3.39 - PUSH [expression]

The PUSH statement is used to push expressions onto the argument stack. This provides a way to swap variables, pass variables to a subroutine and in conjunction with the ST@ statement save floating point variables to protected memory locations for storage. Remember that the stack works in a first in last out fashion.

EXAMPLE:

```
10 A=100 : B=234
20 PRINT A,B
30 PUSH A,B
40 POP A,B
50 PRINT A,B
>RUN

100     234
234     100
```

```
10 A=3 : B=4 : C=5
20 PUSH A,B,C
30 GOSUB 100
40 POP R1,R2
50 PRINT R1,R2

100 POP A,B,C
110 PUSH (A*B)+C
120 PUSH (C*B)+A
130 RETURN
>RUN

23     17
```

3.40 - PRINT, PRINT% or PRINT#

The statement PRINT directs the interpreters output to the serial ports at the default baud rate or the rate set by the BAUD statement. The PRINT uses a separate 256 byte buffer for each of the serial ports allowing unaffected program execution even at very slow baud rates. The print buffers are ring buffers and wrap around if PRINTs are repeated overflows. It is unlikely that a control program would be doing extensive printing but if that is the case then the testing of the print status bits 503, 501, and 500 are important to provide buffer flow control. Every time a PRINT is executed the particular status bit is set high or ON and when the PRINT has been completed the status bit is set low. By checking the status bits the PRINT status can be determined. Multiple variables and expressions can be printed by using a comma to separate them and the final carriage return/line feed can be suppressed by a semi-colon. The percent sign (%) following the PRINT is used to print to the auxiliary RS232 DTE port. The pound sign (#) is used to print to the RS485 port. The status bit for the console is bit 503, the status bit for the DTE port is 501, and the status bit for the RS485 port is bit 500.

EXAMPLE:

```
10 XIL 503: PRINT ADC0,T1,XBY(900H)
.
20 XIL 501: PRINT% "SUPPLY VOLTAGE = ",ADC7*.01955
.
30 PRINT T1,T2,T3,T4,ADC1,ADC2,ADC3,ADC4;
.
40 PRINT CHR(130);
.
50 PRINT #"SEND THIS ON THE 485 LINE WITH NO CRLF";
5  REM proper way to print very fast one after another
10 XIL 503: PRINT"PRINT THIS FAST!!!!!!!!!"
15 FOR T=1 TO 30: NEXT T
20 GOTO 10

5  REM wrong way to print very fast one after another
10 PRINT"THE BUFFER WILL OVERFLOW !!!!!!!!"
15 FOR T=1 TO 30: NEXT T
20 GOTO 10
```

3.41 - SPECIAL PRINT FORMATTING STATEMENTS

TAB([expression])

Including the TAB statement after a print will cause data to be printed out at exact positions.

EXAMPLE:

```
10 PRINT TAB(10),"10th pos",TAB(20),"20th pos"
```

.

SPC([expression])

The SPC statement in the print will cause the number of spaces specified in the expression to be printed.

EXAMPLE:

```
 10 PRINT ADC4,SPC(10),ADC5
```

F([expression])

The F(format) after the print changes the print format to floating point format. This format value is saved and if the format is to be the same throughout then it not need be repeated. The value of the expression can range from 0 to 8, with 0 being free floating with any other value specifying the number of significant digits. By using the # sign the programmer can specify integer format up to a 8 digit combination. F(####.####) displays 4 digits before and 4 after then decimal point. F(##.##) displays 2 and 2 with F(########) being 8 digits max, no decimal.

EXAMPLE:

```
 10 A=1
 20 PRINT F(F6)
 >RUN
 1.0000 E 0

 10 A=23.987
 20 PRINT F(##.##)
 >RUN
 23.98
```

3.42 - PULSE [integer],[(@)integer]

The PULSE statement pulses the image table bit specified by the first integer value for the time specified by the second integer. The pulse bits are tested every second by an interrupt routine in the background and up to a 1 second latency period can be experienced before the actual output turns ON. If an application requires an immediate pulse output, then use the immediate output (OTL!) and use a counter to clear the bit after some period. See S_PULSE.BAS on the demo disk. PULSE works with all lower bits therefore the first integer ranges from 000-255 and pulse time ranges from 1 to 31 seconds. PULSE is slightly different from the DLY and the CTU in that PULSE only has to be executed once! The pulse function works in the back ground and constant execution will cause the specified bit to remain high or ON. PULSE works well in a subroutine where the cycle time is greater than the pulse time.

```
10 XIL 016 : ROS 000
20 XIL 000 : XIH 016 : OST 000 : PULSE 000,005
30 REM read 0 bit and if OFF and 16 high then allow 0 bit
40 REM to be PULSED for 5 seconds
100  XBY(80AH)=20 : REM set pulse time
.
1000 REM here on discrete interrupt
1010 PULSE 100,@010 : RETI
1020 REM pulse for time in location 10
```

3.43 - RESET

The RESET statement generates a warm reset condition just as if the user pressed the reset button on the controller board. If a cold reset is required, the user must clear memory location 2FFH before the RESET statement is executed. This will cause the WARMBOOT statement to act as if it was a cold boot.

EXAMPLE:

```
1000 REM here after a error was trapped
1010 RESET
or
1000 REM clear boot state and do a reset
1010 XBY(2FFH)=0 : RESET
```

3.44 - SIO [2 digit decimal ID],[2 digit IO word address]

The SIO statement is used to transmit IO word data from one controller to another on any the serial port. The programmer can specify SIO, SIO%, or SIO#.

The SIO statement generates the following data packet:

![ID]:[IO word address]:[IO word data]:[Chksum]<CR>

The data is transmitted as ASCII HEX equivalents. The ':'s are set as separators to make the packet readable.

A sample data packet looks like this: !00:00:AA:86<cr>

In this sample packet the ID=00, the word address is 00 (000-007 bits), the data was AA and the Chksum is 86H. The checksum is calculated on the actual binary values that are sent to make
the string. example: '!'+0+':'+0+':'+170d+':' (8 bit ADD no carry then CPL)
One might want to produce this string from a host computer.

3.45 - RIO [ID],[originating IO address],[destination IO address]

The RIO statement is used to read a SIO transmission and store the data in the assigned word location. The RIO statement must be placed in the matching COMINT routine with the proper extension included. The ID and originating IO address are used to make the packet distinctive from multiple packets and more legible in a program. The example below is used to receive our sample packet and write this data to IO word 04.

One additional feature of the RIO is that it reads true like a XIH statement when the packet qualifies and the IO word is written. The data packet is compared byte by byte with the COMINT buffer data and then finally with the packet checksum. If it fails at any time the statement after the RIO is REMed and no data is written. This yields very good reliability. The transmitting controller can use the XSB to test for the "DATA OK!" string, this gives some ACK for received data packets.

EXAMPLE:

```
10   COMINT #,1000: CIC#,1
.
1000 CIC#,0
1010 RIO # 00,00,04: PRINT #"DATA OK!"
1020 CIC#,1: RETI
```

3.46 - SETRTC [day],[date],[month],[yr],[hr],[min],[sec]

The SETRTC statement sets the time and date in the real time clock. The time and date values can be variables. This allows for remote clock setting through custom communication interrupt routines and day light savings time adjustment routines. The SETRTC statement also works at the command line.

EXAMPLE:

```
>SETRTC 6,10,6,94,12,23,0
 100 XIH 056 : GOSUB 1000
< 1000 SETRTC T1,T2,T3,T4,T5,T6,T7 : RETURN
```

3.47 - ST@ [expression]

The ST@ statement pops the argument stack and saves that floating point value in the memory location pointed by the expression. The number is saved down from the specified location. That is if the user specifies location 1805H, the ST@ uses locations 1805H to 1800H. The floating point numbers take 6 bytes for each so be careful when specifying locations.

EXAMPLE:

```
10 PUSH A : ST@ 1805H : REM save A at 1805H<
20 LD@ 1805H : POP B : REM load from 1805H to B
```