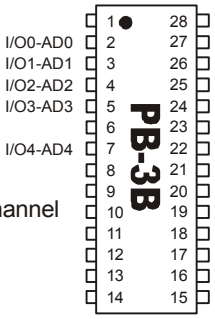
8bit, 5 channel



8bit, 8 channel

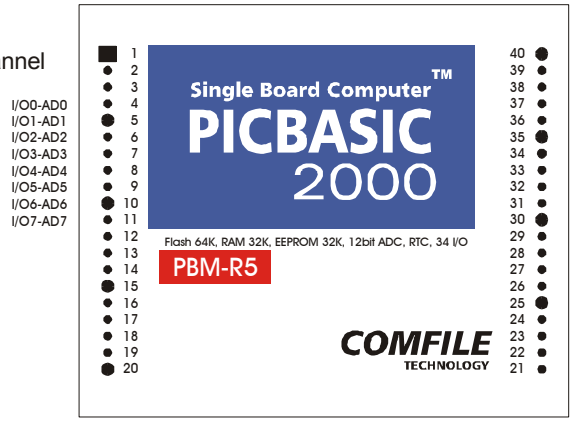


10bit, 8 channel



10bit, 5 channel

10bit, 8 channel



12bit, 2 channel

ADKEYIN ()

Integer variable = ADKEYIN (*port*)

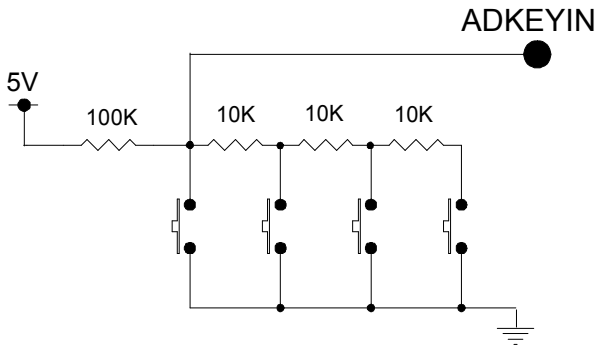
A/D conversion key-input

Port is a Byte variable (0~33) or a port number acceptable A/D input.

EXPLANATION

This instruction receives key inputs using A/D conversion function. ***Port*** must be able to receive A/D input. In the following circuit, 0V is inputted on pushing the first key. 500mV is inputted on pushing the second key. 800mV is inputted on pushing the third key. When all key is released, 5V is inputted. One A/D input port can receive up to ten key inputs at most by using ADKEYIN instruction.

EXAMPLE




```
10   I = ADKEYIN(0)          'Receive A/D key input at port 0.
     PRINT HEX(I)            'Display key value on LCD
     GOTO 10
```

ADDITIONAL INFORMATION

You need to organize 1~10 of extra circuits so that you use ADKEYIN instruction. With the instruction, you can receive a number of key-input through only few ports. When there are eight of A/D ports, you can receive 80 of key-inputs at most. ADKEYIN instruction is not available at 12bit A/D ports.

When symptoms such as incomplete key-input, improper recognition of key-input and etc occur, adjust resistor's value (100K?). Because the symptoms are caused from A/D conversion error by influence of noise etc, they could be resolved by adjusting value of pull-up resistor.

ASC ()

PBM

Integer variable = ASC (*character*)

ASCII code of character

Value is a character (Character is surrounded by " mark. Example is "A") or a Sting variable.

EXPLANATION

This instruction returns ASCII code of the first ***character*** in a string.

EXAMPLE

```
DIM ST AS STRING * 16          'Set 16 as maximum length of string
DIM I AS BYTE
ST = "A"
I = ASC(ST)                    '&H41, which is ASCII code of "A", is stored at variable I.
```

The following table shows ASCII code.

Upper 4bits \ Lower 4 bits	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	B	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	R	s	t	u	v	w	x	y	z	{	|	}		

Read upper 4bits first, and read lower 4bits. For instance, "A" is &H41 and "W" is &H57.

BCD ()

Integer variable = BCD (*value*)

BCD conversion

Value is a constant (0~65535) or a Byte/Integer variable.

EXPLANATION

This instruction converts ***value*** into BCD code. BCD (Binary Coded Decimal) code is decimal expressed by 4bits. Therefore, 16bits is expressed by the four digits and 32bits by the eight digits. If a variable has enough size to restore the result of conversion, upper value will disappear. For an instance, if you save BCD code,12345, into INTEGER type variable, only 2345 will be saved.

Before BCD conversion (Decimal)	After BCD conversion (Decimal)	After BCD conversion (Hexadecimal)
1	1	&H01
2	2	&H02
3	3	&H03
4	4	&H04
5	5	&H05
6	6	&H06
7	7	&H07
8	8	&H08
9	9	&H09
10	16	&H10
11	17	&H11
12	18	&H12
13	19	&H13
14	20	&H14
15	21	&H15

EXAMPLE

```
DIM A AS INTEGER, I AS INTEGER
A = 13
I = BCD(A)              'Convert value of variable A into BCD code and store the result at I.
                        'The result is 19.
LOCATE 0,0
PRINT DEC(I)            'Display 19, the result, on LCD
```

BEEP

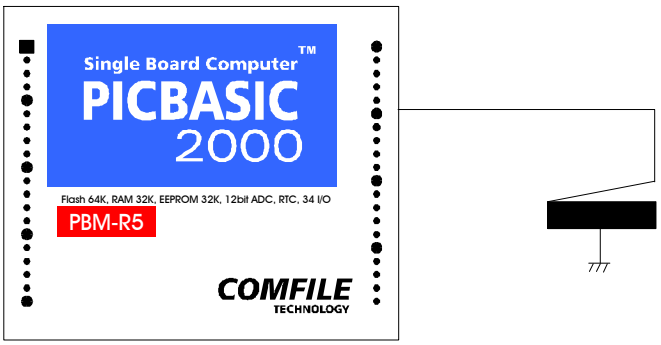
BEEP *port*

Generate key-touch sound

Port is a constant (0~33) or a Byte variable.

EXPLANATION

This instruction helps you to generate key touch sound simply. Connect one of PICBASIC ports to PIEZO, and use the instruction to generate sound which will be generated around 2KHz frequent shortly). PBM series can use the instruction at port 0~15.



EXAMPLE

```
20   IF KEYIN(1)=1 THEN GOTO 20      'Receive a key input at port 1 (it becomes 0 when pushed)
     BEEP 10                         'When the key is pushed, short BEEP sounds.
     GOTO 20
```

BCLR

PBM

BCLR

Clear RS232 buffer

EXPLANATION

This instruction clears buffer of hardware RS232C. Buffer of RS232C is cleared on power-on. After execution of SET RS232C instruction, data received at RX pin are saved in RS232C buffer. BCLR instruction clears all data in the buffer.

BLEN (0)

PBM

Integer variable = BLEN (*0*)

Number of data at RS232C

EXPLANATION

This instruction returns number of data received at buffer of RS232C. The bracketed "(***0***)" is meaningless dummy value.

EXAMPLE

```
      DIM I AS BYTE, J AS BYTE, K AS BYTE
      SET RS232 9600                    'Activate RS232C at 9600 baud rate
      ON RECV GOSUB 100                 'Execute receiving one hundred times
      OUT 19,0
10    PULSE 19,100
      DELAY 100
      IF KEYIN(9) = 0 THEN              'Clear RS232C buffer when key 9 is pushed
      BCLR
      END IF
      GOTO 10

100   K = BLEN(0)                       'Execute ECHO after reading number of data in buffer
 FOR J = 1 TO K
            GET I
            PUT I
      NEXT
      RETURN
```

BREAK

BREAK

Break

EXPLANATION

A program breaks execution on encountering this instruction in executing and sending DEBUG information to PCP. Because this instruction is deducted for debug mode, it can not operate at state that a PICBASIC module is not connected with PC. Execution of this instruction changes PICBASIC studio window to DEBUG mode window displaying the location where execution stop and contents of each variables and so on. Refer to description of "DEBUGGING" in end of this chapter.

CAUTION

If you insert BREAK instruction into an endless loop, it makes endless break causing logical error in operation. Therefore, you should use BREAK instruction in where the BREAK instruction executes once or by key input. In order to escape from endless loop executing BREAK instruction, you should use ERASE ALL button and get out of the loop.

BUSOUT

BUSOUT *value*, [*value*], ...

Transmit specific code to PICBUS

Value is a constant (0~255) or a Byte variable.

EXPLANATION

This instruction transmits numbers to PICBUS pin. PICBUS pin is dedicated port connecting with LCD display module. When you want to display specific code on LCD, use this instruction.

EXAMPLE

```
10   BUSOUT &HA0, &H01, &HA3          'Transmit A0,1,A3 to PICBUS
     GOTO 10
```

BYTEIN ()

Integer variable = BYTEIN (*port block*)

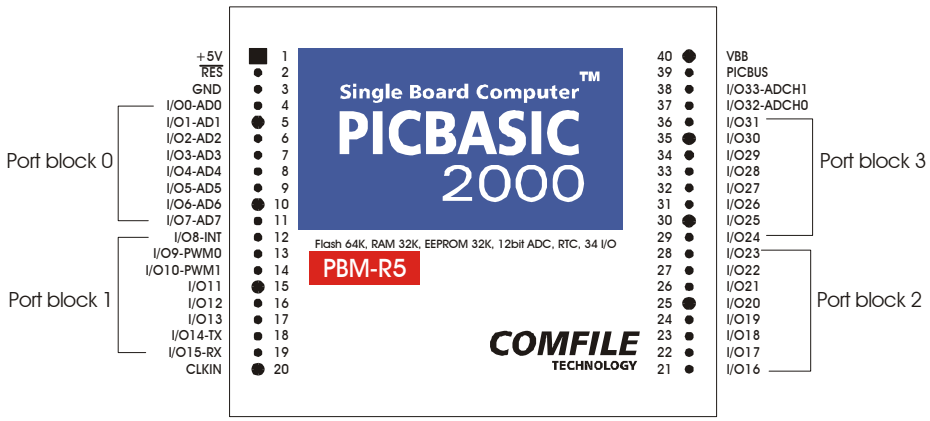
Input by byte

Port block is a port block number (0~3) or a Byte variable.

EXPLANATION

This instruction reads eight ports simultaneously and save them at a variable by 8bits. (Accessing by byte) Port block is a unit of combined eight ports. PB-1B and PB-1S have two port blocks and PBM series has four port blocks.

EXAMPLE




```
DIM I AS BYTE                ‘Declare I as Byte variable
I = BYTEIN(0)                ‘Read the state of port block 0
```

CAUTION

Because port blocks, which BYTEIN instruction is used in, are in input state, you cannot assign an output port at the port blocks. i.e. you must use the eight ports assigned to the port block as input port.

BYTEOUT

BYTEOUT *port block*, *value*

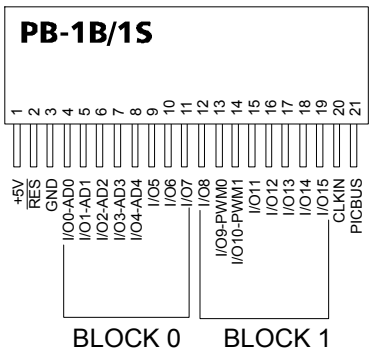
Output by byte

Port block is a port block number and is a constant (0~3) or a Byte variable.
Value is a constant (0~255) or a Byte variable.

EXPLANATION

This instruction output ***value*** to appointed ***port block***, which consists of eight ports. All ports in the ***port block*** assigned by BYTEOUT instruction become output state.

EXAMPLE




```
I = &HAB
BYTEOUT 0,I                ' Output content of I to port block 0
```

CAPTURE ()

PB

Integer variable = CAPTURE (*port, target*)

Capture pulse

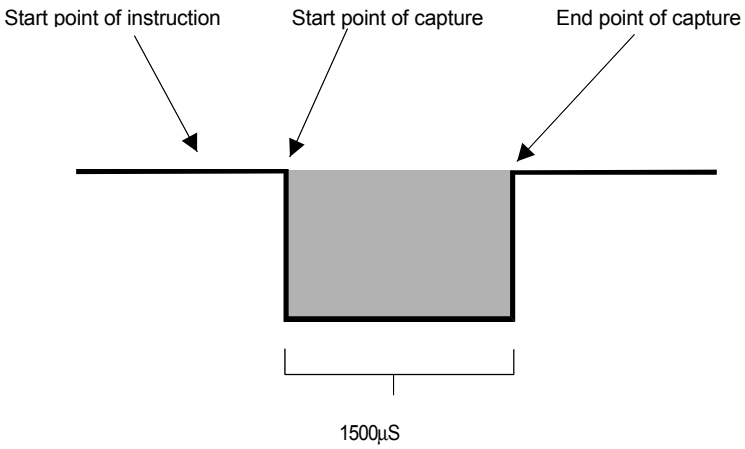
Port is a capture port and is a constant (0~3) or a Byte variable.
Target is a constant (0 or 1 only) and is not available with variable.

EXPLANATION

This instruction measures interval of incoming pulse (HIGH or LOW) from a ***capture port***. When a ***capture target*** is 0, LOW interval is measured. When the capture target is 1, HIGH interval is measured.

EXAMPLE

```
I = CAPTURE(0,0)          'Measure interval of LOW pulse at port 0
```



Right after executing of the instruction, the interval of the first incoming LOW pulse is captured and divided by a certain value(see following table) and returned. CAPTURE instruction starts at <Start point of instruction> and wait for detecting <end point of instruction>. If the <end point of capture> is out of maximum capturing range, it will return 0 not waiting for the <end point of instruction>.

	PB-1B/1S/2S	PB-2H	PB-3B/3H	PBM-R1/R5
Maximum capturing range	1.3sec	0.26sec	0.26sec	0.4sec
Return value	Captured interval/ 20	Captured interval/ 4	Captured interval/ 4	Captured interval/ 7

CHR ()

PBM

String variable = CHR (*value*)

Convert ASCII code into character

Value is a constant (0~255) or a Byte variable.

EXPLANATION

This instruction converts ASCII code into a corresponding character. For instance, CHR(&H41) returns “A”.

EXAMPLE

```
DIM ST AS STRING * 16          'Define 16 as the maximum size of String variable ST
ST = CHR(&H41)                 '"A" is stored at ST.
```

Refer to ASCII code table in description of ASC instruction.

CINT ()

Integer variable = CINT (*value*)

Convert Single type into Integer type

Value is a Single constant/variable.

EXPLANATION

This instruction converts Single variables/constants into 16bit Integer variables/constants.

EXAMPLE

```
DIM L1 AS INTEGER
DIM S1 AS SINGLE
S1 = 1234.5678
L1 = CINT(S1)
```

CLNG ()

PBM

Long variable = CLNG (value)

Convert Single type into Long type

Value is a Single constant/variable.

EXPLANATION

This instruction converts Single constants/variables into 32bits Long constants/variable. Digits beyond point are cut.

EXAMPLE

```
DIM L1 AS LONG
DIM S1 AS SINGLE
S1 = 1234.5678
L1 = CLNG(S1)                ' 1234 is stored at L1
```

CONST DEVICE

CONST DEVICE = *device*

Declare device

Device could be one of 1B, 1S, 2S, 2H, 3B, R1 and R5. (Use last two digit of model name)

EXPLANATION

This instruction is used to declare a kind of PICBASIC module at the first part of program.

EXAMPLE

```
CONST DEVICE = 3H          'Use 3H module
```

COS ()

PBM

Single variable = COS (*value*)

Compute COSINE value

Value is a Single constant/variable.

EXPLANATION

This instruction returns COSINE value of ***value***.(Angle by radian). ***Value*** must be SINGLE type. If value is not Single type, you must convert it into Single type by CSNG instruction.

EXAMPLE

```
DIM F1 AS SINGLE
F1 = COS(1.0)
```

COUNT ()

Integer variable = COUNT (*parameter*)

Count input

Parameter is a constant (0 or 1) and is not available with variable). 0= sustain, 1= clear.

EXPLANATION

This instruction returns the result of pulse count from CLKIN terminal. The count value of CLKIN terminal is counted by a hardware circuit, and is sustained internally. The instruction returns counted value. If ***parameter*** is 0, the instruction keeps current count value. If ***parameter*** is 1, the instruction clears the count value. Because size of internal counter is 16bits, the maximum countable number is up to 65535. Therefore, when count value is excess the 65535, it rolls over to zero.

EXAMPLE

```
I = COUNT(1)          'Transmit the result of count to I and clear count value
I = COUNT(0)          'Transmit the result of count to I and keep count value
```

The following example is for counting of pulse incoming from CLKIN during specific period.

```
    DIM I AS INTEGER
10  I = COUNT(1)          'Clear count value
    DELAY 100             'Delay for 100mS
    I = COUNT(1)          'Store incoming pulse for 100mS
    LOCATE 0,0
    PRINT DEC(I)          'Display the result on LCD
    GOTO 10               'Repetition
```

CSNG ()

PBM

Single variable = CSNG (*value*)

Convert Integer type to Single type

Value is a Byte, Integer and Long constant/variable.

EXPLANATION

This instruction converts Byte/Integer/Long constants/variable into Singer constants/variable. Because that PICBASIC language does not allow operating Single value and Byte/Integer/Long value together in a operation, you should temporary convert Byte/Integer/Long value into Single value by this instruction.

EXAMPLE

An operation on line 6 in the following program is combined with Single variable and Long variable.

```
DIM L1 AS LONG
DIM S1 AS SINGLE
DIM S2 AS SINGLE
S1 = 1234.5678
L1 = 333
S2 = S1 * L1
```

Therefore, the result of this operation is not predictable and the operation must be corrected as follows:

```
DIM L1 AS LONG
DIM S1 AS SINGLE
DIM S2 AS SINGLE
S1 = 1234.5678
L1 = 333
S2 = S1 * CSNG(L1)          'L1 should be applied to this operation after converting into Single type.
```

All constants used with Single value in an operation must have POINT as follow.

```
S2 = S1 * CSNG(L1) + 3.0
```

CSRON

CSRON

Turn on LCD cursor

EXPLANATION

This instruction turns on a LCD cursor.

CSROFF

CSROFF

Turn off LCD cursor

EXPLANATION

This instruction turns off a LCD cursor.

CLS

CLS

Clear LCD screen

EXPLANATION

This instruction clears LCD screen.

DACOUNT

PBM

DACOUT *port*, *duty rate*, *output*

D/A conversion/output

Port is a constant (0~31) or a Byte variable (PBM series is 0~15)
Duty rate is a constant (0~255) or a Byte variable.
Output is output number of waveform and is a constant (0~65525) or a Byte/Integer variable.

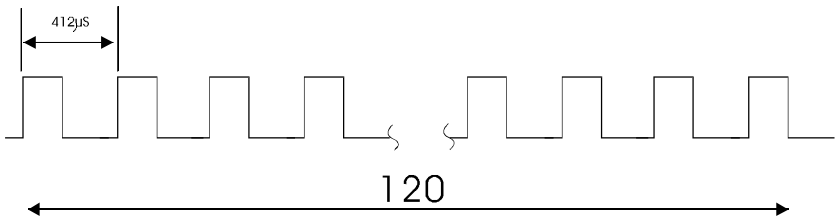
EXPLANATION

For D/A conversion, this instruction outputs PWM wave form to a specific port for appointed time, and make the port high-impedance state (HIGH-Z). To use the instruction, you need an extra circuit consisting of registers, condensers and OP amps. The instruction is similar as PWM instruction in operation. However, DACOUT instruction is free to select ports while PWM instruction is restricted from selecting ports. The following table shows the differences between DACOUT and PWM.

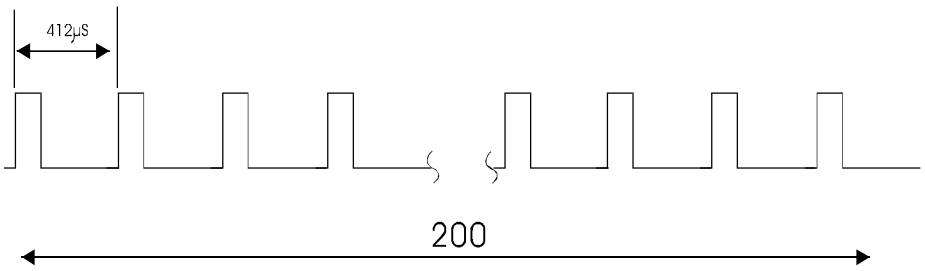
	DACOUT	PWM
PORT (in case of PBM series)	0 ~ 15	9, 10
Output method	software	hardware
Frequency	Fixed 2.4KHz	variable
Continuity of output	N/A	Keep outputting till PWMOFF

EXAMPLE

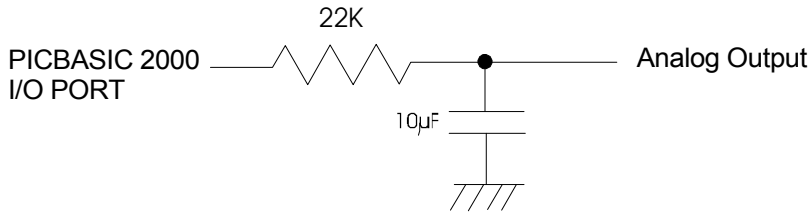
DACOUT 5,100,120



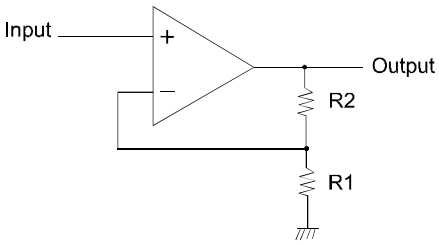
DACOUT 5,10,200



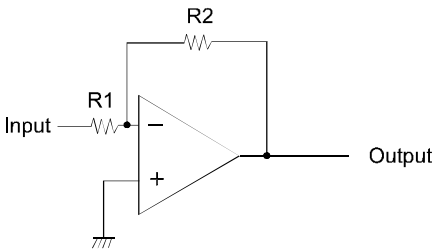
When there is no current consumer at output part, you can get output of D/A conversion easily by organizing the following circuit.



When there is a current consumer at output part, Voltage variation would be occurred. Therefore, you should build a buffer by using OP amp.



Noninverting amplifier circuit



Inverting amplifier circuit

DELAY

DELAY *mS*

Delay by mS

mS is a Byte/Integer constant/variable specifying the length of delay in milliseconds.

EXPLANATION

This instruction generates delay for specified time in mS. The range of delay is 0~65535.

EXAMPLE

```
DELAY 2000                ‘Delay 2000mS
```

DEC ()

DEC (*value*, *position*, *parameter*)

Convert into Decimal string

Value is a Byte/Integer/Long constant/variable.
Position is the maximum length of position. (Up to 10. Default is 5. Constant only)
Parameter is a constant (0 or 1 only) indicating to display "0" in front of a converted string or not. (0=shown, 1=not shown. Default is 1.)

EXPLANATION

This instruction converts constants/variables into decimal string. PB series can use this instruction only for in PRINT instruction. PBM series can store the result of DEC() instruction in a String variable.

EXAMPLE

The following table shows the result of instructions including DEC when variable A has 12345. (b is space)

Instruction	Result
PRINT DEC(A)	12345
PRINT DEC(A,10)	bbbbb12345
PRINT DEC(A,10,0)	0000012345
PRINT DEC(A,3)	345

PBM series can store the result of DEC() instruction in a String variable as follow.

```
DIM ST AS STRING * 16          ' Declare 16 the maximum length of string
DIM I AS INTEGER
I = 1234
ST = DEC(I) + "5678"           ' Conjunct strings
                               'The result, 12345678, is stored at ST
```

EEREAD ()

Integer variable = EEREAD (*address*, *byte*)

Integer variable = EEREAD (*address*)

Read from EEPROM

Address is a Byte/Integer constant (0~&HFFFF)/variable
Byte is a constant (0~255) or a Byte variable and is used to assign how many bytes will be read.

EXPLANATION

This instruction reads assigned ***bytes*** of data from assigned ***address*** of EEPROM. Because EEPROM can retain its contents after RESET, it is normally used to save user's data.
When you use EEREAD instruction in PBM series, you can assign how many bytes will be read from EEPROM. Default value of ***byte*** is 1.

EXAMPLE

```
DIM I AS BYTE
DIM J AS INTEGER
DIM ST AS STRING * 16
I = EEREAD(&HFFF)          ' Read a byte from FFFH and store at I
J = EEREAD(&HF00,2)        ' Read two bytes from F00H and store at J
ST = EEREAD(&HF00,18)      ' Read eighteen bytes from F00H
                           'And store at string variable ST
```

EEWRITE

EEWRITE *address*, *data*, *byte*

EEWRITE *address*, *data*

Read from EEPROM

Address is a Byte/Integer constant (0~&HFFFF)/variable.
Data is a constant (0~255) or a Byte variable.
Byte is a constant (0~255) or a Byte variable indicating how many bytes will be written.

EXPLANATION

This instruction writes assigned bytes of data at assigned address of EEPROM. Data is byte constant/variable.
When you use EEWRITE instruction in PBM series, you can assign how many bytes will be written at EEPROM. When *variable* is omitted, it is 1.

EXAMPLE

```
DIM I AS BYTE
DIM ST AS STRING * 16
EEWRITE 0, I                'Write value of I at address 0
EEWRITE 0, ST, 18           'Write eighteen bytes of String variable ST at address 0
                            'The actual length of sting area is added to maximum length.
                            (Actual length = maximum length + 2)
```

ADDITIONAL DESCRIPTION ABOUT EEREAD & EEWRITE

In PICBASIC, EEPROM (or Flash memory) is mainly used to store programs. If you do not use up all of the memory space for program, you can use the rest of the EEPROM space by EEWRITE/EEREAD instructions. Most of cases, EEPROM is not entirely filled with program. Because storing program starts from lower addresses, higher addresses could remains unused. Because 4Kbytes of program memory in PB-3X/1S is 0~&HFFF and a program will be written from address 0, you should write data from &HFFF reversely.

```
    EEWRITE &HFFF, R1
    EEWRITE &HFFE, R2
    EEWRITE &HFFD, R3

R1 = EEREAD(&HFFF)
    R2 = EEREAD(&HFFE)
    R3 = EEREAD(&HFFD)
```

At this moment, you must not let one of them overlap with the other, program area and data area. If it happens, unpredictable result may occur. Therefore, when you use EEWRITE and EEREAD, you should always make sure of the highest address of program. The highest address of program will be displayed at lower part of screen as “Code Size : XXX byte” right after downloading.

PB-1B	PB-1S, 3B, 3H	PB-2S	PB-2H
0~&H7FF (2K)	0~&HFFF (4K)	0~&H1FFF (8K)	0~&H3FFF (16K)

Because PBM series modules have extra EEPROM space for EEWRITE and EEREAD, you do not have to worry about the overlap and can use memory from address 0.

Extra EEPROM space

PBM-R1	PBM-R5
0 ~ &H1FFF (8K)	0 ~ &H7FFF (32K)

```
EEWRITE &H0, R1
EEWRITE &H1, R2
EEWRITE &H2, R3
```

EPADIN ()

PBM

Integer variable = EPADIN (*row*, *column*)

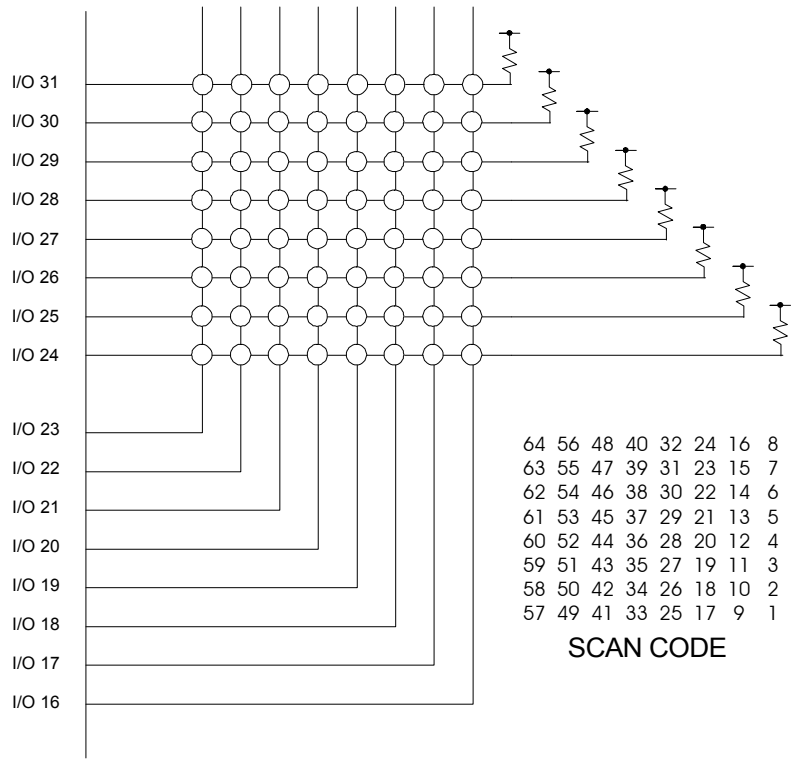
Keypad input

Row is number of rows from four to 8. (N/A with variables)
Column is number of columns from 4 to 8. (N/A with variables)

EXPLANATION

This instruction is able to read key matrix up to 8x8 by using ports, 16~31. Number of ***rows*** and ***columns*** are adjustable. If there is no key-input, then, it returns 0. If some, it returns scan code of corresponding key.

Keypad circuit must be organized as the following picture so that ports from 16 to 23 are output ports (***Row***) and Ports from 24 to 31 are input ports (***Column***). The input ports must be equipped with Pull-up resister (Around 5k~10k).

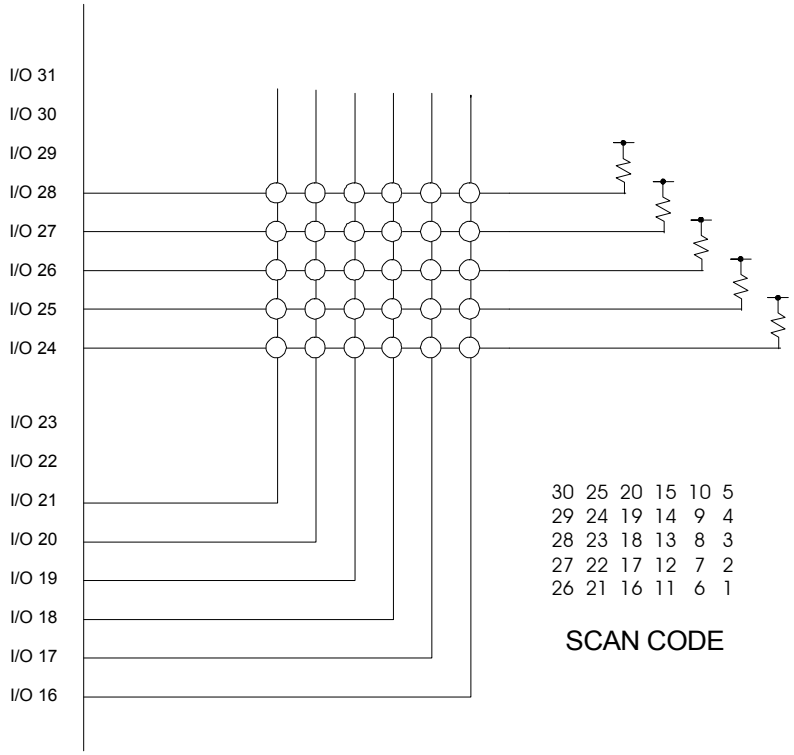



```
I = EPADIN(8,8)          'Read keypad by 8X8
LOCATE 0,0
PRINT DEC(I)             'Display scan code on LCD
```

EXAMPLE

In case that the key is less than 8 x 8, you can adjust rows and columns as follows:

Example) 6X5 key matrix input




```
I = EPADIN(6,5)          'Read keypad by 6X5
LOCATE 0,0
PRINT DEC(I)             'Display scan code on LCD
```

CAUTION

Port allocation must be placed in numerical order. Allocation of output ports must start from port 16 and allocation of input ports must start from port 24. If you don't use up all ports, you can use unallocated ports for other purpose. In that case, the ports must be input ports only due that they will output undesired result when used as output ports.

EXP ()

PBM

Single variable = EXP (*value*)

Compute natural logarithm e^x

Value is a Single constant/variable.

EXPLANATION

This instruction returns the natural logarithm of inputted ***value***. The ***value*** must be a Single variable/constant. When the ***value*** isn't Single type, use CSNG function to convert it into Single value.

EXAMPLE

```
DIM F1 AS SINGLE
F1 = EXP(2.0)
```

FREQOUT

PB

FREQOUT *port*, *parameter*

Output desired frequency

Port is a Byte constant (must be 9 or 0)/variable.
Parameter is a Byte constant (1-253)/variable indicating frequency.

EXPLANATION

This instruction outputs consistently, without relation to program process, the waveform of desired frequency to specific port. It uses PWM port (9 or 10) and keeps outputting the waveform of desired frequency until encountering OFF command of PWMOFF. FREQOUT is available in PB series. PBM series can get same result by adjusting cycle of PWM instruction.

EXAMPLE

```
FREQOUT 9,131        'Output 1KHz of waveform through port 9. (Case of 5G)
```

ADDITIONAL INFORMATION

The following table shows frequency output of each model.

PB-1B/1S/2S

Parameter	Frequency
1	258Hz
10	267.5Hz
20	278.7Hz
40	304.7Hz
80	374Hz
100	421.8Hz
191	***1KHz***
200	1.17KHz
230	2.5KHz
253	21.9KHz

PB-2H/3B/3H

Parameter	Frequency
1	1.227KHz
10	1.274KHz
20	1.324KHz
40	1.439KHz
80	1.77KHz
100	2KHz
153	***3KHz***
200	5.58KHz
230	12KHz
253	104KHz

You could find out that the higher the frequency band is, the wider variation width of frequency per parameter is. Maximum parameter is 253 and when parameter is 254 or 255, there isn't frequency output.

CAUTION

FREQOUT instruction cannot drive port 9 and port 10 simultaneously. FREQOUT instruction can use one of those two ports. While you use FREQOUT instruction, you cannot PWM instruction. Contrary, while you use PWM instruction, you cannot FREQOUT instruction.

FLOAT ()

PBM

String variable = FLOAT (*value*, *position1*, *position2*)

Converting Single type into Sting type

Value is a Single constant/variable.
Position1 is the maximum positional number for integral part. It is possible up to 10 and default value is 9. (N/A with variable)
Position2 is the maximum positional number for fractional part. It is possible up to 10 and default value is 6. (N/A with variable)

EXPLANATION

This instruction converts Single constants/variables into Single string.

EXAMPLE

The following table shows the results of each instruction in accordance with variation of parameter.
(SN is 1234.56789 and ***b*** is space)

Instruction	Result
PRINT FLOAT(SN)	***bbbb***1234.56789
PRINT FLOAT(SN,4)	1234.5678
PRINT FLOAT(SN,4,2)	1234.56
PRINT FLOAT(SN,2,2)	34.56

To express a Byte/Integer/Long value in forms of Single value having point, you must execute conversion by CSNG instruction.

```
DIM A AS INTEGER
DIM SN AS SINGLE
A = 123456
SN = CSNG(A)                ' Convert into SINGLE type by CSNG instruction
PRINT FLOAT(SN,6,5)         ' LCD displays 123456.00000
```

CAUTION

There is possibility of errors in process of FLOAT instruction converting a Single value into String. For instance, when an actual stored value is 4.0, FLOAT instruction could express the value 3.999999. This is caused that PICmicro, main processor of PICBASIC modules, cannot perfectly completes Floating point number conversion of IEEE 745 due to its limitation of hardware. i.e., the actual stored value is 4.0. It is just expressed 3.999999 when it is converted into string by FLOAT instruction.

FOR...NEXT

FOR *variable1*=*start* TO *end* [STEP–increment]...NEXT *variable2*

FOR…NEXT loop

Variable is a Byte/Integer/Long variable.
Start value is a Byte/Integer/Long constant/variable specifying the initial value of variable.
Increment is a Byte constant (-128~+127)/variable indicating variation of variable1

EXPLANATION

This instruction sets ***variable1*** as ***start value,*** and repeats commands/instructions between FOR and NEXT until the ***variable1*** reaches ***end value***. *Variable1* is increased or decreased by ***increment***. For instance, "FOR I=0 TO 50" will executes commands/instructions between FOR and NEXT 51 times. It starts on I=1 and finishes on I=50. ***variable2*** can be omitted. ***Increment*** is not available in PB series. In PBM series, you can adjust increment or decrement of ***variable1*** by STEP instruction. STEP instruction can use –128 ~ +127 as ***increment*** value.

EXAMPLE

```
FOR I=0 TO 50
      BYTEOUT 1,I                  'Repeat with increment of I from 0 to 50
NEXT I

FOR I=0 TO 50
      FOR J=0 TO 100
          PRINT DEC(J)             'Repeat 101 * 51 times
      NEXT J
NEXT I
```

'The following example is available in only PBM series.

```
FOR I=0 TO 50 STEP 3
      BYTEOUT 1,I                  'Repeat with increment of I by 3 from 0 to 50.
NEXT I

FOR I=50 TO 0 STEP -1
      BYTEOUT 1,I                  'Repeat with decrement of I by -1 from 50 to 0.
NEXT I
```

GET

PBM

GET *variable*, *delay*, *jump label*

Receive RS232C data by hardware

Variable is a Byte/String variable receiving data from RS232C buffer.
Delay is an Integer constant/variable specifying wait-time for data when no data in RS232C buffer.
Jump label is where to jump to, in case that there is no incoming data after delay.

EXPLANATION

This instruction receives data through hardware RS232C and is available with only PBM series. SET RS232 instruction must precede GET instruction that reads received data at RX terminal (I/O port 15). Hardware RS232 is full-duplex buffering method receiving data from RX terminal while another routine is processing. GET instruction can read data by FIFO(First in, First OUT). SERIN, another instruction for receiving RS232C by using software half-duplex method, has no RS232C buffer. Therefore, it can read data only while the instruction is processing. Refer to Depth analysis of RS232C for the details of difference between GET instruction and SERIN instruction.

Delay is time to wait for incoming data when there is no data in RS232C buffer. If there is no data after ***delay,*** a program jump to where ***jump label*** indicates. If you don't give jump label specific value, the program goes to next line. If the incoming data comes continuously, you can receive it to String variable.

EXAMPLE

```
SET RS232 9600      'Initialize hardware RS232C at 9600 baud rate
DIM I AS BYTE
GET I,100           'Receive data for 100mS in waiting state (If no receipt, go to next line.)
GET I               'If receipt of data, store it at I, otherwise, go to next line
GET I,100,ERR       'Receive data for 100mS in waiting state (If no receipt, go to ERR.)
```

ADDITIONAL INFORMATION

Both GET instruction and PUT instruction uses hardware RS232C port. The hardware RS232C has 96bytes of receiving buffer. When the buffer is used up with data, it overwrites from the first parts of buffer. Therefore, the data in the receiving buffer must be read before the buffer is full by GET instruction. (For this, uses ON RECV interrupt function). GET instruction reads data from buffer not read data from RX pin directly. A circuit for receiving/storing RS232 data from RX pin should work constantly by using SET RS232 instruction at the first part of a program.

GOTO

GOTO *label*

Unconditional jump

Label is a point/line where to jump

EXPLANATION

This instruction makes program jump to specified line number or label.

EXAMPLE

```
GOTO 10          'Jump to label 10
```

ADDITIONAL INFORMATION

There are two kinds of identifiers, line number and label, used to point a specific area of a program. Line number is, what we call, a kind of label consisting of numbers.

```
10          PRINT   DEC(I)
            GOTO 10
```

Label is suitable for subroutine's name, etc. It must start with an alphabet and finish with colon (:).

```
            GOSUB DELAY10
            ;
DELAY10:    FOR   I =0   TO   10
            NEXT I
            RETURN
```

You can use up to 255 of characters for name of ***jump label***.

Valid label	Invalid label
NOKEY: DELAY_RTN: ACCIN_PROC:	123AB: Started with digit IN: Name of instruction

GOSUB...RETURN

GOSUB *label*, RETURN

Call subroutine

Label is a subroutine/line where to jump.

EXPLANATION

This instruction calls subroutine and returns.

EXAMPLE

```
10          GOSUB DELAY10           'Call subroutine, DELAY10
            GOTO  10
            ;
DELAY10:    FOR   I =0   TO   10
            NEXT I
            RETURN                  'Returns after subroutine, DELAY 10
```

ADDITIONAL INFORMATION

Because PICBASIC modules have limited stacks, it is impossible to superpose subroutines above maximum depth level. It means that you must not call subroutines from inside of a subroutine more than maximum depth level. (Stack is returning address of a subroutine and 2 bytes of return address is used by one GOSUB instruction.) It is ok using subroutines in same depth. But, nesting subroutines over maximum depth causes problem. In addition, more, because PICBASIC doesn't show any error message for this problem, you must very careful with nesting subroutines. The maximum subroutine depth of PB series is 7 level while that of PBM series is 10 level.

Return must be placed on the end of a subroutine jumping to GOSUB. If instructions such as GOTO etc terminate subroutine, stack will be overflowed in a program.

```
        'Valid subroutine                   'Invalid subroutine

        GOSUB    1000                               GOSUB 1000
                                                2000
        :                                   :
1000    I = I + 1    'Subroutine    1000    I = I + 1    'Subroutine
        RETURN                              GOTO 2000
```

HEX ()

HEX (*value*, *digit*, *indicator*)

Convert an Integer constant/variable into hexadecimal string

Value is an Integer type constant/variable.
Digit is the maximum (up to 8) positional digit that you want to display. (Default is 4.)
Indicator determines to place "0" on space or not. (0=display, 1=not display. Default is 1.)

EXPLANATION

This instruction converts a constant/variable into hexadecimal string. The instruction is available with only PRINT instruction in PB series while it is able to store the result of conversion in string variable in PBM series.

EXAMPLE

This is the result of HEX instruction when variable A has &HABCD.
(**b** is substituted for space.)

Operation	Result
PRINT HEX(A)	ABCD
PRINT HEX(A,8)	B**bbb**ABCD
PRINT HEX(A,8,0)	0000ABCD
PRINT HEX(A,3)	BCD

PBM series is able to save the result of HEX instruction in a string variable. If you use a Long variable, positional number must be eight (8) to express the result completely.

```
DIM ST AS STRING * 16          'Set maximum length of string 16
DIM I AS LONG
I = &H1234ABCD
ST = HEX(I,8)                  ' If digit, 8, is omitted, result is ABCD.
```

IF...THEN

IF *expression* THEN...ELSE...

Conditional statement

EXPLANATION

If an ***expression*** is true, lines of program below 'THEN' executes. If the ***expression*** is not true, lines of program after 'ELSE' execute.

EXAMPLE

IF statement has two types of line form, they are one line form and multi line form. One line form is completed within one line. (In one line form, command appears after THEN.)

```
IF   I > 10   THEN   GOTO 10                    ' Jump to address 10 if I is larger than 10.
IF   I = 10   THEN   J = 20   ELSE J = 30       ' Put 20 into J if I is 10 otherwise, put 30 into J.
```

Multi line form has no command after THEN and must be used with END IF. Multi line form can use ELSEIF.

```
IF I > 99 THEN
        J = ADIN(0)             ' When I is larger than 99,
ELSE
        J = ADIN(1)             ' When I is same as 99 or smaller than 99,
END IF

IF I < 10 THEN
        J = ADIN(0)             ' If I is smaller than 10,
ELSEIF I < 80 THEN              ' There must be no space between ELSE and IF like ELSEIF.
        J = ADIN(1)             ' If I is smaller than 80,
ELSE
        J = ADIN(2)             ' other cases
END IF
```

CONDITIONAL EXPRESSION

Conditional expression: [variable] compare operator [variable/constant] logic operator....

Compare operator : < > <= >= <> =
Logic operator: AND, OR

CAUTION #1

When using "IF" statement, laying two or more conditions continuously by using brackets is not available. For an example, if there is a conditional statement as follows;

```
IF ((A > 1) AND (A < 10)) OR ((B > 10) AND (B < 20)) THEN
        K = K + 1
END IF
```

As a processor does not recognize brackets, above conditional statement will be operated just like the followings.

```
IF A > 1 AND A < 10 OR B > 10 AND B < 20 THEN
        K = K + 1
END IF
```

Therefore, the first conditional statement will end in different value from desired result.
In this case, the conditional statement must be separated two “IF” statements.

```
IF A > 1 AND A < 10 THEN
        K = K + 1
END IF
IF B > 10 AND B < 20 THEN
        K = K + 1
END IF
```

CAUTION #2

Operation of comparison must be performed with same data type. SINGLE type must be compared with SINGLE type. When comparing INTEGER type data (BYTE, INTEGER, LONG) with SINGLE type data, the INTEGER type data must be converted into SINGLE type by CSNG instruction.

```
IF F1 < CSNG(I1) THEN GOTO 100          ' When F1=SINGLE and I1=INTEGER
```

CAUTION #3

Comparison between string variables is not available.

```
DIM ST1 AS STRING * 12
DIM ST2 AS STRING * 12
IF ST1 = ST2 THEN GOTO 100
```

IN ()

Integer variable = IN (*port*)

Port input

Port is a constant (0~31) or a Byte variable.

EXPLANATION

This instruction read the state (HIGH or LOW) of specific ***port***. When it is HIGH, the instruction returns 1. When LOW, it returns 0.

EXAMPLE

```
DIM  I  AS  BYTE          ' Declare I byte type
I = IN(0)                 ' Read the state of port 0
```

ADDITIONAL INFORMATION

All port of PICBASIC modules is bidirectional. As INPUT/OUTPUT direction of the ports is can be set freely by instruction, ports shall be INPUT port on INPUT instruction and be OUTPUT port on OUTPUT instruction automatically. When a port is INPUT port, it is at high-impedance (floating) state. In case of other programming language such as C and Assembly, I/O instruction must be used with declaring I/O direction of ports.

In case of ports of TTL input, ports is recognized as HIGH when input voltage is higher than 1.4V and as LOW when input voltage is lower than 1.3V.
In case of ports of Schmitt trigger input, ports is recognized as HIGH when input voltage is higher than 3.4V and as LOW when input voltage is lower that 3.3V.

REFERENCE

All ports become HIGH state (high-impedance) at initial state (when electric power turns on).

KEYIN ()

Integer variable = KEYIN (*port*, [*chattering delay*])

Key input

Port is a constant (0~31) or a Byte variable.
Chattering delay is time by mS and a constant (0~255). Default value is 20mS.

EXPLANATION

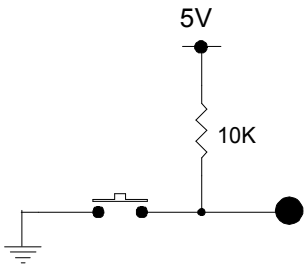
This instruction reads the state of specific ***port***, HIGH or LOW, after eliminating chattering. The instruction is used mainly to switch input. The ***chattering delay*** is specified by mS. For instance, "KEYIN (0, 100)" reads the HIGH/LOW state of ***port 0*** after 100mS of ***chattering delay***. If you omit the ***chattering delay***, it becomes default value, 20mS)

EXAMPLE

```
DIM I AS BYTE              'Declare byte type
I = KEYIN(0,100)           ' Read key input state of port 0 after 100mS of delay
I = KEYIN(0)               ' Read key input state of port 0 after 20mS of delay
```

ADDITIONAL INFORMATION

The following picture#1 is general key input circuit. It turns LOW state when switch is pushed.



(Picture #1) Key input circuit

(Picture#2) Chattering

Mechanical switching components generate chattering, which is vibration of voltage during switching process as picture #2. (It usually comes out around 10mS.) If you don't eliminate Chattering at S/W input, there may occur errors such as double-operation by one input. Therefore, you must perform elimination of Chattering in S/W input circuit. PICBASIC resolves this by software. I.e. in case of KEYIN (0, 100), it recognizes a signal as input when it receives same signal during 100mS.

KEYDELAY

PB

Integer variable = KEYDELAY (*key function*, *value*, *delay1*, *delay2*)

Key-input delay

Key function is one of ADKEY(), KEYIN() and PADIN().
Value is a constant and value to be returned (normally 0) when there is no input.
Delay1 is a constant (0~255) and wait-time for repeat.
Delay2 is a constant (0~255) and repeat–time.

EXPLANATION

This instruction sets delay of key input. ***Key instruction*** can be one of ADKEYIN, KEYIN and PADIN. The result of this instruction is same as the result of ***Key instruction***. The instruction just makes delay specified by ***Delay1*** and ***Delay2*** to return the result. The instruction is not available in PBM series.

EXAMPLE

```
10 I = KEYDELAY( KEYIN(0), 1, 30, 10)          'In case of KEYIN
     IF I = 1 THEN GOTO 10
     'If there is key input, then executes here.
     GOTO 10

10 I = KEYDELAY( ADKEYIN(0), 0, 30, 10)        'In case of ADKEYIN
     IF I = 0 THEN GOTO 10
     'If there is key input, then executes here.
     GOTO 10
     LOCATE 0,0
     PRINT DEC(I)
     GOTO 10

10 I = KEYDELAY( PADIN(0), 0, 30, 10)          'In case of PADIN
     IF I = 0 THEN GOTO 10
     'If there is key input, then executes here.
     LOCATE 0,0
     PRINT DEC(I)
     GOTO 10
```

LCDINIT

LCDINIT

Initialize LCD

EXPLANATION

This instruction initializes LCD including data on a display.

LEFT ()

PBM

String variable = LEFT (*string*, *value*)

Cut string from left

String is a String variable
Value is a Byte constant/variable.

EXPLANATION

This instruction cut a string from left as much as specified value.

EXAMPLE

```
DIM ST AS STRING * 16          'Set 16 as the maximum length of string
DIM RE AS STRING * 16
ST = "I LOVE YOU"
RE = LEFT(ST,5)
LOCATE 0,0
PRINT RE                       'LCD displays "I LOVE"
```

LEN ()

PBM

Integer variable = LEN (*string*)

Length of string

Sting is a String variable.

EXPLANATION

This instruction returns real length of string.

EXAMPLE

```
DIM ST AS STRING * 16          'Set 16 as the maximum length of string
DIM I AS INTEGER
ST = "COMFILE"
I = LEN(ST)                    ' Result of I is 7.
```

LOCATE

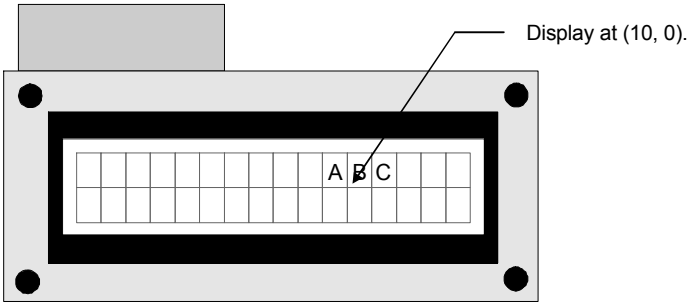
LOCATE *X coordinate*, *Y coordinate*

Appoint position of display

X coordinate is a constant (0~255) or a Byte variable.
Y coordinate is a constant (0~255) or Byte variable.

EXPLANATION

This instruction appoints position of display on LCD. Coordinates of the left top position on LCD is (0,0). The maximum size of coordinates is different in accordance with LCD models. In case of 16X2 LCD, you can use 0~15 of X coordinate and 0~1of Y coordinate.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0											A	B	C			
1																

EXAMPLE

```
LOCATE 10,0
PRINT “ABC”
```

LOG ()

PBM

Single variable = LOG (*value*)

Logarithmic function

Value is a Single constant/variable.

EXPLANATION

This instruction returns the logarithmic value of a single constant/variable. A constant/variable, which will be inputted into blank, must be single type. Otherwise, the constant/variable must be converted into single type.

EXAMPLE

```
DIM F1 AS SINGLE
F1 = LOG(1.0)
```

LOG10 ()

PBM

Single variable = LOG (*value*)

Common logarithmic function

Value is a Single constant/variable.

EXPLANATION

This instruction returns the common logarithmic value of a single constant/variable. A constant/variable, which will be inputted into blank, must be single type. Otherwise, the constant/variable must be converted into single type.

EXAMPLE

```
DIM F1 AS SINGLE
F1 = LOG10(1.0)
```

MID ()

PBM

String variable = MID (*string*, *point*, *length*)

Cut string

String is a String variable.
Point is a constant (0~31) or a Byte variable where to be cut from.
Length is a constant (0~31) or a Byte variable to be cut.

EXPLANATION

This instruction cuts desired ***length*** from an appointed ***point*** in the middle of a ***string***. Point is in order of 1, 2, 3… from left. For instance, When a string, ST, is "ABCDEFG", MID(ST, 2, 3) results "BCD".

EXAMPLE

```
DIM ST AS STRING * 16          'Set 16 the maximum length of string
DIM RE AS STRING * 16
ST = "I LOVE YOU"
RE = MID(ST,3,4)
LOCATE 0,0
PRINT RE                       'LCD displays "LOVE".
```

ON...GOTO

ON *parameter* GOTO *label1*, *label2*, *label3*, ...

Branch by parameter

Parameter is a Byte variable. (The maximum value is 127.)
Label is a label or line where to jump to.

EXPLANATION

This instruction jumps to a ***label*** where a ***parameter*** indicates. If the ***parameter*** is 0, then it jumps to ***label 1*** and if the ***parameter*** is 1, it jumps to ***label 2***.

EXAMPLE

```
ON I GOTO 100, 200, 300
                    'When I=0, jumps to 100.
                    ' When I=1, jumps to 200.
                    ' When I=2, jumps to 300.
                    ' When I is other value, executes next line without jump.
```

ON...GOSUB

PBM

ON parameter GOSUB label1, label2, label3, ...

Call subroutine by parameter

Parameter is a Byte variable. (The maximum value is 127.)
Label is a label/line of subroutine where to be called.

EXPLANATION

This instruction calls subroutine by ***parameter***. When the ***parameter*** is "0", then it calls ***lable1***. when "1", calls ***label2***. (you can use up to 127th label.). When it encounters "RETURN", it is back to next line of ON GOSUB instruction.

EXAMPLE

```
ON I GOSUB 100, 200, 300
                    'When I=0,    call address 100.
                    'When I=1,    call address 200.
                    'When I=2,    call address 300.
                    'When I is other value, executes next line without call.
```

ON INT () GOSUB

PBM

ON INT (*edge*) GOSUB *label*

Ports interrupt

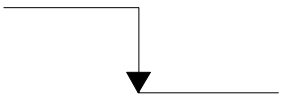
Edge is a constant (0 or 1).
Label is a label/line of subroutine where to be called.

EXPLANATION

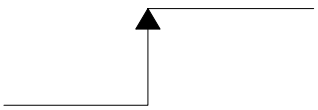
When rising/falling edge is detected at port 8 of PBM series, this instruction calls specific ***label***. If the ***edge*** set 0, a falling edge (HIGH ? LOW) will be detected. Contrary, if the edge set 1, a rising edge (LOW ? HIGH) will be detected. The instruction is a kind of port interrupt by Basic language so as to match variation of a port expeditiously. The instruction can be used only once in the first part of a program. (Not available to use the instruction more than one time simultaneously)

EXAMPLE

```
ON INT(0) GOSUB 10          'While a program runs, if a falling edge occurs at port 8, label 10 will be called.
```



```
ON INT(1) GOSUB 10          'While a program runs, if a rising edge occurs at port 8, label 10 will be called.
```



REFERENCE

The port 8 of PBM series is specially designed to detect a edge. It is impossible to use other ports to detect an edge. Though PB series has "ON INT" instruction also, there are differences from PBM series in grammar. Interrupt of PBM series can only detects variation of edges. It is impossible to detect level by interrupt of PBM series.

ON INT ()=x GOSUB

ON INT (*port*) = *level* GOSUB *label*

Ports interrupt

Port is a Byte variable or a port number (0~31) which is able to receive interrupt input.
Level is a constant (0 or 1).
Label is a label/line of subroutine where to be called.

EXPLANATION

This instruction reads state of a port and executes specific routine if the state is matched with given condition. The instruction is a port interrupt function by Basic language and is used to response variation of port state quickly. The instruction can be used only once in the first part of a program. (Not available to use the instruction more than one time simultaneously)

EXAMPLE

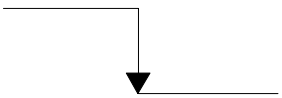
```
ON INT(0) = 0 GOSUB 20      ' While a program executes, if port 0 becomes LOW, executes line 20.
10 GOTO 10                  'If not in interrupt process, executes endless loop.

20  OUT 1,1
    RETURN
```

REGARDING EDGE INTERRUPT...

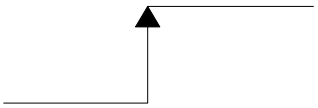
Edges can be detected at a specific port by "ON INT" instruction. Port 5 of PB-1x/2x series, port 16 of PB-3B and port 24 of PB-3H are designed to detect edges. When those ports are used by "ON INT" instruction, they operate as an edge interrupt. The edge interrupt occurs where variation of an edge interrupt port status from LOW to HIGH (Rising edge) or HIGH to LOW (Falling edge) is detected.

EXAMPLE

```
ON INT(5) = 0 GOSUB 10      ' While a program executes, if there occurs falling edge at port5, call line10.
```



```
ON INT(5) = 1 GOSUB 10      ' While a program executes, if there occurs rising edge at port5, call line10.
```



ON RECV GOSUB

PBM

ON RECV GOSUB *label*

RS232C receiving interrupt

Label is a label/line to be called when an interrupt occurs.

EXPLANATION

This instruction calls specific ***label*** when RX pin (I/O 15th) receives data while RS232C hardware is used. If RS232C receives data while a program operates, the program stores the data in extra buffer, and read the data from the extra buffer after jumping to specific subroutine. The instruction can be used only once in the first part of a program. (Not available to use the instruction more than one time simultaneously)

EXAMPLE

```
ON RECV GOSUB 10        'During operation of program, if RS232C data is received, jump to line10.
;
;
;
10  GET I               'Get one byte from buffer.
```

ON TIMER () GOSUB

ON TIMER (*interval*) GOSUB *label*

Time interrupt

Interval is a constant (0~20) indicating interval of interrupt.
Label is a label/line to be called when an interrupt occurs.

EXPLANATION

This instruction performs specific routine indicated by ***label*** every ***interval***. The instruction is a kind of Time-interrupt function by Basic language. The instruction can be used only once in the first part of a program.

Interval	1B/1S/2S	2H/3B/ 3H	PBM-R1/R5
0	0.5 second	0.105 second	0.10 second
1	1 second	0.210 second	0.21 second
2	2 second	0.420 second	0.42 second
3	3 second	0.630 second	0.63 second
4	4 second	0.840 second	0.84 second
5	5 second	1.05 second	1.05 second
6	6 second	1.26 second	1.26 second
7	7 second	1.47 second	1.47 second
8	8 second	1.68 second	1.68 second
9	9 second	1.89 second	1.89 second
10	10 second	2.10 second	2.10 second

EXAMPLE

```
ON TIMER(0) GOSUB 10          'In case of PBM-Rx, call line10 every 10 seconds.
```

ADDITIONAL INFORMATION

When you use “ON TIMER” instruction, you make sure that a subroutine, which is called by label, must finish its execution within specific ***interval***. If a subroutine having one second interval has two seconds of execution time, the subroutine will not operate properly and not make desired result. Therefore, you must make execution time of a subroutine less than a period appointed by ***interval***.

OUT

OUT *port*, *value*

Port output

Port is a constant (0~31) or a Byte variable indicating port number.
Value is a constant (0 or 1) or Byte variable.

EXPLANATION

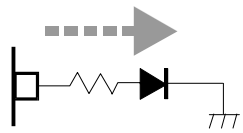
This instruction outputs HIGH or LOW to specific ***port***. When ***value*** is 0, it outputs LOW. When ***value*** is 1, it outputs HIGH.

EXAMPLE

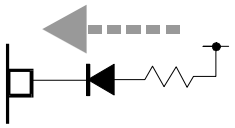
```
OUT 0,1          'Make port0 HIGH
OUT 0,0          'Make port0 LOW
```

STATE OF PORT OUTPUT...

The ports of PICBASIC module flow around 25mA in state of output. i.e. when the ports are at output state, they can receive 25mA of current (Sink current) while when the ports are at input state, they can output 25mA of current (Source current).

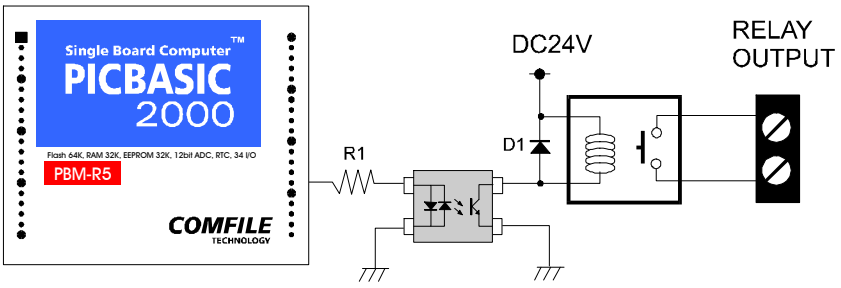


Output current from a port



Input current into a port

20mA of current is not enough to drive directly devices such as relay, solenoid etc. In this case, you had better amplify the output signal by using a photo coupler (or TR) as the following picture.



OUTSTAT ()

OUTSTAT (*port*)

Verify output state

Port is a constant (0~31) or a Byte variable indicating port number.

EXPLANATION

This instruction reads current-outputting value. IN() instruction convert appropriate port into input port and read a state of the pin. OUTSTAT() instruction reads a value in a output buffer. This instruction is mainly used to verify an outputting value.

EXAMPLE

```
I = OUTSTAT(0)        'Read value being outputted to port0.
```

ADDITIONAL INFORMATION

The instructions related to port are as follows;

-IN() : make corresponding port input, and read the state of pin outside
-OUT() : make corresponding port output, and change pin state.
-OUTSTAT() : make corresponding port output, and read content of internal output buffer

PADIN ()

PADIN (*port block*)

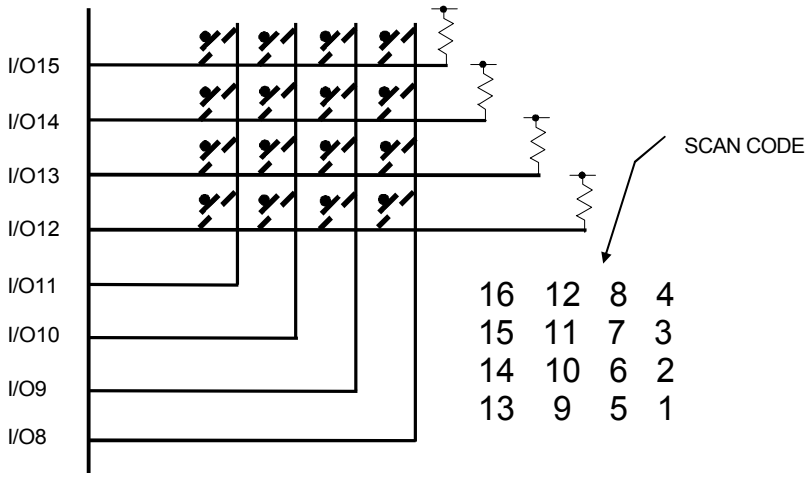
Keypad input

Port block is a constant (0~3) or a Byte variable indicating port block number.

EXPLANATION

This instruction reads 4X4 key matrix from specific port block. It is convenient to read input of dial keypad. If there is a key input, it returns 0. Otherwise, it returns scan code of the key.

The circuit of key pad must be organized as the following picture. Lower 4bits are for output and higher 4bits are for input. Higher 4bits must be pulled-up (5K~10K).



In case of PBM-R1/R5, you can use PADIN instruction at port block 1, 2 and 3.

EXAMPLE

```
I = PADIN(1)            'Read keypad from port block 1.
```

PEEK ()

PEEK (*file register*)

Read file register

File register is a file register number (0~&HFF, Not available with variable)

EXPLANATION

This instruction reads a file register of main MCU directly. You can see various file registers in the file register map (Refer to data book). Some of those registers are out of control by PICBASIC. PEEK and POKE instruction allow you to access those registers.

EXAMPLE

```
I = PEEK(&H85)          ‘ Read the content of register &H85
                        'Read the PORTA state of register &H85
```

POKE

POKE *file register*, *value*

Write file register

File register is a file register number (0~&HFF, Not available with variable)
Value is a constant 90~255) or Byte variable.

EXPLANATION

This instruction writes value directly into a file register of MCU. You can find various registers in File Register Map (refer to data book) which can be accessed by PEEK and POKE instruction.

EXAMPLE

```
POKE &h5,0          'Clear register &H5
                     'Register &H5 accesses PORTA
```

ADDITIONAL INFORMATION

Because that PEEK and POKE, instruction could take critical effect on PICBASIC hardware, if you are not well used in controlling PIC16F87X, don't try it.

POW ()

PBM

Single variable = POW (*value X*, *value Y*)

X^Y value

Value X and ***Value Y*** are a Single constant/variable.

EXPLANATION

This instruction calculates a value of X^Y. ***Value X*** and ***value Y*** must be single constants or variables. If they are not single type, you should convert them into single type with CSNG instruction as the following example.

EXAMPLE

```
DIM F1 AS SINGLE
F1 = POW(2.0, 3.0)          '8.0 is inputted into F1.
```

PLAY

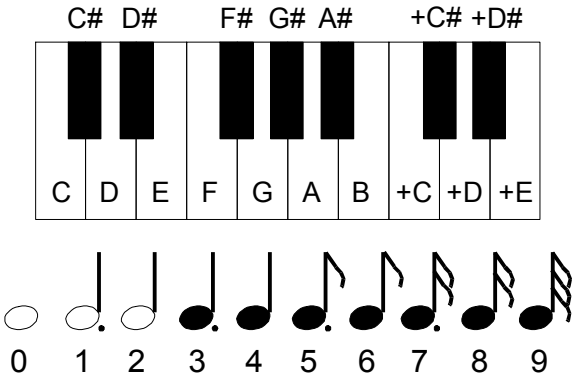
PLAY *port*, *data*

Play music

Port is a constant (0~31) or a Byte variable indicating port number.

EXPLANATION

This instruction plays simple melody. A appropriate port should be connected to a PIEZO or a speaker. PBM series can use only 0^{th} ~15^{th} ports for play. The followings are how to make ***data***.



WHEN DATA IS C4,
Two parameters are used to express one tone. One is an alphabet (A~G) determining interval of the tone and the other is number determining length of the tone. Therefore, “C4” means Do of quarter note.

WHEN DATA IS +C#5,
One tone can be expressed by four parameter at most. “#” means higher tone by one semitone and “+” means higher tone by one octave. Therefore, +C#5 means a tone is higher tone than Do by one semitone and one octave. Its length is 5. (Quarter note.)

C5E5G5+C1
Play do, mi, sol, do.

EXAMPLE

```
PLAY 1, "C5C7D4C4F4E2C5C7D4C4G4F2C5C7+C4A4F4E4D2A#5A#7”
PLAY 1, "G4E6E6G6E6C4D4E6D6C6E6G4+C5G7+C5G7+C6G6E4G4D6”
PLAY 1, "C9D9E9F9G9"
                                    ‘Output to port1
```

PRINT

PRINT *string* [*value1*, *value2*, ...]

Display on LCD

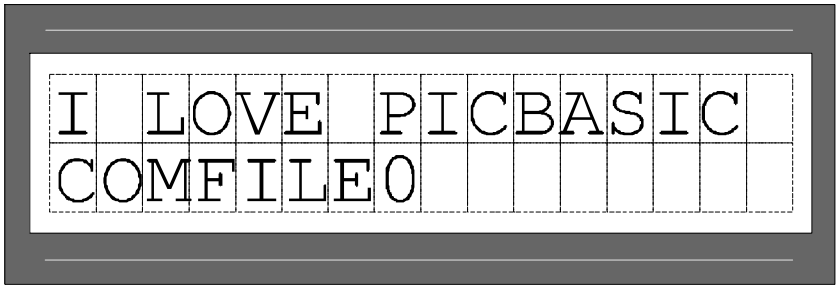
String is a String like "ABC". (String variable is available in PBM series.)
Value is a constant (0~255) or a Byte variable.

EXPLANATION

This instruction displays characters and digits on a LCD. When you input ***values***, the corresponding ASCII codes will be displayed on a LCD. If you want to display digits in the forms of decimal or hexadecimal, use conversion instruction such as DEC or HEX.

EXAMPLE

```
LOCATE 0,0
PRINT "I LOVE PICBASIC"
PRINT "COMFILE",&H30
```



Example of ELCD162

PULSE

PULSE *port*, *pulse width*(μs)

PULSE *port*

Output pulse

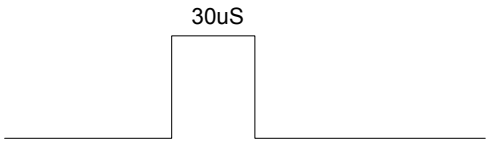
Port is a constant (0~31) or a Byte variable indicating port number.
Pulse width is time of microsecond unit and is an Integer constant (0~65535)/variable.

EXPLANATION

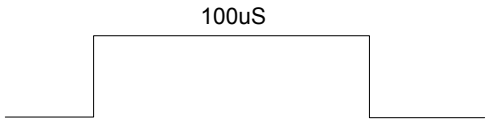
This instruction outputs pulse to specific ***port*** as long as given ***pulse width*** (μs). In effect, it inverts pulse as long as given ***Pulse width*** and returns. ***Pulse width*** can be omitted. (When it is omitted, it becomes 18 μs.) PBM series use PULSE instruction to drive a RC servomotor by adjusting ***pulse width***. PB series cannot adjust ***pulse width***(it outputs around 2~3 μs constantly.).

EXAMPLE

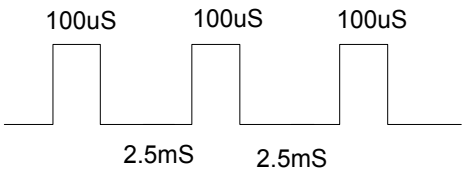
```
OUT 3,0
PULSE 3,30
```




```
OUT 3,0
PULSE 3,100
```




```
OUT 3,0
FOR I = 0 TO 2  'Output pulse three times
PULSE 3,100
DELAY 2
NEXT
```



Above example is a program outputting pulse three times. As 2mS of DELAY instruction is used at LOW state, there is needed additional 0.5 mS to execute FOR…NEXT instruction.

PUT

PBM

PUT *data*

Output RS232C data

Data is a Byte constant/variable or a String constant/variable.

EXPLANATION

This instruction outputs data through RS232C hardware. The data is outputted through I/O port 14 in the format of 8bit, none parity and 1 stop bit. Before using this instruction, you must execute SET RS232 instruction.
Data must be Byte type. If you use Integer or Long type data, only lower 8bits will be transmitted. Single data cant be transmitted. If you use String variable as ***data***, all content of String variable will be transmitted.

EXAMPLE

```
SET RS232 4800
DIM I AS BYTE
I = &Ha0
PUT I                          'Send content of variable I. (Carriage return character wont be sent.)

SET RS232 4800
DIM ST AS STRING * 30
ST = "I LOVE PICBASIC 2000"
PUT ST                         'Send content of String ST. (Carriage return character wont be sent.)
```

CARRIAGE RETURN CHARACTER

Other Basic languages such as QBASIC etc transmit carriage return character, 13 and 10 automatically. Those character are kind of command code moving to the first part of next line.
However, PBM series cannot use the carriage return character.

PWM

PWM *port*, *duty rate*, *cycle*

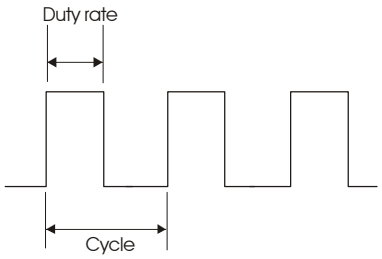
PWM *port*, *duty rate*

Generate PWM wave

Port is a constant (9 or 10) or a Byte variable.
Duty rate is a Byte constant/variable. (In case of PBM series, Integer constant/variable because of 10bits)
Cycle is a Byte constant/variable.

EXPLANATION

This instruction outputs PWM to specific ***port***. ***Duty rate*** is adjustable in the range between 0 and 1023. The larger the ***duty rate*** is, the wider the HIGH pulse is. ***Cycle*** is a value determining width of one cycle. If it is omitted, it becomes 255. In case of PB series, you cannot adjust a value of cycle, and the value is fixed 255. Only ***port 9*** and ***port 10*** can be used for PWM output. PWM instruction keeps outputting PWM waveform until power-off or PWMOFF instruction.



You can change the cycle of PWM waveform by adjusting ***cycle***. The following table shows the relation between ***cycles*** and cycle of PWM waveform.

Cycle value	Frequency	Duty rate range
10	28.4KHz	0~40
100	3KHz	0~400
255	1.22KHz	0~1023

Range of duty rate changes in accordance with variation of ***cycle***. Input range of duty rate cannot exceed quadruple cycle value. (If it exceeds, an appropriate port becomes HIGH state.) Make certain that a changed cycle value will affect both port 9 and port 10. i.e., you cannot use different cycle value at port9 and port 10.

In case of PB series, you cannot adjust the cycle value. It is fixed as 255. Frequency of each model is as the following table.

1B/1S/2S	2H	3B/3H
Frequency 256Hz, 8bits resolution	Frequency 1.22KHz, 8bits resolution	Frequency 1.22KHz, 8bits resolution

If you want to output wave having different frequency in PB series, you should use FREQOUT instruction. You cannot use FREQOUT instruction in PBM series (You can same result as FREQOUT instruction by adjusting a cycle value of PWM instruction.).

EXAMPLE

```
10    I = ADIN(0)      'Store the result of A/D input at I
      PWM 9, I         'Output PWM waveform by duty rate I to port 9
      GOTO 10
```

PWMOFF

PWMOFF port

Stop outputting PWM wave

Port a constant (9 or 10) or a Byte variable.

EXPLANATION

Once PWM instruction is executed, the corresponding port keeps outputting PWM. To stop the output, you should use PWMOFF instruction. (Ports for PWM output (port9 and port10) are in PWMOFF state at power-on reset.)

EXAMPLE

```
PWMOFF 9          'Stop outputting PWM at port9
PWMOFF 10         'Stop outputting PWM at port10
```

RESET

RESET

Initialize main processor

EXPLANATION

This instruction initializes a processor by software. It takes effect as electric power reset.

EXAMPLE

```
RESET              'Reset processor
```

RIGHT ()

PBM

Sting variable = RIGHT (*string*, *value*)

Cut right part of string

String is a String variable.
Value is a Byte constant/variable.

EXPLANATION

This instruction cuts part of string constant/variable as much appointed from the right.

EXAMPLE

```
DIM ST AS STRING * 16              'Set maximum length of string 16
DIM RE AS STRING * 16
ST = "I LOVE YOU"
RE = RIGHT(ST,3)
LOCATE 0,0
PRINT RE                           ' "YOU" is displayed on LCD.
```

RND (0)

RND (*0*)

Random number

EXPLANATION

This instruction generates random number (16bits integer). 0 in the brackets is meaningless dummy value.

EXAMPLE

```
    I = RND(0)              'Store random number into variable I
    PRINT DEC(I)            'Display I
```

The following example is a program outputting result of RND() by sound. If you connect port2 to PIEZO, you can listen funny sound.

```
    DIM J AS BYTE
10  J = RND(0)              ' Store random number into J
    SOUND 2, J, 3           'Output sound pulse having J as a tone
    GOTO 10
```

SERIN

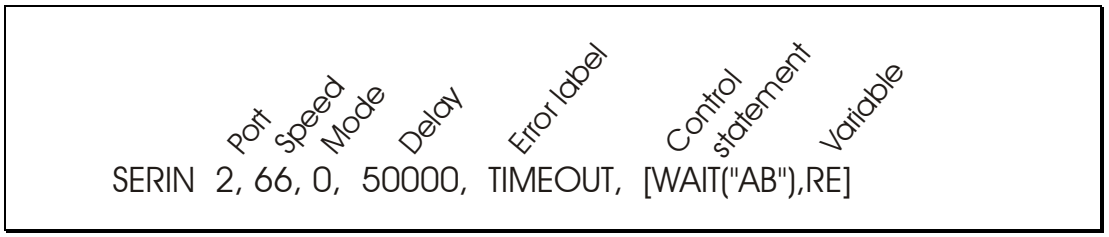
SERIN *port*, *speed*, *mode*, *delay*, *label* [*control statement*, *variable*]

Receive RS232C

Port is a constant or a Byte variable specifying port number to receive RS232 (PBM series can use only port0~15.)
Speed is a constant specifying transmission speed of RS232C. It could be different in according to baud rate.
Mode is a constant (0 or 1) indicating to invert PWM. If mode is 0, PWM is not inverted. Otherwise, PWM is inverted.
Delay is a constant or a Integer variable specifying latency time(μs) for input. If receiving RS232C does not finish within delay time, program jumps to the ***error routine label***. Otherwise, it proceeds to next line.
Label is a label/line where to be jump when latency for input exceeds ***delay***.
Control statement specifies control option for receiving RS232C such as WAIT, UNTL and SKIP. UNTL and SKIP are not available in PBM series.
Variable is a Byte variable/array or a String variable specifying where received data is stored in.

EXPLANATION

This instruction receives data through RS232C by software. You can freely use ports and delay. Received data will be stored at a receiving variable. (Transmission format of RS232C is fixed as 8bit, 1 stop bit, No Parity.)



Each PICBASIC modules has unique transmission speed in according to baud rate as show below table. These rates are same in SEROUT instruction.

Baud rate	Transmission speed (1B/1S/2S)	Transmission speed (2H/3B/3H)	Transmission speed (PBM-R1/R5)
300			3260
600			1620
1200			810
2400	138		400
4800	66	207	196
9600	30	103	93
19200	11	47	40
38400			14

EXAMPLE

```
SERIN   2, 196, 0, 5000, TIMEOUT, [I]      'Receive data from port 2 at 4800 baud rate.
                                           'Jump to TIMEOUT routine if there is no data
                                           'Input in five seconds.
```

ADDITIONAL INFORMATION

Receiving control statement #1: WAIT

You can make Multi-drop communication, which is a method assigning a kind of address to each receiving device, by WAIT. (This is frequently used in RS485/433 communication.) SERIN instruction waits for appointed characters (2bytes) by WAIT, and stores data following the appointed characters close behind into an appointed variable.

```
SERIN 2, 66, 0, 50000, TIMEOUT, [WAIT("AB"),I]    'Receive data from port2 at 4800 baud rate.
                                                  'Wait for receiving character A, B. After 'receiving
                                                  character A, B, store data 'following behind into I, and
                                                  finish a program.
```

Receiving control statement #2: until

When array are used to receive several bytes simultaneously, if a specific character is received, receiving process is stop.

```
SERIN 2, 66, 0, 50000, TIMEOUT, [UNTIL("*"),A(0)~10]    'During receiving 10 bytes to array A, 'if * is received,
                                                         'stop receiving.
```

Receiving control statement #3: skip

When array are used to receive several bytes simultaneously, if a specific character is received, the specific character is skipped.

```
SERIN   2, 66, 0, 50000, TIMEOUT, [SKIP("R"),A(0)~10]    'During receiving 10 bytes to array A, character R
                                                          will be skipped.
```

When you use WAIT, UNTIL and SKIP all together, you must use them in order of WAIT, UNTIL and SKIP.

*** PBM series cannot use UNTIL and SKIP**.

Incase of receiving several bytes.....

You can use an array or a string variable for a receiving variable. When you use an array, you can receive several bytes simultaneously by using "~". (It is used when data is sent continuously)

```
SERIN 2, 93, 0, 50000, TIMEOUT, [I(0)~5]
        'When an array is used for a receiving variable, you can appoint number of receiving 'byte. 'Receive
        5bytes from I(0). i.e. receive bytes, I(0), I(1), I(2), I(3), I(4)

SERIN   2, 93, 0, 50000, TIMEOUT, [ST~5]
        'Receive 5bytes to string variable ST. at this moment, the last byte of data must be '"0" because that "0"
        means end of string.
        'Make certain not to exceed maximum range of string declared by DIM at first part of a program.
```

RS232C receiving interrupt in PB series

RS232C receiving interrupt stops executing instructions of a program other than SERIN when RS232C data transmission is detected, and executes receiving RS232C. In order to make this, edge interrupt is used.

```
ON INT(5)=0 GOSUB RS232_INT
:
:   'main program
:

RS232_INT:
        SERIN 5,11,0,1000,TIMEOUT,[BF]
TIMEOUT:
        RETURN
```

When falling edge (Start bit of RS232C) at port 5 is detected, a program is stop its current execution, and jumps to RS232_INT routine to wait for RS232C transmission.
In this moment, make certain that the first data can not received. Because it takes a couple of mS for PICBASIC modules to detect the edge and jump to SERIN instruction. Therefore, a data sender had better send dummy data,00, as the first data.

SEROUT

SEROUT *port*, *speed*, *mode*, *interval*, [*data*]

RS232C transmission

Port is a constant or a Byte variable specifying port number to receive RS232 (PBM series can use only port0~15.)
Speed is a constant specifying transmission speed of RS232C. It could be different in according to baud rate.
Mode is a constant (0 or 1) indicating to invert PWM. If mode is 0, PWM is not inverted. Otherwise, PWM is inverted.
Interval is a constant specifying space between bytes (by mS).
Data is a Byte constant/variable or a string variable.

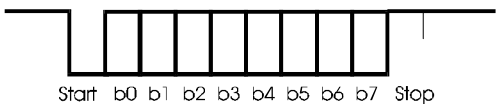
EXPLANATION

This instruction outputs RS232C data controlled by software. Transmission format, 8bits,none parity and one stop bit, is fixed. You can set ports output state freely and do inverting output. Data can be string (ASCII code), constant and variable etc.

```
                Port Speed Mode Interval Data
SEROUT  3, 66, 0, 1, [&HA0]
```

The data must be byte type. If integer or long type data is transmitted, only lower 8bits will be sent. You can not use single type data in SEROUT.

The following picture shows waveform of RS232C. (Fixed at 8bit, 1 Stop bit, None Parity)



The instruction is almost same as hardware PUT instruction. It has additional advantages such as setting port, inverting output and setting delay between bytes.

Each model has its transmission speed. The following table shows transmission speed of each model in accordance with baud rates.

Baud rate	Transmission speed (1B/1S/2S)	Transmission speed (2H/3B/3H)	Transmission speed (PBM-R1/R5)
300			3260
600			1620
1200			810
2400	138		400
4800	66	207	196
9600	30	103	93
19200	11	47	40
38400			14

EXAMPLE

```
SEROUT 3, 196, 0, 1, [&HA0]
        'Output to port 3 (4800 baud rate, non parity, interval: 1milisecond)

SEROUT 1, 93, 1, 0, ["PICBASIC 2000",13,10]
'Output to port1 (9600 baud rate, non invert, no interval, String output)
```

ADDITIONAL INFORMATION

When you need to send only ASCII code, use converting instruction such as DEC, HEX, etc for converting digits into ASCII code.

```
SEROUT 1, 93, 1, 0, [DEC(I)]          'Send value of I by decimal
SEROUT 1, 93, 1, 0, [HEX(I)]          'Send value of I by hexadecimal
```

When Sting variable is used, all content of String variable is sent.

```
DIM ST AS STRING * 16
ST = “PICBASIC 2000”
SEROUT 1, 93, 1, 0, [ST]      ’”PICBASIC 2000” is sent.
```

SERVO

PB

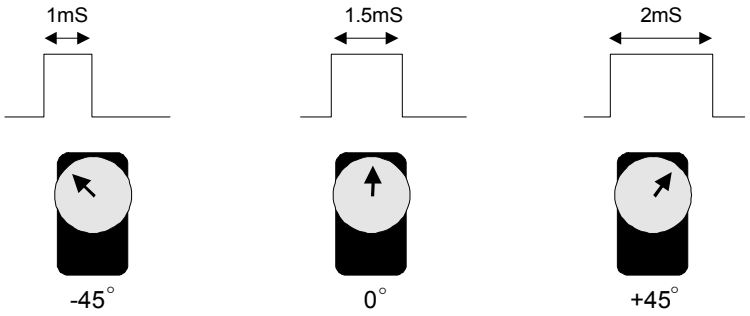
SERVO *port*, *position*

Generate RC SERVO motor control signal

Port is a constant or a Byte variable specifying port number. (0~31)
Position is a constant or a Byte variable (0~255).

EXPLANATION

This instructions generates signal to control a RC SERVO motor. You can make positioning of a RC SERVO motor simply.



When a RC SERVO motor is provided with 1mS of pulse sixty times per a second, it is positioned at -45°. When 1.5mS, it is positioned at 0°. When 2mS, it is positioned +45°. (a RC SERVO motor rotates only within 180°.) PBM series can't use SERVO instruction. To drive a RC SERVO motor in PBM series, use PULSE instruction.

EXAMPLE

The following example is for PB-1B/1S/2S. Port 0 is used for pulse output.

```
DASI:   SERVO 0, 333            'Generates 1mS of pulse at port 0
        DELAY 10
        GOTO DASI
```

Usage of SERVO instruction is little different by each model.

	PB-1B / 1S / 2S	PB-2H, 3B, 3H
Unit	3uS	0.8uS
To generate 1mS of pulse	SERVO 0, 333	SERVO 0, 1250

CAUTION ON SERVO INSTRUCTION

RC servo motor needs continued pulse supply sixty times per one second to hold current position. Otherwise, it cannot hold desired position or, if loaded, it becomes uncontrollable. To provide pulse steadily, you had better organize the endless loop shown above example.
However, in case of controlling several RC servomotors simultaneously, it may cause problem to control all RC servomotor by software. Therefore, there needs dedicated controller for RC servomotor such as SMC-PRO (Comfile Technology).

SET ONINT

PBM

SET ONINT *ON*/*OFF*

Turn on/off edge interrupt

EXPLANATION

This instruction activates/inactivates edge interrupt of port 8. The initial state of ON INT(0) GOSUB instruction is ON. If you use the instruction without declaration of ON INT, you will encounter error.

EXAMPLE

```
SET ONINT ON          'Activate Edge interrupt
SET ONINT OFF         'Inactivate Edge interrupt
```

SET ONRECV

PBM

SET ONRECV *ON*/*OFF*

Turn on/off RS232C receiving interrupt

EXPLANATION

This instruction activates/inactivates RS232 receiving interrupt of port 8. The initial state of ON RECV GOSUB instruction is ON. If you use the instruction without declaration of ON RECV, you will encounter error.

EXAMPLE

```
SET ONRECV ON         'Activate RS232 receiving interrupt
SET ONRECV OFF        'Inactivate RS232 receiving interrupt
```

SETONTIMER

PBM

SET ONTIMER *ON*/*OFF*

Turn on/off timer interrupt

EXPLANATION

This instruction activates/inactivates timer interrupt of port 8. The initial state of ON TIMER(0) GOSUB instruction is ON. If you use the instruction without declaration of ON TIMER, you will encounter error.

EXAMPLE

```
SET ONTIMER ON                  'Activate timer interrupt
SET ONTIMER OFF                 'Inactivate timer interrupt
```

USAGE OF INTERRUPTS IN PB SERIES

All interrupts in PB series is activated constantly. If you want to stop an interrupt temporarily, you should check a corresponding flag and return instantly at first part of an interrupt routine.

```
    DIM   INT_EN AS BYTE            'Declare interrupt enable flag
    INT_EN = 0                      'Enable state
    ON TIMER(0) GOSUB 100           'When occurring timer interrupt, line 100 executes.
    :
    GOTO 10                          'Main loop

100 IF INT_EN = 1 THEN RETURN       'When a interrupt enable flag is 1, the interrupt is skipped
     :
     :
     RETURN                         'End of interrupt routine
```

In above example, if some other routine made INT_EN 1, the interrupt routine at line 100 will not execute.

SET PICBUS

SET PICBUS *HIGH*/*LOW*

Set transmission speed of PICBASIC

EXPLANATION

This instruction adjusts transmission speed of LCD output port (PICBUS). Initial value is set 19200 baud rate. To change into 4800 baud rate, use SET PICBUS LOW. To return 19299 baud rate, use SET PICBUS HIGH.
All serial LCD modules have a jumper adjusting baud rate. To display characters on LCD properly, parameter of SET PICBUS instruction must coincide with baud rate adjusted by jumper in LCD.

CAUTION

The initial baud rate of all PICBASIC models having firmware v.27 or higher is 19200. All serial LCD modules provided by Comfile Technology inc has 19200 of initial baud rate.

SET RS232

PBM

SET RS232 baud rate

Declare RS232C communication

Baud rate is fixed value.

EXPLANATION

This instruction is used to declare use of RS232 hardware. In order to use GET or PUT instruction, you must use this instruction in the first part of program. Once SET RS232 is activated, PICBASIC assign port14 and port15 as RS232C ports and initialize them. You can use one of baud rates shown in following table.

SET RS232 2400	2400 baud rate
SET RS232 4800	4800 baud rate
SET RS232 9600	9600 baud rate
SET RS232 19200	19200 baud rate
SET RS232 38400	38400 baud rate
SET RS232 57600	57600 baud rate
SET RS232 76800	76800 baud rate
SET RS232 96000	96000 baud rate
SET RS232 115200	115200 baud rate

EXAMPLE

The following program displays data from PC on LCD, and ECHO.

```
    SET RS232 4800                  'Declare hardware RS232 at 4800 baud rate
    DIM DATA AS BYTE
    LCDINIT
    CSROFF
10  GET DATA,100
    IF DATA=0 THEN GOTO 10
    PRINT CHR(DATA)                 'Display received data on LCD
    PUT DATA
    DATA = 0
    GOTO 10
```

SHIFTIN ()

SHIFTIN (*CLK port*, *DATA port*, *mode*, *bit*)

Ports interrupt

CLK port is a constant or a Byte variable specifying port number-generating clock (OUTPUT)
DATA port is a constant or a Byte variable specifying port number reading data (INPUT)
Mode

0 = Starting with LSB (Least Significant Bit). Sampling after CLK rising
1 = Starting with MSB (Most Significant Bit). Sampling after CLK rising
2 = Starting with LSB (Least Significant Bit). Sampling after CLK falling
3 = Starting with MSB (Most Significant Bit). Sampling after CLK falling

Bit is a constant specifying bit number (up to 16. Default is 8)

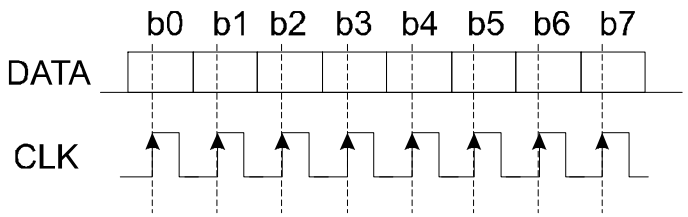
EXPLANATION

This instruction receives Shift Input. It is mainly used to read data from devices such as I^2C, SPI and so on etc using ***CLK*** and ***DATA.***

```
                    CLK port
                      Data port
                        Mode
                          Bit
A = SHIFTIN(3, 4, 0, 8)
```

EXAMPLE

```
I = SHIFTIN(3,4,0)          'Port3 outputs clock. Port4 receives data input.
                            'Mode is 0. The result is stored at I.
```



SHIFTOUT

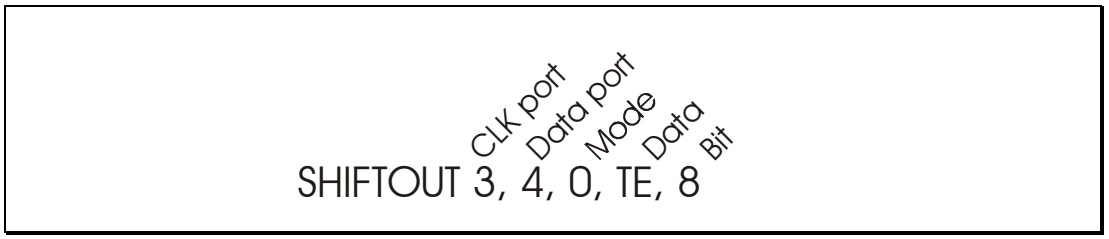
SHIFTOUT *CLK port*, *DATA port*, *mode*, *data*, *bit*

Serial output

CLK port is a constant or a Byte variable specifying port number-generating clock (OUTPUT)
DATA port is a constant or a Byte variable specifying port number outputting data (INPUT)
Mode
0 = Starting with LSB (Least Significant Bit).
1 = Starting with MSB (Most Significant Bit).
2 = Starting with MSB (Most Significant Bit). Generating ACK (ACKNOLOGY)
Bit is a constant specifying bit number (up to 16. Default is 8)

EXPLANATION

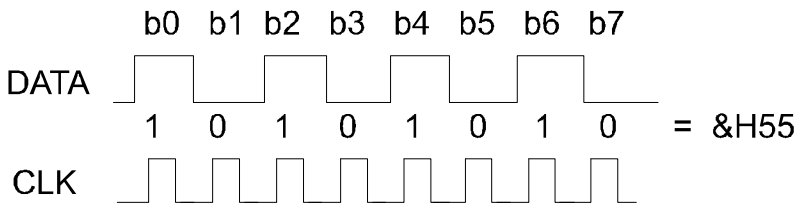
This instruction is for SHIFT output. There are three kinds of modes, LSB priority, MSB priority and I^2C.



Mode 2 is used for I^2C protocol, which needs ACK every 8 bits during transmission.

EXAMPLE

```
SHIFTOUT 0,1,0,&H55                '0 is clock, 1 is data, mode 0 and output &H55
```



SIN ()

PBM

Single variable = SIN (*value*)

SIN function

Value is a Single constant/variable.

EXPLANATION

This instruction returns SIN value of ***value***.(Angle by radian). ***Value*** must be Single type. If value is not Single type, you must convert it into Single type by CSNG instruction.

EXAMPLE

```
DIM F1 AS SINGLE
F1 = SIN(1.0)
```

*** Angle should be expressed by radian. (Radian = (Angle * 3.14) /180)**

SOUND

SOUND *port*, *interval*, *length*, [, ...]

Generate sound

Port is a constant or a Byte variable specifying port number.
Interval is a Byte constant/variable.
Length is a Byte constant/variable.

EXPLANATION

This instruction generates simple sound. To use this instruction, you should connect PIEZO, etc to specific port. ***Interval*** is pitch of sound such as do, re, mi, fa and sol. (Each sound has unique frequency and SOUND instruction generates the frequency to make sound. Approximately, when 233~139 are inputted, you can get sound of do, re, mi, fa, sol, la and this in one octave. The larger ***Length*** is, the longer sound last. (When ***Length*** is 16, sound lasts around Quarter note.) PBM series can use port 0~15 as output of SOUND.

EXAMPLE

```
SOUND   5, 239, 10          'Generate sound of 239(interval) and 10(length) at port5
SOUND   5, 239, 10, 159, 10 'you can use intervals and lengths in a row.
```

SQR ()

PBM

Single variable = SQR (*value*)

Square root

Value is a Single constant/variable.

EXPLANATION

This instruction returns a square root of ***value***. ***Value*** must be single type. If value is not Single type, you must convert it into Single type by CSNG instruction.

EXAMPLE

```
DIM F1 AS SINGLE
F1 = SQR(4.0)              ' 2.0 is stored at F1.
```

STEPOUT

PB

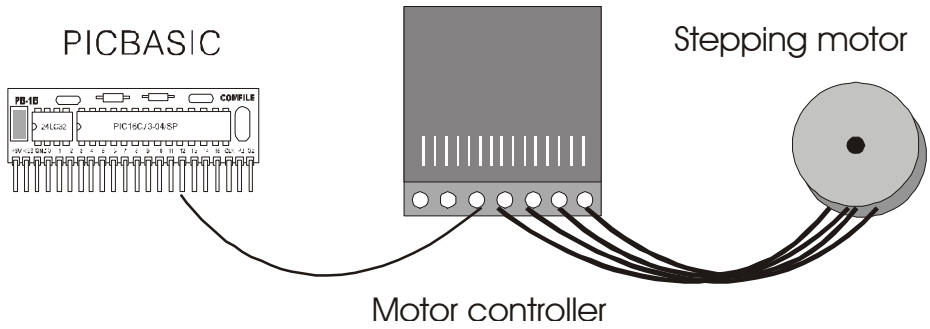
STEPOUT *port*, *interval*, *number of times*, *stop port*, *stop level*

Output pulse for controlling stepping motor

Port is a constant (0~31) or a Byte variable specifying port number.
Interval is a constant (1~255) or a Byte variable specifying pulse interval.
Number of times is a constant (1~65535) or a Integer variable.
Stop port is a constant (0~31) specifying port number for stop input.
Stop level is a constant (0 or 1)

EXPLANATION

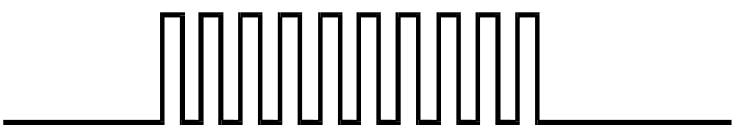
This instruction output wave specific ***number of times*** every ***interval***. By supplying the wave to a stepping motor drive, you can drive a stepping motor simply. Generally, a stepping motor used for positioning needs extra drive for connecting a controller. By providing the controller with pulse, the stepping motor will rotate as much as provided pulse. If a stepping motor rotates 1.8° per one pulse, providing 10 of pulse makes the stepping motor rotate by 18°.



EXAMPLE

```
STEPOUT   2, 50, 10          'Output wave ten times every 50 interval to port2.
                             ' A motor rotates by 18°.
```

```
STEPOUT   2, 20, 10          'Output wave ten times every 20 interval.
                             'The motor rotates faster.
```



Pulse interval means one cycle of pulse. The smaller the value of ***interval*** is, the shorter a pulse is (a motor rotates faster.). The larger the value of ***interval*** is, the longer a pulse is. (A motor rotates slower.) The following table shows pulse interval of each PICBASIC module.

Pulse interval (1~255)	1B,/1S/ 2S	2H/3B/ 3H
1	11.6KHz	60.9KHz
2	10.3KHz	53.2KHz
5	7.6KHz	38.47KHz
10	5.3KHz	26.31KHz
20	3.3KHz	16.13KHz
50	1.54KHz	7.463KHz
100	0.819KHz	3.937KHz
200	0.423KHz	2.024KHz
255	0.333KHz	1.597KHz

It is possible to stop during operation. When ***stop port*** become LOW or HIGH, outputting pulse is stopped.

```
STEPOUT   2, 10, 500, 3, 0        ' Output wave 500 times every 10 interval.
                                  'If port3 becomes LOW during operation, stop outputting.
```

TABLE ()

Table (*parameter*, *value1*, *value2*, *value3*, ...)

Converting table

Parameter is a Byte variable. Its value must not exceed 127.
Value is a constant. (0~255)

EXPLANATION

This instruction executes table-conversion by a ***parameter***. When a ***parameter*** is 0, it returns ***value 1***. when a ***parameter*** is 1, it returns ***value 2***. (Number of ***value*** is available up to 127 and must be 8bits value (0~255).

EXAMPLE

```
I = TABLE (J,192,22,165,9,34)

    'If J=0, I=192.
    'If J=1, I=22.
    'If J=2, I=165.
    'If J=3, I=9.
    'If J=4, I=34.
    'If J is other value, I=0.
```

TOGGLE

TOGGLE *port*

Invert output state

Port is a constant 90~31) or a Byte variable specifying port number.

EXPLANATION

This instruction inverts state of a specific port. i.e., LOW to HIGH and High to LOW.

EXAMPLE

```
OUT 0,1                 'Make port0 HIGH
FOR I=0 TO 7
TOGGLE 0                'Invert port0
NEXT I
OUT 0,0                 'Make port0 LOW
```

TIME ()

PBM

TIME (*address*)

Read real-time data

Address is a constant or a Byte variable specifying address number.

EXPLANATION

PBM-R5 module has a built-in real-time clock chip (DS1302). The chip has a dedicated clock crystal (32.768 KHz) and can renewal automatically second, minute, hour, day, month and year. The chip has several memory blocks. Each block contains data of second, minute, hour and so on in the forms of BCD code. The data will be read by TIME().

Address	Content	Range	Composition of bit							
0	Second	0~&H59		10				1		
1	Minute	0~&H59		10				1		
2	Hour	0~&H23				10		1		
3	Date	1~&H31				10		1		
4	Month	1~&H12				10		1		
5	Day	1~&H7						1		
6	Year	0~&H99		10				1		

EXAMPLE

```
SS = TIME(0)          ' Read current second
MM = TIME(1)          ' Read current minute
HH = TIME(2)          ' Read current hour
LOCATE 0,0
PRINT HEX(SS)         ' Because of BCD value, expressed by HEX function.
```

ADDITIONAL INFORMATION

PBM-R5 has built-in super condenser (0.1F) for real-time backup. This condenser is charged during operation. When PICBASIC module is in OFF state, it operates as battery. (If it is full-charged, it will last around six months)

***** This instruction is available in only PBM-R5.***

TIMESET

PBM

TIMESET *address*, *data*

Change real-time data

Address is a constant (0~6) or a Byte variable.
Data is a constant (0~255) or a Byte variable.

EXPLANATION

This instruction revises data of real-time clock (DS1302) in PBM-R5 module. (Factory default is dummy value)

Address	Content	Range	Composition of bit							
0	Second	0~&H59		10				1		
1	Minute	0~&H59		10				1		
2	Hour	0~&H23				10		1		
3	Date	1~&H31				10		1		
4	Month	1~&H12				10		1		
5	Day	1~&H7						1		
6	Year	0~&H99		10				1		

EXAMPLE

```
                    'Must be hexadecimal type for storing in the forms of BCD
TIMESET 1, &H30     'Set current minute 30
TIMESET 2, &H2      'Set current hour 2
TIMESET 3, &H10     'Set current date 10
TIMESET 4, &H1      'Set current month January
```

The following table shows the relation between BCD code, hexadecimal and decimal. Because that TIM() and TIMESET() instructions use BCD code, it is much easier to store/display by hexadecimal.

Time (BCD code)	Hexadecimal	Decimal
1	&H01	1
2	&H02	2
3	&H03	3
4	&H04	4
5	&H05	5
6	&H06	6
7	&H07	7
8	&H08	8
9	&H09	9
	&H0A	10
	&H0B	11
	&H0C	12
	&H0D	13
	&H0E	14
	&H0F	15
10	&H10	16
11	&H11	17
12	&H12	18
13	&H13	19
14	&H14	20
15	&H15	21
16	&H16	22
17	&H17	23
18	&H18	24
19	&H19	25
	&H1A	26
	&H1B	27
	&H1C	28
	&H1D	29
	&H1E	30
	&H1F	31
20	&H20	32
21	&H21	33
22	&H22	34
23	&H23	35
24	&H24	36

VAL ()

PBM

Integer variable = VAL (string)

Convert string into Integer value

String is a Sting variable.

EXPLANATION

This instruction converts string into Integer value. When a string expresses Integer value such as “12345”, it convert into figures (12345). When a string is not value such as “KOREA”, it converts into 0. The result of this instruction must be stored at Integer variable.

EXAMPLE

```
DIM ST AS STRING * 16          'Set maximum length of string 16
DIM I AS INTEGER
ST = “12345”
I = VAL(ST)                    '12345 is stored at I.
```

VALSNG ()

PBM

Single variable = VALSNG (*string*)

Convert string into Single value

String is a String variable.

EXPLANATION

This instruction converts string into Single value. When a string expresses Sing value such as “12345.5678”, it convert into figures (12345.678). When a string is not value such as “KOREA”, it converts into 0. The result of this instruction must be stored at Single variable.

EXAMPLE

```
DIM ST AS STRING * 16      'Set maximum length of string 16
DIM C AS SINGLE
ST = “12345.5678”
C = VALSNG(ST)             '12345.5678 is stored at ST.
```

PB to PBM conversion in program

Although most of programs which are programmed by PB series, PB-1B, PB-1S, PB-2S, PB-2H, PB-3B and PB-3H can be executed by PBM series, because that some of instructions for PB series are changed or excluded, you need to modify a program programmed by PB series to use PBM series. Refer to the following description carefully for converting a program.

INSTRUCTION OF PB SERIES

Instruction	Conversion
STEPOUT	Use PULSE and FOR..NEXT instruction
FREQOUT	Adjust frequency value of PWM instruction
KEYDELAY	Use DELAY instruction

DIFFERENT USAGE BETWEEN PB SERIES AND PBM SERIES

Instruction	Additional function
FOR..NEXT	STEP is available (-128 ~ 127)
EEWRITE	Assigning byte number for WRITE is available.
EEREAD	Assigning byte number for READ is available.
PWM	Adjusting cycle value is available
PULSE	Adjusting length of pulse (when omitted, 1mS)
DELAY	Delay up to 65535mS (PB series is 255mS)

CUT-OFF FUNCTION IN PBM SERIES

Instruction	Additional function
SERIN	Excludes control statement, SKIP and UNTIL
ON INT	Only Edge interrupt is detectable at port8. - excludes check function for level variation.

ETC

Instruction	Details
Variable	Add LONG type, STRING type, SINGLE type
Array	Extended Subscript (0~65535)
Bit/Byte designator	DOT (.) ➔ COLON (:)
SET PICBUS	Initial value is HIGH, 19200 baud rate.

Chapter 3.

PICBASIC Programming

RS232C Advanced analysis

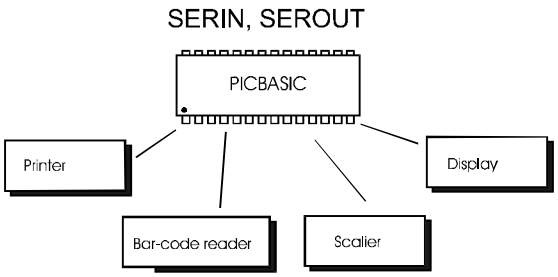
When programming in PICBASIC, you will encounter the situation-needing interface with other devices. Most common interface is very RS232C communication. It is widely used in PC as well as other devices such as card reader, small-size printer, display, etc.
PICBASIC language includes various instructions for RS232C. the instructions are SERIN, SEROUT, PUT and GET. There also are instructions for dealing with receiving interrupt such as ON RECV etc.
To work with RS232C communication, you should need understand of RS232C communication standard, usage of corresponding instructions and how to handle a counterpart device.

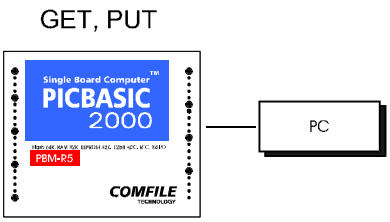
First, I would like to describe two methods to control RS232C by PICBASIC. One is SERIN and SEROUT instructions, which control RS232C on software. These instructions do not use hardware circuit of RS232C inside of a microchip to control RS232C communication. They use normal I/O port to execute RS232C communication. On the other hand, PBM series have GET and PUT instructions. Because that these instructions use hardware RS232C circuit built in microchip, they are able to provide with lots of functions that SERIN and SEROUT instructions can not provide with. However, SERIN and SEROUT instruction also have some functions that GET and PUT instructions cannot provide with. Therefore, you should completely understand the differences between those two methods, and select a proper method or combine them in accordance with desired purpose.

The following table shows the differences between above two methods.

Content	Software RS232C	Hardware RS232C
Instruction	SERIN, SEROUT	GET, PUT BCLR, ON RECV, BLEN
Principle	Controlling RS232C communication by software of interpreter	Using built-in hardware RS232C communication port.
Port	Free to select port. (PBM series can use ports from 0 to 15)	RX: port15 TX: port14
Baud rate	PBM series: 300 ~ 19200 PB series: 2400 ~1 or 9200	2400 ~ 115200
Receiving buffer	N/A	Available (PBM has built-in 96 bytes of receiving buffer)
Receiving interrupt	N/A	Available
Duplex	Half-duplex	Full-duplex
Signal level	Invert/non-invert	N/A
WAIT character receiving control	Available (up to 2 bytes)	N/A (by software)
Model	PB series, PBM series	PBM series

The most significant strong point of SERIN/SEROUT instruction is free to select port and uses several ports simultaneously. On the other hand, GET/PUT instruction can use only port 14/15.

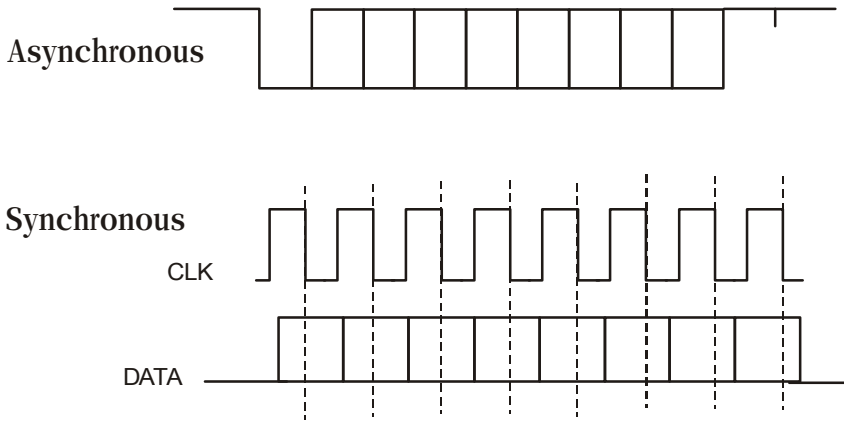




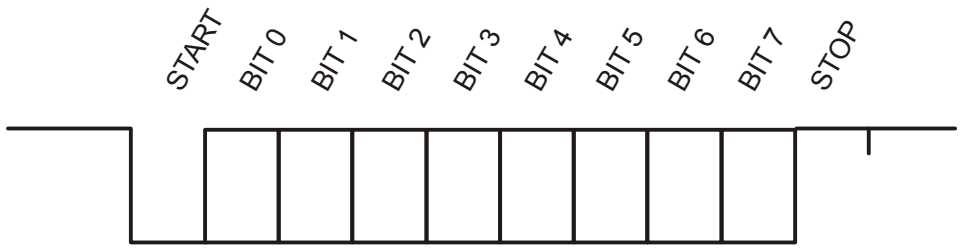
The most strong point of GET instruction is receipt of data during other instructions. SERIN cannot receive data during other instruction.

Basic of RS232C

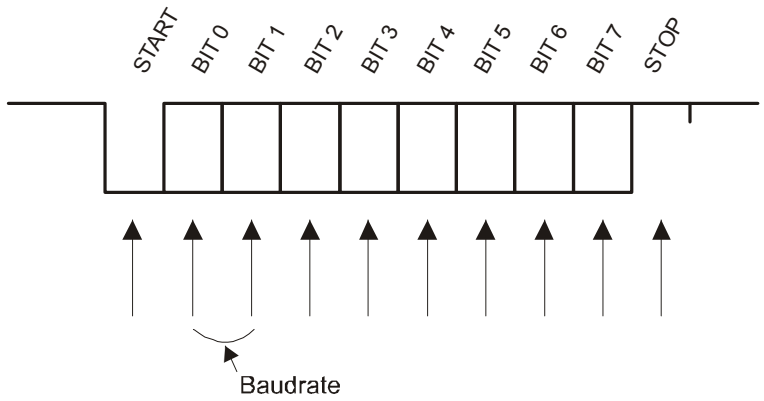
There are two kinds of serial communication, Asynchronous and Synchronous. Synchronous communication needs another clock signal to synchronization between a transmitter and a receiver. SHIFTIN/SHIFTOUT instruction of PICBASIC is used for the synchronizing communication. Because that RS232C is asynchronous communication, there is no need of clock signal. Instead of that, RS232C communication needs synchronization of baud rate between a transmitter and a receiver. A transmitter sends data at appointed baud rate, a receiver deal with data of the baud rate.



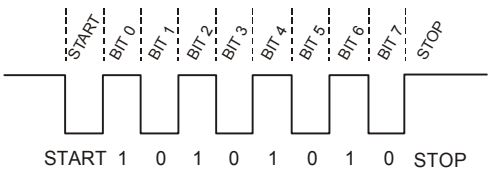
Data of RS232C is composed of one START bit, eight DATA bit and one STOP bit. This composition is called a protocol of 8BIT/NONE parity/1STOP bit. It is also expressed with abbreviated form, 8,N,1. In addition, there is another protocol composed of seven DATA bits, ODD parity and two STOP bits.
However, PICBASIC is designed to use only 8BIT/NONE parity/1STOP bit protocol, which is most commonly used. The following picture shows RS232C format composing one set of DATA. The format in the picture is LSB FIRST because that data transmission starts with lower bits.



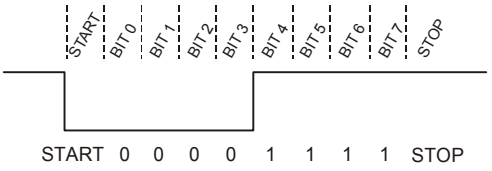
In RS232C communication, a sender only has to output data in the forms as above picture at appointed baud rate. However, a receiver should do work that is much more complicated. At first, the receiver should detect START bit above all. Once, after detecting falling edge of START bit, it starts sampling DATA bits at appointed interval in accordance with appointed baud rate. Receiving one set of DATA is finished by combining HIGH or LOW status of DATA bits incoming through a receiving port.



The following picture expresses waveform of actual data transmission.

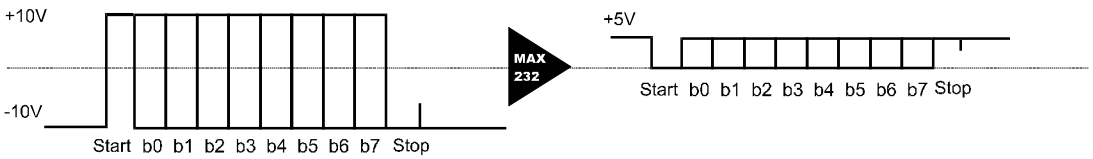


Hexadecimal: 55 Binary: 01010101

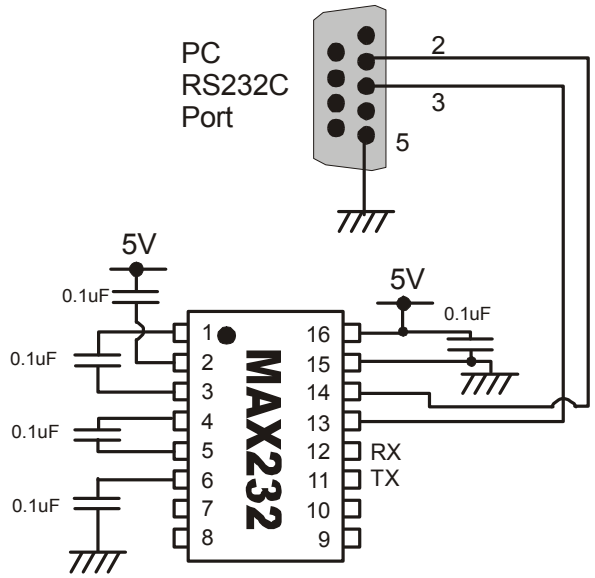


Hexadecimal: F0 Binary: 11110000

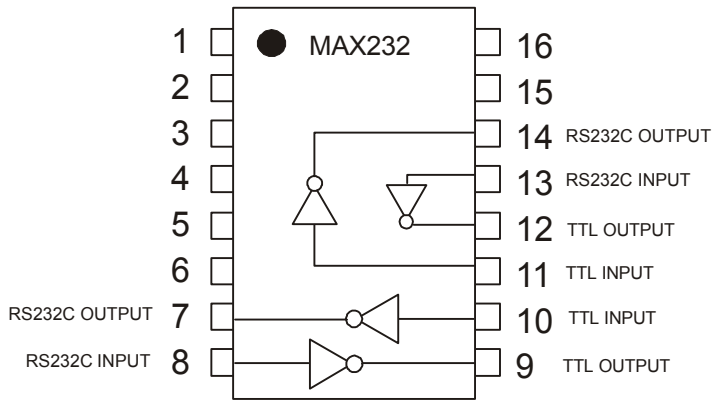
In above picture, a RS232C sender outputs HIGH signal in waiting state. If transmission starts, the sender outputs LOW signal. This is TTL level (0~5V) of RS232C transmission and is called Positive logic.
In actual application of RS232C such as PC and so on, RS232C adopts Negative-true logic and ±12V of level in order to reducing bad influence of noise in long-distance transmission .



Therefore, there needs an additional circuit for converting these ±12V Negative-true logic RS232C signal into 5V positive logic RS232 signal. MAX232 is most commonly used chip for this work. You can find the MAX232C chip in most of RS232C circuit. PICBASIC PnP board also contains MAX232C and its circuit is as follows.

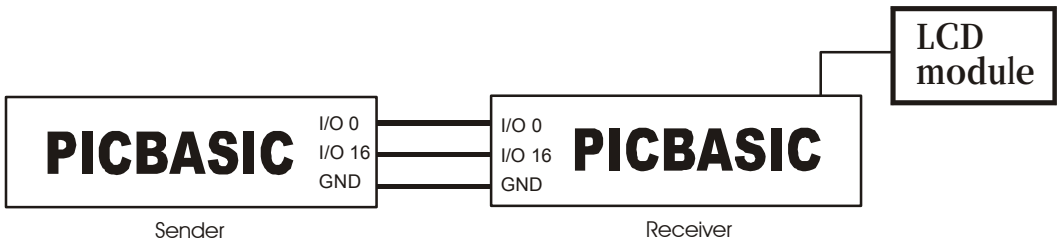


MAX232 is a kind of inverter operating in different voltage levels.(TTL level is 0~5V range.)



Communication between PICBASIC

To make RS232C communication between PICBASIC modules, connect port 0, port 16 and GND of two PB-3B modules as the following picture, and connect a LCD module to a receiver module.



Input the following program into a sender module.

```
'Sending program
        CONST TX_PIN = 0
        CONST BAUD = 103
        DIM I AS BYTE
10      SEROUT TX_PIN,BAUD,0,1,["PICBASIC"]
        DELAY 1000
        GOTO 10
```

Input the following program into a receiver module, and execute. Then, you will see "PICBASIC" on the LCD.

```
    'Receiving program
        CONST RX_PIN = 0
        CONST BAUD = 103
        DIM I AS BYTE
        DIM A(10) AS BYTE
        SET PICBUS HIGH
10      SERIN RX_PIN,BAUD,0,5000,10,[A(0)~8]
        FOR I = 0 TO 7
                LOCATE I,0
                PRINT A(I)
        NEXT I
        GOTO 10
```

This program, as a simple example for communication between PICBASIC modules, makes the sender module to send data every one second and makes the receiver to stay at waiting state/display received data instantly. However, because that actual programs add the receiver various function, the receiver can not be waiting state for RS232C signal all the time. Therefore, there needs a trick to make the receiver.

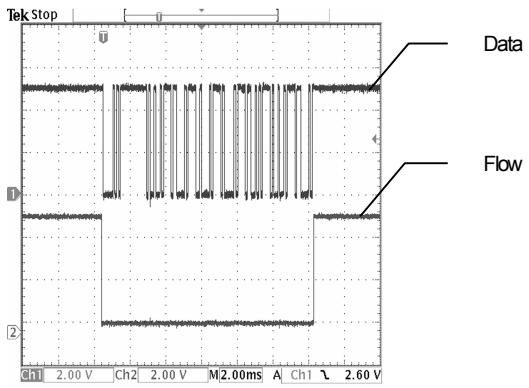
Because SERIN instruction does not have buffer function, it can receive data incoming right after its execution. Therefore, receiver's program should inform a RS232C sender through extra I/O pin when it is ready to receive data. This is called Flow control.

The sender's program watches an assigned pin for Flow control, and sends data when it receives a signal from the receiver. If the sender's program should execute other works except sending data, it is hard to watch the Flow control pin all the time. In this case, use Edge interrupt. The following program is an example of using Edge interrupt.

```
'Sender's program with Flow control
        CONST TX_PIN = 0
        CONST FLOW = 16
        CONST BAUD = 103
        DIM I AS BYTE
        ON INT(16)=0 GOSUB 10
      OUT TX_PIN,1                              'Set sending pin HIGH at first part of a program
100     GOTO 100                                'Executes main program (endless loop)
10      DELAY 1                                 'Set some delay to let a receiver prepare
        SEROUT TX_PIN,BAUD,0,1,["PICBASIC"]
        RETURN
```

```
'Receiver's program with Flow control
        CONST RX_PIN = 0
        CONST FLOW = 16
        CONST BAUD = 103
        DIM I AS BYTE
        DIM A(10) AS BYTE
        SET PICBUS HIGH
        OUT FLOW,1                      ' Set Flow pin HIGH
10      DELAY 1000                      ' Execute other works
        DELAY 1000
        OUT FLOW,0                      ' Send a sender a signal indicating that a receiver is ready to
                                        'Receive data (Set Flow pin LOW)
        SERIN RX_PIN,BAUD,0,5000,10,[A(0)~8]
        OUT FLOW,1                      ' Set Flow pin HIGH after receiving data
        FOR I = 0 TO 7
                LOCATE I,0
                PRINT A(I)
        NEXT I
        GOTO 10
```

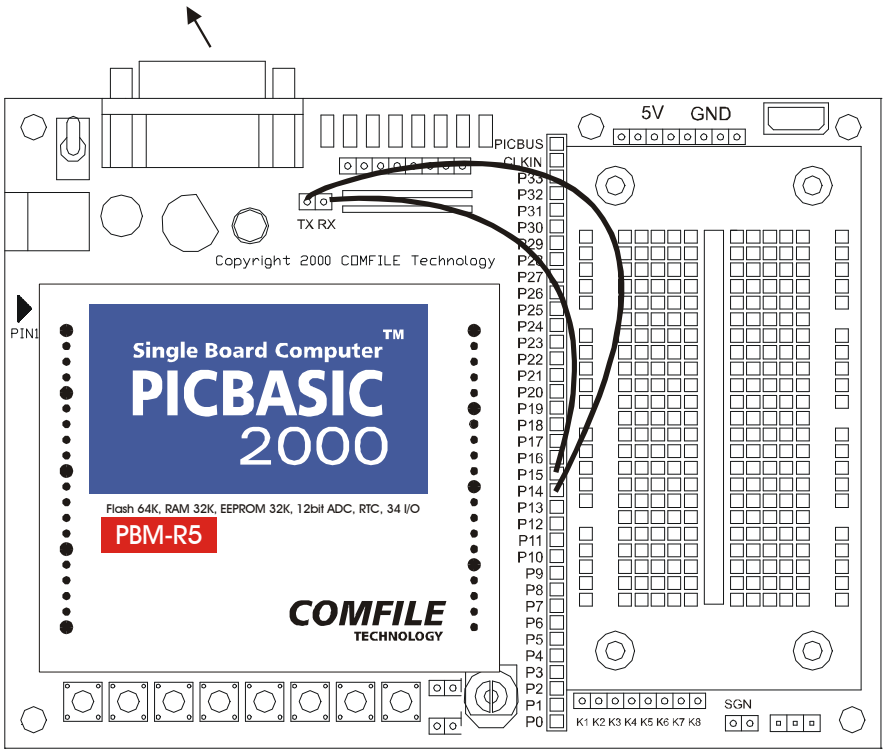
The following picture shows the result of measurement with an oscilloscope for above example communication. As you can see in the picture, sending data starts after Flow drops to LOW state. With this Edge interrupt method, timing problem of communication can be resolved for both of sender and receiver.



Communication with PC

To communicate PBM-R5 with PC through RS232C, connect PBM-R5 to PC through PICBASIC PnP board as following picture. (Connect port 15 to RX, and port 14 to TX with 1:1 cable)

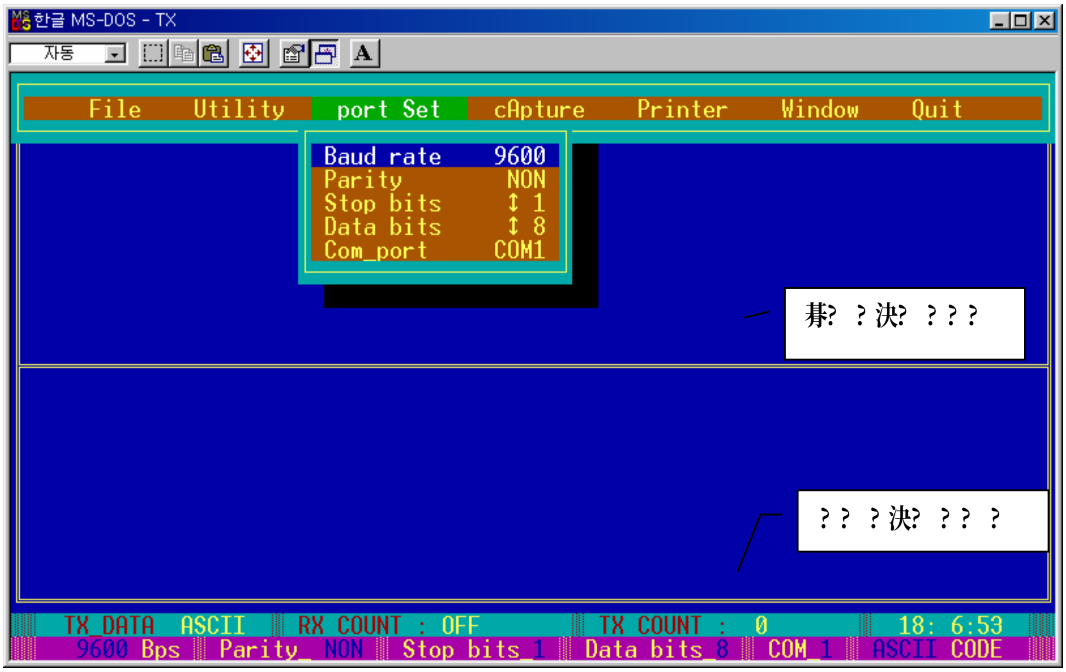
Connecting to RS232C port of PC with one to one cable



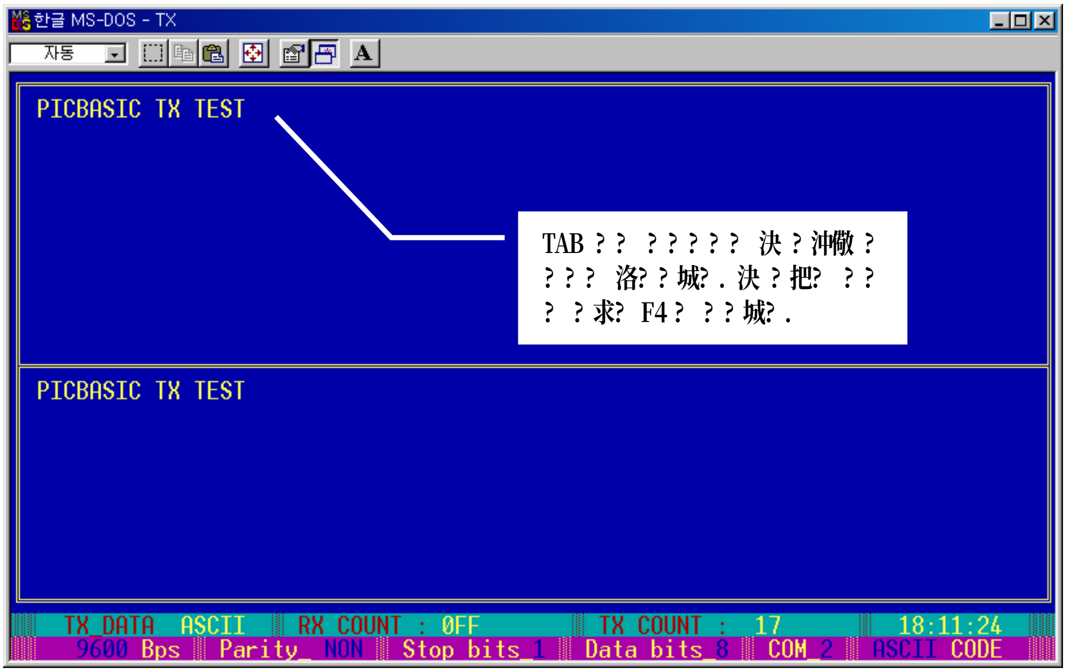
Then, executes PICBASIC studio, and download the following program. The program will FEED BACK data from PC.

```
SET RS232 9600
DIM J AS BYTE
ON RECV GOSUB 100
10                  GOTO 10
100     GET J
                    PUT J
                    RETURN
```

PC ? , ? ? ? ? ? ? TX.EXE ? ├┴. ? ? ? ? ? ? ? ? ? ? ? ? ? www.comfile.co.kr ? ? ? ? ? ? ? ├┴. DOS ? ? ? (? ? Command ? ? ?)? ? TX ? ? ? ? ? ? ? ? ? F10(? ⓤ? ?)? ? ? ? ┴? ? ? ? ? ? ? ? ? ? ├┴. ? ⓤ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴. ? ? ? ? ? ? ? 8 ? ? NONE ? ? ? , 1 STOP BIT ? ? ? ? ? ? ? ? .



? ? ? ? ? ? ? ? ? ? ? ? ◆, ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ◆? ? ? ? ? ? ? ? ? ? ? , ? ? ? ? ? ? ◆? ? ? ? ? ? ? ? ? ? ? ├┴. ? ├┴. ? ? ? ? ? TAB ? ? ? ? ? ? ? ? ? ◆? ? ? RS232C ? ? ? ? ? ? ├┴. ? ? ? ? ? ? PICBASIC ? ? ? ? ? ? ? ? ? ┴? ? ? ? ? ? ? ◆? ? ? ? ? ? ECHO ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴.

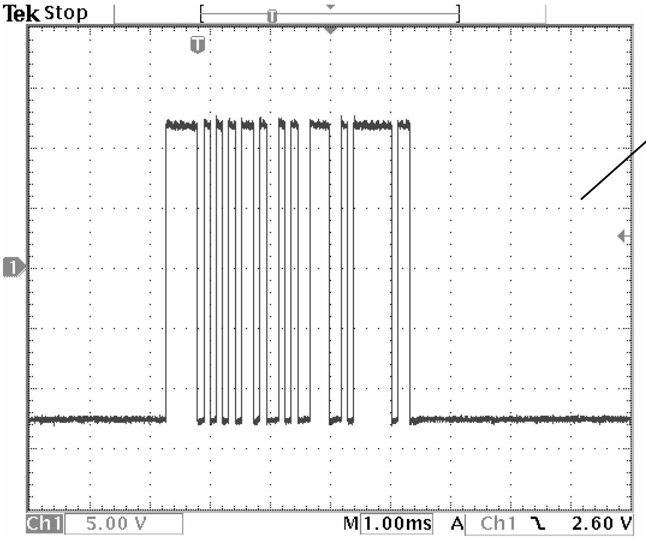


TX ? ? ? ? ? ? ? ? ? ? ? ┴? ? ? ? ├┴.

F4 : ? ? ? ? ? ? ? (? ? ?)
F6 : ? ? ? ? ? ? (? ? ? ? ┴ ? ? ? ? ? ? ?)
F7 : ASCII ? ? ? HEX ? ? ? ?
F8 : ? ? ? ? ? (? ? ? ? ? ? ? ? ? ? ? ? ESC ? ? ?)
ESC : ? ? ? ? ? ?
TAB : ? ? ? ? ? ? ? ? ◆? ? ? RS232C ? ? ? ? ?
F10 : ? ? ? ⓤ ? ? (? ? ? ? ㎡? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?)

Check communication port of PC

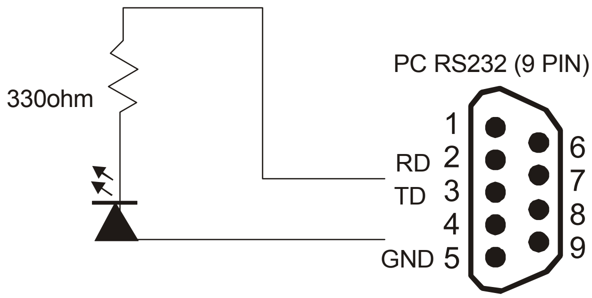
? ? ? ? ? ? ? ? ? ? ? ? ┴? , PC ? ? ? ? ? ? ? ? な, ? ? PICBASIC PnP Board ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴. ? ? ? ? ? ? ? ? ? ? ? ? ┴? , PC ? RS232C ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴. ? ? ? ? ? ? ? ? ? ? ? ? ? ? PC ? ? ? ? ? RS232C ? ? ? 3 ? ? ? ? ? ? ? ? ? ? ├┴. ? ? ? ? ? ? ? ? ? ? ? ? TAB ? ? ? ? ? ? ┴ ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴.



決 ? ? ? ? 瑚? ? ? 洛穢 ? ? ? 勾堆? ? ? 穢 ? ? 城? . 丼 ? ? ? 5V 崎憮決? 丼? ? ? 1mS ? ? 洛穢 ? ? 城? .

堆? ±12V 汞 砒笛決 櫝? ? ? ? ? ? 汾? 丼 ? ? 城? .

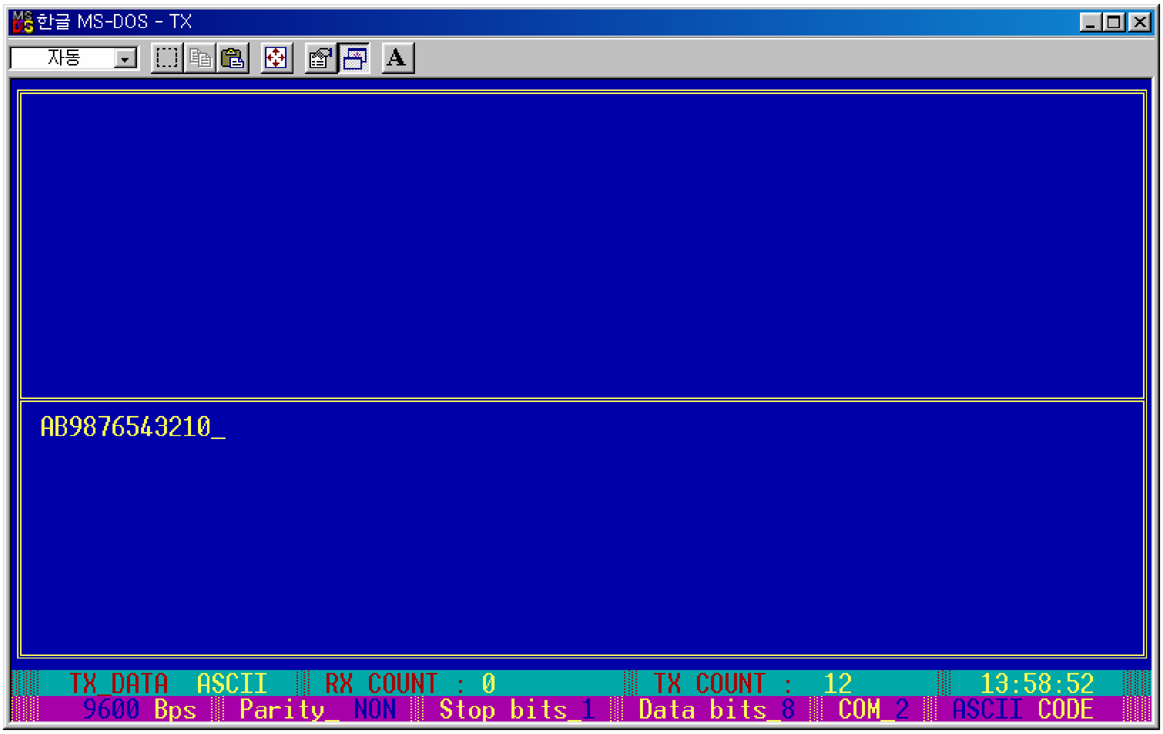
? ? ? ? ? ? ? ┴? ┴? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴.



LED ? ? ? ? ? ? ┴? PC ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴. ? ? ? ? ? ? ? ? ? ? ┴? , ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? , ? ? ? ? ? ? ? ? ? , ? ├? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴.

GET ? ? ? ? ? ? ? ? ? ? ?

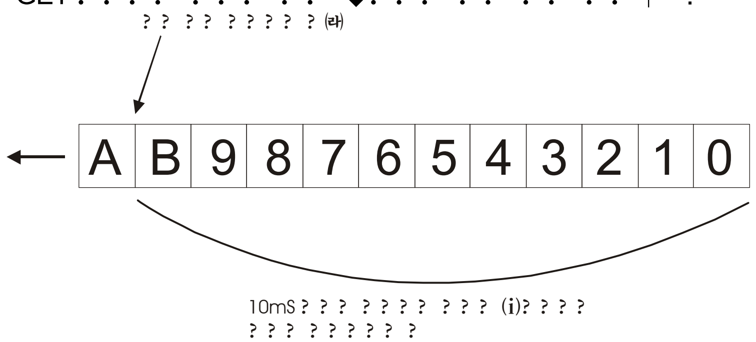
PC ? ? ? ? ? ? ? ? ◆? ├ ┴. ◆? ? ? "AB"? ? ? ? ? ? mg? 10 ? ? ◆? ? ? ├┴. ? ? ? ? 12 ? ? ? ? ├┴.



? ? ? ? ? ? ? ? TAB ? ? ? ? ? ┴ ? ? ? ? ? ? ? ? ? ? ? ├┴. PICBASIC ? ? ┴? ? ? ? ? ? ? ? ? ? ? ? ? ├┴. ┴? ? ? PICBASIC ? ? ? LCD ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴.

```
    SET RS232 9600              ' RS232C ? ? ?  ? ⓑ?
    DIM I AS BYTE
    DIM A(12) AS BYTE
    ON RECV GOSUB 100           ' ? ? ?  ? ㈑? ?  100 ? ? ? ?  ? ?
10  GOTO 10                     ' ? ? ? ? ?  ? ? ? ? ?  ? ?

100 DELAY 10                    ' ? ? ?  ? ? ?  ? ? ? ? ?  ? ? ? ?
    FOR I = 0 TO 11             ' 12 ?  ? ? ?  ? ?
        GET A(I)            ' 12 ? ?  (i)? ? ?  ? ? ? ?
        LOCATE I,1
        PRINT CHR(A(I))         ' LCD ?  ? ? ?  (i)? ? ?  ? ?
    NEXT I
    RETURN
```

PBM-R5 ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? , PC ? TX ? ? ? ? ? ? ? TAB ? ? ? ? ? LCD ? ? "AB9876543210"? ? ? ? ? ? ? ├┴. ? ? ? ? ? ? ? ? DELAY 10 ? ? ? ? ? ? ? ? ? ? ? ? ├┴. ON RECV ? , ? ? ? ? ? ? ? ? "A"? ? ? ? ? ? ? ? , ? ? ? ? ? ? ? ? ? ? 100 ? ? ? ? ? ? ? ? ? ├┴ .? ? ? ? ? "A"? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? , ? ? ? GET ? ? ? ? ? ? ? ◆? ? ? ? ? ? ? ? ? ├┴. ? ? ? DELAY 10 ? ? ? ? 10mS ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? , ? ? ? ? ? ◆? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴. ? ? ? ? ? FOR..NEXT ? ? ? ? ? ? ? GET ? ? ? ? ? ? ? ? ? ◆? ? ? ? ? ? ? ? ? ├┴.



GET ? ? ? ? ? ? ? ? ◆? ├┴. PBM-R5 ? ? ? ? ? SET RS232 ? , GET ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴. ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? "? ? ? ◆? ? ? x ? ? ? ? ? ? ? ? "? ├┴. ? ? ? ? ? ? 9600 ? ? ? ? ? ? ? ? ? ? ? ? ? 1mS ? ├┴. ? ? ? ? ? ? 11 ? ? ◆? ? ? ? ? ? ? ? ? ? ? ? ? ? ? 10mS ? ? ? ? ├┴. ? ? ? ? ? ? ? ? ? ? (1 / ? ? ? ?) x 10 ? ? ? ? ? ? ? ? ├┴.

? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? BLEN ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴. BLEN ? ? ? ? ? ? ? ? ? ? ◆? ? ? ? ? ? ? ? ? ? ? ? ? ├┴.

```
    SET RS232 9600                  ' RS232C ? ? ? ? ㉥?
    DIM I AS BYTE
    DIM A(12) AS BYTE
    ON RECV GOSUB 100              ' ? ? ? ? ㈑? ? 100 ? ? ? ? ? ?
10  GOTO 10                         ' ? ? ? ? ? ? ? ? ? ? ? ? ? ?

100 IF BLEN(0) < 12 THEN RETURN     ' ? ? ? 12 ? ? ? ? ? ? ? ? ? ? ?
    FOR I = 0 TO 11                 ' 12 ? ? ? ? ? ?
        GET A(I)            ' 12 ? ? (i)? ? ? ? ? ? ?
        LOCATE I,1
        PRINT CHR(A(I))             ' LCD ? ? ? ? (i)? ? ? ? ?
    NEXT I
    RETURN
```

? ? ? ? ? ? ? ? ? ? ? ◆? ? ? ? ? ? ? ? ? , ? ? ? ? ?┴ ? ? ? ? ? ? ? ? ?

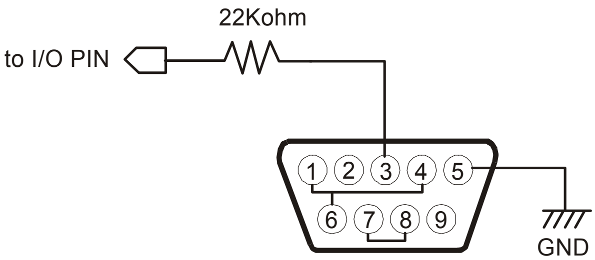
? ? ? ? ? ? ? ? ? ? ? ?┴? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴. (? ? ? ? ? ? ? ? ? ? ? ? ? ┴? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ├┴.) DELAY ? ├┴. ? ? ? ? ? ? ? ? ? ? ? ? ? GOTO 10 ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? , ? ├┴.

SERIN ? ? ? ? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ? ? ? SERIN ? ? ? ? ? ? ◆? ? ? ? ? ? ? ? ? ? ? ? ├┴. SERIN ? ? ? ? ? ? ? ? ? ? ? SERIN ? ? ? ? ? ? ? ? ? ◆? ? ? ? ? ? ? ? ? ┴? ? ? ├┴. SERIN ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? , ? ? I/O ? ? ? ? ? ? ? ? ◆? ? ? ? ? ? ├┴.

```
    DIM I AS BYTE
    DIM A(12) AS BYTE
10  SERIN 15,93,0,5000,10,[A(0)~12]
    FOR I = 0 TO 11
        LOCATE I,0
        PRINT CHR(A(I))
    NEXT I
    GOTO 10
```

SERIN ? ? ? ? ? ? ┼? ? ? ? ? ? ? ? ? ? ┴? ? ? ? ? ? ? ? ┴? ? ? ├┴. ? ? ? ? ? ◆? ? ? 12 ? ? ? ? ? ? , GET ? ? ? ? ? ? ? 10 ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? , ┴? ? ? ? ? ? ? ? ? ? ? , SERIN ? ? ? ? ? ? ? SERIN ? ? ? ? ? ? ? ? ? ? ? ┴? ? ? ? ? ? ├┴. SERIN ? ? ? ┼? ? ? ? ? ? ? ┬├┴. ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? , ? MAX232 ? ? ? ? ? ? ? ? , PC ? ? ? ? ? ◆? ? ? ? ? ? ? ? ? ├┴. (PICBASIC I/O ? ? ? ? ? ? ? ┴? ±12V ? ? ? ? ? ? ? ? ? ? ? ? ├┴.)



PC RS232C PORT
(DB-9 Mail Connector Side)

MEMO