TANDY
ELECTRONICS
The biggest name
in little computers

MICROCOMPUTER NEWSLETTER

Volume No. 7 — Feb. March, 1980

Prices may vary at individual Stores and Dealers

NEW PRODUCTS

HARDWARE:

The TRS-80 Vox Box, speech recognition unit allows you to control and instruct your Level II TRS-80 and to enter data by using words and phrases. Focus on the work or play in hand without having to type on the keyboard.

— Simple to use — connects to either the TRS-80 or an Expansion Interface.

— Dynamic push-to-talk microphone included.

— All driver software provided, plus 3 applications programs.

The Vox Box can be 'taught' to recognize up to 32 words and phrases — in ANY language! 26-1181
Vox Box $199.95

SPACE SAVER DESK

If you have a need for a desk for your TRS-80, but find that our System Desk is too large, then we now have a desk which will suit you. It is only 95cm by 60 cm yet it will hold a TRS-80, cassette recorder, monitor, Expansion Interface, Quick Printer and 1 or 2 disk drives and still leave plenty of work space. 26-1304
Space Saver Desk $129.95

SOFTWARE: The following programs are now in stock.

Casino Games:
This package consists of these 6 games —

•**Craps** — plays the popular dice game of Craps. The computer rolls the dice and the players place their bets.

•**Roulette** — Place your bets and watch the wheel spin. Computer pays odds of from 1:1 to 35:1. If you have a 'system', you can try it without risking any hard cash!

•**Slot Machine.** Turns your TRS-80 into a one-armed bandit. Play a 'three wheeler', each wheel has 20 positions which gives 8000 combinations. Average payout is 97% unless you strike a bad run.

•**Baccarat** — One of the most complex and intriguing games ever conceived. Your TRS-80 acts as both banker and dealer, so you don't have to worry about 'illegal plays'. It also plays 'Chemin de Fer', which is a variation of baccarat.

•**Keno** — This is a similar game to Bingo (or Lotto). You can win up to $25,000 for just a one dollar bet. There are 8 different ways to do it, and up to 73 different winning combinations. Of course there are several thousand ways to lose your money. REquires Level II and 16K RAM. 26-1806
Casino Games $29.95

•**Wheel of Fortune.** Try this old favourite. You can bet on 7 out of 40 numbers with odds of from 1:1 to 40:1. Spin again? Just press ENTER.

REAL ESTATE

This is the first Tandy program to deal with a specific industry. At present 3 volumes are available, dealing with the following:

Vol. 1 — Mortgage payments and balance
Mortgage amortization
Wraparound mortgage
Yield and interest

Vol. 2 — Compound interest
Rate of return
Resale analysis
Stepped income analysis

Vol. 3 — After tax capitalization
Income and expense projection
Present worth
Appreciation and depreciation

Catalogue numbers are 26-1571, 26-1572 and 26-1573 respectively.
Real Estate (each) $59.95

ELIZA

Meet ELIZA, this facinating program will carry on a conversation with the operator on any subject you care to discuss. ELIZA will analyze your input and alter its responses accordingly. If you have a Voice Synthesizer ELIZA will even speak to you!

Don't be fooled by ELIZA's apparent intelligence, it is only a simulation of artificial intelligence. Requires 16K RAM. 29-1908.
Eliza $24.95

CHECKERS-80

If you are tired of playing Microchess then try our new program Checkers-80. This machine language program plays checkers at 2 levels of difficulty with look ahead of either 4 or 6 moves. No illegal moves or skipped jumped allowed! Don't be surprised if this program forces you to start playing a better game. Requires 16K RAM. 26-1907
Checkers-80 .... $14.95

HINTS & ERRORS:

If you are using error trapping routines in DISK BASIC you should be aware of the following:

The Level II Manual uses the formulae;

true error code = ERR/2 + 1
or
(true error code − 1)*2 = ERR

This is correct for Level II errors how-ever in DISK BASIC this is incorrect use instead;

true error code = ERR/2
or
(true error code)*2 = ERR

In the Level II Manual on page A/16 it states the minimum requirement for a line in BASIC as 5 bytes plus 1 byte for each reserved word or BASIC command. In case you were wondering how Level II stores its commands in 1 byte, the answer is, it places a numeric value of from 128 to 250 in place of the command name. To examine these codes and their corresponding reserved words, run the following program.

```
10 X = 128:CLS:FOR I =
5712 TO 6175
20 PRINT CHR$(PEEK(I) AND 127);:
IF PEEK (I + 1) > 127
   PRINT " " , X,:X = X + 1
30 NEXT
```

LEVEL I

Although many programs for using TRS-80 Level I as a four function calculator have been printed, the following must be the simplest

```
10 INPUT A:PRINT A
20 GOTO 10
```

This will not work with Level II as you cannot enter +, −, * or / into a numeric variable.

Budget Management (26-1603) Note

Budget Management requires a 110 column (or wider) printer, such as our Line Printer I (26-1152) or (26-1150)

TRS-80 CLASSROOM

TRS-80 users have expressed confusion over th cause and cure of "garbage digits" which appear in double precision numbers. The purpose of this article is to precisely describe the problem and suggest some remedies.

Try typing the following into a Level II or a Disk TRS-80:

```
A =.1
B = .5
PRINT A,B
.1                        .5
```

This is, of course, what you would expect. Now enter .1 and .5 as double precision values:

```
A# =.1
B# =.5
PRINT A#,B#
.1000000014901161         .5
```

.1 (Which can not be represented exactly in binary) did not convert to a double precision value correctly.

Here are three methods to make sure that BASIC knows you want a double precision value. Each of the following three lines is a different method.

```
A# = 0.1D
B# = 0.100000000
A = .1:C# = VAL(STR$(A))
PRINT A#,B#,C#
   .1          .1          .1
```

The first two methods are for numbers you actually type in (as part of the BASIC program). You must put a 'S' on the end of the number, or type more than seven digits. This will make BASIC use a double precision value.

The third method is the one most people miss. Here you force BASIC to take a single precision variable and re-interpret the value as double precision. Integer values (whole numbers) are no problem in double precision arithmetic. Look at the following program:

```
10 HOURS = 12
20 OVERTIME = 0
30 IF HOURS > 8 THEN OVERTIME
   = HOURS - 8:HOURS = HOURS
   - OVERTIME
40 RATE = 5.65
50 R# = VAL(STR$(RATE))
60 H# = VAL(STR$(HOURS))
70 0# = VAL(STR$(OVERTIME))
80 PAY# =  H# *R# + 0# *2 *R#
90 PRINT PAY #
```

On lines 10 through 40, only single precision values (and whole numbers) are used, so no problems will occur. But in line 80, we want double precision PAY # out of single precision HOURS, OVERTIME and RATE. This is where problems arise. In lines 50 to 70, we fix all the single precision numbers into double precision values. Because of lines 50 to 70, lines 80 and 90 will produce the correct gross pay. Note that the '2' is used on line 80 with no ill effect. Remember, numbers without fractional parts (nothing after the decimal point) will not bother double precision arithmetic. If the overtime rate were 1.5 instead of 2, A '1.5D should be used to insure that the correct double precision value is used.

Disk BASIC users can use a 'DEF FN' statement to make programming easier. The following program repeats the one above, but uses a 'DEF FN' function which fixes double precision numbers:

```
 5 DEF FNX#(A) = VAL(STR$(A))
10 HOURS = 12
20 OVERTIME = 0
30 IF HOURS > 8 THEN
   OVERTIME =
   HOURS - 8:HOURS = HOURS -
   OVERTIME
40 RATE = 5.65
50 PAY # = FNX(HOURS)*
   FNX#(RATE) + FNX#(OVERTIME)
   *2*FNX#(RATE)
60 PRINT PAY #
```

Simply put the FNX#( ) any variable or constant that has a decimal portion. If you are experienced in binary arithmetic, you should be able to tell when you need to apply this fix function. When in doubt, put FNX# around any number you question.

Why do "garbage digits" appear? The answer lies in the way BASIC stores numbers and converts them from single precision to double precision.

Recall our first two numbers, A and B. The internal hexadecimal representations of these two numbers are:

```
Decimal        Hexadecimal

 .1    =    125   76   204  205
 .5    =    128    0     0    0
```

Note the zeros on the end of the .5 re-presensation in hexadecimal. This means that .5 is not an infinite repeating fraction in hexadecimal (or binary). But, when .1 is converted to hexadecimal, it becomes an infinite repeating fraction. In binary it is 0.11001100110011001100 ... Now let's look at the hexadecimal values for A# and B#.

```
Decimal        Hexadecimal

 .1  =  125  76  204  205  0  0  0  0
 .5  =  128   0    0    0  0  0  0  0
          (4 zeros added to end)
```

When BASIC converts a single precision number (.1 or .5) to a double precision number (.1 or .5) to a double precision value, you'll note that four zeros were simply added to the end of the single precision representation. Since .5 was not an infinite repeating binary fraction (it has zeros on the end), adding more zeros to the end does not affect the value. The number .1, however, is no longer correct. The correct hexadecimal value is';

125 76 204 204 204 204 204 205

Why didn't BASIC figure out that the last four numbers were not zeros? The answer is   , you can't get something out of nothing. BASIC must be explicitly told what the last four numbers are, or it will use zeros (because BASIC simply dosen't know).

Integrating TRSDOS® 2.2 Information into TRSDOS Manual

In order to integrate the TRSDOS 2.2 release information into our current TRSDOS manual, renumber the release pages as follows:

Cover Letter — 1.7

Memo	New #	Memo	New #
1	1-8	2	3-2.5
3	1-9	4	1-10
5	4-10.5	6	7-2.5
7	7-2.7	8	7-2.81
9	7-2.82	10	7-2.83
11	7-10.5	12	7-18.51
13	7-18.52	14	5-10
15	5-11	16	5-12
17	5-13	18	5-14
19	5-15	20	5-16
21	1-11		