BCC52CX

Technical Manual

Release 1.0
2/16/1988

Micromint, Inc.
115 Timberlachen Circle
Suite 2001
Lake Mary, FL, 32746

COPYRIGHT

Micromint, Inc.
115 Timberlachen Circle
Suite 2001
Lake Mary, FL, 32746

DISCLAIMER

While we have attempted to provide accurate and up to date information in this manual, MICROMINT INC. makes no representations or warranties respecting its contents. We reserve the right to make periodic changes to the text and to issue new editions of this manual without notification.

Occasionally we refer to other manufacturers' products. Such references do not constitute an endorsement of these products, but are included for the purpose of illustration or clarification. We do not intend such technical information and interface data to supersede information provided by individual manufacturers.

Table of Contents

Table of Contents (continued)

1.0 An Overview of the BCC52CX

The BCC52CX uses the new Micromint 80C52 CMOS 8-bit microcontroller chip which contains a ROM resident 8K byte BASIC interpreter. The BCC52CX computer/controller board has the 80C52, 64K bytes of RAM space, 64K bytes of EPROM space, a 27C256 EPROM programmer, 3 parallel ports, a serial terminal port with auto baud rate selection (RS232 - or - RS422) , a serial printer port, and is bus compatible with the BCC BUS and all the BCC series expansion boards.

It is particularly well suited for process control providing IF THEN, FOR NEXT, DO WHILE/UNTIL, ONTIME, and CALL statements among its broad repertoire of instructions (Figure 1.0-1 block diagrams the hardware and Figure 1.0-2 lists the software features). Calculations are handled in integer or floating point math and fully supported with trigonometric and logical operators. Because of its low system overhead it is extremely fast and efficient.

Unlike most one-shot EPROM programmers that fill the entire contents of an EPROM regardless of the application program size, BCC52CX treats the EPROM as "write once" mass storage for BASIC programs.

When a BASIC application program is saved to EPROM, it is tagged with an identifying ROM number and stored only in the amount of EPROM required to fit the program (plus header and EOF). Additional application programs can be stored to the same EPROM and recalled for execution by requesting a particular ROM number. A 27C256 EPROM affords 32 Kbytes of mass storage space (24 Kbytes is available if any I/O is used.) When it is full (a non-destructive EPROM FULL error will indicate this), simply erase the present EPROM or insert another. Finally, since this pseudo mass storage exists in directly addressable memory space (as opposed to cassettes or disks), it runs at full CPU speed and stored application programs are instantly accessible.

The BCC52CX bridges the gap between expensive intelligent control capabilities and hard to justify price sensitive control applications. Its full floating point BASIC is fast and efficient enough for the most complicated tasks while its cost effective design allows it to be considered for many new areas of implementation.

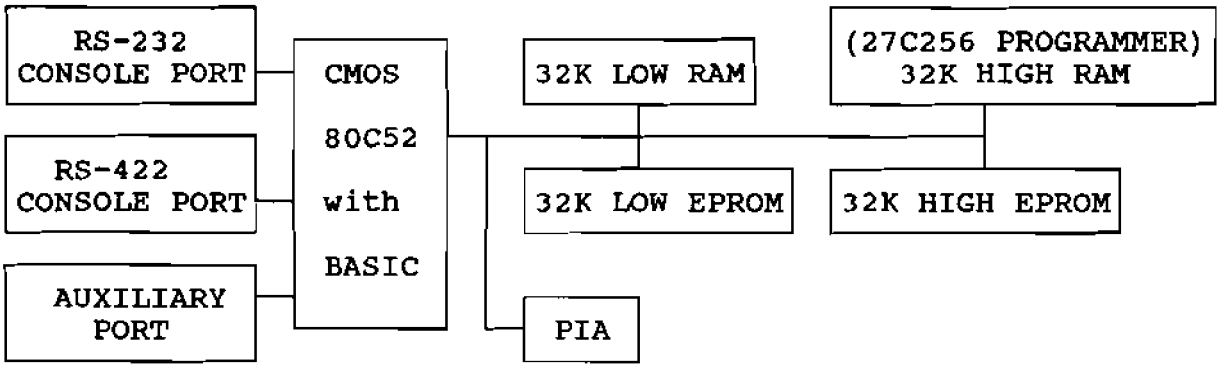


Figure 1.0-1 Block Diagram of the BCC52C

COMMANDS	STATEMENTS	OPERATORS
RUN	BAUD	+ (ADD)
CONT	CALL	/ (DIVIDE)
LIST	CLEAR	** (EXPONENTIATION
LIST#	CLEARS	* (MULTIPLY)
LIST@	CLEARI	- (SUBTRACT)
NEW	CLOCK0	LOGICAL AND
NULL	CLOCK1	LOGICAL OR
RAM	DATA	LOGICAL XOR
ROM	READ	LOGICAL NOT
XFER	RESTORE	ABS
PROG	DIM	INT
PROG1	DO-WHILE	SGN
PROG2	DO-UNTIL	SQR
PROG3	END	RND
PROG4	FOR-TO-STEP	LOG
PROG5	NEXT	EXP
PROG6	GOSUB	SIN
FPROG	RETURN	COS
FPROG1	GOTO	TAN
FPROG2	ON-GOTO	ATAN
FPROG3	ON-GOSUB	= (EQUAL)
FPROG4	IF-THEN-ELSE	> (GREATER THAN)
FPROG5	INPUT	< (LESS THAN)
FPROG6	LET	<> (NOT EQUAL)
	ONERR	ASC
	ONEX1	CHR
	ONTIME	CBY
	PRINT	DBY
	PRINT#	XBY
	PRINT@	GET
	PH0.	IE
	PH0.#	IP
	PH0.@	PORT1
	PH1.	PCON
	PH1.#	RCAP2
	PH1.@	T2CON
	PGM	TCON
	PUSH	TMOD
	POP	TIME
	PWM	TIMER0
	REM	TIMER1
	RETI	TIMER2
	STOP	XTAL
	STRING	MTOP
	UI0	LEN
	UI1	FREE
	UO0	PI
	UO1	
	LD@	
	ST@	
	IDLE	
	RROM	

Figure 1.0-2 Language Features of 80C52-BASIC

2.0 The BCC52CX Computer/Controller Board

The BCC52CX computer/controller is a single board controller/ development system. The BCC52CX's 19 chip circuit is a compact 4 1/2 by 6 1/2 inches. It contains 128K RAM/EPROM space, an EPROM programmer, 3 parallel ports, a console serial port (RS232/RS422), and a serial output printer port.

There are five main sections to the BCC52CX board: processor, address decoding and memory, parallel I/O, serial I/O, and EPROM programmer.

2.1 Processor

The BCC52CX computer/controller board is based on Micromint's 80C52-BASIC chip which is a preprogrammed version of the 80C52 microcontroller (block diagrammed in Figure 2.1-1). The 80C52 is a CMOS version of INTEL's 8052AH 8 bit controller chip. The 80C52 contains 8K bytes of on-chip ROM, 256 bytes of RAM, three 16 bit counter/timers, 6 interrupts, and 32 I/O lines. In the 80C52-BASIC the ROM is a masked BASIC interpreter and the I/O lines are redefined to address, data, and control lines.

The 80C52-BASIC has a 16 bit address and 8 bit data bus (the 8 least significant address bits (A0-A7) and the data bus (D0-D7) are multiplexed together similar to the 8085 and Z8). When the chip is powered up it sizes consecutive external memory from 0000H to end of memory (or memory failure) by alternately writing 55H and 00H to each location. A minimum of 1K bytes of RAM is required for BCC52CX to function (32K bytes is standard) and RAM must be located starting at 0000H.

Three control lines, RD (pin 17), WR (pin 16), and PSEN (pin 29) partition the address space as 64K bytes each of program and data memory. However, when DATA memory and PROGRAM memory are separated, user called assembly routines and EPROM programming are unsupported in data memory. For that reason, the BCC52CX as designed, is addressed as separated DATA and PROGRAM from 0000H - 7FFFH and (user selectable) combined or separated DATA/PROGRAM memory from 8000H - 0FFFFH. The addressing logic is as follows:

1) The RD and the WR pins on the 80C52 chip enable RAM memory from 0000H to 7FFFH. Addresses are used to decode the chip select (CS) for the RAM devices and RD and WR are used to enable the OE and WE or (WR) pins respectively.

2) PSEN is used to enable EPROM memory from 2000H (optionally 0000H) to 7FFFH. Addresses are used to decode the chip select (CS) for the EPROM devices and PSEN is used to enable the OE pin.

3) Between 8000H and 0FFFFH both RD and PSEN are used to enable either EPROM or RAM memory. RD and PSEN are logically "ANDED" through U5, a 74HCT08. The WR pin on the chip is used to write to RAM memory in this same address space. (optionally the DATA/PROGRAM spaces can be separated as in 1 & 2 above)

BCC52CX reserves the first 512 bytes of External Data Memory to implement two "software" stacks. These are the control stack and the arithmetic stack or Argument Stack. Understanding how the stacks work is only necessary if the user wishes to link 80C52-BASIC and 80C52 Assembly Language routines. The details of how to link to assembly language are covered in the Assembly Language Linkage section of the MCS BASIC-52 User's Manual.

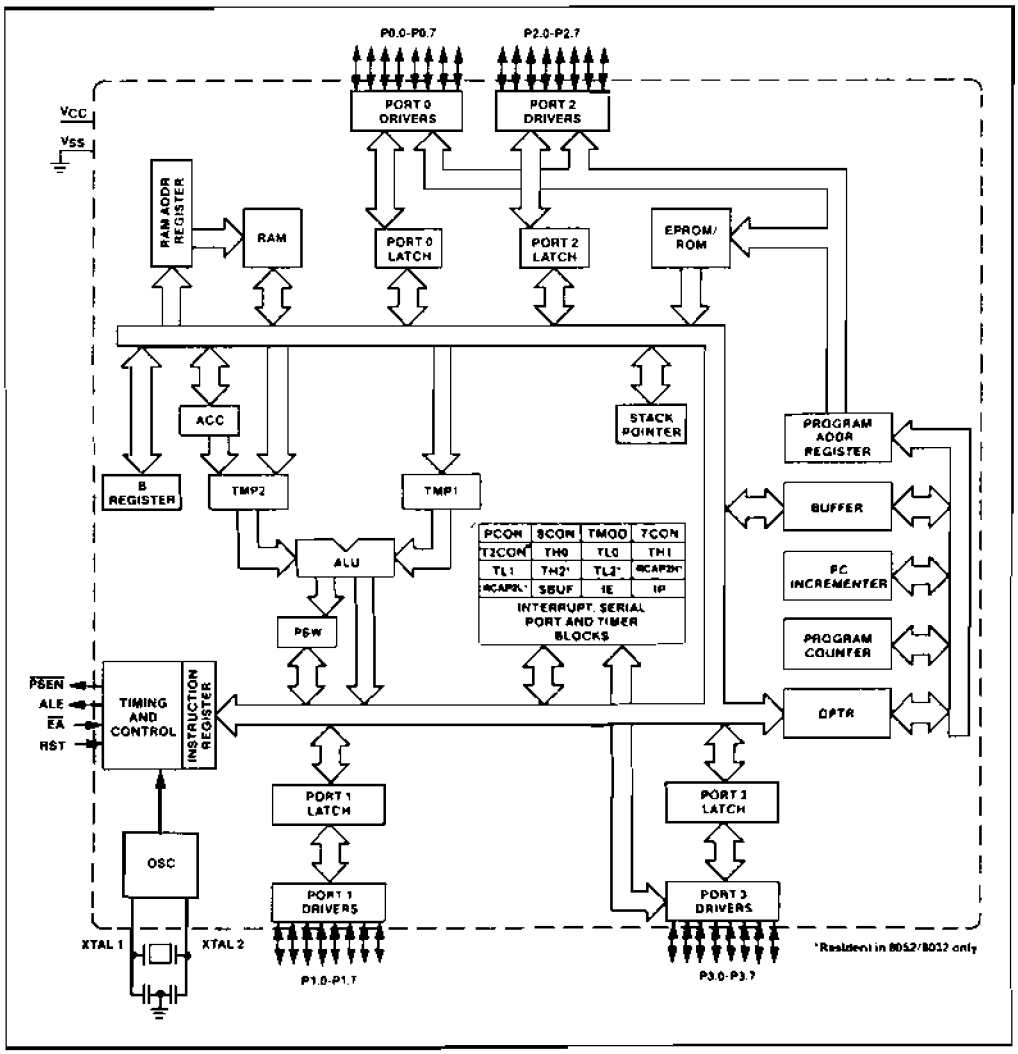


Figure 7-1. MCS-51 Architectural Block Diagram

Figure 2.1-1 Block Diagram of Micromint's 80C52-BASIC Chip

The control stack occupies locations 96 (60H) through 254 (0FEH) in external RAM memory. This memory is used to store all information associated with loop control (i.e. DO-WHILE, DO-UNTIL, and FOR-NEXT) and BASIC subroutines (GOSUB). The stack is initialized to 254 (0FEH) and "grows down."

The Argument Stack occupies locations 301 (12DH) through 510 (1FEH) in external RAM memory. This stack stores all constants that BASIC is currently using. Operations such as Add, Subtract, Multiply, and Divide always operate on the first two numbers on the Argument Stack and return the result to the Argument Stack. The argument stack is initialized to 510(1FEH) and "grows down" as more values are placed on the Argument Stack. Each floating point number placed on the Argument Stack requires 6 BYTES of storage.

The stack pointer on the 80C52 (Special Function Register, SP) is initialized to 77 (4DH). The 80C52's stack pointer "grows up" as values are placed on the stack.

2.2 Address Decoding

The 80C52 uses all of the first 32K (0H-7FFFH) as split memory. DATA (RAM) memory is enabled by the 80C52 RD line and PROGRAM (EPROM) memory is enabled by the 80C52 PSEN line. The BCC52CX can enable the same memory by either RD or PSEN only on addreses above 8000H. (This differs slightly from the BCC52, using 8K devices allowed the user a bit more flexibility.) A thorough understanding of each space is essential.

DATA MEMORY - Basic programs execute out of DATA memory. All memory mapped I/O (PIAs, A-Ds, D-As, etc.) are accessed through DATA memory. Source code input with TEDIT is stored in DATA memory as is the assembled Object code using ASM or ASMS (NOTE the keyword here is stored - OBJECT CODE CANNOT BE EXECUTED FROM DATA MEMORY.)

CODE MEMORY - The basic interpreter executes out of PROGRAM memory from within the processor. Extensions to BASIC, written in assembly language, execute from PROGRAM memory. Interrupt jump-vectors are in PROGRAM memory. Assembly language routines, such as 'CALLS' from BASIC are executed out of PROGRAM memory.

The 3 most significant address lines (A13-A15) are connected to a 74HCT157 decoder chip, U10, which separates the addressable range into 8-8K memory segments, each with its own chip select (2Y0-1Y3). The 4 lower chip selects (0000H, 2000H, 4000H & 6000H) are OR'd to enable the low bank of memory devices, U12, a 32K X 8 static RAM (DATA memory) and U14, a 32 X 8 CMOS EPROM (optional)(PROGRAM memory). The upper chip selects (8000H, 0A000H, 0C000H & 0E000H) are OR'd to enable the high bank of memory devices, U13, a 32K X 8 static RAM (optional)(DATA memory) or CMOS EPROM (optional)(combined DATA/PROGRAM memory) and U15, a 32K X 8 CMOS EPROM (optional)(PROGRAM memory). U13 is also the EPROM programming socket intended for 27C256 CMOS EPROMS.

A second decoder, U14, partitions E800H-EFFFH as eight 256 byte I/O blocks. Rather than simply using the available E000H strobe from U10 alone which would occupy a 2000H address space for a single PIA chip, IC16 allows many peripherals to share one 8K address block by using only a 256 byte address range. This addressing convention is consistent with other BCC expansion boards and it is easy to configure a 64 channel A/D or 64 channel power I/O system using this board with a number of peripheral cards and still end up with a good deal of DATA memory space for BASIC programs.

NOTE - Using I/O peripherals requires DATA memory space. The upper 8K of DATA memory space is used for memory mapped I/O, this lowers the available room in a 27C256 to 24K. The programmer will only allow 24K of the possible 32K to be used when jumper JP3 selects I/O at 0E000H.

2.3 Parallel I/O

The BCC52CX contains an 82C55 PIA which provides three 8 bit software configurable parallel ports. The three I/O ports, labeled A, B, and C and a write only mode configuration port occupy 4 consecutive addresses in one of the 8 jumper selectable I/O blocks. With jumper JP3 selecting I/O at 0E000H, U11 decodes this block into 8 possible ranges (0E800H, 0E900H, 0EA00H, 0EB00H, 0EC00H, 0ED00H, 0EE00H & 0EF00H.) Using the XBY() operator in BASIC, data can be written to and read from this PIA (you are probably more familiar with PEEK and POKE.

PEEK at a value in address location 0E802H with:

```
PRINT XBY(0E802H)
```

and POKE a value into address location 0E802H with:

```
XBY(0E802H)=VALUE
```

The three parallel ports (24 bits) are connected, with two grounds, to a 26 pin flat ribbon cable header.

2.4 Serial I/O

There are two serial ports on the BCC52CX board. One is for the console I/O terminal (J1/RS232 or J7&8/RS422) and the other is an auxiliary serial output (J5/RS232) frequently referred to as the line printer port. When using an 11.0592 MHz crystal, the console port does auto baud rate determination on power up (a preset baud rate can alternatively be stored in EPROM as well). It will function at 19200 bps with no degradation in operation. The console serial port format is 8 data bits, no parity, 1 stop bit.

The BAUD[expr] statement is used to set the baud rate for the line printer port. In order for this statement to properly calculate the baud rate, the crystal (special function operator -XTAL) must be correctly assigned (e.g. XTAL = 9000000). BASIC assumes a crystal value of 11.0592 MHz if no XTAL value is assigned.

The main purpose of the software line printer port is to let the user make a "hard copy" of program listings and/or data. The command LIST# and the statement PRINT# direct outputs to the software line printer port. If the BAUD[expr] statement is not executed before a list# or print# command/statement is entered, the output to the software line printer port will be at about 1 bps and it will take a long time to output something. It is necessary to assign a BAUD rate to the software printer port before using LIST# or PRINT#. The maximum baud rate that can be assigned by the BAUD statement depends on the crystal but, 4800 bps is a reasonable maximum rate.

The MC145406 level shifter, U8, converts the TTL logic levels from the console and line printer ports to RS-232 (the TTL serial lines are also connected to the bus to allow use of the TERM-MITE smart terminal board without RS-232 voltages). A pair of 74176 differential bus transceivers, U18 and U19, provide the same serial console I/O, as U8 but RS-422 (half or full duplex).

The BCC52CX requires only about 100 milliamps at +5V to function. +/- 12V is required for external RS-232C communication and +12.5V for EPROM programming.

2.5 EPROM Programmer

One of the more unique and powerful features of the BCC52CX is that it has the ability to execute and save programs in an EPROM. The 80C52 chip actually generates all of the timing signals needed to program 27C256 EPROMs. Saving programs in EPROMS is a much more attractive alternative to diskettes or cassette tape, especially in control and/or noisy environments.

Port 1, bit 4 (U2 pin 5) is used to provide a 1 or 50 millisecond programming pulse. The length of the programming pulse is determined by whether we are programming with the standard or the fast programming algorithm. BCC52 calculates the length of the programming pulse from the assigned crystal value. The accuracy of this pulse is within 10 CPU clock cycles. This pin is normally in a logical high (1) state. It is asserted low (0) to program the EPROMs.

Port 1, bit 5 (U2 pin 6) is used to enable the EPROM programming voltage. This pin is normally in a logical high (1) state. Prior to the EPROM programming operation, this pin is brought to a logical low (0) state. This pin is used to turn the high voltage, 12.5 volts required to program the EPROMS on or off.

BCC52CX does not save a single program on an EPROM (unless the program reaches the limits of the EPROM). In fact, it can save as many programs as the size of the EPROM memory permits. The programs are stored sequentially in the EPROM and any program can be retrieved and executed. This sequential storing of programs is referred to as the EPROM FILE. The following commands permit the user to generate and manipulate the EPROM FILE.

RAM (cr) and ROM [integer] (cr)

These two commands tell the BCC52CX interpreter whether to select the current program (the current program is the one that will be displayed during a List command and executed when Run is typed) out of RAM or EPROM. The RAM address is assumed to be 512 (200H) and the EPROM address begins at 32,784 (8010H)

RAM

When RAM (cr) is entered, BCC52CX selects the current program from RAM Memory. This is usually considered the "normal" mode of operation and is the mode that most users interact with the command interpreter.

ROM

When ROM [integer] (cr) is entered BCC52CX selects the current program out of EPROM memory. If no integer is typed after the ROM command (i.e. ROM (cr)), BCC52CX defaults to ROM 1. Since the programs are stored sequentially in EPROM the integer following the ROM command selects which program the user wants to run or list. If you attempt to select a program that does not exist (i.e. you type in ROM 8 and only 6 programs are stored in the EPROM) the message 'Error: Prom Mode' will be displayed. The error is non destructive and you can retype the correct command.

The BCC52CX does not transfer the program from EPROM to RAM when the ROM Mode is selected and you cannot EDIT a program in ROM. Attempting to do so will result in an error message.

Since the ROM command does not transfer a program to RAM, it is possible to have different programs in ROM and RAM simultaneously. The user can "flip" back and forth between the two modes at any time. Another added benefit of not transferring a program to RAM is that all of the RAM memory can be used for variable storage if the program is stored in EPROM. The System Control Values - MTOP and FREE always refer to RAM not EPROM.

XFER (cr)

The XFER (transfer) command transfers the current selected program in EPROM to RAM and then selects the RAM mode. After the XFER command is executed, the user may edit the program in the same manner any RAM program may be edited.

PROG (cr)

The PROG Command programs the resident EPROM with the current selected program (programming is the only time that the +12.5V programming voltage needs to be applied). The current selected program may reside in either RAM or EPROM. After PROG (cr) is typed, BCC52CX displays the number in the EPROM File the program will occupy.

PROG1 (cr)

Normally, after power is applied to the BCC52CX device, the user must type a "space" character to initialize the 80C52's console port. As a convenience, BCC52CX contains a PROG1 command. What this command does is program the resident EPROM with the baud rate information. So, the next time the BCC52CX is "powered up," i.e. reset, the chip will read this information and initialize the serial port with the stored baud rate. The "sign-on" message will be sent to the console immediately after BASIC device completes its reset sequence. The "space" character no longer needs to be typed.

PROG2

The PROG2 command does everything the PROG1 command does, but instead of "signing-on" and entering the command mode, the BCC52CX immediately begins executing the first program stored in the resident EPROM.

By using the PROG2 command it is possible to run a program from a reset condition and never connect the BCC52CX board to a console. In essence, saving PROG2 information is equivalent to typing ROM 1 and RUN in sequence. This is ideal for control applications, where it is not always possible to have a terminal present. In addition, this feature permits the user to write a special initialization sequence in BASIC or Assembly Language and generate a custom "sign-on" message for specific applications.

See the MCS BASIC-52 manual for information on additional PROG statements (PROG3-PROG6).

Extra circuitry needed for programming the 27C256

The processor must read the EPROM to find the EOF marker of the last program saved. The *PE (programming enable) line has dropped, normally this turns on the programming voltage. If the 12.5 volts were applied to the EPROM, any *CS (chip select) to the EPROM would program a byte. To prevent this, the *PE is held high (off) until the first *PP (programming pulse) is detected. Since the *PP comes after the reads to the EPROM, the EPROM is not programmed until the processor has found its place, following the last EOF marker. *PP is OR'd with *CS and now takes over until programming is complete.

3.0 Powering up the Board

Power requirements for the BCC52CX are as follows:

+ 5 Volts +/- 5% @ 100 ma
+ 12 Volts +/- 20% @ 10 ma (required only for RS-232)
- 12 Volts +/- 20% @ 10 ma (required only for RS-232)
+ 12.5 Volts +/- 2% @ 50 ma
(required only for 27C256 EPROM programming)

After applying power the BCC52CX:

1) Clears the internal 80C52 memory
2) Initializes the internal registers and pointers
3) Tests, clears, and sizes the external memory

BASIC then assigns the top of external RAM to the System Control Value (MTOP) and uses this number as the random number seed. BASIC assigns the default crystal value, 11.0592 Mhz, to the System Control Value (XTAL) and uses this default value to calculate all time dependent functions, such as the EPROM programming timer and the interrupt driven Real Time Clock. Finally, BASIC checks external memory location 8000H to see if the baud rate information is stored. If the baud rate is stored, BASIC initializes the baud rate generator (the 80C52's Special Function Register -T2CON) with this information and signs on. If not, BASIC interrogates the serial port input and waits for a space character to be typed (auto baud rate detection).

If you have entered nothing on the console device BASIC will sit there waiting and appear inoperative to the uninitiated. Simply type a space and the console device should display the following:

```
*MCS-51(tm) BASIC V1.1*

READY
>
```

To see if the processor is operating correctly, type:

```
>PRINT XTAL, TMOD, TCON, T2CON
```

(BASIC should respond with the control and special function values)

```
 11059200 16 244 52

READY
>
```

CAUTION - The 12.5 volt programming power source should only be applied **after** the BCC52CX is powered up and should be disconnected **before** the BCC52CX is powered down. **Failure to do so may permanently damage your EPROM ! !**

4.0 A Word About the BASIC

The BCC52CX is oriented toward process control and is significantly more powerful than a tiny BASIC. Since most of you are familiar with BASIC, individual instructions such as DO WHILE and FOR NEXT etc. will not be described here. A few of the pertinent features which demonstrate the exceptional small package performance of the BCC52CX will be discussed.

MCS BASIC-52 contains a minimum level line editor. Once a line is entered the user may not change the line without re-typing the line (see ROM A or ROM A&B for editing, renumbering, etc.) However, it is possible to delete characters while a line is in the process of being entered. This is done by entering a Rubout or Delete character (7FH). The Rubout character will cause the last character entered to be erased from the text input buffer. Additionally, a control-D will cause the entire line to be erased.

4.1 Variables and Expressions

The range of numbers that can be represented in BCC52CX is:

+1E-127 to + .99999999E + 127.

There are eight digits of significance. Numbers are internally rounded to fit this precision. Numbers may be entered and displayed in four formats: integer, decimal, hexadecimal, and exponential. **EXAMPLE: 129, 34.98, 0A6EH, 1.23456 E+3**

Integers are numbers that range from 0 to 65535 or 0FFFFH. All integers can be entered in either decimal or hexadecimal format. A Hexadecimal number is indicated by placing the character "H" after the number (e.g. 170H). When an operator, such as AND requires an integer, BASIC will truncate the fraction portion of the number so it will fit the integer format. All line numbers are integers.

A variable can be either a letter, (i.e. A, X, I), a letter followed by a number, (i.e. Q1, T7, L3), a letter followed by a One Dimensioned expression, (i.e. J(4), G(A+6), I(10*SIN(X))), or a letter followed by a number followed by a One Dimensioned expression (i.e. A1(8), P7(DBY(9)), W8(A+B). Variables that include a One Dimensioned expression are often referred to as dimensioned or arrayed variables. Variables that only involve a letter or a letter and a number are called SCALAR variables.

BASIC allocates variables in a "static" manner. That means each time a variable is used, BASIC allocates 8 bytes specifically for that variable. This memory cannot be de-allocated on a variable by variable basis. That means if you execute a statement like Q=3, later on you cannot tell BASIC that the variable Q no longer exists and "free up" the 8 bytes of memory that belong to Q. The user can clear the memory allocated to variables with a CLEAR statement.

Relative to a dimensioned variable, it takes BASIC a lot less time to find a scalar variable. That's because there is no expression to evaluate in a scalar variable. If you want to make a program run as fast as possible, use dimensioned variables only when you must. Use scalars for intermediate variables, then assign the final result to a dimensioned variable.

An expression is a logical mathematical expression that involves Operators (both unary and dyadic), Constants, and Variables. Expressions can be simple or quite complex, i.e. 12*EXP(A)/100, H(1)+55, or (SIN(A)*SIN(A)+COS(A)*COS(A))/2. A "stand alone" variable [var] or constant [const] is also considered an expression.

4.2 Real Time Operation after RUN

After RUN (cr) is typed all variables are set equal to zero, all BASIC evoked interrupts are cleared and program execution begins with the first line number of the selected program. The RUN command and the GOTO statement are the only way the user can execute a program in the Command mode. Program execution may be terminated at any time by typing a control-C on the console device.

Unlike some BASIC interpreters that allow a line number to follow the Run command (i.e., RUN 100), BASIC-52 does not permit such a variation on the Run command. Execution always begins with the first line number. To obtain the same functionality as the RUN[ln num], use GOTO[ln num] in the direct mode.

The CLOCK1 statement enables the software real time clock in the BCC52CX. The special function operator time is incremented once every 5 milliseconds after the CLOCK1 statement has been executed. The CLOCK1 statement uses timer/counter 0 in the 13-bit mode to generate an interrupt once every 5 milliseconds. Because of this, the special function operator time has a resolution of 5 milliseconds.

BASIC automatically calculates the proper reload value for timer/counter 0 after the crystal value has been assigned (i.e., XTAL=value. If no crystal value is assigned, MCS BASIC-52 assumes a value 11.0592 MHz). The special function operator time counts from 0 to 65535.995 seconds. After reaching a count of 65535.995 seconds time overflows back to a count of zero.

The interrupts associated with the CLOCK1 statement cause BASIC programs to run at about 99.6% of normal speed. That means that the interrupt handling for the real time clock feature only consumes about .4% of the total CPU time. This is very small interrupt overhead. The CLOCK0 (zero) statement disables or "turns off" the real time clock feature.

The TIME statement is used to retrieve and/or assign a value to the real time clock after the CLOCK1 statement enables it. TIME = 5 presets the real time clock to 5 seconds while ONTIME 30,100 causes the program to jump to line 100 when the real time clock reaches 30 seconds.

Finally, PWM is an interesting statement that might be useful to literally add bells and whistles to your next control application. PWM stands for Pulse Width Modulation. What it does is generate a user defined pulse sequence on U2 pin 3.

The statement appears as PWM 50,50,100. The first expression following PWM is the number of clock cycles the pulse will remain high. A clock cycle is equal to 1.085 microseconds (11.0592 MHz Xtal). The second expression is the number of clock cycles the pulse will remain low and the third expression is the total number of cycles the user wishes to output. All expressions in the PWM statement must be valid integers (i.e. between 0 and 65535 (0FFFFH) inclusive) and the minimum value for the first two expressions is 20.

These are only a few of the 103 commands, statements, and operators in BASIC-52. The INTEL MCS BASIC-52 User's Manual describes them in detail.

5.0 Using the 82C55 Parallel Ports

Three parallel input/output ports are provided on the BCC52CX Computer/Controller Board with an 82C55 peripheral interface adapter chip. The 24 I/O bits and two grounds are available on a 26 pin header (J6) between the 82C55 and the RS-232C connector.

The 82C55 is a programmable interface device. The 24 I/O lines are divided into three ports, A, B, and C, configured either as input, output, or handshaking lines under software control. A control register defines the characteristics of the 24 I/O bits. To the BCC52CX, the three I/O ports and control register appear as three memory addresses as follows. These addresses can be user set to select one of eight possible locations.

JP3 BASE	+ JP4 OFFSET	= PORT A	PORT B	PORT C	CONTROL REGISTER
0E000H	800H	0E800H	0E801H	0E802H	0E803
"	900H	0E900H	0E901H	0E902H	0E903
"	0A00H	0EA00H	0EA01H	0EA02H	0EA03
"	0B00H	0EB00H	0EB01H	0EB02H	0EB03
"	0C00H	0EC00H	0EC01H	0EC02H	0EC03
"	0D00H	0ED00H	0ED01H	0ED02H	0ED03
"	0E00H	0EE00H	0EE01H	0EE02H	0EE03
"	0F00H	0EF00H	0EF01H	0EF02H	0EF03

Figure 5.0-1 Parallel Port Memory Address Possibilities

The example shows a Base Address of 0E000H and an Offset Address of 0800H selected giving a PORT A address of 0E800H. (0E000H + 800H = 0E800H)

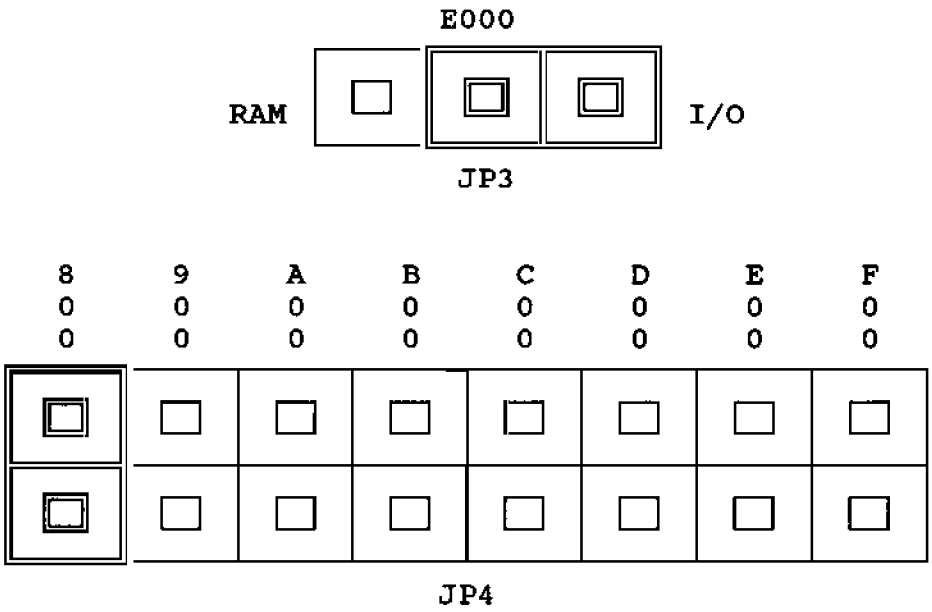


Figure 5.0-2 Parallel Port Addressing Example

The 24 lines are divided into two groups of 12 lines, group A (port A and the upper half of port C) and group B (port B and the lower half of port C). The functional configuration of each port by the system software. In essence, the BCC52CX outputs a control word to the 82C55 which contains information such as "mode", "bit set", "bit reset", etc., that initializes the 82C55 (the control register is a write only location and the control word cannot be read by examining the contents of the control register address).

When the power to the expansion board is turned on, the 82C55 is in an unknown configuration. Before the ports can be used they must be initialized by loading a control word into the control register address. For example, the BASIC statement 'XBY(0E803H)=80H' will load the value 80H into the control register. The value 80H sets all three ports to mode 0 operation and output (bit 7 of the control register must always be set to logic 1). At this point, 8 bit values can be directed to the specific ports with the 'XBY(address)' command. Outputing 56 hex from port B is done simply by 'XBY(0E801H)=56H'.

The three ports can just as easily be configured all as mode 0 inputs by loading 9BH as the control word. The command is 'XBY(0E803H) = 9BH'. Reading the value of input port A then becomes 'PRINT XBY(0E800H)'. The following is a list of control word values for some typical 82C55 port configurations. To use any of them, simply load the control register address with the XBY command.

Control Word Value	Port A	Port B	Port C
80H	output	output	output
89H	output	output	input
82H	output	input	output
8BH	output	input	input
9BH	input	input	input
92H	input	input	output
99H	input	output	input
90H	input	output	output

Figure 5.0-3 82C55 (U17) Port Configuration

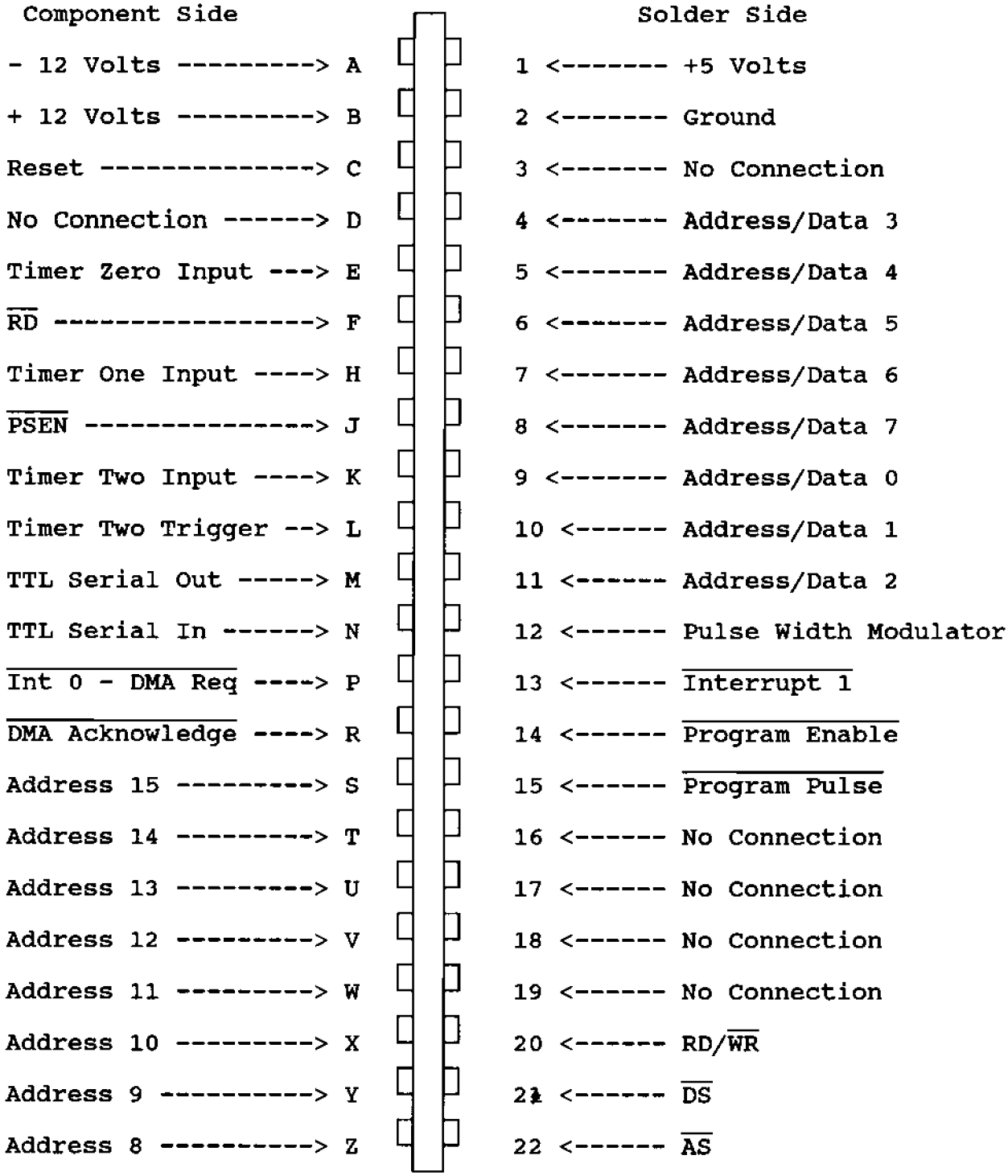
A complete specification of the 82C55 is not included in this manual. Should you need to configure the 82C55 for some more complicated I/O configuration or require the use of handshaking refer to an 82C55 DATA sheet by one of the following manufacturers:

Intel Corporation
3065 Bowers Avenue
Santa Clara
California 95051

NEC Electronics U.S.A. Inc.
One Natick Executive Park
Natick
Massachusetts 01760

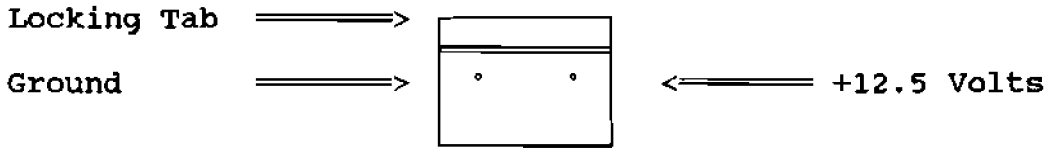
National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara
California 95051

BCC BUS Pin Configuration
(End View - Edge Card Connector)



J2

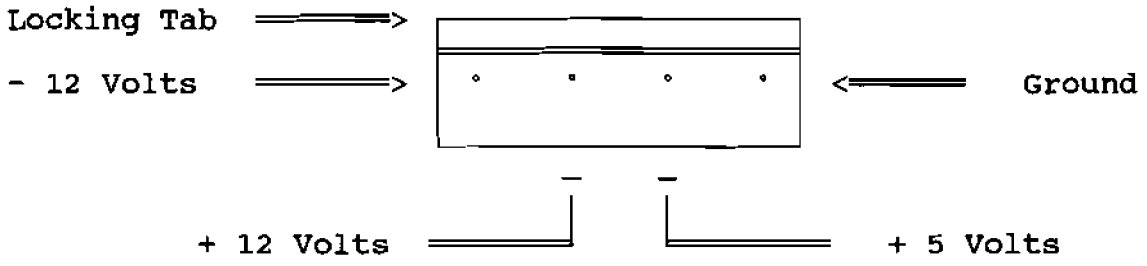
12.5 Volt Programming Voltage Input
(2 Pin Right Angle Molex Connector - End View)



2 Pin Molex Right Angle Connector

J3

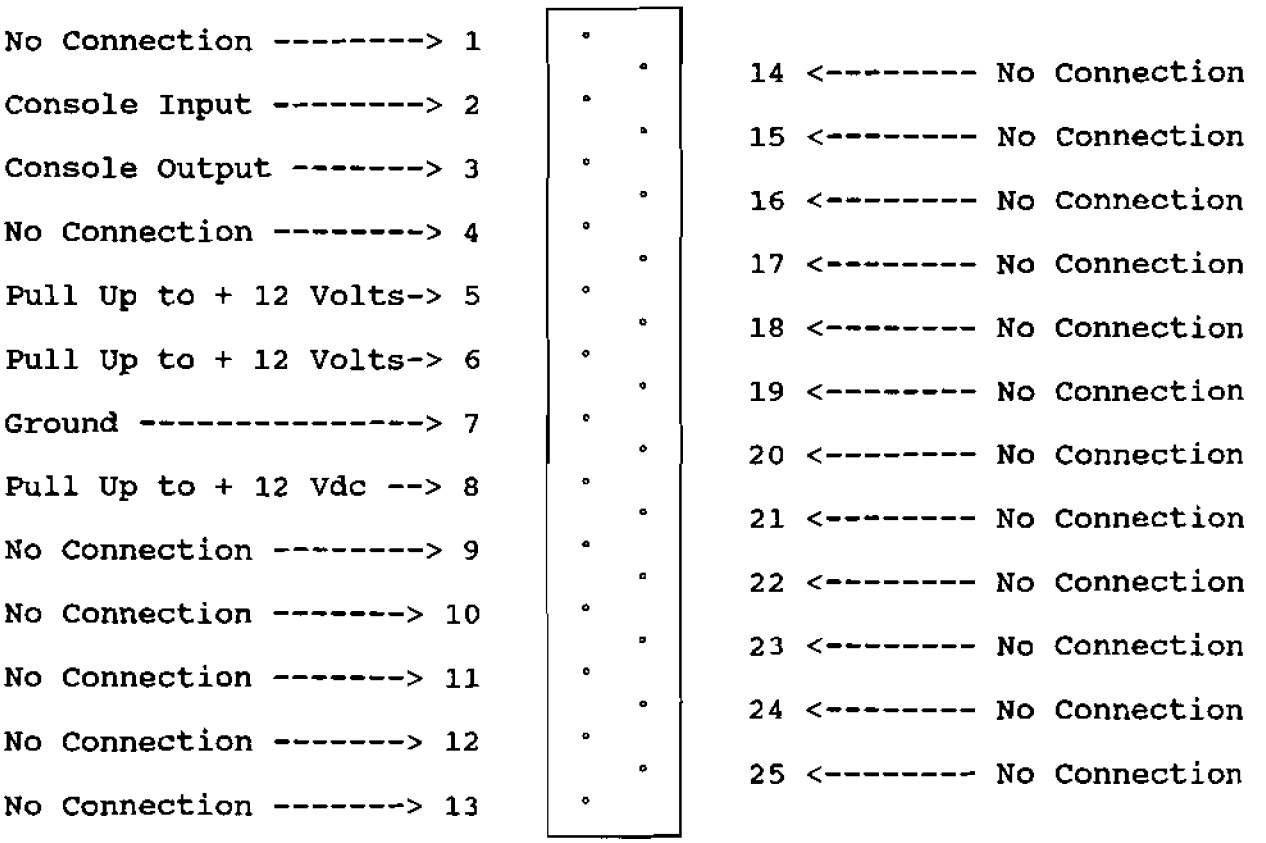
4 Pin Molex Power Connector (Top View)



NOTE : Jumpers JP1 - JP9 are shown as shipped by the factory. They are set for 32K of contiguous RAM in U12 (DATA memory only), 24K of contiguous EPROM in U14 (PROGRAM memory only) and 24K of combined DATA/PROGRAM memory in U13 (the TOP 8K reserved for I/O)

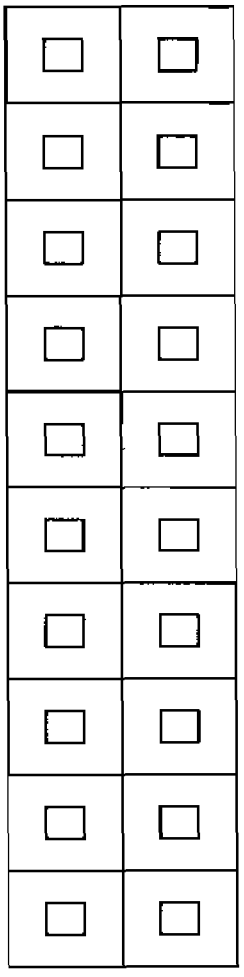
J4

Console Serial Connector
(DB-25S RS-232 Connector - Top View)



J5

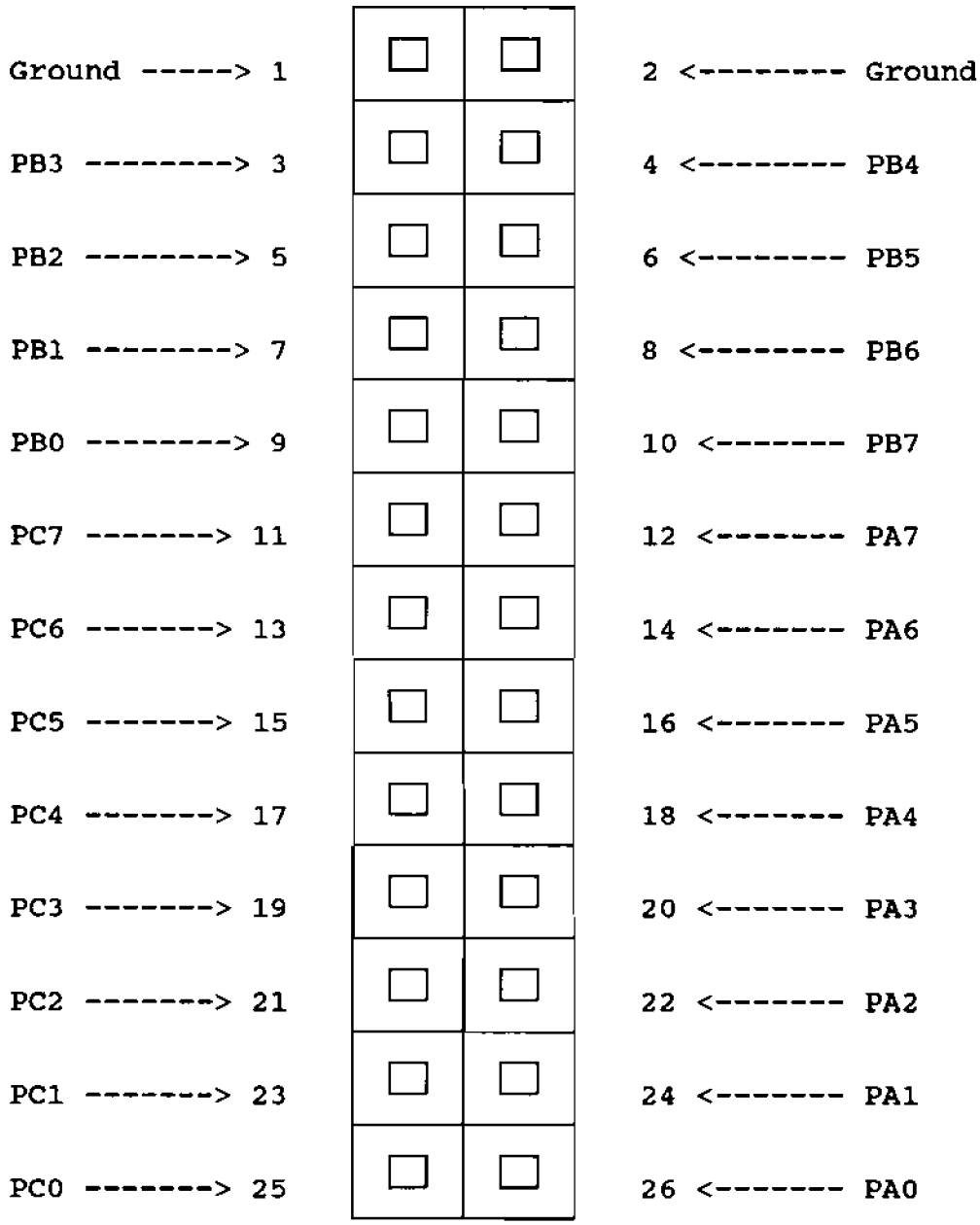
Serial Printer Connector
(2 X 10 Pin Berg Header - Top View)



No Connection -----> 1		14 <----- No Connection
Aux Serial Out ----> 2		15 <----- No Connection
No Connection -----> 3		16 <----- No Connection
No Connection -----> 4		17 <----- No Connection
Pull Up to +12 Vdc > 5		18 <----- No Connection
No Connection -----> 6		19 <----- No Connection
Ground ------------> 7		20 < Pull Up to +12 Vdc
No Connection -----> 8		21 <----- No Connection
No Connection -----> 9		22 <----- No Connection
No Connection -----> 10		23 <----- No Connection

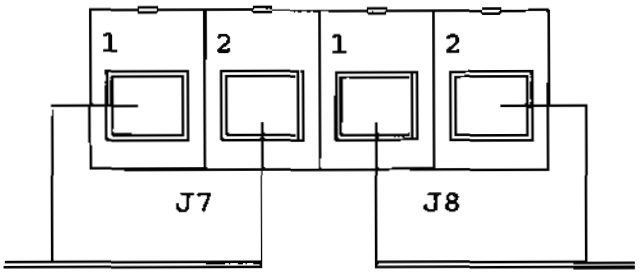
J6

82C55 Parallel Port Connector (Top View)
(2 X 13 Pin Berg Header)



J7 & J8

RS-422 Connector (Top View)
(1 X 4 Screw Terminal Connector)



FULL = TRANS

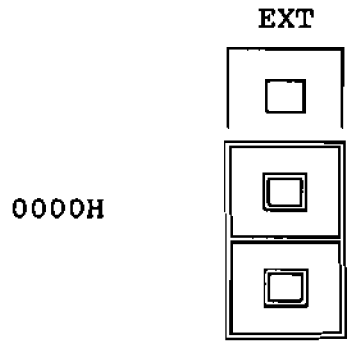
HALF = TRANS/
RECEIVE

FULL = RECEIVE

HALF = +5 VOLTS (1)
GROUND (2)

JP1

U2 MODE SELECTION Jumper
(1 X 3 Pin Berg Header - Top View)

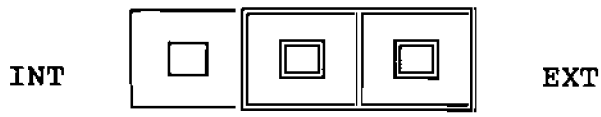


<== Jumper shown selecting INTERNAL CODE at 0000H

***** NOTE ***** BCC52CX board is shipped **WITHOUT** JP9 installed. User may install this header for 80C31 operation. Jumpering will ground pin 31 of U5. Refer to Intel's MICROCONTROLLER HANDBOOK for 8031 information.

JP2

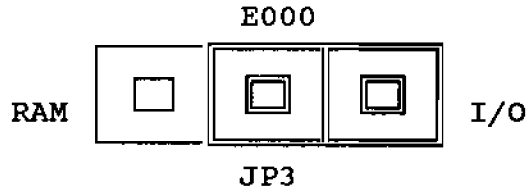
U14 Function Selection
(1 X 3 Pin Berg Header - Top View)



PROGRAM Memory 00000H

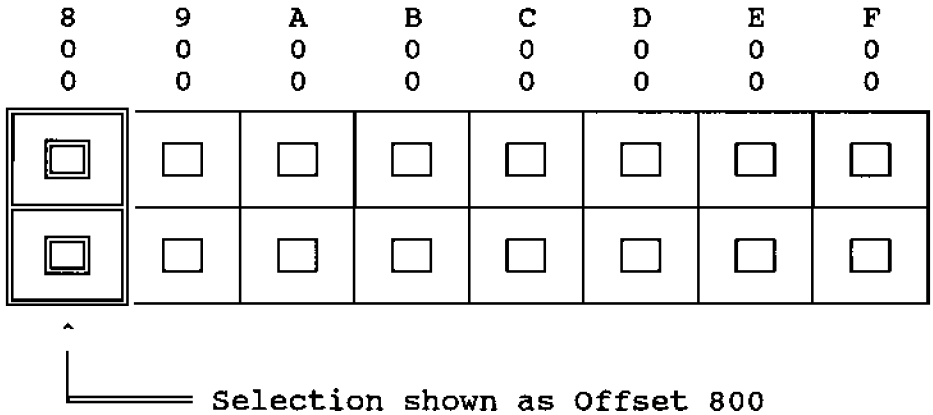
JP3

0E000H Function Selection
(1 X 3 Pin Berg Header - Top View)



JP4

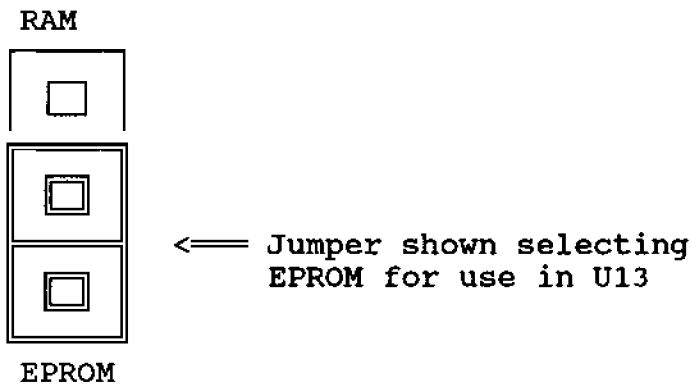
82C55 Offset Address Selection
(2 X 8 Pin Berg Header - Top View)



Base Address = 0E000H (JP3)
Offset Address = + 800H (JP4)
Actual Address = 0E800H

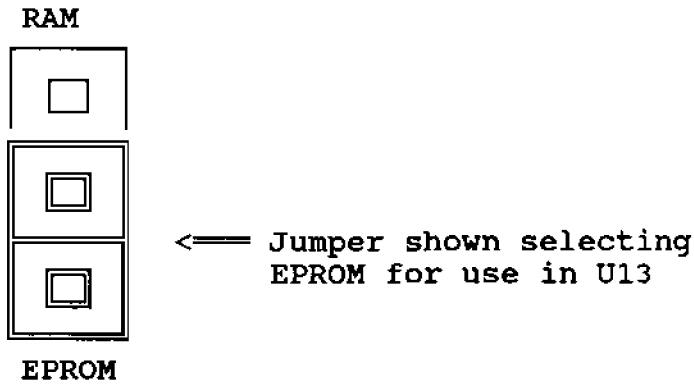
JP5

U13 RAM/EPROM SELECTION (pin 27)
(1 X 3 Pin Berg Header - Top View)



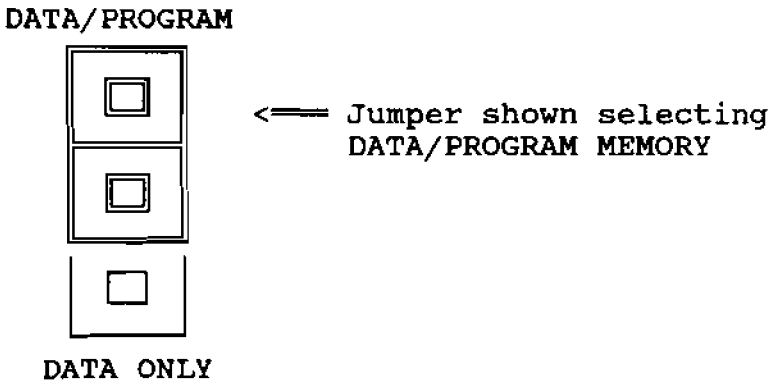
JP6

U13 RAM/EPROM SELECTION (pin 1)
(1 X 3 Pin Berg Header - Top View)



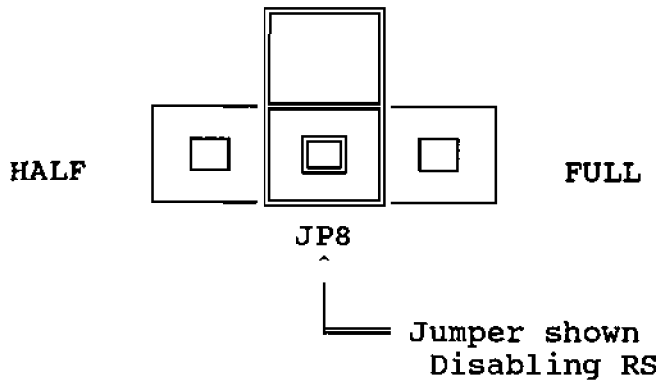
JP7

U13 DATA Memory or DATA/PROGRAM Memory Selection
(1 X 3 Pin Berg Header - Top View)



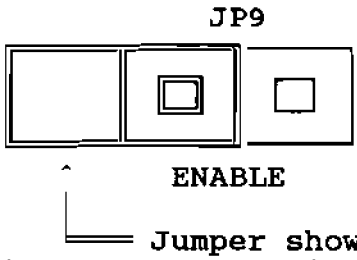
JP8

RS422 Selection
(1 X 3 Pin Berg Header - Top View)



JP9

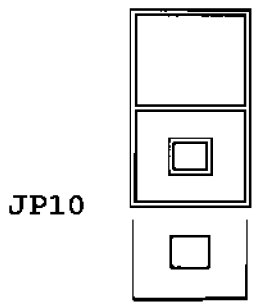
TERMINATION SELECTION
(1 X 2 Pin Berg Header - Top View)



Jumper shown
Disabling 100 OHM termination of U18

JP10

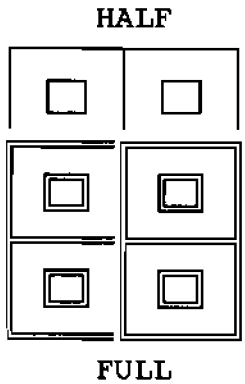
TERMINATION SELECTION
(1 X 2 Pin Berg Header - Top View)



<== Jumper shown
Disabling 100 OHM
termination of U19

JP11

RS422 HALF/FULL DUPLEX SELECTION
(2 X 3 Pin Berg Header - Top View)

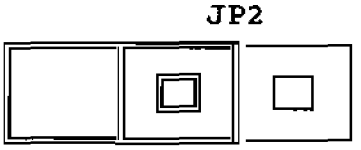


<== Jumpers shown
Selecting FULL DUPLEX

JP12

Internal 12 Volt Selection
(for programming)

(1 X 2 Pin Berg Header - Top View)



Selection shown
Disabling Internal 12 volts

8.0 Interfacing with other Micromint Boards

The BCC52CX is completely bus compatible with Micromint's expansion boards which were originally developed for use with the Z8 controller boards. This section discusses some aspects of interfacing these boards to the BCC52CX by supplying a graphical representation of a 64K address space along with the specific board's address space requirements plus any supplemental notes.

8.1 MB08 & MB04 Mother (Passive backplane) Board

The MB0x is used to bus together the BCC52CX with any of the Micromint BCC BUS compatible boards. The MB04 is used in any application where only require 1 to 4 boards are required. The MB08 has 8 slots and may be expanded with a short cable (BCC42), which ties together two MB08s. Each MB08 can be powered by its own Power Supply. Alternately, each MB08 can be powered separately if necessary.

8.2 The Mother Board Enclosure

The MB04 is designed to mate with a cardguide-cardcage enclosure. The CC04 (6 inch cardcage) can hold one MB04 (CC01 can hold two MB04s and the CC02, four MB04s.)

The MB08 is likewise designed to mate with a cardguide-cardcage enclosure. The CC01 (10 inch cardcage) can accommodate one MB08, while the CC02 (19 inch cardcage) will hold two MB08's.

8.3 Powering the Mother Board

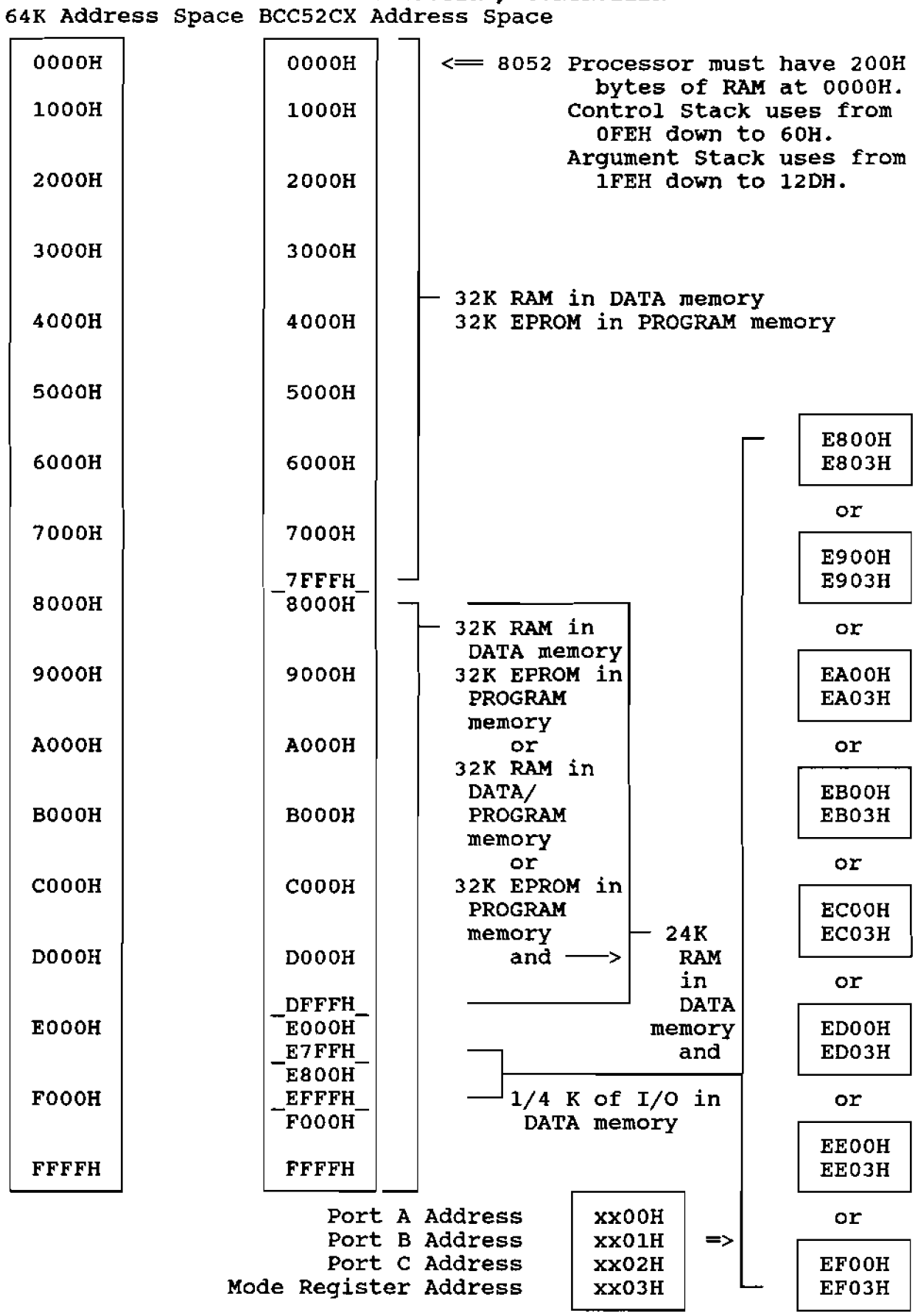
The MB0x may be powered with Micromint's Universal (UPS01) or Standard (UPS11) power supply.

The UPS05 is a card style version of the Standard power supply. It will mount in the cardcage with the rest of the system.

The UPS10 is a metal enclosed switcher.

	+5 V	+12 V	-12 V
UPS11 -	.9A	.3A	.1A (-5 V)
UPS01 -	.3A	50ma	50ma
UPS05 -	1A	.3A	.1A
UPS10 -	5A	.7A	.3A

Figure 8.31 Micromint Power Supply Specifications


64K Address Space
BCC52CX Address Space
0000H
1000H
2000H
3000H
4000H
5000H
6000H
7000H
8000H
9000H
A000H
B000H
C000H
D000H
E000H
F000H
FFFFH
0000H
1000H
2000H
3000H
4000H
5000H
6000H
7000H
7FFFH
8000H
9000H
A000H
B000H
C000H
D000H
DFFFH
E000H
E7FFH
E800H
EFFFH
F000H
FFFFH
<== 8052 Processor must have 200H bytes of RAM at 0000H.
Control Stack uses from 0FEH down to 60H.
Argument Stack uses from 1FEH down to 12DH.
32K RAM in DATA memory
32K EPROM in PROGRAM memory
32K RAM in DATA memory
32K EPROM in PROGRAM memory
or
32K RAM in DATA/PROGRAM memory
or
32K EPROM in PROGRAM memory
and
24K RAM in DATA memory
and
1/4 K of I/O in DATA memory
E800H
E803H
E900H
E903H
EA00H
EA03H
EB00H
EB03H
EC00H
EC03H
ED00H
ED03H
EE00H
EE03H
EF00H
EF03H
Port A Address
Port B Address
Port C Address
Mode Register Address
xx00H
xx01H
xx02H
xx03H
=>