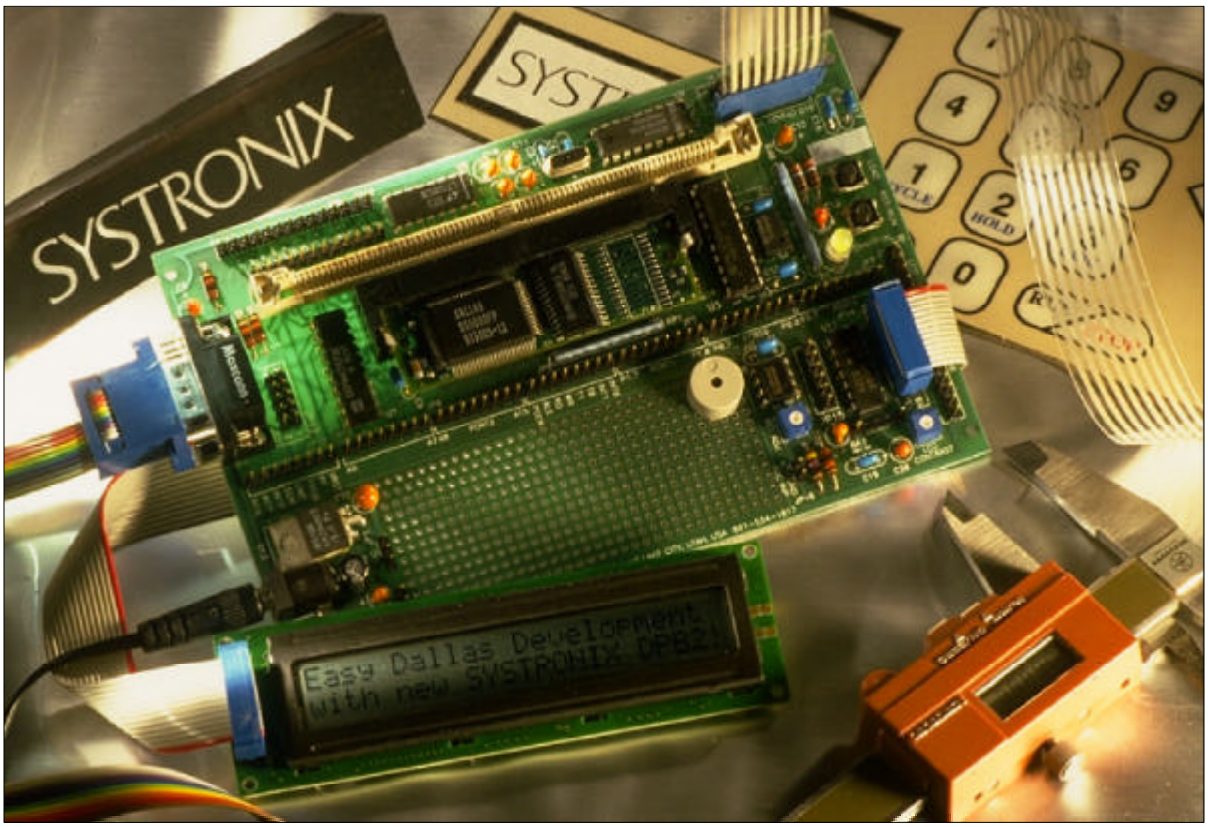


If the idea of powerful and painless 8051-family code development sounds attractive, consider the new 8051 BASIC cross-compiler from Systronix. BCI51 is the first BASIC cross-compiler to offer ease of use, a short learning curve, and the power and flexibility to grow with your requirements. BCI51 and BCI51 Pro are *true real-time compilers*, NOT modified interpreters. This means you get *true real-time interrupts* and a re-entrant function library. The photo shows our DPB2 Universal Dallas Development System - one board supports the whole DIP and SIMM Dallas Secure Microcontroller Line!

BCI51

8051 Family BASIC Compiler

The Systronix commitment: *We pledge to provide you with embedded programming tools offering exceptional ease of use, outstanding performance, and unmatched value.*

8051 FAMILY

Easy 8051 programming!

With BCI51, creating professional quality 8051 programs has never been easier! Your code will be fully compatible with a wide range of 8-bit microcontrollers including CMOS or NMOS 8031, 8032, 8051, and 8052 devices. BASIC saves you time in development and documentation.

How does BCI51 combine ease of use with sophistication? The BCI51 compiler contains an Artificial Intelligence engine which analyzes your program and configuration directives. The AI engine automatically allocates and manages the hardware resources of your target microcontroller, and generates optimal startup and interrupt handling code. You can specify as few or as many configuration options as you wish. The AI engine does the rest for you, and writes all configuration options to a file for your analysis.

BCI51 has the power and flexibility to grow with your requirements. In-line assembly code, sophisticated error handling, and a variety of configuration options provide you with a tool that is simple to use yet sophisticated enough for all your needs. BCI51's syntax is very similar to PC BASICs. Many users prototype in QuickBASIC and then compile with BCI51.

BCI51 Features

BCI51 supports interrupt-driven, ring buffered, full duplex serial I/O, an additional serial output, and a Pulse Width Modulation output, even on a two dollar 8031! Arrays may have up to 65535 elements. Strings may be up to 255 characters long, and each may be a different length. BCI51 includes a string concatenation operator to easily combine string variables and literal text. BCI51 provides signed and unsigned 8 and 16 bit integer math. BCI51 identifies many syntactical errors such as improperly nested control loops at the time of compilation, when they are most easily and inexpensively corrected. BCI51 also provides extensive, flexible run-time error handling - you can even determine which BASIC line caused the error!

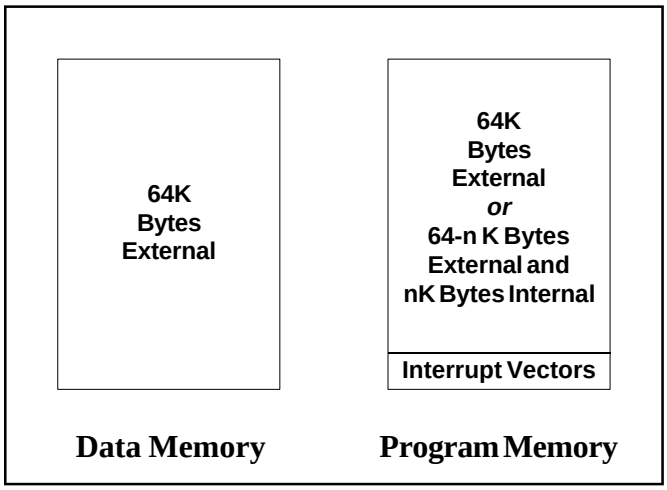
BCI51's runtime libraries and AI engine automatically take care of microcontroller reset, initialization, serial I/O, interrupt handling, startup code, and all interrupt vectors. All numerical routines are fully re-entrant. You can concentrate on your application instead of mundane programming chores.

Custom I/O Drivers!

With BCI51 you can easily create your own custom input and output drivers such as input from a keypad and output to a liquid crystal display. Your program can even redirect I/O during runtime. For example, if your program encountered an error, output could be directed to the serial port instead of an LCD. Examples of this are included in the printed reference manual and on the program disk.

Flexible Configuration Features:

- Specify text files to include in your source code (just like in C!).
- Define the address of your program, data, and stack.
- Specify baud rates for your console and printer I/O.
- Specify target microcontroller options.
- Control error messages and line number tracing.
- Optionally initialize all variables to 0 and strings to "" at startup.
- Configure error handlers and Control C.



8051 Family Memory

Exceptional Support:

Extensive documentation, phone hotline, 24 hour FAX, and a 24-hour BBS provide you with outstanding support. All Systronix products have a 30-day money-back guarantee.

From the simple to the sophisticated, BCI51 gives you exceptional compiler value.

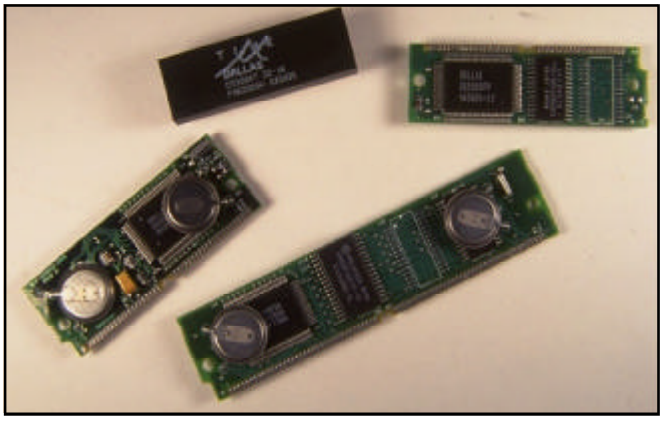
Devices Supported:

8031, 80C31, 8032, 80C32, 8051, 80C51, 8052, 80C52, 8751, 87C51, 8752, 87C52. with external RAM. BCI51 PRO adds support for Dallas DS5000/1/2, DS2250/1/2 and DS5000/1/2FP, including 'T' versions. Supports DS80C320/520/530 families in 8051-compatability mode

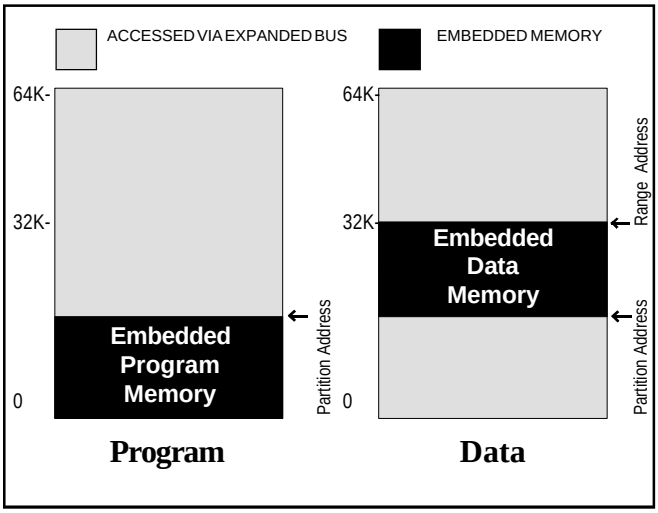
Requirements: MS-DOS computer with at least 512K bytes of available RAM and MS-DOS 3.0 or later. Also runs great in a Windows 3.X or Windows 95 DOS box.

"Great software. Great package, great support. Keep up the good work!"
Derek Howard, Tech Electronics

DS5000 FAMILY



The DS5000 & DS2250/1/2 Secure Microcontrollers



DS5000 Memory Map

Board on a Chip!

- 8051 - compatible microcontroller core.
- Up to 128K Bytes nonvolatile memory.
- 10 year lithium backup battery.
- Realtime Clock and Calender.
- Powerfail Warning, Watchdog Timer, and Automatic Power Down.
- Internal memory leaves I/O ports free for your use.
- Replaces several discrete chips.
- Just add a crystal and a power supply!

The DS5000 is a high performance 8-bit CMOS microcontroller that contains the equivalent of a single board computer all in one compact package with many unique benefits. BCI51 is the only compiler with special support for the DS5000 family, including the 2250 series.

The DS5000 contains an embedded lithium cell that preserves the contents of the embedded RAM, internal registers, and powers the real-time clock/calender for ten years in the absence of any external power source. It has the ability to resume execution when power is re-applied as if the power failure had not occurred at all. The DS5000 has a Power Fail Warning Interrupt, Automatic Power Down mode, a Watchdog timer, and flexible Power On Restart options. These features ensure crashproof operation and unprecedented software adaptability. BCI51 with the DS5000 extensions adds easy to use BASIC language support for all these unique hardware functions.

DS5000 I/O includes a full duplex serial port capable of asynchronous or synchronous operation, 32 parallel lines, and two external interrupts. BCI51 adds a unidirectional asynchronous serial output with programmable parity, and a Pulse Width Modulated output. You can easily use the tools provided by BCI51 to create your own application-specific serial or parallel I/O drivers.

Security Locks and Encryption

In addition, the DS5000 also provides security lock logic and software encryption, preventing unauthorized access to your Program/Data stored in the embedded memory. If anyone attempts to steal your code, the DS5000 simply erases itself.

Quick, Low Cost Development

No EPROM Programmer or Emulator needed!

The DS5000 features a Serial Loader program contained within a special on-chip ROM area. The DS5000 can be programmed through its serial port at up to 19200 baud. You can reprogram or update your software remotely over a serial communications line or a standard telephone line using a modem.

DS5000TK

The DS5000TK development kit enables immediate evaluation of any DS5000 series device. The kit includes program loading and terminal emulation software, a 16 MHz DS5000T-32K processor, user's guide, and In-System Loader Pod. The In-System Loader Pod enables you to load application software into a DS5000 device while the pod is installed in your target system. With the kit, you can quickly reprogram the DS5000 without detailed knowledge of the chip's Serial Load Mode. The DS5000TK also provides an RS232 port which can assist you during debugging, even if your target has no serial I/O. This kit, with the BCI51 DS5000 support, delivers a complete, easy-to-use development and programming system at an unbeatable price.

The DS5000 Family

- The **DS5000T** features a Real Time Clock and Calender that keeps time independent of any external power. 8 or 32K embedded RAM.
- The **DS5000FP, DS5001FP, and DS5002FP** are 80 pin flat packs and require an external battery and RAM. These products are intended for very high volume applications.
- The **DS2250(T), DS2251(T),** and **DS2252(T)** are essentially a DS5000 in a SipStick package with up to 128k bytes of RAM on board.

Requirements: MS-DOS Computer with at least 512K bytes of memory, MS-DOS 3.0 or later, and a serial port.

"[BCI51] makes a potent development combination with the DS5000TK."
***EDN*, January 20, 1992**

BASIC-52

BASIC-52 Acceleration!

BCI51 is the only 8051 compiler that helps you to combine a BASIC-52 interpreter program with a compiled program! Call a compiled program from BASIC-52, pass values back and forth, and return to BASIC-52 as many times as you wish. If you have a current BASIC-52 program that you need to speed up, you can easily compile some or all of it. For the ultimate in speed, incorporate in-line assembly code. No other programming tool gives you such seamless integration of interpreted, compiled, and assembly language code!

BCI51 is the *only* 8051 BASIC compiler written specifically for real-time embedded control; it is not just a warmed-over derivative of the 8052AH-BASIC interpreter. In general, BCI51 supports the syntax of BASIC-52 while freeing you from many of its limits. BCI51 offers enhanced performance in areas such as: interrupt response, error handling, serial I/O, data structures, string handling, and many more. These improvements are clearly documented in the extensive BCI51 manual.

Interpreters vs. Compilers

The BASIC-52 interpreter is easy-to-use and compact, but has its restrictions: leisurely execution and I/O speed, no true interrupt handling (BASIC-52 "interrupts" are actually handled in a polled fashion), and limited error recovery. These are critical aspects of real-time control applications. You could spend one or two thousand dollars to struggle with an 8051 C Compiler, but why? BCI51 delivers the performance of a compiler with the familiarity of BASIC.

Interpreters

The main advantages of an interpreter are an interactive development environment and compact program size. When you type RUN, each tokenized command in your interpreted program is read from memory, and referenced in a token table. The token is subsequently converted into a series of nested CALLs to assembly language routines which perform the appropriate functions. Your program is essentially 'expanded' to assembly code one command at a time. For this reason, interpreted programs are very compact, but execute slowly. Because the interpreter analyzes your source program one token at a time, it has no overall view of your code. Many errors (such as GOTO a non-existent line number) are not detectable until you actually run your program.

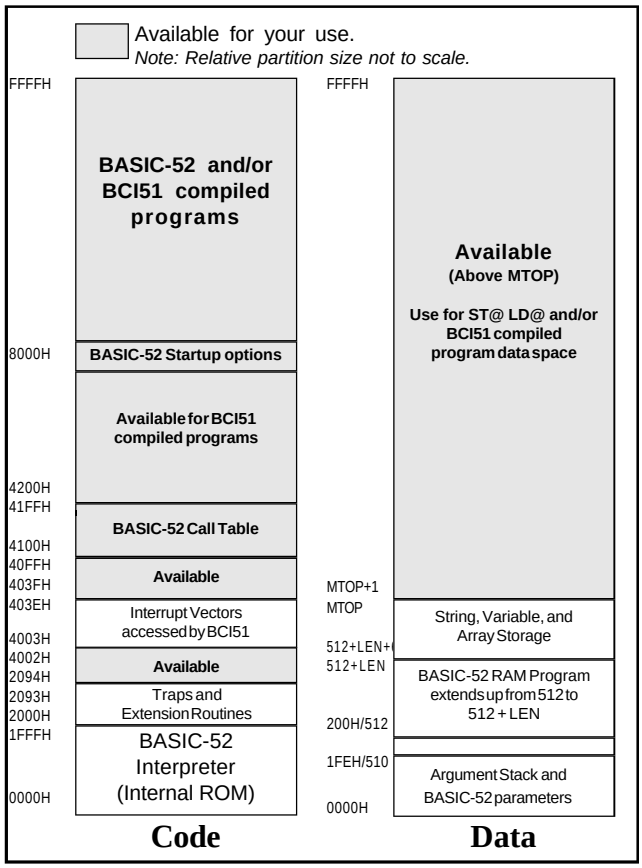
Compilers

The BCI51 compiler converts your BASIC source to assembly language one time only, (at the time of compilation). The 8051 processor executes compiled code much more quickly than it interprets a tokenized program. Because the compiler analyzes your program in its entirety, many errors can be detected and fixed prior to ever running your code. This global view also gives a compiler the ability to optimize. Compilers use a mixture of 'in-line' code and calls to 'run-time libraries' . Libraries include often used routines such as math operations. Compiled programs execute much more quickly than interpreted programs but are somewhat larger.

Inside BCI51

The BCI51 compiler is a multi-pass optimizing compiler. The compiler emits an assembly code source file which can be viewed and edited prior to assembly. The assembly code source is then assembled to 8051 executable machine code by the included Systronix assembler. Other supported assemblers include Intel, Avocet, or compatibles. The Systronix A51 assembler may also be used by itself!

The BCI51 run-time function library is written in hand-optimized assembly code; it is not simply a modification of the BASIC-52 interpreter ROM. Systronix is continually improving the compiler and libraries. Unlimited telephone hotline support, 24 hour FAX and BBS lines, free maintenance releases, and reduced-cost upgrades are available to registered users.



BASIC-52 Memory Map

Requirements: Access to interrupt vectors at 4003H-402FH and code/data space not used by your BASIC-52 program.

"[With BCI51] I went from a dead start to a shipped prototype in 18 days, including a custom serial I/O driver. On average, it's running 153 times faster than the BASIC-52 interpreter!"

Mark Mueller, Mueller Broadcast Design

BASIC, C, & Assembly (a technical note...)

Programming languages are like religion or politics - the question of "right" or "best" is a matter of interpretation, opinion, and personal preference. When selecting a tool for any job, it is advisable to ask: what is the best choice in my current situation? Perhaps the "best" language could be defined as that which gets the job done for you, with the greatest ease and utility while consuming the minimum of time and effort. The BCI51 compiler is a BASIC compiler, written in C. The BCI51 libraries are written in hand-optimized 8051 assembly language. This gives us a unique perspective - familiarity with C, BASIC and assembly code.

Assembly code is like a Formula-One racing car - the ultimate in performance, but requiring constant attention by specialized mechanics. A Formula-One requires a high degree of skill to use and is dangerous in the hands of an inexperienced driver. C is similar to a Ferrari - finicky and expensive, but easier to drive than a Formula-One. A Ferrari requires considerable skill to reach its maximum performance. BASIC? - a Volvo with airbags. Maximum safety and reliability, with reasonable performance - and *anyone* can drive it. Consider the rough-and-tumble industrial embedded programming environment to be like an icy road with treacherous turns, on which you want to make a safe trip in the shortest time possible. Which would you choose?

Assembly language in the hands of a skilled and experienced programmer offers the ultimate in flexibility and control. Unfortunately, you will either need to purchase libraries even for simple functions such as math operations and serial I/O or you will need to write them all yourself! Assembly code is notoriously time-consuming to write and difficult to modify and maintain. Rather like building a brick house one grain of sand at a time. Assembly code is cryptic and requires massive commenting to be understandable.

C is a powerful but difficult-to-master language which originated on large mainframe computers. C was deliberately written by serious computer programmers for other serious computer programmers. Programming in C is like building a brick house one brick at a time. It is more standardized than any other language, and offers the promise of portability across different execution platforms or targets. This can be an important consideration in workstations and personal computers. Among full-time computer scientists, and in academia, C is often the language of choice.

More people know BASIC than any other language. BASIC was specifically written for people who are not full-time programmers, and as such was designed to be easy to use. It is possible to learn the rudiments of BASIC in just a few hours. BASIC is relatively close to English, tends to be self documenting even without comments, and is highly intuitive. Programming in BASIC is like building a house with prefabricated modular panels. For all these reasons, BASIC is the most frequently taught language in public schools. Some version of BASIC is shipped with most personal computers. BASIC dialects are widely used to program high-end electronic production and test equipment.

There is a major difference in how C, assembly and BASIC handle errors. The C language has very little error checking or data validation. For example, allocate 10 bytes to a character array (a text string). A C compiler will be perfectly happy to let you write 1000 bytes to that string! This clobbers 990 bytes of whatever happens to follow that string in memory. Assembly language is even worse - it's completely up to you to perform even the most low-level data typing manually. In marked contrast, BCI51 and most BASICs feature extensive built-in data checking and validation which help you develop your code with a minimum of errors.

How much time, money and patience do you have? Would you rather focus your attention on chasing obscure bugs in your code or concentrating on your application? All other things being equal, you can write and debug an embedded application in BASIC faster than any other language. Many people call us to express their amazement after developing significant amounts of working code the same day they receive BCI51.

BCI51 and BCI51-Pro Benefits and Features

Professional Embedded BASIC Develop professional quality 8051 code with BASIC! Write applications from simple to sophisticated. You don't have to be a "firmware guru" or spend weeks learning complicated, expensive tools. Runs on any MS-DOS PC with 512K RAM and a hard disk. Outputs assembly source code and Intel Hex files. AI engine helps with otherwise laborious setup and configuration tasks. True real-time compiled (NOT interpreted) code.

Pick Your Target Compatible with most any 8031/32, 8051/52 or BASIC-52 system (NMOS or CMOS). Access a full 64K of code and 64K of data. Virtually no hard-coded limits. Bi-directional serial I/O, serial printer output, Pulse Width Modulator, all supported even on a two dollar 8031! BCI51-Pro supports Dallas Secure Microcontrollers.

Super I/O Interrupt-driven bidirectional serial port with output and type-ahead ring buffers. Serial printer output with programmable parity, data bits and stop bits. Built-in library support for your own custom polled or interrupt-driven I/O drivers.

Easy Assembly Code Your BASIC program may contain unlimited instances of in-line assembly code. Call the BCI51 run-time libraries from your in-line assembly code (complete, commented library source code is available in the Assembly Language Programmer's Toolkit). BCI51 generates an assembly source code listing. And you can use the included 8051 assembler separately.

Powerful Error Handling Sophisticated error checking catches most program errors at compile time. Flexible run-time error handling options for robust embedded control systems. Every run-time error can be trapped. You even get the BASIC line number in which the error occurred!

Flexible Data Four integer data types (signed and unsigned 8 or 16 bit), arrays of any length (up to 64K), any number of strings of different lengths, and a string concatenation operator! FAST integer and character math (floating point upgrade coming).

Works with or without BASIC-52 BCI51 is the only 8051 compiler that helps you to combine a BASIC-52 interpreter program with a compiled program! Call a compiled program from BASIC-52, pass values back and forth, and return to BASIC-52 as many times as you wish. Choose coordinated or independent run trapping and interrupt handling.

Custom Solutions We can enhance BCI51 with BASIC operators specific to your needs, while you maintain control of the supporting library code. Contact us to discuss your requirements.

Real Support Extensive documentation, phone hotline, FAX, email, and web site. Commitment to ongoing enhancements and a 30-day money-back satisfaction guarantee.

Systronix, Inc.

555 South 300 East #21

Salt Lake City, UT, USA 84111
TEL: +1-801-534-1017
FAX: +1-801-534-1019
WEB: www.systronix.com
EMAIL: info@systronix.com

"Your products have made development a joy!"
James Mason, Programmer

Sample Program

This example calls a compiled program from BASIC-52 and transfers values in both directions. It illustrates in-line assembly code, enhanced ONERR functions, multiple-statement lines, and more.

```
     ; Configuration section
#TARGET 8052 BASIC52                ;this configures our target
#DATA START 1000H                   ;compiled data stored beginning here
#CODE START 4200H                   ;we'll CALL 4200H from BASIC-52
#PRINTER MODE=4800,7,E,2            ;serial printer 4800 baud 7 data, even par, 2 stop
#PRINT COMPLETE OFF                 ;for maximum performance

     ; Declaration section
UNSIGNED INTEGER VAR1, VAR2
STRING STRING1$(33), STR2$(13)      ;any name ending in '$' can be a string

     ; Program body
100 ONERR 500 : CLOCK1              ;set up ONERR handler location
#ASM
     cpl     T0                     ; in line assembly code
#ASM_END
110 STRING1$ = "Some ": STR2$ = "concatenation"
     : STRING1$ = STRING1$ + "string " + STR2$ + "!"
     : ? STRING1$ : ?# STRING1$," ...on serial printer port"
120 POP VAR1, VAR2                  ;get variables from BASIC-52 argument stack
130 ? "From BASIC-52: ", VAR1, " and ", VAR2
140 PUSH MSEC                       ;push value to BASIC-52
190 ? VAR1/0                        ;deliberate error - divide by zero
200 ? "DONE!" : GOTO 999
500 : ? "Error of errvalue ", ERRVALUE, " in line ", ERRLINE
999 END
```

BASIC-52 program:

```
100  PUSH 12345 : PUSH 32451 : CALL 4200H : POP X : PRINT "From BCI51:",X
```

Output:

```
Some string concatenation!
From BASIC-52: 32451 and 12345
ERROR (Line 190): Divide by Zero
Error of errvalue 10 in line 190
From BCI51: 165
```

BCI51-Pro Dallas Secure Microcontroller

Additional Keywords

Configuration Directives

TARGET
INCLUDE
ON WATCHDOG
ON POWERUP
ON WARMRESET
ON RUNTRAP
ON POWERFAIL

Watchdog Instructions

WDOG_ON
WDOG_OFF
WDOG_RST

I/O Instructions

PORT0
PORT1
PORT2
PORT3

Embedded Clock/ Calender Instructions

DS_ECC_RD
DS_ECC_WR
DS_ECC_RUN
DS_ECC_MODE
DS_AMPM
DS_YEAR
DS_MONTH
DS_DATE
DS_DAY
DS_HOUR
DS_MIN
DS_SEC
DS_HSEC

Printed on Recycled Paper

BCI51 Keywords

Configuration Directives	Operators & Instructions				
BASIC52	/	CBY	GOTO	PCON	SGN
CHECK MATH	-	CHR	IDLE	PH0. and PH1.	SPC
CHECK ISTACK	+	CLEAR	IE	PH0.# and PH1.#	SQR
CMOS	+ (string)	CLEARI	IF-THEN-ELSE	PH0.@ or PH1.@	ST@
CODE START	*	CLEARS	INPUT	POP	STRING
CONSOLE	**	CLOCK0	INT	PORT0	T2CON
CONTROL C	<	CLOCK1	IP	PORT1	TAB
DATA START	<=	CR	LD@	PORT2	TCON
DEFAULT	<>	CTOP	LEN	PORT3	TIME
INCLUDE	= (string)	DATA	LET	PRINT or P. or ?	TIMER0
INTERRUPT HANDLERS	= (assignment)	DBOT	LF	PRINT# or P.# or ?#	TIMER1
ERROR MESSAGES	= (relational)	DBY	MSEC	PRINT@ or P.@ or ?.@	TIMER2
INITIALIZE VARIABLES	>	DO-UNTIL	MTOP	PUSH	TMOD
IRAM	>=	DO-WHILE	NOT	PWM	UI0
ISTACK START	ABS	DTOP	NULL	RCAP2	UI1
MODE	AND	END	ON-GOTO	READ	UO0
PRINT COMPLETE	ASC	ERRLINE	ON-GOSUB-RETURN	REM	UO1
PRINTER	BASLINE	ERRVALUE	ONERR	RESTORE	XBY
RUN TRAP	BS	FOR-TO-STEP-NEXT	ONEX1	RETI	XOR
TARGET	CALL	GET	ONTIME	RND	
TIMER1	CBOT	GOSUB-RETURN	OR	SEED	
TIMER2					
TRACE LINE NUMBERS					
XTAL					