TBAS52

A BASIC Interpreter for the TAPR TUC52
Version 2.0.05
December 7, 1997

Table of Contents

Alphabetical Index of Commands, Instructions and Operators

1. INTRODUCTION

This is the documentation for TBAS52, Version V2.0.05, a derivation of Intel's BASIC-52[1] (MCS-51 BASIC V1.1) for the TAPR TUC52 universal controller board.

This manual documents only the enhancements and fixes to Intel's BASIC-52 for the TUC52 board.

2. DEFINITIONS AND CONVENTIONS:

2.1 BBRAM

Battery-backed RAM residing on the TUC52 board for storage of BASIC programs. This memory (as defined for TBAS52 operation) is located at address 8000h on the TUC52 board.

2.2 TAPR

Tucson Amateur Packet Radio, a national ham radio club with a focus on data communications by means of radio.

2.3 TBAS52

TAPR BASIC 52, the name given to this customized version of Intel's BASIC-52 tailored for operation on the TUC52 board.

2.4 TUC52

TAPR Universal Controller 52[4], an 80C51/80C52 universal controller board with up to 64K of EPROM, up to 64K battery-backed RAM (BBRAM), 82C55-based parallel I/O, and an I^2C serial interface bus.

2.5 "POST ASSEMBLY PATCH TABLE"

A fixed location table in TBAS52 code space that contains constants of limits and other values that can be changed when programming TBAS52 into an EPROM without the need for reassembly of the TBAS52 source code.

2.6 NAMING CONVENTIONS/TERMINOLOGY

In the Intel BASIC-52 manual[1] (page 4), a STATEMENT is defined to mean several things. To prevent misunderstandings in the multiple use of the term STATEMENT, the new term INSTRUCTION is used here.

The TBAS52 BASIC naming conventions/terminology in this manual are defined as follows:

2.6.1 COMMAND

Can only be executed at the READY prompt level (i.e. RUN, CONT, LIST, etc.), but not in the body of a BASIC program.

2.6.2 INSTRUCTION

Can be executed (in general) both at the READY prompt level and in a program, but cannot return a value (i.e. it cannot be on the right side of an '=' or PRINT statement).

2.6.3 OPERATOR

Can be executed at both the READY prompt level and in a program, and (in general) it can be on either side of an "=" to set or return a value (i.e. PORT1, XTAL, MTOP, etc.).

2.6.4 STATEMENT

A combination of one or more variables and expressions related to each other by means of a COMMAND, INSTRUCTION, or OPERATOR. A single BASIC line can contain one or more STATEMENTs separated by the ":" character.

"`10 PRINT X : A = B : IF X = Y GOTO 100`" is a BASIC line with three BASIC STATEMENTs in it.

Some BASIC Keywords do not fit the above definitions exactly. The Keywords FREE, GET, LEN, and PI can't be on the left side of "=", but most closely fit the definition of being OPERATORs.

2.6.5 KEYWORD ENHANCEMENT TYPES

In reference to this manual's naming conventions, TBAS52 Keyword enhancements are defined as follows:

COMMANDS:	ERASE
	IDUMP
	LOAD
	PROG
INSTRUCTIONS:	CLOCK
OPERATORS:	BIT
	DBIT
	XBIT
	XBITW
	XBYW

3. MODIFICATION SUMMARY:

1. MTOP has been modified to allow a reserved space for custom data storage area above BASIC program space.
2. Line editing now supports the backspace function (^H) as well as DELete.
3. Due to the fact that the EPROM programming features do not work when executing BASIC-52 from external EPROM and that the TUC52 does not support EPROM programming, all EPROM programming features have been disabled.
4. The PROG command has been redefined to allow storage of BASIC programs in BBRAM instead of EPROM.
5. The ERASE/ERASEALL commands have been added to remove programs from BBRAM.
6. The IDUMP command has been added to allow the code stored in BBRAM to be exported to be programmed into EPROM.
7. The ILOAD command has been added to allow assembly code to be loaded into external RAM (XRAM) for access by BASIC.
8. The XBY operator now supports 8 or 16 bit operations.
9. The CLOCK instruction now defaults to on (CLOCK1) and supports an 8-channel frequency counter (0 to 65534 Hz), a 16-channel pulse counter (0 to 65534 pulses on each channel), and a 16 input channel state change flag register operation.
10. The XBIT and XBITW operators have been added to allow reading and writing of individual bits of bytes and words in external RAM (XRAM).
11. The DBIT operator has been added to allow reading and writing of individual bits in internal RAM (IRAM).
12. The BIT operator has been added to allow reading and writing of individual bits in bit-addressable RAM.
13. Specific CALLs have been added to support access to the I^2C clock/calendar.
14. Specific generic CALLs have been added to support access to I^2C devices.
15. Autobauding for DS80C320 family of processors is supported.
16. ^W^C was enhanced to include display of Version number.

3.1 MODIFICATION NOTES

1. TBAS52 will only operate on 8052 class microcontrollers. It does **not** work on the 8051 class of microcontrollers.

2. TBAS52 has similar minimum hardware requirements as BASIC-52. That is, 768 bytes of RAM in XRAM space starting at address 0000h. It will require a least a 27C256 EPROM for code, which is larger than the 27C64 EPROM required for BASIC-52.

3. TBAS52 retains only the Intel documented internal addresses of BASIC-52. These are Reset (0000h), assembly I/F (0030h), TechBits[3] bug fix I/F (03A4h), and documented locations at 20xxh and 40xxh.

4. BASIC KEYWORD ENHANCEMENTS

The new BASIC keywords added to TBAS52 as enhancements to BASIC-52 operate in a similar manner as the standard BASIC-52 keywords. All new keywords should be treated as reserved for use as commands, instructions, or operators, and should **not** be used as variable names.

The new or enhanced keywords are as follows:

- BIT
- CLOCK
- DBIT
- ERASE
- IDUMP
- ILOAD
- PROG
- XBIT
- XBITW
- XBYW

4.1 BIT

The BIT operator allows examination and modification of the value of an individual bit in bit-addressable RAM.

Syntax:	BIT(*bexpr*) [= {*expr*}]
Function:	Read/write a bit value of bit-addressable RAM.
Mode:	Command, Run
Use:	BIT(7) = 0 ; clear Bit 7 (20.7h)

The BIT operator retrieves or assigns a value to the designated bit-addressable RAM bit. The bit is addressed with the (*bexpr*) expression where {*bexpr*} is the bit-addressable RAM bit. Valid bit address values for the BIT operator are 0 to 127 (00 - 7Fh).

The value returned when reading a bit-addressable bit using the BIT operator is a '0' (logic low) or '1' (logic high). When writing a value to the bit using the BIT operator, the {*expr*} must evaluate to a valid integer value of 0000h through FFFFh (0 - 65535). Any non-zero value will write a '1' to the bit-addressable bit while only a 0 value will write a '0' to the bit-addressable bit.

ERRORS:	"BAD SYNTAX"	missing '(' or ')' in "(*bexpr*)"
	"BAD ARGUMENT"	expression not integer 0 through 65535 or '*bexpr*' is not evaluated to be 0 through 127

EXAMPLE1:
```
100 X = BIT(4)                   ; read value of bit 4 into X
110 BIT(2) = X                   ; write value from X into bit 2
120 BIT(7) = BIT(0)              ; copy value from bit 0 into bit 7
130 IF BIT(3) = 1 GOTO 200       ; goto line 200 if 20.3h set
140 ? BIT(7), BIT(6), BIT(5)     ; print bits 7, 6, 5 values
150 BIT(23h) = BIT(23h) .XOR. 1  ; toggle bit 24.3h
160 A = BIT(C)                   ; read value of bit in C into var A
```

EXAMPLE2:
```
100 BIT(75) = 0                  ; disable frequency counter (29.3h)
110 BIT(78) = 0                  ; disable pulse counter/state change (29.6h)
120 BIT(79) = 0                  ; disable TIME function (29.7h)
:
200 BIT(4Bh) = 1                 ; enable frequency counter (29.3h)
210 BIT(4Eh) = 1                 ; enable pulse counter/state change (29.6h)
220 BIT(4Fh) = 1                 ; enable TIME function (29.7h)
```

Note: Even though many of the 8052 Special Function Registers are bit-addressable (Bits 128-255), the BIT operator does not allow access to these bits. Only bits in internal RAM locations 20h to 2Fh are accessed by the BIT operator.

Note: The read-modify-write operation required to modify any bits by the BIT operator is protected from interrupts during this operation.

4.2 CLOCK

The CLOCK instruction is enhanced to support an 8-channel frequency counter and a 16-channel pulse counter in addition to the TIME function.

Syntax:	CLOCK0
Function:	Halts the incrementing of the TIME variable, halts the frequency counter on T1, and halts the port pulse counter and state change detector operation.
Mode:	Command, Run
Use:	CLOCK0

Syntax:	CLOCK1
Function:	Enables incrementing of the TIME variable, enables the frequency counter on T1, and enables the port pulse counter and state change detector operation (default after reset).
Mode:	Command, Run
Use:	CLOCK1

In addition to the TIME operation of BASIC-52, the CLOCK instruction now controls the Timer 0 based frequency counter, pulse counter, and state change detector.

The CLOCK1 instruction enables operation of the 5 millisecond Timer 0 to increment the TIME variable, control Timer 1 as an 8-channel 16-bit counter with the channel selected by Port 1, count positive transitions on 16 bits of I/O, and detect state changes on another 16 bits of I/O.

The CLOCK0 instruction disables operation of the 5 millisecond Timer 0, which stops the incrementing of the TIME variable, stops the frequency counter, and stops the I/O pulse counter and state change detector.

In addition, the four elements of the Timer 0 operation can be individually disabled using bits in internal memory to remove the associated BASIC performance overhead of any unused functions. The TIME function is enabled/disabled (1/0) by bit 29.7h (Bit 79). The frequency counter operation is enabled/disabled (1/0) by bit 29.3h (Bit 75) and the pulse counter and state change detector operations are enabled/disabled (1/0) by bit 29.6h (Bit 78) (Note that the pulse counter and state change detector **cannot** be individually controlled). These bits can be set or cleared before or after the CLOCK1 instruction, but a CLOCK1 instruction must be active for any of these functions to operate.

As a change from BASIC-52; after reset, TBAS52 automatically executes the equivalent of a CLOCK1 instruction and bits 29.3h, 29.6h, and 29.7h are set to defaultly enable the TIME, frequency counting, and pulse counting and state change operations, opposite of what occurs in BASIC-52. Therefore, if a BASIC-52 program is used under TBAS52 that does not use any of the CLOCK functions, a CLOCK0 instruction can be added to the start of the program to remove the CLOCK associated performance overhead.

More details are provided later in the "Enhancement Details" section (page 31).

4.3 DBIT

The DBIT operator allows examination and modification of the value of an individual bit of bytes in internal RAM.

Syntax:	DBIT(*aexpr*,*bexpr*) [= {*expr*}]
Function:	Read/write a bit value in internal RAM.
Mode:	Command, Run
Use:	DBIT(22h,7) = 0 ; clear IRAM bit 22.7h

The DBIT operator retrieves or assigns a value to the designated IRAM location's bit. The bit is addressed with the (*aexpr*,*bexpr*) expression where {*aexpr*} is the internal RAM address and {*bexpr*} is the bit (3,4 is byte 3, bit 4). Valid address/bit values for the DBIT function are:

Any valid BASIC expression that results in:
- 00h,0 through FFh,7

The value returned when reading a bit using the DBIT operator is a '0' (logic low) or '1' (logic high). When writing a value to the bit using the DBIT operator, the {*expr*} must evaluate to a valid integer value of 0000h through FFFFh (0 - 65535). Any non-zero value will write a '1' to the bit while only a 0 value will write a '0' to the bit.

ERRORS:	"BAD SYNTAX"	- missing '(' or ')' or ',' in "(*aexpr*,*bexpr*)"
	"BAD ARGUMENT"	- expression not integer 0 through 65535 or
		- '*aexpr*' is not evaluated to be 00 - FFh or
		- '*bexpr*' is not evaluated to be 0 through 7

EXAMPLE1:
```
100 X = DBIT(2,4)                    ; read value of 02.4h into X
110 DBIT(3,2) = X                    ; write value from X into 03.2h
120 DBIT(2,7) = DBIT(0,0)            ; copy value from 00.0h into 02.7h
130 IF DBIT(0,3) = 1 GOTO 200        ; goto line 200 if 00.3h set
140 ? DBIT(1,7),DBIT(1,6),DBIT(1,5)  ; print 01.7, 01.6, 01.5 values
150 DBIT(1,3) = DBIT(1,3) .XOR. 1    ; toggle bit 01.3
160 A = DBIT(B,C)                    ; read bit value of IRAM byte in B
                                     ; with bit number in C into var A
```

EXAMPLE2:
```
100 DBIT(41,3) = 0        ; disable frequency counter (29.3h)
110 DBIT(41,6) = 0        ; disable pulse counter/state change (29.6h)
120 DBIT(41,7) = 0        ; disable TIME function (29.7h)
:
200 DBIT(29h,3) = 1       ; enable frequency counter (29.3h)
210 DBIT(29h,6) = 1       ; enable pulse counter/state change (29.6h)
220 DBIT(29h,7) = 1       ; enable TIME function (29.7h)
```

Note: The read-modify-write operation required to modify any bits by the DBIT operator is protected from interrupts during this operation.

4.4 ERASE BB-RAM

The ERASE, ERASEn, and ERASEALL commands are used to clear battery-backed RAM.

4.4.1 ERASE

The ERASE command is used to clear all BASIC programs in BBRAM from 8000h to the end of the last stored BASIC program. This is a "smart" erase in that only PROG-stored BASIC programs are cleared and any assembly programs (or data) above the last BASIC program in BBRAM will not be cleared. The clearing operation sets all cleared bytes to FFh to emulate an erased EPROM.

Syntax:	ERASE
Function:	Clear all stored BASIC programs in BBRAM.
Mode:	Command
Use:	ERASE

4.4.2 ERASEn

The ERASEn command is used to clear all BASIC programs in BBRAM from the beginning of program 'n' (where 'n' is the number displayed during the PROG operation) to the end of the last stored BASIC program. This is a "smart" erase in that only PROG-stored BASIC programs are cleared and any stored BASIC programs before program 'n' and any assembly programs (or data) above the last BASIC program in BBRAM will not be cleared. This allows for some simple "file" operations in which a program can be copied to RAM for editing and update, erased from BBRAM and then have the updated program replaced in BBRAM. The clearing operation sets all cleared bytes to FFh to emulate an erased EPROM.

Syntax:	ERASEn
Function:	Clear all stored BASIC programs from 'n' on in BBRAM.
Mode:	Command
Use:	ERASE4

Note: ERASE1 is equivalent to ERASE

ERRORS: "PROM MODE" - 'n' is not a valid stored program in BBRAM

4.4.3 ERASEALL

The ERASEALL command is used to clear all of BBRAM from 8000h to the BBRAM_LIMIT (currently 0CFFFh). This allows all BASIC programs, assembly programs, and data stored in the BBRAM to be cleared. The BBRAM_LIMIT is stored in the "post assembly patch table" for customization if need be.

Syntax:	ERASEALL
Function:	Clear all of the BBRAM.
Mode:	Command
Use:	ERASE

Note: After execution of any of the ERASE commands, if the interpreter is in the ROM mode, it will be returned to the RAM mode. This prevents the errors that would occur if ERASE was used to clear the currently selected ROM program and the RUN command was then executed.

4.5 IDUMP

The IDUMP command is used to hex dump the contents of the BBRAM.

Syntax:	IDUMP {*addr*}
Function:	Output a hex dump of the BBRAM
Mode:	Command
Use:	IDUMP or IDUMP 9000h

The IDUMP command is used to output to the console port an Intel hex dump of program/data stored in the BBRAM. The stand-alone IDUMP command (without the {*addr*}) outputs the BBRAM bytes from 8000h to the end of the last stored BASIC program in BBRAM. If the IDUMP {*addr*} form of the command is used, the BBRAM bytes from 8000h to the {*addr*} value or BBRAM_LIMIT (whichever is lower in value) is output to the console port.

The IDUMP command supports the standard XOFF/XON (^S/^Q) software flow control in case the recipient of the data cannot keep up with the data rate. The IDUMP command can also be aborted with a control-C (^C).

The IDUMP output is in the form of standard 8-bit Intel hex records with each record (line) containing 16 data bytes. The address field of each data record is offset by -8000h so that the data can be programmed into a standard 27C256 and installed on the TUC52 board in place of the BBRAM. The end of the IDUMP data is a standard end-of-file (EOF) record - ":00000001FF".

ERRORS: "BAD ARGUMENT"	- {*addr*} not integer 0 through 65535
	- {*addr*} value less than 8000h

4.6 ILOAD

The ILOAD command is used to load hex data into memory.

Syntax:	ILOAD {*addr*}
Function:	Input hex data into memory
Mode:	Command
Use:	ILOAD or ILOAD 9000h

The ILOAD command is used to input Intel hex formatted data into XRAM memory from the console serial port. The stand-alone ILOAD command (without the {*addr*}) inputs the data directly to the addresses specified in the Intel hex record address field. If the ILOAD {*addr*} form of the command is used, the addresses of all data are offset by the {*addr*} value (truncated to 16 bits) before being placed in memory.

The ILOAD command inputs data until it receives an end-of-file (EOF) record, a corrupted record (checksum fails), or is aborted by control-C (^C). If no start address record is received (and no starting address is included in the EOF record), when ILOAD completes, it returns to the interpreter **READY >** prompt. If a start address record is received with a non-zero starting address, after receiving an EOF record, the processor will start executing the loaded code at the indicated start address (or the start address offset by the {*addr*} value if that form of ILOAD was used).

The ILOAD command supports the following standard (and non-standard) Intel hex records:

```
:xxaaaa00dd....ddcc   - standard Data record
:0000000000           - zero length Data record == EOF record
:00aaaa00cc           - zero length Data record == EOF/start record
:00000001FF           - standard EOF record
:00aaaa01cc           - EOF record with starting addr "aaaah"
:02000002aaaacc       - Start record with starting addr "aaaah"
```

If the hex data contains both a start record and an EOF record with a non-zero starting address, the starting address included in the EOF record overrides the starting address in the start record.

ERRORS:	"BAD ARGUMENT"	- {*addr*} not integer 0 through 65535
	"CHECKSUM"	- calculated checksum failed on record

Note1: For maximum loading speed, all interrupts are disabled during the ILOAD process.

Note2: All characters received before the starting ':' of a record and after the checksum byte in a record are ignored.

Note3: No verification is performed on the stored data. That is, data can be attempted to be stored in ROM, I/O space, or anywhere else in the XRAM data space without limitations.

4.7 BB-RAM PROGRAMMING COMMANDS

The PROG commands have been changed to store BASIC programs in non-volatile BBRAM. Several different startup options are available.

The programming commands assume there is a RAM storage device starting at location 8000h in external data memory (XRAM). Startup options are stored beginning at location 8000h and BASIC programs beginning at location 8010h. For simplicity, this space is referred to as BBRAM in this manual. As many programs can be stored in BBRAM as there is room for up to the upper BBRAM_LIMIT (currently 0CFFFh). Programs are stored sequentially and any program can be later executed (RROM), selected (ROM), retrieved (XFER), modified, and stored again. Also, these programs can be cleared (ERASE).

For additional related information, refer to the Intel or Systronix BASIC-52 manuals.

The following commands store BASIC programs and startup options in BBRAM:

1. PROG
2. PROG1
3. PROG2
4. PROG3
5. PROG4
6. PROG5
7. PROG6

Syntax:	**PROG**
Function:	Saves the currently selected program in BBRAM. Displays the BBRAM file number used.
Mode:	Command
Use:	PROG

Syntax:	**PROG1**
Function:	Saves the current console serial port (not the printer port) baud rate. TBAS52 reads this value on power up or reset, sets the console serial port baud rate from this value, displays the "sign-on" message on the console port, and enters the Command mode without requiring receipt of a space character.
Mode:	Command
Use:	PROG1

Syntax:	**PROG2**
Function:	Saves the current console serial port (not the printer port) baud rate. TBAS52 reads this value on power up or reset, sets the console serial port baud rate from this value, enters the Run mode and begins executing the first stored BBRAM program. No "sign-on" message is displayed on the console port. This allows a program to be run after reset without connecting the TUC52 board to a terminal. This is equivalent to the ROM1, RUN command sequence.
Mode:	Command
Use:	PROG2

Syntax:	**PROG3**
Function:	Saves the current console serial port (not the printer port) baud rate and the system control value MTOP. On power up or reset, TBAS52 reads these values to set the console serial port baud rate and clears the external data memory (XRAM) to the stored MTOP address, then displays the "sign-on" message on the console port and enters the Command mode without requiring receipt of a space character.
Mode:	Command
Use:	PROG3

Syntax:	**PROG4**
Function:	Saves the current console serial port (not the printer port) baud rate and the system control value MTOP. On power up or reset, TBAS52 reads these values to set the console serial port baud rate and clears the external data memory (XRAM) to the stored MTOP address, enters the Run mode and begins executing the first stored BBRAM program. No "sign-on" message is displayed on the console port. This allows a program to be run after reset without connecting the TUC52 board to a terminal. This is equivalent to the ROM1, RUN command sequence.
Mode:	Command
Use:	PROG4

Syntax:	**PROG5**
Function:	Saves the current console serial port (not the printer port) baud rate and the system control value MTOP. On power up or reset, TBAS52 reads the serial port value to set the console serial port baud rate. If external data memory location 005Fh contains the value 0A5h, TBAS52 will **not** clear the external data memory. TBAS52 then displays the "sign-on" message on the console port and enters the Command mode without requiring receipt of a space character. This allows a program or data in external data space to be saved during a reset. If this saved program in external data space is to be executed immediately after reset, a 34h should be placed in external memory location 005Eh. This is called the "Run Trap Option". If no RAM program is present when in the Run Trap Mode, TBAS52 will continually display the "READY" message on the console port.
Mode:	Command
Use:	PROG5

Syntax:	**PROG6**
Function:	Similar in operation to PROG5, PROG6 saves the current console serial port (not the printer port) baud rate and the system control value MTOP. On power up or reset, TBAS52 reads the serial port value to set the console serial port baud rate. If external data memory location 005Fh contains the value 0A5h, TBAS52 will **not** clear the external data memory. TBAS52 then CALLs external program memory location 4039h, which **must** contain the start of a custom assembly language RESET routine. Upon return from this routine, TBAS52 examines the CARRY and ACCUMULATOR bits to determine one of three additional options to be used.
Option 1:	If CARRY = 0, TBAS52 enters the auto-baud routine and waits for receipt of a space character on the console serial port.
Option 2:	If CARRY = 1 and ACCUMULATOR Bit 0 = 0, the BBRAM stored rate is used to set the console serial baud rate and the "sign-on" message is displayed on the console port.

Option 3:	If CARRY = 1 and ACCUMULATOR Bit 0 = 1, the BBRAM stored rate is used to set the console serial baud rate and TBAS52 will execute the first stored program in BBRAM after returning from the custom RESET routine.
Mode:	Command
Use:	PROG6

4.7.1 PROG Errors

4.7.1.1 "PROGRAMMING"

This error indicates that a verify of an attempt to store data in BBRAM failed. This error should not occur unless the BBRAM is not installed or there has been a failure of the BBRAM. Note that a partial BASIC program may have been stored before this failure occurred. If this has happened and it is attempted to be run, TBAS52 may hang, crash, or produce unexpected results. This partial program should be cleared with the ERASEn command and the PROG operation repeated.

4.7.1.2 "NO ROOM"

This error indicates that an attempt was made to save a program in BBRAM that is larger than the available room left in the BBRAM (BBRAM_LIMIT). The ERASE command can be used to clear space in BBRAM and then the PROG operation can be repeated.

4.7.2 PROG Notes

1. Programs stored with the PROG command will overwrite any data/assembly code stored in BBRAM above the last saved program (i.e. no test is made to ensure that the BBRAM needed is in an erased state "FFh" before saving the program).

2. The BASIC-52 assembly language interface to programming EPROMs (OpByte 04h) has been disabled since it easy enough at the assembly level to use "MOVX" instructions to write to BBRAM.

4.8 XBIT & XBITW

The XBIT and XBITW operators allow examination and modification of the value of an individual bit of bytes or words in external RAM.

Syntax:	XBIT(*aexpr*,*bexpr*) [= {*expr*}]	
	XBITW(*aexpr*,*bexpr*) [= {*expr*}]	
Function:	Read/write a bit value in external RAM.	
Mode:	Command, Run	
Use:	XBIT(7F31h,7) = 0	; clear state change reg. channel 7
	XBITW(7F30h,12) = 0	; clear state change reg. channel 12

The XBIT and XBITW operators retrieve or assign a value to the designated XRAM byte or word location's bit. The bit is addressed with the (*aexpr*,*bexpr*) expression where {*aexpr*} is the external RAM address and {*bexpr*} is the bit (3,4 is XRAM byte 3, bit 4). Valid address/bit values for the XBIT function are:

Any valid BASIC expression that results in:
- 0000h,0 through FFFFh,7

Valid address/bit values for the XBITW function are:

Any valid BASIC expression that results in:
- 0000h,0 through FFFFh,15

The value returned when reading a bit using the XBIT or XBITW functions is a '0' (logic low) or '1' (logic high). When writing a value to the bit using the XBIT or XBITW operators, the {expr} must evaluate to a valid integer value of 0000h through FFFFh (0 - 65535). Any non-zero value will write a '1' to the bit while only a 0 value will write a '0' to the bit.

The XBITW operator treats the bits in the byte at address {*aexpr*} as bits 15 to 8 and the bits in the byte at address {*aexpr*+1} as bits 7 to 0.

Use caution when using the XBIT or XBITW operators to process the individual bits of an I/O device mapped into external memory. Many such devices have registers (Note 1) or ports (Note 2) that when read do not return the last value written to them, or just cannot return the actual status of the physical port pins (Note 3).

Note 1. Registers may be read-only and thus return an unexpected value after an attempted read.

Note 2. I/O devices that are double-buffered by a register, where the register holds an intermediate value and the actual port status is returned only after shifting-in the register.

Note 3. I/O devices that only support writing and do not support reading will return a deceptive/false 00h or 0FFh (or other undetermined value).

In such cases, any PRINT XBIT, XBIT read statements, and "IF XBIT ..." will operate improperly.

Also, since XBIT uses a read-modify-write mechanism to change any bits in a byte, port bits not selected by the XBIT write operation may be inadvertently changed. In this scenario, it is recommended that the BASIC program keep a "shadow" port in memory of the I/O port, use the XBIT operator to change any bits in the "shadow" port, and then use the XBY operator to copy the "shadow" port to the I/O port address. The read-modify-write operation required to modify any bits by the XBIT operator is protected from interrupts during this operation.

XBIT Internal Operation:

- Clear bit (read-modify-write statement)
  XBIT(100h,3) = 0
  is equivalent to
  XBY(100h) = XBY(100h) .AND. NOT(2**3)

- Set bit (read-modify-write statement)
  XBIT(100h,3) = 1
  is equivalent to
  XBY(100h) = XBY(100h) .OR. (2**3)

ERRORS:	"BAD SYNTAX"	- missing '(' or ')' or ',' in "(*aexpr*,*bexpr*)"
	"BAD ARGUMENT"	- expression not integer 0 through 65535 or
		- '*aexpr*' is not evaluated to be 0000 - FFFFh or
		- '*bexpr*' is not evaluated to be 0 through 15 or
		- '*bexpr*' is not evaluated to be 0 through 7

EXAMPLE1:
```
100 X = XBIT(2,4)                     ; read value of 0002.4h into X
110 XBIT(3,2) = X                     ; write value from X into 0003.2h
120 XBIT(2,7) = XBIT(0,0)             ; copy value from 0000.0h into 0002.7h
130 IF XBIT(0,3) = 1 GOTO 200         ; goto line 200 if 0000.3h set
140 ? XBIT(1,7),XBIT(1,6),XBIT(1,5)   ; print 01.7, 01.6, 01.5 values
150 XBIT(1,3) = XBIT(1,3) .XOR. 1     ; toggle bit 0001.3
160 A = XBIT(B,C)                     ; read value of XRAM byte in B
                                      ; with bit number in C into var A
```
EXAMPLE2: (uses "shadow" port)
```
100 REM    XBIT Shadow Port Example
110 REM    Assumes Top of Memory (MTOP) is 7FFFh
120 MTOP = 7FEFh                    ; move MTOP for shadow variables
130 HWPORT = 0F900h                 ; use variable to reference hardware
140 SHPORT = MTOP + 1               ; shadow variable for port (above MTOP)
150 XBY(SHPORT) = 0                 ; initialize shadow variable for port
160 XBY(HWPORT) = XBY(SHPORT)       ; initialize hardware from shadow var
:
500 XBIT(SHPORT, 7) = 1             ; set bit 7 in shadow variable
510 XBY(HWPORT) = XBY(SHPORT)       ; update hardware from shadow variable
:
700 XBIT(SHPORT, 7) = 0             ; clear bit 7 in shadow
710 XBIT(SHPORT, 0) = 1             ; set bit 0 in shadow
720 XBY(HWPORT) = XBY(SHPORT)       ; update hardware from shadow variable
```

4.9 XBY & XBYW

The XBY operator has been enhanced so that it can access 16-bit words in external data memory as well as bytes.

Syntax:	XBY({*expr1*}) [= {*expr2*}]
Function:	Read/Write a byte of external memory (XRAM)
Mode:	Command, Run
Use:	XBY(99) = 0A5H or PRINT XBY(8000H)

This original version of the XBY instruction retrieves or assigns a byte value to the external data memory. The {*expr1*} must evaluate to a valid integer address of 0 to 0FFFFh (65535) and {*expr2*} must be a valid byte integer between 0 to 0FFh (255).

ERRORS:	"BAD ARGUMENT"	- {*expr1*} not integer 0 through 65535
		- {*expr2*} not integer 0 through 255

Use XBYW to access external data memory as 16 bit values.

Syntax:	XBYW({*expr1*}) [= {*expr2*}]
Function:	Read/Write a word of external memory (XRAM)
Mode:	Command, Run
Use:	XBYW(1099) = 0A55AH or PRINT XBY(8001H)

This 16-bit version of the XBY instruction retrieves or assigns a word value to the external data memory. The {*expr1*} must evaluate to a valid integer address of 0 to 0FFFFh (65535), which is the address of the high byte of the word. Likewise, {*expr2*} must be a valid word integer between 0 to 0FFFFh (65535) and is stored in most significant digit (MSD), least significant digit (LSD) format.

ERRORS:	"BAD ARGUMENT"	- {*expr1*} not integer 0 through 65535
		- {*expr2*} not integer 0 through 65535

Note: The read or write access to the word is protected from interrupts during the consecutive byte accesses by the processor.

EXAMPLE:
```
10 REM Display Frequency & Pulse Counter Tables, and State Change Reg.
20  REM Get Table addresses from "post assembly patch table"
40  FREQ_VALS = 200Ah : PULSE_VALS = 2010h : STATE_VALS = 2016h
60  FADDR = CBY(FREQ_VALS) * 256 + CBY(FREQ_VALS + 1)
70  PADDR = CBY(PULSE_VALS) * 256 + CBY(PULSE_VALS + 1)
80  SADDR = CBY(STATE_VALS) * 256 + CBY(STATE_VALS + 1)
100 PRINT : PRINT "Frequency Values:"
110 FOR X = 0 TO 15 STEP 2  : REM Display Frequency table
120   PRINT XBYW(FADDR + X),
130 NEXT
200 PRINT : PRINT "Pulse Values:"
210 FOR X = 0 TO 31 STEP 2 : REM Display Pulse table
220   PRINT XBYW(PADDR + X),
230 NEXT
300 PRINT : PRINT "State Change Register: ",
310 PH1. XBYW(SADDR)
```

4.10 I^2C SUPPORT

This section describes the specific CALLs available to support access to the clock, calendar, and alarm of the PCF8583 device. Also described are specific CALLs that provide generic read and write access of devices on the I^2C bus. The I^2C bus on the TUC52 board are on the processor pins P1.6 (SCL) and P1.7 (SDA).

4.10.1 Read Seconds Since Midnight

The function CALL 2100H is used to obtain the number of seconds since midnight.

Syntax:	CALL 2100H, {*seconds variable*} [, {*result variable*}]
Function:	Read time from clock and return seconds since midnight.
Mode:	Command, Run
Use:	CALL 2100H, SECS, RESULT

The CALL 2100H function reads the current time from the PCF8583 clock/calendar device on the I^2C bus, calculates the number of seconds that have elapsed since midnight, and puts that value into the {*seconds variable*}. An optional {*result variable*} is used to return the results of interaction with the I^2C bus. The following is a list of the possible results:

0 = OK
1 = No ACKnowledge was received from the I^2C device
2 = I^2C Bus Fault (SCL and/or SDA lines stuck low)

ERRORS: "BAD SYNTAX" - missing first ',' or
- missing {*seconds variable*} or
- missing {*result variable*} after 2nd ','

EXAMPLE:
```
10 REM Display Elapsed Seconds Since Midnight
1100 CALL 2100H, SECS, RSLT
1110 IF RSLT <> 0 THEN 10000
1120 PRINT "Seconds =", SECS
:
10000 REM Error Handler
10010 IF RSLT = 1 THEN PRINT "I2C No ACKnowledge Error" : RETURN
10020 IF RSLT = 2 THEN PRINT "I2C Bus Fault Error" : RETURN
10030 PRINT "Unknown Error:", RSLT
10040 RETURN
```

4.10.2 Set Clock Date and Time

The function CALL 2102H is used to set the clock date and time of the PCF8583 clock/calendar.

Syntax:	CALL 2102H, {*XRAMaddr expr*} [, {*result variable*}]
Function:	Set the clock date/time.
Mode:	Command, Run
Use:	CALL 2102H, 4000H, RESULT

The CALL 2102H function sets the date and time in the clock portion of the PCF8583 clock/calendar device on the I^2C bus. The date and time are taken from an ASCII string in XRAM at the address in {*XRAMaddr expr*}. The format of this ASCII string is: yymmdd/hhnn/ss where "yy" is a two digit year, "mm" is a two digit month, "dd" is a two digit day, "hh" is a two digit hour, "nn" is a two digit minute, and "ss" is a two digit second (i.e. "951230/1535/45" for 3:35:45 PM on Dec. 30, 1995). All digits (use leading zero's) and the '/' must be present or a "BAD ARGUMENT" error will occur.

The 12 hour AM/PM time format and the Weekday feature of the PCF8583 are not supported.

An optional {*result variable*} is used to return the results of interaction with the I^2C bus. The following is a list of the possible results:

0 = OK
1 = No ACKnowledge was received from the I^2C device
2 = I^2C Bus Fault (SCL and/or SDA lines stuck low)

ERRORS:	"BAD SYNTAX"	- missing first ',' or
		- missing {*XRAMaddr expr*} or
		- missing {*result variable*} after 2nd ','
	"BAD ARGUMENT"	- missing or mis-placed '/' character

EXAMPLE:
```
10 REM Set Clock Date and Time
20 STRING 22,20
30 XMEM = 4000H
1300 INPUT "Clock Date/Time Data (yymmdd/hhmm/ss): ",$(0)
1310 FOR I = 1 TO 14
1320   XBY(XMEM+I-1) = ASC($(0),I)
1330 NEXT I
1340 CALL 2102H, XMEM, RSLT
1350 IF RSLT <> 0 THEN 10000
:
10000 REM Error Handler
10010 IF RSLT = 1 THEN PRINT "I2C No ACKnowledge Error" : RETURN
10020 IF RSLT = 2 THEN PRINT "I2C Bus Fault Error" : RETURN
10030 PRINT "Unknown Error:", RSLT
10040 RETURN
```

4.10.3 Read Clock Date and Time

The function CALL 2104H is used to read the clock date and time of the PCF8583 clock/calendar.

Syntax:	CALL 2104H, {*XRAMaddr expr*} [, {*result variable*}]
Function:	Read the clock date/time.
Mode:	Command, Run
Use:	CALL 2104H, 4000H, RESULT

The CALL 2104H function reads the date and time in the clock portion of the PCF8583 clock/calendar device on the I^2C bus. The date and time are stored as an ASCII string in XRAM at the address in {*XRAMaddr expr*}. The format of this ASCII string is: yymmdd/hhnn/ss where "yy" is a two digit year, "mm" is a two digit month, "dd" is a two digit day, "hh" is a two digit hour, "nn" is a two digit minute, and "ss" is a two digit second (i.e. "951230/1535/45" for 3:35:45 PM on Dec. 30, 1995).

The 12 hour AM/PM time format and the Weekday feature of the PCF8583 are not supported.

An optional {*result variable*} is used to return the results of interaction with the I^2C bus. The following is a list of the possible results:

0 = OK
1 = No ACKnowledge was received from the I^2C device
2 = I^2C Bus Fault (SCL and/or SDA lines stuck low)

ERRORS: "BAD SYNTAX" - missing first ',' or
- missing {*XRAMaddr expr*} or
- missing {*result variable*} after 2nd ','

EXAMPLE:
```
10 REM Read Clock Date and Time
20 STRING 22,20
30 XMEM = 4000H
1200 CALL 2104H, XMEM, RSLT
1210 IF RSLT <> 0 THEN 10000
1220 FOR I = 1 TO 14
1230   ASC($(0),I) = XBY(XMEM+I-1)
1240 NEXT I
1250 ASC($(0),I) = 13 : REM Terminate string
1260 PRINT $(0)
:
10000 REM Error Handler
10010 IF RSLT = 1 THEN PRINT "I2C No ACKnowledge Error" : RETURN
10020 IF RSLT = 2 THEN PRINT "I2C Bus Fault Error" : RETURN
10030 PRINT "Unknown Error:", RSLT
10040 RETURN
```

4.10.4 Set Alarm Date and Time

The function CALL 2106H is used to set the alarm date and time of the PCF8583 clock/calendar.

Syntax:	CALL 2106H, {*XRAMaddr expr*} [, {*result variable*}]
Function:	Set the alarm date/time.
Mode:	Command, Run
Use:	CALL 2106H, 4000H, RESULT

The CALL 2106H function sets the date and time in the alarm portion of the PCF8583 clock/calendar device on the I^{2}C bus. The date and time are taken from an ASCII string in XRAM at the address in {*XRAMaddr expr*}. The format of this ASCII string is: yymmdd/hhnn/ss where "yy" is a two digit year, "mm" is a two digit month, "dd" is a two digit day, "hh" is a two digit hour, "nn" is a two digit minute, and "ss" is a two digit second (i.e. "951230/1535/45" for 3:35:45 PM on Dec. 30, 1995). All digits (use leading zero's) and the '/' must be present or a "BAD ARGUMENT" error will occur.

The 12 hour AM/PM time format and the Weekday feature of the PCF8583 alarm are not supported. Note that the PCF8583 does not use the Year for its alarm function, but it is required for consistency. Also note that the alarm is turned off when setting alarm date and time (see "Set Alarm Interrupt State" section on page 26).

An optional {*result variable*} is used to return the results of interaction with the I^{2}C bus. The following is a list of the possible results:

0 = OK
1 = No ACKnowledge was received from the I^{2}C device
2 = I^{2}C Bus Fault (SCL and/or SDA lines stuck low)

ERRORS:	"BAD SYNTAX"	- missing first ',' or
		- missing {*XRAMaddr expr*} or
		- missing {*result variable*} after 2nd ','
	"BAD ARGUMENT"	- missing or mis-placed '/' character

EXAMPLE:
```
10 REM Set Alarm Date and Time
20 STRING 22,20
30 XMEM = 4000H
1500 INPUT "Alarm Date/Time Data (yymmdd/hhmm/ss): ",$(0)
1510 FOR I = 1 TO 14
1520   XBY(XMEM+I-1) = ASC($(0),I)
1530 NEXT I
1540 CALL 2106H, XMEM, RSLT
1550 IF RSLT <> 0 THEN 10000
:
10000 REM Error Handler
10010 IF RSLT = 1 THEN PRINT "I2C No ACKnowledge Error" : RETURN
10020 IF RSLT = 2 THEN PRINT "I2C Bus Fault Error" : RETURN
10030 PRINT "Unknown Error:", RSLT
10040 RETURN
```

4.10.5 Read Alarm Date and Time

The function CALL 2108H is used to read the alarm date and time of the PCF8583 clock/calendar.

Syntax:	CALL 2108H, {*XRAMaddr expr*} [, {*result variable*}]
Function:	Read the alarm date/time.
Mode:	Command, Run
Use:	CALL 2108H, 4000H, RESULT

The CALL 2108H function reads the date and time in the alarm portion of the PCF8583 clock/calendar device on the I^2C bus. The date and time are stored as an ASCII string in XRAM at the address in {*XRAMaddr expr*}. The format of this ASCII string is: yymmdd/hhnn/ss where "yy" is a two digit year, "mm" is a two digit month, "dd" is a two digit day, "hh" is a two digit hour, "nn" is a two digit minute, and "ss" is a two digit second (i.e. "951230/1535/45" for 3:35:45 PM on Dec. 30, 1995).

The 12 hour AM/PM time format and the Weekday feature of the PCF8583 alarm are not supported. Note that the PCF8583 does not use the Year for its alarm function.

An optional {*result variable*} is used to return the results of interaction with the I^2C bus. The following is a list of the possible results:

0 = OK
1 = No ACKnowledge was received from the I^2C device
2 = I^2C Bus Fault (SCL and/or SDA lines stuck low)

ERRORS:	"BAD SYNTAX"	- missing first ',' or
		- missing {*XRAMaddr expr*} or
		- missing {*result variable*} after 2nd ','

EXAMPLE:
```
10 REM Read Alarm Date and Time
20 STRING 22,20
30 XMEM = 4000H
1400 CALL 2108H, XMEM, RSLT
1410 IF RSLT <> 0 THEN 10000
1420 FOR I = 1 TO 14
1430   ASC($(0),I) = XBY(XMEM+I-1)
1440 NEXT I
1450 ASC($(0),I) = 13 : REM Terminate string
1460 PRINT $(0)
:
10000 REM Error Handler
10010 IF RSLT = 1 THEN PRINT "I2C No ACKnowledge Error" : RETURN
10020 IF RSLT = 2 THEN PRINT "I2C Bus Fault Error" : RETURN
10030 PRINT "Unknown Error:", RSLT
10040 RETURN
```

4.10.6 Set Alarm Interrupt State

The function CALL 210AH is used to control the alarm (and interrupt) functions of the PCF8583 clock/calendar.

Syntax:	CALL 210AH, {*alarm expr*} [, {*result variable*}]
Function:	Set the alarm function.
Mode:	Command, Run
Use:	CALL 210AH, 0, RESULT

The CALL 210AH function controls the alarm interrupt functions of the PCF8583 clock/calendar device on the I^{2}C bus. This function allows an alarm to be cleared, set for an alarm interrupt every day at a specific time, or set for an alarm interrupt on a specific date and time. After setting the alarm date and time, this function must be called to enable the alarm as it is disabled by the function that is used to set the alarm date and time. The control value is taken from {*alarm expr*}. If the control value is not a supported value, a "BAD ARGUMENT" error will occur. The following is a list of the four valid alarm control values:

0 = off	- Clear an alarm interrupt or turn off alarm
1 = daily	- Enables a daily alarm
2 = reserved	- Reserved
3 = dated	- Enables an alarm for a specific date/time

Note: Enabling an alarm also clears the alarm interrupt.

An optional {*result variable*} is used to return the results of interaction with the I^{2}C bus. The following is a list of the possible results:

0 = OK
1 = No ACKnowledge was received from the I^{2}C device
2 = I^{2}C Bus Fault (SCL and/or SDA lines stuck low)

ERRORS:	"BAD SYNTAX"	- missing first ',' or
		- missing {*alarm expr*} or
		- missing {*result variable*} after 2nd ','
	"BAD ARGUMENT"	- {*alarm expr*} not 0 to 3

EXAMPLE:

```
10 REM Set Alarm Control
:
1600 ALARM = 0 : REM Alarm off and clear interrupt
1610 GOTO 1810
:
1700 ALARM = 1 : REM Daily alarm
1710 GOTO 1810
:
1800 ALARM = 3 : REM Dated alarm
1810 CALL 210AH, ALARM, RSLT
1820 IF RSLT <> 0 THEN 10000
:
10000 REM Error Handler
10010 IF RSLT = 1 THEN PRINT "I2C No ACKnowledge Error" : RETURN
10020 IF RSLT = 2 THEN PRINT "I2C Bus Fault Error" : RETURN
10030 PRINT "Unknown Error:", RSLT
10040 RETURN
```

4.10.7 Read Days Since First of Year

The function CALL 210CH is used to obtain the number of days since the beginning of the current year.

Syntax:	CALL 210CH, {*days variable*} [, {*result variable*}]
Function:	Read date from clock and return days since Jan. 1.
Mode:	Command, Run
Use:	CALL 210CH, DAYS, RESULT

The CALL 210CH function reads the current date from the PCF8583 clock/calendar device on the I^{2}C bus, calculates the number of days that have elapsed since the first of the year, and puts that value into the {*days variable*}. An optional {*result variable*} is used to return the results of interaction with the I^{2}C bus. The following is a list of the possible results:

0 = OK
1 = No ACKnowledge was received from the I^{2}C device
2 = I^{2}C Bus Fault (SCL and/or SDA lines stuck low)

ERRORS: "BAD SYNTAX" - missing first ',' or
- missing {days variable} or
- missing {result variable} after 2nd ','

EXAMPLE:
```
10 REM Display Elapsed Days Since Jan 1
1100 CALL 210CH, DAYS, RSLT
1110 IF RSLT <> 0 THEN 10000
1120 PRINT "Days =", DAYS
:
10000 REM Error Handler
10010 IF RSLT = 1 THEN PRINT "I2C No ACKnowledge Error" : RETURN
10020 IF RSLT = 2 THEN PRINT "I2C Bus Fault Error" : RETURN
10030 PRINT "Unknown Error:", RSLT
10040 RETURN
```

4.10.8 Read Days Since Reference Date: Jan. 1, 1996

The function CALL 210EH is used to obtain the number of days since the beginning of the reference year 1996.

Syntax:	CALL 210EH, {*days variable*} [, {*result variable*}]
Function:	Read date from clock and return days since Jan. 1, 1996.
Mode:	Command, Run
Use:	CALL 210EH, DAYS, RESULT

The CALL 210EH function reads the current date from the PCF8583 clock/calendar device on the I^{2}C bus, calculates the number of days that have elapsed since Jan. 1, 1996, and puts that value into the {*days variable*}. An optional {*result variable*} is used to return the results of interaction with the I^{2}C bus. The following is a list of the possible results:

0 = OK
1 = No ACKnowledge was received from the I^{2}C device
2 = I^{2}C Bus Fault (SCL and/or SDA lines stuck low)

ERRORS:	"BAD SYNTAX"	- missing first ',' or
		- missing {*days variable*} or
		- missing {*result variable*} after 2nd ','

EXAMPLE:
```
10 REM Display Elapsed Days Since Jan 1, 1996
1100 CALL 210EH, DAYS, RSLT
1110 IF RSLT <> 0 THEN 10000
1120 PRINT "Days =", DAYS
:
10000 REM Error Handler
10010 IF RSLT = 1 THEN PRINT "I2C No ACKnowledge Error" : RETURN
10020 IF RSLT = 2 THEN PRINT "I2C Bus Fault Error" : RETURN
10030 PRINT "Unknown Error:", RSLT
10040 RETURN
```

4.10.9 Copy Data from XRAM to I^{2}C Device

The function CALL 2110H is used to copy data from XRAM data space to an I^{2}C device.

Syntax:	CALL 2110H, {*I^{2}Cdev*},{*cexpr*},{*xexpr*},{*pexpr*} [,{*result variable*}]
Function:	Write data to I^{2}C device.
Mode:	Command, Run
Use:	CALL 2110H, 0A0H, 100, 4000H, 20H, RESULT

The CALL 2110H function copies data from XRAM data space and writes it to an I^{2}C device on the I^{2}C bus. The device written to is determined by the { *I^{2}Cdev*} expression (the TUC52 has a PCF8583 RAM device at 0A0h and a PCF8581/8582 EEPROM device at 0A4h). The number of bytes transferred is determined by the {*cexpr*} (count expression). The source of the bytes copied is in {*xexpr*} (XRAM address expression) and the I^{2}C device destination of the copy operation is determined by the value in {*pexpr*} (PCF address expression).

A maximum of 256 bytes can be copied at a time. More than that and a "BAD ARGUMENT" error will occur. This function is not prevented from copying data to the clock, timer, or alarm space of the PCF8583 address space. This is the responsibility of the BASIC programmer using this function.

An optional {*result variable*} is used to return the results of interaction with the I^{2}C bus. The following is a list of the possible results:

0 = OK
1 = No ACKnowledge was received from the I^{2}C device
2 = I^{2}C Bus Fault (SCL and/or SDA lines stuck low)

ERRORS:	"BAD SYNTAX"	- missing separating commas or
		- missing {*I^{2}Cdev*} or missing {*cexpr*} or
		- missing {*xexpr*} or missing {*pexpr*} or
		- missing {*result variable*} after last ','
	"BAD ARGUMENT"	- {*cexpr*} not 0 to 256
		- {*pexpr*} greater than 255
		- {*cexpr*} + {*pexpr*} greater than 256

EXAMPLE:

```
10 REM Save BASIC Variable in PCF8583 RAM
100 TSTVAR = 123.45 : PMEM = 20H : I2CDEV = 0A0H
120 GOSUB 1300
:
1300 XMEM = 4000H
1310 PUSH TSTVAR
1320 ST@ XMEM+6
1330 CALL 2110H, I2CDEV, 6, XMEM, PMEM, RSLT
1340 IF RSLT <> 0 THEN 10000
1350 RETURN
:
10000 REM Error Handler
10010 IF RSLT = 1 THEN PRINT "I2C No ACKnowledge Error" : RETURN
10020 IF RSLT = 2 THEN PRINT "I2C Bus Fault Error" : RETURN
10030 PRINT "Unknown Error:", RSLT
10040 RETURN
```

4.10.10 Copy Data from I^{2}C Device to XRAM

The function CALL 2112H is used to copy data from an I^{2}C device to XRAM data space.

Syntax:	CALL 2112H, {*I^{2}Cdev*},{*cexpr*},{*pexpr*},{*xexpr*} [,{*result variable*}]
Function:	Read data from I^{2}C device.
Mode:	Command, Run
Use:	CALL 2112H, 0A4H, 100, 20H, 4000H, RESULT

The CALL 2112H function copies data from an I^{2}C device on the I^{2}C bus and writes it to XRAM data space. The device written to is determined by the {*I^{2}Cdev*} expression (the TUC52 has a PCF8583 RAM device at 0A0h and a PCF8581/8582 EEPROM device at 0A4h). The number of bytes transferred is determined by the {*cexpr*} (count expression). The source of the bytes copied is in {*pexpr*} (PCF address expression) and the XRAM destination of the copy operation is determined by the value in {*xexpr*} (XRAM address expression).

A maximum of 256 bytes can be copied at a time. More than that and a "BAD ARGUMENT" error will occur. This function is not prevented from copying data from the clock, timer, or alarm space of the PCF8583 address space. This is the responsibility of the BASIC programmer using this function.

An optional {*result variable*} is used to return the results of interaction with the I^{2}C bus. The following is a list of the possible results:

0 = OK
1 = No ACKnowledge was received from the I^{2}C device
2 = I^{2}C Bus Fault (SCL and/or SDA lines stuck low)

ERRORS:	"BAD SYNTAX"	- missing separating commas or
		- missing {*I^{2}Cdev*} or missing {*cexpr*} or
		- missing {*pexpr*} or missing {*xexpr*} or
		- missing {*result variable*} after last ','
	"BAD ARGUMENT"	- {*cexpr*} not 0 to 256
		- {*pexpr*} greater than 255
		- {*cexpr*} + {*pexpr*} greater than 256

EXAMPLE:

```
10 REM Restore BASIC Variable from PCF8583 RAM
100 PMEM = 20H : I2CDEV = 0A0H
200 GOSUB 1400
210 PRINT TSTVAR
:
1400 XMEM = 4000H
1410 CALL 2112H, I2CDEV, 6, PMEM, XMEM, RSLT
1420 IF RSLT <> 0 THEN 10000
1430 LD@ XMEM+6
1440 POP TSTVAR
1450 RETURN
:
10000 REM Error Handler
10010 IF RSLT = 1 THEN PRINT "I2C No ACKnowledge Error" : RETURN
10020 IF RSLT = 2 THEN PRINT "I2C Bus Fault Error" : RETURN
10030 PRINT "Unknown Error:", RSLT
10040 RETURN
```

5. ENHANCEMENT DETAILS

This section describes some of the technical details and operational considerations of some of the enhancements to TBAS52.

5.1 MTOP LIMITS

When TBAS52 starts operation, its determination of MTOP is different than BASIC-52. BASIC-52 would test and clear all external data memory from 0 to 0E000h. When it detected a memory failure from non-existent RAM or reached the 0E000h limit, it would set MTOP to that value. TBAS52 will also test and clear memory starting at 0, but only up to the limit MTOP_LIMIT (currently 3FFFh), and sets MTOP to the last valid memory location found or MTOP_LIMIT, whichever is lower.

Also, TBAS52 places a restriction on the setting of MTOP using the MTOP instruction (which BASIC-52 didn't do). If the user attempts to set MTOP to a value above MTOP_MAX (currently 7EFFh), a "BAD ARGUMENT" error occurs, preventing MTOP from being set higher than MTOP_MAX.

Note: A knowledgeable user can modify the interpreter location that stores MTOP directly to circumvent the built-in limit. In a similar manner, the MTOP value stored in BBRAM by the PROG3, PROG4, PROG5, and PROG6 commands could also be directly modified to circumvent the above limit. Either of these methods could be used, if necessary, for testing special features or testing modifications of the limits in the "post assembly patch table" before actually programming a new EPROM.

5.2 LINE EDITING

TBAS52 now supports the use of the backspace character (^H) to delete a character entered at the console port. The backspace operates the same as the DEL character in that the cursor is moved back one space, a space character is sent, and the cursor is again moved back one space to erase a character on a terminal screen.

5.3 ^W^C DESCRIPTION, VERSION NUMBER

TBAS52 uses the keystroke sequence ^W^C (control-W followed by a control-C) to display its version number, developers' name, company, and date.

```
"*MCS-51(tm) BASIC V2.0.05* WRITTEN BY JOHN KATAUSKY, INTEL CORP. 1985
 Modified for TUC52 by Daniel L. Karmann 1994-1996"
```

The version number of TBAS52 is available to a BASIC program using the CBY operator starting at location CODE 20B3h. It is in the form of: major_number.minor_number.devl_number (i.e. 2.0.05).

EXAMPLE:

```
100  REM Print TBAS52 Version Number
110  PRINT "Version ",
120  FOR X = 0 to 5
130    PRINT CHR(CBY(20B3h + X)),
140  NEXT X
150  PRINT
```

5.4 TIME OPERATION

The TIME operator in TBAS52 operates the same as in BASIC-52 except that it defaults to **on** and has an additional control bit. The TIME variable can be disabled by bit 29.7h in internal memory in addition to the CLOCK0 instruction. This allows the TIME to be disabled without disabling the frequency counter or pulse counter or state change detector operations, which is what the CLOCK0 instruction would do.

This change in operation was required by the change in the operation of the interrupt service routine to support the frequency counter, pulse counter, and state change detector operation in addition to the TIME function. The Timer 0 five millisecond interrupt is still enabled by the CLOCK1 instruction, but now only updates the TIME variable if the additional bit 29.7h is set in internal memory.

The following outlines the current Timer 0 interrupt routine:

```
If Bit 29.3h set
    Do Frequency Counter operations
If Bit 29.7h set
    Update TIME variable
If Bit 29.6h set
    Do Pulse Counter/State Change operations
```

At Reset, bits 29.3h, 29.6h, and 29.7h are set and an effective CLOCK1 operation is performed to defaultly enable the TIME, frequency counting, pulse counting, and state change detector operations.

5.5 FREQUENCY COUNTER OPERATION

The frequency counter is an addition to the Timer 0 interrupt service routine to count and store the frequency of 8 channels of frequency input. The 8052 Timer 1 is used as a hardware counter to count the frequency on its T1 pin. This operation is controlled by the CLOCK instruction and bit 29.3h in internal memory. Bit 29.3h allows the frequency counter to be stopped without stopping the TIME or pulse counting or state change detector operation.

The channel routed to the 8052 T1 pin is determined by three bits of the 8052 Port 1 controlling an external multiplexer. These three bits are P1.5 (A2), P1.4 (A1), and P1.3 (A0). In operation, the frequency counter code first selects channel 0, clears the Timer 1 counter, and waits 5 milliseconds. Then the Timer 1 counter is enabled for 1 second. At the end of one second, the Timer 1 counter is stopped, its value is read and then stored in memory in a table starting at 7F00h (7F00h is channel 0, 7F02h is channel 1, etc.). The multiplexer is then set to channel 1 and the operation repeats for all 8 channels at which time it repeats for channel 0 again.

In operation, this means that each channel is counted for 1 second approximately every 8 seconds. These frequency values are then available to a BASIC program using the XBYW operator. For maximum flexibility, the address of the frequency counter table is stored in the code memory "post assembly patch table" at location 200Ah (default: 7F00h). Since the XBYW operation is a 16 bit operation protected from interrupts and the frequency counter operation is controlled by interrupts, frequent use of the XBYW operator in a BASIC program will introduce some jitter into the frequency counter (and TIME) results.

In BASIC-52, Timer 1 is used for EPROM programming, PWM output, and serial line printer (LPT) output. These are exclusive uses of Timer 1, but in TBAS52, Timer 1 must be shared with the frequency counter operation and the PWM and LPT functions (EPROM programming is no longer supported). In TBAS52, this operates as follows: when not performing a PWM or LPT operation, Timer 1 is dedicated to the frequency counting operation. When performing a PWM or LPT operation, the frequency counting operation is suspended (effectively a CLOCK0 is performed) and Timer 1 is reprogrammed for the standard BASIC-52 mode of operation. At the end of the PWM or LPT operation, Timer 1 is then reprogrammed for hardware counting operation. The external multiplexer is set to channel 0 and the counting operation (if it was enabled before the PWM or LPT) continues. Since no Timer 0 interrupt occurs during the PWM or LPT operation, no invalid counting data is accumulated in the frequency counter table.

The result of the above scenario is the counting operation is conducted until a PWM or LPT operation is needed, and restarted after the operation is completed. The minor side effect of this operation is that the frequency counting is not continued on the channel that was selected when the PWM or LPT operation occurred, but it restarts at channel 0.

At Reset, unless a PROG5 or PROG6 reset is active with external RAM location 005Ah set to A5h, the frequency counter storage table at 7F00h (to 7FFFh) is cleared. The frequency counter operation (as well as TIME and pulse counting) is then enabled starting with channel 0.

The TBAS52 specifications on the frequency input to the T1 pin of the 8052 is 0 to 65534 Hz. The actual specifications limit the upper frequency on the T1 pin to 1/24th the oscillator frequency of the 8052, but in most designs this frequency would overflow the Timer 1 counter, indicated by 65535 (FFFFh) as the frequency value in the storage table.

5.6 PULSE COUNTER OPERATION

The pulse counter is an addition to the Timer 0 interrupt service routine to count and store the pulses on 16 bits of two XRAM-mapped I/O port inputs.

The 8052 Timer 0 interrupt service routine is used to sample the level of 16 bits on two I/O ports in external memory. This operation is controlled by the CLOCK instruction and bit 29.6h in internal memory. Bit 29.6h allows the pulse counter to be stopped without stopping the TIME or frequency counting operation (it also stops the state change detector).

In operation, the pulse input to an XRAM port pin (not a processor port pin) is sampled once approximately every 20 milliseconds and a count is incremented on a high sample following a previously low sample. In actual operation (to spread out processing), the low and high I/O ports are alternately sampled every 10 milliseconds. The address of the low port is stored in the code memory "post assembly patch table" at location 200Ch (default: F9FDh) and the high port address is stored at location 200Eh (default: FAFDh).

When a positive transition is detected on an XRAM port pin, the count for that pin (channel) is incremented in memory in a table starting at 7F10h (7F10h is channel 0, 7F12h is channel 1, etc.). The 16-bit count value is limited to 65535 (FFFFh), so the count is not allowed to overflow. These pulse count values are then available to a BASIC program using the XBYW operator. For maximum flexibility, the address of the pulse counter table is stored in the code memory "post assembly patch table" at location 2010h (default: 7F10h). The least significant bit of the low I/O port is considered channel 0 with its most significant bit considered channel 7. In a similar manner, the least significant bit of the high port is considered channel 8 with its most significant bit considered channel 15.

In TBAS52, when a PWM output or serial line printer (LPT) output operation are executed, the pulse counting operation is suspended (effectively a CLOCK0 is performed). Since no Timer 0 interrupt occurs during the PWM or LPT operation, multiple transitions on any of the counter port pins may be missed, resulting in possible incorrect count data in the pulse counter table.

At Reset, unless a PROG5 or PROG6 reset is active with external RAM location 005Ah set to A5h, the pulse counter storage table at 7F10h (to 7FFFh) is cleared. The pulse counter operation (as well as TIME and frequency counting) is then enabled.

Note: Due to the larger amount of stack space needed by the pulse counting routines, if the processor is executing a very complex BASIC statement when a Timer 0 interrupt occurs, the pulse counter will not update its counters if it is determined that there is not enough stack space. This will result in a delayed transition detection (about 5 milliseconds), but should not present any problems with most slow mechanical inputs.

5.7 State Change Operation

The state change detector is an addition to the Timer 0 interrupt service routine to detect and store the changes on 16 bits of two XRAM-mapped I/O port inputs.

The 8052 Timer 0 interrupt service routine is used to sample the level of 16 bits on two I/O ports in external memory. This operation is controlled by the CLOCK instruction and bit 29.6h in internal memory. Bit 29.6h allows the state change detector to be stopped without stopping the TIME or frequency counting operation (it also stops the pulse counter).

In operation, the input to an XRAM port pin (not a processor port pin) is sampled once approximately every 20 milliseconds and any change in state from a previous sample is noted. In actual operation (to spread out processing), the low and high I/O ports are alternately sampled every 10 milliseconds. The address of the low port is stored in the code memory "post assembly patch table" at location 2012h (default: F9FDh) and the high port address is stored at location 2014h (default: F9FEh).

When a level change is detected on an XRAM port pin, a flag bit for that pin (channel) is set in memory in a 16-bit register starting at 7F30h (7F30h is channels 15 to 8, 7F31h is channels 7 to 0). These state change register values are then available to a BASIC program using the XBYW operator. For maximum flexibility, the address of the state change register is stored in the code memory "post assembly patch table" at location 2016h (default: 7F30h). The least significant bit of the state change register is considered to be channel 0 with its most significant bit considered to be channel 15.

In TBAS52, when a PWM output or serial line printer (LPT) output operation are executed, the state change detector operation is suspended (effectively a CLOCK0 is performed). Since no Timer 0 interrupt occurs during the PWM or LPT operation, transitions on any of the state change port pins may be missed. If an even number of transitions on any of the state change port pins are missed, the transitions will not be detected and shown in the state change register. If an odd number of transitions on any of the state change port pins are missed, after the PWM or LPT operation is completed, the last transition will be detected and shown in the state change register.

After a BASIC program detects a change in state on any of the 16 channels in the state change register, the program should use the XBYW(), XBIT(), or XBITW() operators to clear the register bits of any channels of interest to be able to detect additional transitions on those channels.

At Reset, unless a PROG5 or PROG6 reset is active with external RAM location 005Ah set to A5h, the state change register at 7F30h (to 7FFFh) is cleared. The state change detector operation (as well as TIME and frequency and pulse counting) is then enabled.

Note: Due to the larger amount of stack space needed by the state change detector routines, if the processor is executing a very complex BASIC statement when a Timer 0 interrupt occurs, the state change detector will not be updated if it is determined that there is not enough stack space. This will result in a delayed transition detection (about 5 milliseconds), but should not present any problems with most slow mechanical inputs.

5.8 PCF8583 Clock/Calendar/RAM Operation

The PCF8583 clock/calendar device (device address 0A0h) on the I^2C bus on pins P1.6 (SCL) and P1.7 (SDA) of the processor is supported in TBAS52 by several CALL 21xxH functions. Most of the features of the PCF8583 are directly supported by these CALLs, but some of the features are not.

In addition to a clock/calendar, the PCF8583 also contains an alarm function, timer function, and 224 bytes of battery-backed RAM. The TBAS52 implementation directly supports nearly all functions but the timer function. Also, the clock/calendar time and alarm functions do not support the 12 hour AM/PM format or the use of the weekday. One of the limitations of the PCF8583 is its storage of the year. Only a four year duration is built in with year 00 considered a leap year. Sacrificing one of the 224 bytes of RAM, the TBAS52 clock functions use location 10H to store the 2 digit BCD representation of the year. This location is updated from the built-in year bits every time the read seconds since midnight function (CALL 2100H), read date/time function (CALL 2104H), read days since first of year function (CALL 210CH), or read days since Jan. 1, 1996 function (CALL 210EH) are called.

The functions that read (CALL 2112H) and write (CALL 2110H) the RAM in the PCF8583 are not restricted from reading or writing the clock, timer, alarm, or control locations. This means that the BASIC programmer is responsible for not accessing locations 00 to 10H (unless it is understood what the effects are).

The alarm control function (CALL 210AH) is designed to always enable an alarm interrupt when any alarm is enabled. The alarm interrupt is always disabled when setting the alarm date and time. When the PCF8583 generates an alarm interrupt, this interrupt is cleared by either turning off alarms or resetting the alarm type (it is not necessary to turn off the alarms when resetting an alarm). If the INT pin of the PCF8583 is not connected to the processor, location 0, bit 1 of the PCF8583 can be read (CALL 2112H) to determine if an alarm occurred (1 = alarm occurred).

For a detailed description of the complete PCF8583 operations, see the Philips[5,6] data sheet for the PCF8583.

5.8.1 Alarm

Use the alarm functions as described below.

5.8.1.1 Setting an Alarm

- Set the alarm date/time (CALL 2106H)
- Enable the alarm type (CALL 210AH)

Note: The "daily" alarm uses only the hour, minutes, and seconds values and the "dated" alarm uses only the month, day, hour, minutes, and seconds values. No alarm uses the year value, but it must be included.

5.8.1.2 Resetting or Clearing an Alarm and Its Interrupt

- Enable or turn off the alarm type (CALL 210AH)

5.9 PCF8581/PCF8582 EEPROM OPERATION

The PCF8581 or PCF8582 EEPROM device (device address 0A4h) on the I^{2}C bus on pins P1.6 (SCL) and P1.7 (SDA) of the processor is supported generically in TBAS52 by the CALL 2110H write and CALL 2112H read functions.

The CALL 2110H function has been designed to be very general purpose in nature so that alternate supplier devices can be used. Some alternate suppliers are Atmel, Cypress, Microchip, National Semiconductor, SGS-Thomson, and Xicor. The "page-write" feature of the EEPROM devices is not supported since it can vary from device-to-device from different manufacturers.

On the PCF8581, which is only a 128 byte device, any read or write above address 127 (and below 256) results in a read or write of the lower 128 bytes. The BASIC programmer is responsible to ensure this "wrap" does not inadvertently occur.

6. TBAS52 MEMORY MAP

This memory map is a subset of the TUC52 Memory Map.

6.1 CODE MEMORY

```
0000h-1FFFh - BASIC Interpreter
2000h       - reserved
2001h-2002h - Intel BASIC-52 flags
2003h       - reserved
2004h-2005h - MTOP_LIMIT       ---\
2006h-2007h - MTOP_MAX             \
2008h-2009h - BBRAM_LIMIT           \
200Ah-200Bh - FREQ_VALS              \
200Ch-200Dh - PULSE_PORT_LO           >  "post assembly patch table"
200Eh-200Fh - PULSE_PORT_HI          /
2010h-2011h - PULSE_VALS            /
2012h-2013h - STATE_PORT_LO        /
2014h-2015h - STATE_PORT_HI       /
2016h-2017h - STATE_VALS       --/
2018h-203Fh - BASIC Interpreter
2040h-209Fh - Intel BASIC-52 reserved
20A0h-20B2h - BASIC Interpreter
20B3h-20B8h - BASIC Interpreter Version Number
20B9h-20FFh - BASIC Interpreter
2100h-2113h - CALLs Table
2114h-2C05h - BASIC Interpreter
2C06h-3FFFh - Available for code expansion
4000h-41FFh - Intel BASIC-52 reserved (Ints & BASIC CALLs)
4200h-7FFFh - Available for code expansion
8000h-FFFFh - External code space
```

6.2 EXTERNAL DATA MEMORY

```
‡0000h-01FFh  - TBAS52 buffer and stack space
 0200h-3FFFh  - TBAS52 BASIC program storage space
 4000h-7EFFh  - TBAS52 available data storage space
 7F00h-7FFFh  - TBAS52 BASIC reserved storage space
 8000h-CFFFh  - BBRAM BASIC program storage/EPROM space
 D000h-EFFFh  - reserved
 F000h-FFFFh  - reserved I/O space
```

‡Note 1: Locations 0128h and 0129h, which were used by BASIC-52 for EPROM programming, are now used by the pulse counter routines.

‡Note 2: Locations 012Ah and 012Bh, which were used by BASIC-52 for EPROM programming, are now used by the state change detector routines.

6.3 INTERNAL MEMORY ENHANCEMENTS

See the Intel documentation[1] for full description of internal RAM.

```
29.0h-29.2h   - Frequency channel mux state value
29.3h         - Frequency counter control bit (0 = off)
29.4h-29.5h   - Pulse counter state variable
29.6h         - Pulse counter/State change control bit (0 = off)
29.7h         - TIME control bit (0 = off)
44h           - Frequency counter state variable
```

6.4 CODE MEMORY ENHANCEMENT DESCRIPTIONS

The Code Memory Enhancements are implemented via the "post assembly patch table".

Location	Function
2004h-2005h	These locations contain the limit (default 3FFFh) that the TBAS52 interpreter uses to set MTOP when testing for the top of memory to use for BASIC programs. These locations are called MTOP_LIMIT and the value is stored in MSD:LSD format.
2006h-2007h	These locations contain the limit (default 7EFFh) that the TBAS52 interpreter uses to determine the maximum value MTOP can be set to using the MTOP operator. These locations are called MTOP_MAX and the value is stored in MSD:LSD format.
2008h-2009h	These locations contain the limit (default CFFFh) that the TBAS52 interpreter uses to determine the upper limit for BASIC programs using the PROG command, and the upper limit used by the ERASEALL and IDUMP commands. These locations are called BBRAM_LIMIT and the value is stored in MSD:LSD format.
200Ah-200Bh	These locations contain the address (default 7F00h) of the start of the frequency counter table where the 8 channels of frequency data is stored. Each channel has two bytes of frequency data. This address is the address of channel 0, followed 2 bytes later by channel 1, etc. to channel 7. These locations are called FREQ_VALS and the value is stored in MSD:LSD format.
200Ch-200Dh	These locations contain the address (default F9FDh) of the low port of the pulse counter channels (first 8 channels). These locations are called PULSE_PORT_LO and the value is stored in MSD:LSD format.
200Eh-200Fh	These locations contain the address (default FAFDh) of the high port of the pulse counter channels (second 8 channels). These locations are called PULSE_PORT_HI and the value is stored in MSD:LSD format.
2010h-2011h	These locations contain the address (default 7F10h) of the start of the pulse counter table where the 16 channels of pulse counter data is stored. Each channel has two bytes of pulse counter data. This address is the address of channel 0, followed 2 bytes later by channel 1, etc. to channel 15. These locations are called PULSE_VALS and the value is stored in MSD:LSD format.
2012h-2013h	These locations contain the address (default F9FDh) of the low port of the state change channels (first 8 channels). These locations are called STATE_PORT_LO and the value is stored in MSD:LSD format.
2014h-2015h	These locations contain the address (default F9FEh) of the high port of the state change channels (second 8 channels). These locations are called STATE_PORT_HI and the value is stored in MSD:LSD format.

2016h-2017h	These locations contain the address (default 7F30h) of the start of the state change register where the 16 channels of state change information is stored. Each channel is represented by a bit in the state change register. This address is the address of the top 8 channels followed in the next byte by the bottom 8 channels. This format allows the status of all 16 channels to be represented and read as a 16 bit word by the XBYW operator. These locations are called STATE_VALS and the value is stored in MSD:LSD format.
20B3h-20B8h	TBAS52 Version Number ASCII string in the form of: major_number . minor_number . devl_number (i.e. 2.0.05).

7. ADDED KEYWORD TOKENS

TOKEN	KEYWORD	TYPE
D4h	XBIT	OPERATOR
D5h	DBIT	OPERATOR
D6h	BIT	OPERATOR
F9h	ERASE	COMMAND
FAh	IDUMP	COMMAND
FBh	ILOAD	COMMAND
FEh	Reserved for Token Table bug fix	

8. BASIC-52 BUG FIXES

1. Code now correctly executes the auto startup EPROM program that contains user functions using the user expansion mechanism. This fix is compatible with the Intel recommended, now not needed, work-around (TechBits[3] 10/10/86).

2. Corrected BASIC-52 bug which lost the character of a variable if it was followed by a space and was the first character of the last built-in keyword (FPROG in BASIC-52) or the first character of the last user-added keyword.

3. Corrected BASIC-52 bug which crashed interpreter or executed user token 1Dh if user extensions were enabled and a BASIC line ended with the ':' character.

9. BASIC-52 VS. TBAS52 OPERATIONS

The following documents the changes and operational differences of TBAS52 with respect to BASIC-52.

9.1 MTOP OPERATOR CHANGED

Self-test of memory limited to 3FFFh and setting of MTOP limited to 7EFFh.

9.2 FPROG COMMAND DISABLED

Generates `ERROR: BAD SYNTAX` message.

9.3 FPROGX COMMANDS DISABLED

Generates `ERROR: BAD SYNTAX` message.

9.4 PGM INSTRUCTION DISABLED.

Generates `ERROR: BAD SYNTAX` message.

9.5 PROG COMMAND CHANGED

No EPROM programming supported, instead, stores BASIC programs in BBRAM.

9.6 PROGX COMMANDS CHANGED.

No EPROM programming supported, instead, stores startup options in BBRAM in the same manner as it did in EPROM with BASIC-52.

9.7 XBY OPERATOR CHANGED

Supports word operations (XBYW) in addition to byte operations.

9.8 CLOCK1 OPERATION CHANGED

CLOCK1 also controls a frequency counter, a pulse counter, and a state change detector in addition to the TIME variable. The default state for the CLOCK operation is now on (CLOCK1) after reset.

9.9 LPT PIN MOVED

The serial line printer output pin (LPT) has been moved from P1.7 to P1.1 to allow use of P1.7 as I^2C SDA pin.

9.10 BACKSPACE IMPROVEMENTS

Backspace support added to line editing - in addition to the DELete character.

9.11 ANOMALIES

OpByte 25h, which is the assembly language extension interface to the XBY operator, now requires an additional value on the argument stack to indicate the operation type, XBY or XBYW. If this is not done, an "A-STACK" error is likely to occur, but at the assembly level it is better to access XRAM directly with the "MOVX" assembly instruction.

10. REFERENCES

1. Intel MCS® BASIC-52 USERS MANUAL, Order Number: 270010-003, Intel Corporation, 1986
2. BASIC-52 PROGRAMMING, First Edition, Revision 1.1E, Systronix, Inc., November 1989
3. Intel TECHBits BASIC-52 (ECO), Reference No. MIC-23, B. Jones, Intel Corporation, October 10, 1986
4. TUC52 CIRCUIT DESCRIPTION (DRAFT), Issue 0.02, Paul Newland, ad7i, January 25, 1994
5. DATA HANDBOOK, I^{2}C Peripherals for Microcontrollers, Philips Semiconductors (Signetics), 1992
6. DATA HANDBOOK IC12, I^{2}C Peripherals, Philips Semiconductors, 1996

11. ACRONYMS & DEFINITIONS

To be added at a future time.

12. SPECIAL CONSTANTS INDEX