3.1 - BAUD, BAUD %,[integer], or BAUD #,[integer]

The statement BAUD allows the programmer to switch the baud rates of the serial ports. The baud may be changed in the program at any time. The baud statement only allows a single digit integer, no variables are allowed. This statement has been extended for the BAC552ES controller, the % sign represents the auxiliary RS232 DTE port and the # sign represents the RS485 port. There are two more status bits which indicate the activity of the new serial ports. Bit 501 is for the Aux. RS232 and bit 500 is for the RS485 port. Bit 503 is for the console port, same as the BAC552 controllers. These bits are set high or ON when a PRINT is executed and are turned OFF or set low by the interrupt handling routines when the last character is transmitted.

Console Port	RS232 Aux DTE Port	RS485 Port
BAUD 1 - 300 baud	BAUD %,1 - 300 baud	BAUD #,1 - 300 baud
BAUD 2 - 1200 baud	BAUD %,2 - 1200 baud	BAUD #,2 - 1200 baud
BAUD 3 - 2400 baud	BAUD %,3 - 2400 baud	BAUD #,3 - 2400 baud
BAUD 4 - 4800 baud	BAUD %,4 - 4800 baud	BAUD #,4 - 4800 baud
BAUD 5 - 9600 baud	BAUD %,5 - 9600 baud	BAUD #,5 - 9600 baud
BAUD 6 - 19200 baud	BAUD %,6 - 19200 baud	
	BAUD %,7 - 38400 baud	

NOTE - The BASIC print is queued and the user must ensure that the queue does not wrap around itself or over run. The print queue status should be checked before a PRINT is repeated. The status of any port can be tested by checking bits 503(console), 501(aux DTE) and bit 500(RS485). If the bit is high, then a print is in progress, as soon as the print is done then the bit goes low. The clearing of bits are done in the background by the print interrupt routines. The programmer should always clear the bits at the beginning of the program if the status is unknown. The following examples shows how to use these bits.

EXAMPLE:

```
5  OTU 503: OTU 501: CLEAR B: BAUD %,7
10 XIL 503: PRINT "This will always print properly!"
20 XIL 501: PRINT %"This goes to the aux port"
30 GOTO 10
```

3.2 - CALL [INTEGER]

The CALL statement allows the user to call custom routines in the BAC552ES's CODE memory space. The BAC552ES ROM occupies approximately 0000H-64FFH of CODE memory. The user can use any space above that. There is a configuration jumper at the top right corner of the BAC ROM which allows a 64K device to be installed if there is insufficient space in a 32K device.

EXAMPLE:

```
10  CALL 6500H
```

3.3 - CLEAR

The CLEAR statement alone is used to clear all variables. Extensions of the CLEAR statement are as follows below:

CLEAR I

The CLEAR I statement is used to clear all BASIC invoked interrupts. The CLEAR I resets a interrupt internal flags and sets all interrupt control bits to OFF. Therefore it is necessary to re-enable the control switches if interrupts are used.

CLEAR S

The CLEAR S statement is used to reset all BASIC control stacks to their reset value. This statement should be used when trapping control or argument stack errors.

CLEAR R

The CLEAR R statement is used to clear all relay bits (0-255) to a zero or OFF state. If the user wants to selectively clear only certain bits or words then a combination of OTU and CLW statements can be used. **CLEAR R only clears the lower half of the image table (000-255)!!!**

CLEAR D

The CLEAR D statement is used to clear all DLY statements to a reset state as the RST does. CLEAR D does all 128 delays at once.

CLEAR C

The CLEAR C statement is used to clear all CTU statements to a zero count. CLEAR C does all 128 counters at once.

CLEAR E

The CLEAR E statement clears the error LED and resets all error trapping. The ONERR statements will have to be re-executed after the CLEAR E is run.

CLEAR P

The CLEAR P statement resets all PULSE statements to a zero state.

CLEAR B

The CLEAR B statement resets the PLW and the PRINT pointers to zero. The CLEAR B also clears the PRINT in progress bits 503,501 and 500.

CLEAR O

The CLEAR O statement resets all one-shots at once.

3.4 - CLW [(@)integer]

The statement CLW clears image table words to the low or OFF state. The integer can range from 00-62. Note that only 2 digits are required but when using an indexed integer the value must be 3 digits long. The CLW is used to quickly clear 8 bits at a time. The CLW does not clear word 63 because those upper 8 bits are mapped to the special timing bits.The CLW must be used to clear bits higher than 256, the CLEAR R only clears bits below 256. The @ variation is very useful for clearing multiple words in the upper block of bits (256-499).

EXAMPLE:

```
10 CLW 00 : CLW 01 : REM clear two words
20 FOR X=32 TO 62 : REM clear all upper words 32 to 62
30 XBY(800H)=X
40 CLW @000 : REM multiple words with @[integer]
50 NEXT X
```

3.5 - COMINT [ln], COMINT %,[ln], or COMINT #,[ln]

The COMINT statement sets up the line number to GOSUB when a carriage return or a user specified character is received on the serial port. The default character is a CR, but the programmer can change it by altering an internal memory location. See Internal Memory locations for these addresses. The COMINT has been extended to support the additional serial ports on the new BAC552ES controller. The user can set up three separate COMINT subroutines (example: COMINT 1000, COMINT %,3000 and COMINT #,5000). The %, is for the Aux. DTE port and the # is for the RS485 port.

The COMINT is a BASIC interpreter interrupt and not a processor hardware interrupt, some slight latency can be experienced. Every character that is received on the serial ports is buffered in three separate 256 byte ring buffers. A BASIC interpreter flag is set when a (CR) or a user character is received on any port with the communication interrupt enabled (CIC). BASIC must then finish off the current statement that it is executing before it can service the interrupt. This does not mean that the interrupt is missed, a small delay may be experienced. The COMINT interrupts have priority over all other interrupts and can be controlled OFF or ON at any time by the CIC statement. Once the CIC is switched ON for the console port the normal CTRL-C break is disabled. This allows the programmer to provide a custom break.

CIC[1/0], CIC%,[1/0] or CIC#,[1/0]

The CIC statement provides a means of controlling the enabling and disabling of the communication interrupt. The ON/OFF switch uses a '0' to represent OFF and a '1' to represent ON. The CIC can be used at any time to control the interrupt. **NOTE: Every time the communication interrupts are enabled (CIC 1) the serial buffer is pointed to zero and the buffer is nulled to all zeros. This means that received data will be lost.**

RETI

The RETI statement clears the internal flags that generated the interrupt and returns program execution to where it was interrupted from. It is the same for all interrupt types.

EXAMPLE:

```
 5  REM setup comm interrupt, clear counter and turn comint ON
10 COMINT 1000: CIC 1
11 COMINT %,2000: CIC %,1: REM Aux port set up and enabled     15
STRING 100,10

1000 REM communication interrupt routine
1010 CIC 0
1020 INPUT $(0)
1030 PRINT" THE RECEIVED DATA IS ", $(0)
```

```
1040 CIC 1: RETI
2000 CIC %,0
2010 XSBE %"BAC552ES": OTL 0022020 CIC %,1: RETI
```

3.6 - CTU[integer],[(@)integer]

The CTU statement requires two values, first the Counter number ranging from 000-255 and the second is the counter preset value up to 255. The CTU works just the same as other control statements, the CTU is true once the accumulated value equals the preset. The counter value is accumulated every time it is read by the interrupter. Placing the CTU in the main loop of a program would just quickly count up the number of loops. It may be desired to count program loops and some examples will be shown.The purpose of the CTU is event counting and qualifying statements should be placed before it. The CTU can also be placed in interrupt invoked subroutines. The @ option allows that integer value to be indexed.

CTR[(@)integer]

The CTR statement clears the counters accumulated value and done bit. The counter will not wrap around. Once the preset is reached the counter is done and CTR is required to clear it.

EXAMPLES:

```
10 CTU 000,201 : CTR 000
20 XCT 000,050 : GOSUB somewhere at 50th loop
30 XCT 000,100 : GOSUB at 100th loop
40 XCT 000,150 : XIH 016 : PRINT"150th and 16 ON"
50 XCT 000,200 : CTU 001,100:CTR 001: PRINT"20,000thloop!"
999 GOTO 10

10 ONINT 0,1000
20 DIC 0,1

1000 CTU 255,100:CTR 255:DIC 0,0:PRINT"100 transitions"
1010 REM after 100 counts turn OFF interrupt
1020 RETI

10 XIH 017:DLY 001,100:RST 001:CTU 003,100:CTR 003:JMP 001
20 LBL 001:PRINT"Long delays can be obtained 10,000 sec!"
30 XIL 017:CTR 003:RST 001:REM if low reset all
```

3.7 - DATA READ RESTORE

DATA

The DATA statement specifies the expressions that are to be retrieved by the READ statement. If there are multiple expressions per line then they must be separated by a comma (,).

READ

The READ statement retrieves the expressions specified in the DATA statement and re-assigns them to the variable(s) in the READ statement. Multiple variables in the READ must also be separated by a comma (,).

RESTORE

The read pointer is reset back to zero so the data can be read by another READ statement.

EXAMPLE:

```
10 FOR X=1 TO 4
20 READ A,B,C
30 PRINT A,B,C
40 NEXT X
50 RESTORE
60 FOR X=1 TO 6
70 READ D,E
80 PRINT D,E
90 NEXT X
100 DATA 1,2,3,4,5,6,7,8,9,9+1,9+2,9+3
>RUN
1  2  3
4  5  6
7  8  9
10 11 12
1  2
3  4
5  6
7  8
9  10
11 12
```

3.8 - DATE, DATE%, or DATE#

The DATE statement sends the real time clock date to the serial port. The DATE statement only sends the day of the month value, the month value and the year value. No day of the week is sent but, the day can be read from a clock memory location if required. Date is sent DATE-MONTH-YEAR. The DATE has been extended on the BAC552ES.

EXAMPLE:

```
10 DATE : REM send DATE to RS-232 + crlf
20 DATE; : REM send DATE to RS-232 no crlf
30 DATE# : REM send DATE to RS-485 + crlf
40 DATE#; : REM send DATE to RS-485 no crlf
50 DATE%  : REM send DATE to the Aux. DTE port
```

3.9 - DIM [array],[array],...,[array]

The DIM statement is used to allocate memory for arrays. The default for a un-dimensioned array is 10 bytes and the maximum size for a single array is 254 elements. The amount of memory required is equal to the number of elements times the 6 bytes plus 1 element. Therefore A(254) requires 1530 bytes. Once an array has been dimensioned it cannot be re-dimensioned.

EXAMPLE:

```
10 DIM A(100), B(50), C1(20)
```

3.10 - DLY [integer],[(@)integer]

The DLY statement requires two integer values to be supplied. The first is the delay number ranging from 000-127 and the second is the length of the time delay ranging from 001-255 (000 is forced to a 1 internally). The delay works just as the XIH or XIL instructions. If the DLY is false (time not elapsed yet) the program execution is passed to the next line. If the DLY is true then execution continues after the DLY. The time delay value can be indexed to a memory location allowing variable delay settings. Once a DLY has executed once the DLY is said to be enabled and the time value is counted down to zero. If the delay value was changed while the delay was in progress the new time would not be loaded until the DLY was RST (reset) and then enabled again.

The DLY must be in an active program area to execute the statements after it becomes true. If the program loop was longer than 1 sec then the resolution is affected (statements after would not be executed until the program got back to the DLY again). In order to change the delay in progress the memory location relating to the specific DLY would have to be changed. By reading the correct location the current time left can be seen. Refer to memory mapping for these addresses. DLYs can be cascaded together in series and there is no limit to how may, they just execute one after another.

The programmer should always uses DLYs in ascending order. The DLY value is decremented in the background by a interrupt routine. The number of DLYs that are serviced is equal to the highest number currently enabled by the user program. That means that if the application uses 4 DLY statements and the user chooses DYL 121, DLY 122, DLY 123, and DLY 124 processor time is wasted checking DLY 000 through DLY 120. The most time efficient choice would be DLY 000, DLY 001, DLY 002, and DLY 003.

Now every time the processor services the delays only 4 are checked instead of 124. Although the time required to service the delays is very small, it is just a good choice to use the processor time wisely.

RST [(@)integer]

The RST resets the specified DLY to an un-enabled state. The RST clears the DLY "done bit" allowing a restart. Until the RST the DLY will always be true to execution. The CLEAR D resets all DLYs at one time and indexing can be used to reset small specific groups of DLYs.

EXAMPLE:

```
10 XIH 016 : DLY 000,010 : ?"Here after 10 secs"
20 XIL 016 : RST 000 : REM reset DLY if 16 goes low

10 DLY 000,002:OTL 000:DLY 001,002:OTU 000:RST 000:RST 001
20 XIC 017 : DLY 002,255 :DLY 003,045 :?"300 SECONDS"
30 XIL 017 : RST 002: RST 003
40 XIC 050 : DLY 127,050 : RST 127 : GOSUB 1000
50 DLY 100,255 : DLY 101,255 : DLY 102,255 : JMP 010
```

```
60 LBL 010 : DLY 103,235 : ?"1000 SECS" : JMP 011
70 LBL 011 : RST 100 : RST 101 : RST 102 : RST 013
80 XIH 100 : DLY 000,255
90 PRINT"Time left is ",XBY(501H),"seconds"
```

3.11 - DO UNTIL [rel expr]

The DO UNTIL loop will execute all statements within the DO UNTIL statements, for the relational expression. As soon as the relational expression becomes true then the loop is finished. Nesting in the DO UNTIL is allowed.

EXAMPLES:

```
10 A=0
20 DO
A=A+1 : PRINT A
40 UNTIL A=3
50 PRINT"LOOP DONE AT ",A
>RUN

1
2
3
LOOP DONE AT 3
```

```
10 A=0 : B=0
 20 DO : A=A+1 : DO B=B+1 30
30 PRINT A,B
40 UNTIL B=2
50 B=0
60 UNTIL A=2
>RUN

1     1
1     2
2     3
2     4
```

3.12 - DO WHILE [rel expr]

The DO WHILE is similar to the DO UNTIL loop. The statements within the DO WHILE are executed while the relational expression is true. Once it becomes false the loop is completed.
Nesting is also allowed as in the DO UNTIL loop.

EXAMPLES:

```
10 DO
20 B=B+1
30 PRINT B
40 WHILE B<5
50 PRINT "DONE AT ",B
>RUN
1
2
3
4
5
DONE AT 6
```

3.13 - EEPROM [address+R/W],[location],[data]

The statement EEPROM reads or writes data to serial EEPROM devices on the I2C expansion bus. The last bit in the EEPROM address is the control switch for the read or write. If the address was 0A0H then the statement would perform a write operation. If the address was 0A1H then a read would be performed. The location value can be a variable ranging from 0 to 255D. The data value can be a variable for WRITE operations, but for READS it can only be a variable! The following example illustrates this and there is a demo program EEPROM.BAS at the back of this manual.

EXAMPLE:

```
 10 EEPROM 0A0H,1,101: REM write 101 to location 1
 15 EEPROM 0A0H,A1,B1: REM write data in B1 to location A1
 20 EEPROM 0A1H,1,X: REM read location 1 and store data in
variable X
```

3.14 - EXIO [integer],[integer]

The statement EXIO reads and writes to the expansion I/O. The integer after the EXIO is a 2 digit value which equals the address of the expansion board to be accessed and the single digit integer following the comma is the board configuration number. The expansion boards provide an expansion of 16 bits and that can be all inputs, all outputs, or 8 inputs and 8 outputs. The configuration number tells the controller how the image table is updated (reading or writing). There are only 3 possible configurations allowed for expansion boards. The first (0) is an all input board, the second (1) is an all output board and the final is (2) which is a combination input/output board. There are no restrictions on how many EXIOs exist in a program. In a typical high speed application the EXIOs can follow a LIO before the loop back or if the devices connected to the expansion boards do not require fast updating then the EXIOs can be executed by a counter after X number of LIOs. The inputs on the current expansion boards are not captured so if an input comes and goes before a EXIO it will be missed!

EXAMPLE

```
10 XIH 040 : OTU 004
20 XIL 040 : OTL 004
30 XIH 020 : OTL 000 : OTL 001
.
99 LIO : EXIO 00,0 : GOTO 10
    or
10 XIH 040 : OTU 004
20 XIL 040 : OTL 004
30 XIH 020 : OTL 000 : OTL 001

98 CTU 000,010:EXIO 00,0:CTR 000: REM after 10 LIOs do a EXIO
99 LIO : GOTO 10
```

3.15 - FOR TO {STEP} NEXT

FOR

The FOR statement sets up the variable used in the FOR NEXT loop.

TO

The TO statement sets up the limits of the variable in the loop.

STEP

The STEP value sets up the increment of the NEXT statement and if STEP is not specified then the default value is 1.

NEXT

The NEXT statement adds then value of the step to the index and then compares the index. If the index is less than or equal to the limit then control passes back to the FOR statement

EXAMPLE:

```
10 FOR T=0 TO 10 STEP 2
20 PRINT T
30 NEXT T
>RUN
0
2
4
6
8
10
```

FOR NEXT loops can also be nested as the DO UNTIL and the DO WHILE. The FOR NEXT loop statement can operate at the command prompt as seen in the example below which displays values from the real time clock locations.

EXAMPLE:

```
>FOR X=0A90H TO 0A96H : PRINT XBY(X) : NEXT X
```

3.16 - FLFP [integer]

The FLFP statement is used to change the state of the specified bit in the input/output image table. If the bit was low then FLFP would set it high and vice versa. FLFPs can be used with push buttons to give a push ON, push again for OFF effect. The FLFP gives the programmer a divide by 2 for bit states. Flipflop works all bits 000-499.

EXAMPLE:

```
10 CTU 100,100 : FLFP 010 : CTR 100
20 REM bit 10 high for first 100 and then low for next
100      30 XIL 016 : ROS
000
40 XIH 016 : OST 000 : FLFP 100
50 XIL 100 : PRINT "BIT 100 OFF"
60 XIH 100 : PRINT "BIT 100 ON"
```

3.17 - GOSUB [line number] RETURN

GOSUB

The GOSUB statement saves the location of the line following the GOSUB to the control stack and then transfers control of the program to the line number in the GOSUB statement.

RETURN

The RETURN statement recovers the saved line number from the control stack and returns the program execution back to that point in the program. The GOSUB - RETURN routines can be nested, while in one subroutine another routine can be called and executed.

EXAMPLE:

```
10 PRINT "MAIN PROGRAM
20 GOSUB 100
30 STOP
100 REM SUBROUTINE HERE
110 FOR A=1 TO 4
120 PRINT A
130 NEXT A
140 PRINT "SUBROUTINE FINISHED"
150 RETURN
```

3.18 - GOTO [line number]

The GOTO statement will transfer the program control to the given line number. If the line does not exist then a invalid line number error would result.

EXAMPLE:

```
10 GOTO 100
20 XIH 009 : GOTO 200
30 DLY 000,010 : GOTO 300
40 GOTO 10
```

3.19 - HIMEM [0,1] or HIMEM F,[0,1]

The HIMEM is a control switch used to redirect the ST@,LD@, XBY and the destination of the MOVE statement. The zero is used to disable the redirect and the one is used to enable it. HIMEM by itself is used to access the upper 64K block of SRAM and HIMEM F is used to access the upper 64K of FLASH memory. Once HIMEM is enabled, every read or write for the memory access statements are redirected. **The FLASH memory only has a 100,000 cycle write endurance. That is more than 100,000 writes to the same location can cause a device failure! The user must be very cautious when developing routines that use the HIMEM F. A loop gone wrong could damage the device! Also the memory location must be ERASED before it can be written to. A failure to have the location ERASED will cause a FLASH ERASE FAILURE ERROR!**

The FLASH can only be erased by using the ERASEH,(sector) command.

Remember that the HIMEM only redirects the destination of the MOVE statement. The source must always be the lower 64K of SRAM.

EXAMPLE:

```
10 HIMEM 1
20 XBY(0)=55
30 XBY(0FFFFH)=129
40 HIMEM 0
50 PRINT XBY(0), XBY(0FFFFH)
.
10 HIMEM F,1: REM SWITCH TO FLASH
20 MOVE 0A90H,0,7: REM MOVE CLOCK TO FLASH LOCATIONS 0-6
30 PRINT XBY(0),XBY(1),XBY(2),XBY(3),XBY(4),XBY(5),XBY(6)
40 HIMEM F,0
```

3.20 - IF THEN ELSE

IF [relational expression]

The IF statement sets up the relational expression to be tested.

THEN

The THEN statement sets up the action to be taken following the relational expression test.

ELSE

The ELSE statement sets up alternate action to be taken following the first relational test. The ELSE statement can be omitted, the result is the program passes to the next line following the IF - THEN statement.

EXAMPLE:

```
10 IF A<10 THEN A=10 ELSE A=B
.
20 IF ADC0>512 THEN OTL 000 ELSE OTU 000
.
30 IF B>20 THEN GOTO 100 ELSE GOTO 200
or
30 IF B>20 THEN 100 ELSE 200
.
40 IF T1>100 THEN CTU 000,005 : OTL 001
or
40 IF T1>100 CTU 000,005 : OTL 001
```

The THEN and ELSE statements can be replaced by any other valid statement as seen in line 30, the GOTOs are replaced by valid line numbers and in line 40 and the CTU replaces the THEN CTU.

3.21 - IIC [INTEGER],[INTEGER]

The statement IIC [address],[number bytes] initiates a master mode transmit or receive. The transmit or receive is determined by the last bit in the address. Therefore, to initiate a master transmission the address would be even (42H) and for receive the address would be odd (43H). The maximum number of bytes is 32. A 128 byte block of SRAM is reserved for IIC data buffers. The first block is for master transmit and is addressed at 0A00H to 0A1FH. The second is for master receive and is addressed at 0A20H to 0A3FH. The third is for slave receive and is addressed at 0A40H to 0A5FH. The last is for slave transmit and is addressed at 0A60H to 0A7FH.

This simple IIC operating system was devised to transfer blocks of data between multiple BAC552 controllers. To transfer data to another controller the master transmit buffer is loaded with data, then the IIC statement is executed. (address and number of bytes must be specified)
Assuming that another controller has been setup with the matching address, the transmitted data would appear in it's slave receive buffer. If the address was odd (master receiver) then data can be requested from the other controllers slave transmitter buffer. Four of the upper image table words are used to test the IIC hardware states for IIC data transfers.

It is recommended that the user become familiar with the Philips IIC bus and how it operates.

IIC bus operational description is available in the Philips Microcontroller Handbook.

A sample demonstration program shows how to block transfer data between two controllers using the IIC bus.

NOTE: Not all user configurations of BAC552 controllers and expansion boards will allow easy implementation of this IIC block data transfer. For example, only one X10 controller can exist in any system because of address conflicts (X10 uses IIC bus communication). Please discuss your application with SYLVA ENERGY SYSTEMS if you have any concerns.

ADR [INTEGER]

The ADR statement sets the I2C address for the BAC552 controller. If the ADR is not executed then the controller will have a default address of 30H. The I2C address uses the upper 7 bits and a address of 31H is the same as 30H. The ADR need only be executed once at the program start.