~~~~~~~~~~~~~~~~~~~~
W-AGE/99 * NEW-AGE/
99 * NEW-AGE/99 * N
EW-AGE/99 * NEW-AGE
/99 * **NEW-AGE/99** *
~~~~~~~~~~~~~~~~~~~~

* by JACK SUGHRUE, Box 459, East Douglas, MA 01516 *

#1

Why NEW-AGE/99? Well, it's been almost a year since my long-running IMPACT/99 columns were rudely interrupted by a car accident which kept me from my beloved ♥ little computing machine.

So why NEW-AGE/99? After my brush with death last spring I initially felt I had been given another shot at Life (definitely with a capital "L") and wanted to reflect this New Age even with my computer. Secondly, "impact" sounded too much like a crash. Thirdly, I think my view from the TI sidelines for so long gave me the distinct perspective of seeing a New Age arrive for us. And, finally, the sharing and caring that was shown to me during my repair time by 99ers worldwide made me realize that my commitment wasn't to the little computer (that kept me occupied in thousands of wee hours) but to the people.

Even now, so long after The Event (always capitalized in my family), I continue to get cards and letters and disks and TI-related thingies from all over the world. This past week I got cards from England, Belgium, and Australia. And a few more from this country and Canada. To all those well-wishers everywhere who were so supportive during my long recovery, I can only say, "Thank you." You were all very instrumental in my rapid comeback. Even the doctors were amazed.

Regarding my operations, the most oft-asked question was, "Will all those metal plates in your head set off alarms in airports?" I have no idea. I haven't flown since. I had the stainless steel one and one of the titanium ones removed in a recent operation, but I still have seven left. Permanently.

And I am back to work on a slightly limited basis. And I DO drive. A new car. My other two-month-old new car was totally demolished when I hit the tree. And - YES! - I was VERY scared the first time I got back behind the wheel once I could see okay again. It had been so long. I'm still cautious, but I'm no longer frightened.

I had also been drifting toward large blimphood, but I lost lots of weight after the accident and have been able to keep it off. Crash diet. So there's another positive.

I really have a thousand TI people to thank, but I must particularly single out Charlie Good, John Willforth, Chris Bobbitt, Timothy Dermody, Tony McGovern, Jim Peterson, Jim Cox, Sister Pat Taylor, and so many patient newsletter editors for support above and beyond the call of duty. Having inadequately given such little thanks when so much more was due, I'll now move right into N-A #1.

Each month I plan to explore in these two pages some TI fairware and some commercialware and a bit of the goings-on.

The first going-on is Bill Gaskell's. I am sorry to learn of his FOUR-A/TALK swan song. For the year he wrote it, it was the most interesting column around. The variety and the history and the enthusiasm and the good writing made FOUR-A/TALK the boost we all needed. It was a very sustaining column that each month also introduced us to all the new goodies available to us (and there are many). The TI World Community will sorely miss your writings, Bill. Hurry up with the

**skiing and golfing and get back to your computer.**

In Bill's last article he talked about John Johnson's most recent BOOT program as of October 1989, good for Hard Drives, etc. For those with the "normal" system of a drive or two but not the techie wizardry, you can still benefit from the BOOT. It may not be the ONLY way to go, but it sure is one of the very best. Someone sent it to me during the time I wasn't able to compute, but when I got back to the keyboard recently I popped it in and had my flabber gasted. First it says, "MUG BOOT LOADING." That's the Miami Users Group (6755 Tamiami Canal Road, Miami, FL 33126).

Then comes the menu: 1 SHOW DIRECTORY; 2 DISPLAY A FILE; 3 RUN A PROGRAM; 4-9 Options 1-6; C TI XB.

#1 shows a directory of any disk (including RAMs) and permits marking of files for viewing, running, deleting. Marking auto-sets files when you return to Main Menu. A text file appears when #2 (DISPLAY) is pressed. An XB or E/A file runs automatically if #3 (RUN) is pressed.

When you press the spacebar another menu (with Options 7-15 and TI XB) is displayed. Press again and a third menu (with 16-24 and TI XB) is shown. This means you can type in anything on the options to VIEW or RUN including LOAD itself, which is handy if you have a bunch of LOAD programs (such as FUNNELWEB) that you want to operate off ANY drive. If you have a DSDD and two drives, for example, you could load onto this MUG BOOT disk all your favorite programs AND another whole disk of favorites on Drive 2. (And a zillion more.) Then you can put your top 24 on this menu and your other top 20 onto your FUNNELWEB menus and on and on.

With this program you could build the perfect environment for yourself, but that's only scratching the surface of this February '89 version I have.

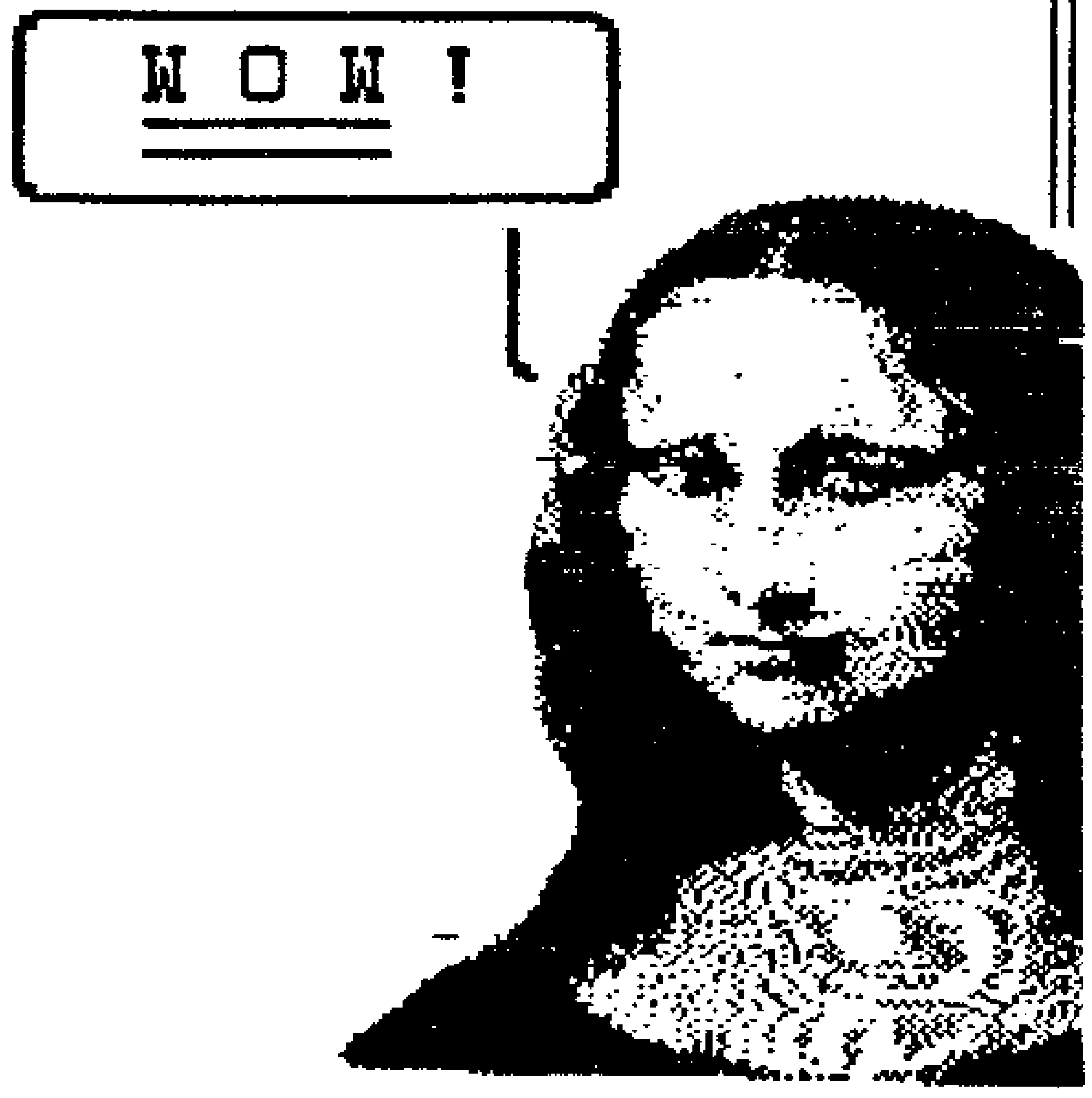
Here are just a few of the single keypress things this will do for your minimum XB disk system: turn screen off; load disk directory; print directory; view text file; print text file; run XB or EA program; EASILY configure up to 24 (actually, unending number) files for autoload; run gram/grom modules; delete files; cycle through and mark groms; toggle XB color interrupt routines off and on; do a CALL routine; change print device; get and display ROM header at >6000 (whatever that means); configure BOOT tracking; display version # and author; toggle instantly between all options; save all configurations; use additional active keypresses in sub menus; and so on.

Could you have dreamed that something so wonderful and so simple to operate could ever exist for our TI? Get the latest version from your user-group library or write to Miami. Don't forget a decent fairware donation.

In his final column Bill Gaskell was surprised by the ingenious FUNNELWEB built-in to change upper to lower case by holding CONTROL/period and running the cursor over the characters (words, sentences, etc.) to be speedily converted. I don't think he realized that Tony McGovern also did a reverse - lower to upper - by holding CONTROL/semi-colon. Tony told me he added this because he was not a good touch typist and found retyping from one case to the other too time consuming. The day after I got this version (4.0 and up), I was typing along without looking at the screen. When I looked up I had 20 lines or so of upper case. Zip. With the cursor and Tony's ingenious keypress changer I was back to typing in seconds. Try it the next time you use FWB (speaking of which, have you tried the new 4.2? It's got some great LARGE changes.) NEW-AGE/99 will detail it all soon.

**[If you use NEW-AGE/99 please put me on your exchange list.]**

# PLUS! v. 2.0



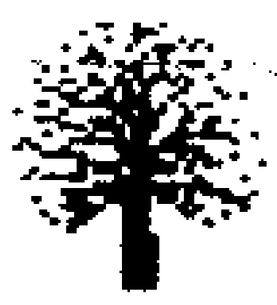
THIS DYNAMIC PACKAGE OF UTILITIES AND WORD-PROCESSING AIDS CREATES ONE OF THE MOST EXCITING ENVIRONMENTS IN THE WORLDS OF TI/99 AND THE GENEVE.

FEW THINGS IN THIS

ARE GOING TO MAKE YOU AS HAPPY AS THIS NEW [AND FINAL!] VERSION OF THE POPULAR PLUS! SERIES, ONE OF THE MOST USED SOFTWARE PACKAGES IN OUR TI WORLD COMMUNITY.

[ORDER PLUS! TODAY & BRING A LITTLE FUN & LOTS OF INGENIOUS PROGRAMS, TUTORIALS & TEMPLATES INTO YOUR LIFE.]

JACK SUGHRUE
BOX 459
E DOUGLAS MA
01516



ALMOST 1400 SECTORS OF PURE DYNAMITE WILL KEEP YOU AWAKE FOR LOTS OF FUN-FILLED NIGHTS! [THE DOCS, FOR EXAMPLE, PRINT OUT TO A 51-PAGE MANUAL/TUTORIAL!]

**PLUS! 2.0** STILL HAS THE TINY PROGRAMS AND TEMPLATES THAT GIVE YOU THE FEELING YOU HAVE A RAMDISK FOR MANY APPLICATIONS EVEN IF YOU HAVE THE MINIMUM TI WRITER [OR FUNNELWEB] AND/OR XB CONFIGURATIONS.

ALL OF THE ORIGINAL PLUS! FILES [PROGRAMS, TEMPLATES, TUTORIALS, MANUAL] ARE NOW UPDATED, IMPROVED, OR ENTIRELY REPLACED. IN ADDITION, NUMEROUS NEW FILES ARE ADDED TO MAKE THIS THE MOST COMPLETE, MOST POWERFUL ENVIRONMENT OF ITS KIND AVAILABLE ANYWHERE AT ANY PRICE!

NOW,
THE GOOD NEWS

PLUS! IS ALMOST TWICE THE SIZE AT THE SAME LOW FAIRWARE PRICE OF $10.
[GROUPWARE PRICE $90 PER USER GROUP CAN END UP COSTING INDIVIDUALS ONLY A DOLLAR OR TWO WITH FULL REGISTRATION.]

-SPECIFY DSSD OR SSSD-

## COMPUTERS and MATH

LOTS of times I hear people talking about "computers" and how GREAT they are for 'mathematics', or sometimes how they need a "BIG" or "FAST" system because they "do a lot of math". This really demonstrates how little understanding we possess of the actual math calculations and computer internal operations involved. Most people's ideas of how well the computer handles 'Math' is based on the fact that the computer
1) can handle calculations repeatedly without having to re-enter each number or value from a keyboard each time (thus reducing our naturally human propensity for making errors); and
2) does not (usually) make logical or mathematical errors in its calculations--It maintains a large number of "significant" digits in each "number" being manipulated, so we consider its results to be "accurate".

So our checkbooks have the 'right' numbers in them (whether they have any money or not!) and our accounts close-out properly and our spreadsheets add up and are easily updated and changed, and it all is 'easy' or at least a lot easier than by 'calculator', or by 'hand'--the way those of us over 35 learned to do it in school in the years known as the "PC" age--"Pre-Calculator".

HOWEVER, we should understand that virtually EVERY calculation which can be done on our computers CAN BE DONE ON A HAND CALCULATOR, or (shudder, dread) with pencil and paper, or even in the dirt with a pointed stick. We just ~~have to 'enter'~~ the numbers and carry through the basic math procedures, step-by-step, for each time we repeat the process with new 'data'. These procedures are based more on the characteristics of our 'numbering systems' than they are on any particular mathematical philosophy. In fact, there is an entire branch of 'programmers' dealing with the generation of algorithms or procedures for calculating complex mathematical functions, using the available computer's 'simple' abilities.

WHAT IS SO GREAT?
What we are actually SEEING when we see the computers quick and accurate math calculations, is the product of clever (or not so clever) software which manipulates the computer into giving us the finished number values, quickly and accurately. In fact, when most people say that the computer is "real good for math", what they really mean is that the programs make it "real easy to get the computer to do our desired calculations for us", the same way every time. It is still up to us, as 'Users', to make sure we don't put in garbage, either in DATA form or in the LOGIC as far as our requested sequences are concerned.

WE also must differentiate between the computers 'power' at math in terms of speed, capacity, accuracy, and complexity. Yes, (All) computers are a lot FASTER than our fingers, and don't lose their concentration like we humans tend to do. But having a 25 Mhtz computer balance a checkbook, instead of a 4 Mhtz system, is much more a matter of ego of the owner than it is a computing necessity! The biggest speed increase by using computers is in the form of not having to re-enter the same sets of numbers over and over, since data entry is the part of most 'computing' processes which is the slowest, anyway, and is fairly independent of the 'speed' of the computer.

As for CAPACITY, if a user actually has a need for either the program or data space for a large spreadsheet or set of accounts, then he might need 512k or several megs of memory, or (in the case of a large bank system) several hundred megabytes of memory on line. But most people who buy computers over-buy substantially, in the mistaken belief that more computer automatically will do the job better. Ever tried to take the square root of single number with your computer? My $4.95 calculator will have the answer before your computer gets booted up, BUT, if you need a calculation series of 4000 square roots done, almost any computer can handle it better--my fingers couldn't stand that anyway!

ACCURACY? Well, maybe, and maybe not! Several comparisons of computers vs. hand-held calculators have shown that computers often are far inferior on actual numeric accuracy, even when comparing fairly expensive computers to fairly cheap calculators. Reason for this is that in calculators, the chips and routines are designed for numeric calculations, rather than the results being generated via routines in the software. (The 4A's number-handling is fairly good, and compares favorably with even some of the newest 16-bit computers and calculators.)

Besides, if you are willing to carry it out far enough, you can, on paper, beat even the 32-bit computers in significant digits of division accuracy. We must keep in mind the contrast between ~~accuracy~~ and precision; accuracy is the correctness of the result, while precision is the degree of preciseness or detail which is displayed or indicated. An incorrectly calculated result of "1.418635843" for the square root of two is 'accurate' only to three significant digits, but indicates an apparent 'precision' of 10 digits. On the other hand, a display of "1.414213562" indicates a 'precision' of 10 digits, but MAY be accurate to 14 digits, as it is in the 4A.

COMPLEXITY of calculations is probably the most over-rated reason for buying a (larger, faster) computer. People believe that because they buy a 'large' computer they can do the 'complex' calculations they need--like percentages and interest rates! The only people who need computers which can handle truly complex math are NASA, on orbital dynamics, and those who work in pure mathmatics or designs work, etc. We too often confuse "complex" with "large-scale," which is often just simple calculations applied to many, many data values, as in spreadsheet recalculations. The complexity of the calculation functions really does not usually require more than the small scratchpad memory in a computer, very similar in size to that of advanced hand calculators--256 bytes or less, usually.

STRUCTURAL DETAILS
Some of the mathmatical software ability is built into our common computers in the 'operating systems', and some of the functional ease is a result of the 'Program', but the truth is this: a 'computer' is unable to do anything with any "numbers" or values EXCEPT add (and subtract, sometimes) or shift these values, and ALL of the values in the common computers are held in binary form, usually 8 bits / byte.

These 'byte' values are used as representational components of what WE consider to be 'numbers' or values. Other than in the special dedicated 'math co-processor' chips, ALL MATH FUNCTIONS are achieved by repeatedly acting on these 'components' of the 'number' in a prescribed (programmed) set of sequences of very simple operations, carried out at a high rate. A computer is then, just a super fast counting machine which (like a Martian) has 8-fingers on each hand, and can have one, two, or even four hands: 8-bit, 16-bit, or 32-bits.

Math chips are considered a big deal in the 'clones' now. The reason is that they more directly carry out the Numeric Functions activity on number values, without direct *and re-iterative procedures by the CPU, and therefore do* 'higher level' functions substantially faster. So, IF your Program usage involves a large number of higher level calculations, as compared to the time spent on 'other stuff', like displays, etc, then this could make a substantial difference, but IF a Program does not USE these types of calculations, then the Math chip sits idle and nothing goes any faster anyway!!!

NUMERIC THEORY:

NUMBERS may exist in several forms: mental, written, imaginary (known NOT to exist but useful mathmatically), fractional (another mental picture of something not really there), decimal, integer, etc.... For our purposes we usually consider "numbers" to be those values which are representative of REAL or possible conditions (like the balance in our checkbook, or the number of correct answers, or the value of 10 divided by 11 ). For these purposes, we either count items for an Integer Value, or we consider a number to have a possible 'parts-of-a-whole' Decimal Value, but in any case, the 'number' is a 'REPRESENTATION' of the concept of a "REAL" number value or 'size'.

Some computer languages or operating systems allow or require declaration of each variable as to type, and allow other number types to be represented and handled, with particular 'rules' as to HOW they are used: Fortran allows 'Imaginary' numbers with a component representing the amount of 'square root of -1' value. Fortran and Pascal, I believe, allow 'double precision' values to be handled, as do some of the IBM Basics(?), for more significant digits. Many of the 8-bit systems only carry 8 or 9 significant digits in their normal mode and therefore NEED a 'Double-Precision' mode.

NUMBER SYSTEMS

HOW very convenient it is, that we have ten fingers, so we can count to ten and then start over, creating groups of tens, and groups of fifties, hundreds, and groups of five hundreds and thousands, each of which can be represented by a LETTER or NUMERAL, a classic Roman concept. Pretty hard to multiply, however! Note that the Number Value is represented by a 'Numeral' or SYMBOL--a Number is a concept of value and the symbol IS NOT 'the number' even if we call it by such a name. The Romans had a couple of problems in their system...lack of enough suitable symbols (the symbol for 'three' is just three of the 'one' symbols together!) AND lack of a symbol for the value representing 'Nothing' (zero) for instance.

What could be better? Well, since the Arabians or Iranians (depending on who you talk to) invented the concept of 'zero', and a set of distinct digits for each value from 'one' to 'nine', we can have groups of tens, and tens of tens, or POWERS Of tens, which is a whole different way of representing the same quantity or counted values. This element of 'place value' of the digits plus the addition of a digit NOT representing a 'count value' (how many items do you count to get to the digit "0" ??) is what makes our mathematical procedures workable, regardless of the base of the system.

Now add a decimal to divide between the 'whole number' values and the 'parts of one' values, and the concept of 'equal but opposite' negative values, and you have what WE consider to be a workable numbering (counting) system. These characteristics of our (decimal) numeric system ALSO allow us to represent 'non-count' and even theoretical values, even those we can't really SEE, like "99.44 percent", and also to easily(?) manipulate these values to represent other related conditions, like PI and the third side of right triangles.

BASES, OTHER THAN FIRST AND THIRD:

Supposedly, if we had sixteen fingers we would count in Hexadecimal groups and the multiplication tables would be tremendous! In such a system, there could be several more 'numbers', such as:
'A'= 11111 11111,
'B'= 11111 11111 1,
'C'= 11111 11111 11,
'D'= 11111 11111 111,
'E'= 11111 11111 1111, and
'F'= 11111 11111 11111
and the value written as "10" would represent 11111 11111 11111 1 items; that is: 0 in the ones column, and 1 in the sixteens column. Two dozen would be (is) written ">18" or "one times sixteen plus 8"; decimal value 255 would be (is) written ">FF" or "fifteen times sixteen plus fifteen times one", and decimal 256 would be (is) written ">100" or "one times sixteen-times-sixteen plus zero times sixteen plus zero times one".

For any given number system, the 'base' or 'radix' of the system is the number at which we 'carry' into the next higher 'place' column. It is also the number of different digits, including zero, used in that system, and is one larger in value than the largest single digit used in that system. Each added 'place' or column is equal to a MULTIPLIER equal to THE BASE to the POWER of that number of columns LESS ONE--the 'place value' of the digits in that particular column.

GOT IT?? In our number systems, the *COLUMNS* have 'place values', and represent the 'powers' of the 'base' number, increasing from right to left: Decimal system = base 10, so--
-- the first column equals a multiplier of "10 to the zero" or 'ones' (ANY NUMBER to the ZERO POWER equals 'ONE');
-- the second column (from right to left) equals "10 to the first power", or 'tens' (ANY NUMBER to the FIRST POWER equals ITSELF);
-- the third column equals "10 to the second power", or 'hundreds';
-- the fourth column equals "10 to the third power", or 'thousands';
-- etc.....

-- the largest 'digit' or numeral symbol is one less than the base, or 10-1=nine, written '9'; and the number of numerals used is ten, including the all-important symbol '0'.
-- if a decimal point is used, all numerals to the right of it are of increasing fractional or 'negative powers of ten': "10 to the -1 power" equals 'one tenth'; second column right equals "10 to the -2" or 'one-hundredths'; etc.... IT IS possible and correct to use decimals with Base 2 and Base 16 numbers, with the same rules, but I sure don't want to try to divide two HEX-Base numbers, especially if fractional parts are involved!!!!

HEXadecimal (Radix 16) numbers used as 'byte' or 'word' values in a 'computer' situation have NO INDICATED DECIMAL and no sign prefix--they have a logical value defined by one numeral per 'Nibble', or two numerals per 8-bit 'Byte', or four Hex numerals per two-byte 'Word':

```
     >5CB3 = 5 times 16 to the third power, (= 4096);
           + C (decimal 12) times 16x16 (=256) ;
           + B (decimal 11) times 16;
           + 3 times 16 to the zero (ones); or:
   >5CB3 = 5x4096 + 12x256 + 11x16 + 3x1 = 23731 (decimal).
```

NOMENCLATURE NOTE: in order to distinquish just what BASE any given 'number' Symbol Group is refering to, we either subscript the base value, OR use a symbol before the 'number', such as:
BASE 2 (Binary) numbers are written-- &11001101, b1101 or 11010111b, or with a subscript of '2';
BASE 10 (Decimal) numbers are written-- +xxxxx or -xxxxx.xxx or without prefix;
BASE 16 (Hexadecimal) numbers are written-- >56A7 (for two bytes) or >3E for 8-bits (one byte) or >3AF7-E462 (for a 4-byte or 32-bit value or 'word').

BINARY SYSTEMS:

COMPUTERS could have been designed with circuits with ten (or sixteen) separate possible conditions or 'states' in each circuit so that each circuit could represent one DIGIT of the number or value, but the cost would have been MUCH more than ten times as much! It is pretty easy to make a dependable 'two-state' or 'on-or-off' circuit, representing "0" and "1", and then hook up several of these in a group to act as a single 'value' in memory, to represent larger values. Since only two conditions exist for each 'counting point' or 'bit' of information, it is called a 'binary' system, and operates on a 'powers of two' base. Zero in a column equals zero of that column value, and one in a column equals exactly that 'power of two' value, starting at the rightmost column as '2 to the zero power' or 'ones'.

Therefore the BIT with the lowest value (ones column) is USUALLY WRITTEN at the RIGHT SIDE of the group of bits, and is called the "Least Significant Bit", or 'LSB', and the bit representing the highest place value is written at the LEFT side of the 'Byte', and is called the 'MSB' for "Most Significant Bit". A further note is that in most 'computer' internal counting functions, the FIRST ITEM in a group is LABELED as being item "0000", the SECOND Item is labeled as "0001", etc. As far as a binary device is concerned, the 'number' "0000" is a perfectly good NUMBER LABEL in a counting sequence, and it is used as such, although when VALUES are related, '0000' is recognized as BEING 'zero'.

The usual size of a grouping of 'bits' in OUR computers is EIGHT bits in a 'BYTE'. The original processors actually operated with 4 bits per byte, and later on a 8-bit basis, then with 16-bits and 32-bits per 'WORD', but so much of the hardware standards have been limited to 8 bits that a 'byte' is still considered to be 8-bits in size, especially in Input-Output contexts. For instance, all modems, keyboards, and even the 9918 Video Chip recognize and use data in 8-bit blocks or Bytes.

One other grouping is of note: a 'Nibble' is a group of 4-bits, usually the first four OR the last four in a byte (not the four in the middle of the byte), and these would be the 'low nibble' and the 'high nibble', accordingly. The advantage to thinking in terms of 'Nibbles' is this: One HEX DIGIT can represent ONE NIBBLE for all values contained in that Four-Bit Nibble, thus making conversions between BINARY and HEX values much easier.

The number values in such a binary system are represented by all the possible sums of the combinations of 'bits', or powers of two-- i.e.:

```
   &10110011 equals (from right to left)
           |-one times   1  =     1
          |--one times   2  =     2
         |---zero times  4  =     0
        |----zero times  8  =     0
       |-----one times  16  =    16
      |------one times  32  =    32
     |-------zero times 64  =    00
    |--------one times 128  =  _128_
           Value (= Sum) =      179
```

The value of all eight bits together ( &11111111 ) happens to be decimal 255, or the same as HEXadecimal >FF, which is why HEX Double Digits is used to REPRESENT values of 8-bit bytes, such as those on disks and in memory locations, and in Assembly Language code, both as byte values and as address locations.

DON'T let all this cause you to believe that these strange numbers actually EXIST in the computer--There is NO SUCH THING as HEX 'numbers' in a computer, nor even Binary nor Decimal. There are only BIT CONDITIONS which represent values between 000 and 255 in each byte, or in a 16-bit machine values from 00000 to 65535 ( two bytes = a "word") can be considered at a time. SO the largest single value which the 16-bit processor can understand or deal with is 65535, or "2 to the 16th power, less one", and it only 'sees' this as a group of 16 binary bits. All that each of these number systems and number symbols ARE is a way of representing these values.

Now for those a little more daring, I will admit that in some cases, the left-most bit (the "128" bit) of the byte is used as a 'sign bit' so that those bytes can contain arithmetic values from 'minus 128 to plus 127', by means of a notation system called "Two's Complement" numbers. SO...I suppose REALLY the range of values of a 8-bit byte are from "-128 to +255" although not all at the same time--the computer HAS TO KNOW whether it is looking at a value to be interpreted as a signed number value or not! And if you consider a "two byte 'word'" as a signed value location, the values which it can contain are EITHER from -32768 to +32767, signed value, OR Integer values from +00000 to +65535. Since the processor treats Binary Bit Values as Integers for their LOGICAL values, the condition of any particular 'word' value being a "signed number" is defined by the program or operating systems or usage.

THE BINARY 4A:

In OUR 16-bit systems, WORD (two bytes) VALUES, in general terminology, and ADDRESSES in particular, can have Logical Values from 00000 to 32767 (decimal value) (>0000 to >7FFF) with equal Arithmetic Values, while the Logical Values from >8000 to >FFFF are considered Negative or "Twos Complement" Arithmetic Values. The FIRST of the two bytes of a 'word' is always the EVEN-NUMBERED byte address, therefore 'WORDS' are always located at EVEN-NUMBER addresses in Memory (addresses which end in a final 'bit' of "0" are "even").

The FIRST byte of a 'word' is also considered to be the "MSBy" or "Most Significant Byte", and its Binary Place-Values are the highest, and the next byte of the pair is the "LSBy" or "Least Significant Byte", of lesser Arithmetic value. The Value of the 'word' is the value of the MSByte times 256, plus the value of the LSByte.

The Two's Complement of a Binary Value is the opposite SIGN but equal absolute VALUE, and is created by inverting all 1's to 0's, and 0's to 1's, then adding 'one'. In DECIMAL form, Two's Complements are formed by SUBTRACTING 65536 from the Value, IF the Logical Value is greater than 32767. The Two's Complement of a Logical 8-bit Value (one byte) is formed by subtracting '256' from the Logical Value greater than 127. This is the way BASIC (in the 4A) keeps up with Arithmetic Negative values and therefore this is why certain addresses in the 4A are NEGATIVE in the CALL LOADs and CALL PEEKs--these lie above >7FFF or +32767 in 'Word Bit-Value'. This is also why there can only be LINE NUMBERS up to 32767--BASIC thinks the 'Word' values above that are Negative, and BASIC dis-allows 'Negative Line Numbers'.
There is a good explanation of a lot of this math madness in Ira McComic's Assembly Language book, pp.12-28.

For each BINARY Word Value, there is a HEX-Notation and Decimal-Notation form to represent that value, and the sequence is:

| | Logical | | | Arithmetic |
|---|---|---|---|---|
| Hex | MSBy-Binary-LSBy | | Decimal | Decimal |
| >0000 | &00000000 | 00000000 | 00000 | +00000 |
| >0001 | &00000000 | 00000001 | 00001 | +00001 |
| >0002 | &00000000 | 00000010 | 00002 | +00002 |
| >0003 | &00000000 | 00000011 | 00003 | +00003 |
| >0004 | &00000000 | 00000100 | 00004 | +00004 |
| >0005 | &00000000 | 00000101 | 00005 | +00005 |
| >0006 | &00000000 | 00000110 | 00006 | +00006 |
| >0007 | &00000000 | 00000111 | 00007 | +00007 |
| >0008 | &00000000 | 00001000 | 00008 | +00008 |
| >0009 | &00000000 | 00001001 | 00009 | +00009 |
| >000A | &00000000 | 00001010 | 00010 | +00010 |
| >000B | &00000000 | 00001011 | 00011 | +00011 |
| >000C | &00000000 | 00001100 | 00012 | +00012 |
| >000D | &00000000 | 00001101 | 00013 | +00013 |
| >000E | &00000000 | 00001110 | 00014 | +00014 |
| >000F | &00000000 | 00001111 | 00015 | +00015 |
| carry "1" to second HEX digit and continue | | | | |
| >0010 | &00000000 | 00010000 | 00016 | +00016 |
| >0011 | &00000000 | 00010001 | 00017 | +00017 |
| >0012 | &00000000 | 00010010 | 00018 | +00018 |
| ....... etc: | | | | |
| approaching >8000 or the "arithmetic negatives": | | | | |
| >7FFD | &01111111 | 11111101 | 32765 | +32765 |
| >7FFE | &01111111 | 11111110 | 32766 | +32766 |
| >7FFF | &01111111 | 11111111 | 32767 | +32767 |
| >8000 | &00000000 | 00000000 | 32768 | -32768 |
| >8001 | &10000000 | 00000001 | 32769 | -32767 |
| >8002 | &10000000 | 00000010 | 32770 | -32766 |
| >8003 | &10000000 | 00000011 | 32771 | -32765 |
| ....... etc: | | | | |
| approaching the limits of 16-bit integers: | | | | |
| >FFF9 | &11111111 | 11111001 | 65529 | -00007 |
| >FFFA | &11111111 | 11111010 | 65530 | -00006 |
| >FFFB | &11111111 | 11111011 | 65531 | -00005 |
| >FFFC | &11111111 | 11111100 | 65532 | -00004 |
| >FFFD | &11111111 | 11111101 | 65533 | -00003 |
| >FFFE | &11111111 | 11111110 | 65534 | -00002 |
| >FFFF | &11111111 | 11111111 | 65535 | -00001 |

cont.>>>

BASICS - MATH // r.lumpkin
Houston Users Group - May 89 -

From the chart above, it is easy to see WHY it is preferable to use HEX notation for 16-bit Word Values--just too many digits otherwise! The conversion process between HEX and BINARY is also VERY EASY, since it is done on a "4-BITS = ONE HEX DIGIT" basis:
--the HEX values from >---0 to >---F correspond to the 'Low Nibble' values from &0000 to &1111 (00 to 15 decimal).
--the NEXT HEX DIGIT over ( >--X- ) corresponds to the SAME series of Binary Bit Values for the NEXT nibble (the left 'nibble' of the Right-hand Byte). The two nibbles of the MSByte also correspond to the two LEFT Hex Digits:

| | | | |
|---|---|---|---|
| >---X | = | &00000000 | 0000xxxx |
| >--X- | = | &00000000 | xxxx0000 |
| >-X-- | = | &0000xxxx | 00000000 |
| >X--- | = | &xxxx0000 | 00000000 |

SO, the conversion is done 4 Binary Bits at a time, to one Hex digit, for each 'nibble', or in reverse.

SCIENTIFIC or EXPONENTIAL NOTATION:

One other method of representing number values needs to be explained. For 'large' or 'very small' numbers (those which otherwise would contain many zeroes) and for which absolute "count" accuracy is not needed, a system using a 'base to a power' is often used. A (usually base 10) group of 'significant digits' or Mantissa (which may be 'signed' as positive or negative) is followed by "E" and then a number representing the 'power of ten' or Exponent corresponding to the magnitude of the value of the first numeral of the mantissa. A decimal ALWAYS follows this first numeral, therefore, the number form is always:
sign; single digit; decimal; fractional part of significant digits; "E"; power-of-ten multiplier indicating the magnitude of the single digit value.

A Positive Exponent indicates a Value is being represented which is LARGER than ONE, and Negative Exponents indicate the Value is SMALLER than ONE--
i.e. +4.4326789 E 14 could be written as "443267800000000." and -1.62E-23 could be written as "-0.0000000000000000000000162". Please NOTE that NEGATIVE numbers are NOT (necessarily) 'small' numbers--they can be quite large, such as the net worth of our national debt: $-1.6E12 or -$1,600,000,000,000.00 which is not the same as $1.6E-12 or $+0.0000000000016 or approximately the interest earned on one penny in 1/100,000 second.

Exponential notation is not usually used where absolute accuracy is needed, such as in book-keeping operations, since the number of significant digits is usually not within acceptable limits of required accuracy; if you need to know the national debt to the exact penny, there is no use in using exponential notation. In fact, in the 99/4A, number values represented in "Scientific" or Exponential form only carry 6 significant digits IN THE DISPLAY, although the actual value in memory is held at 13 or 14 digits accuracy.

NUMERIC VALUES AND ACCURACY IN THE 4A:

Our little 4A's only allow one Number Form for NUMERIC VARIABLES in (X)Basic: Floating Point. These 'number values' which are associated with named VARIABLES ('VALUE', 'X', 'N', etc) always occupy 8 bytes IN MEMORY, with one byte of 'decimal point location' info, AND 7 bytes of 'Radix 100' components, very similar to the Exponential Form, but using a 'BASE' of '100'. This is NOT the method of storage of LINE NUMBERS (two bytes as a signed 'Word') nor the method of holding NUMERIC CONSTANTS in Program Lines (stated numbers like " = 123456789.98765432 " in Program Lines in memory or on disks are held as ASCII CHARACTERS).

This Radix-100 form of number storage always uses eight bytes to store the value of a VARIABLE used in Basic or Extended Basic, even if the value is only '1' or '16' or '1001', or if the value is '832,579,452,631,444.7' . The 4A holds 13 OR 14 significant digits in the second thru eighth bytes (each equals HEX >00 to >63 for Decimal 00 to 99) plus a sign bit in the second byte if the value represented is negative, while the first byte holds the 'multiplier' or Base-10 exponent which runs from 00 to +127 and from -128 to -001 (complemented form of binary byte). The decimal is always considered to follow the first significant digit.

In theory, you could have up to 500 calculational errors in the 14th place before the displayed 10th digit was effected, since BASIC rounds off its display, which is why you can take 1 and divide by 3, (which actually equals .33333333333333333333333333... ) and then multiply BY 3 (which equals .99999999999999999999999999...) but on the screen you will SEE " 1 ", which is a rounded-off display of the actual product stored in memory: ' 9.9999999999999 E-1 '. Since the TI maintains such accuracy, and since it suppresses the decimal display on 'near integer' values, it can and does use these FLOATING POINT values as if they were integers, both in math operations and in Statement (directive) arguments.

All numeric calculations are done at 13 or 14 digits accuracy, and displayed with 10 significant digits, rounded off, except if the Exponential form must be used because of the 'size' (too large or small for 10 digits, due to zeroes), in which case only 6 significant digits will be DISPLAYED, although all 14 will be maintained in memory for calculations purposes. This means that the 4A runs all numeric values at close to what is considered "Double Precision" in 8-Bit machines.

BASICS-IV, MATH--r.lumpkin, pg 9

Houston Users Group - May 89 -

There IS A WAY to display more than 10 digits of a number--use Extended Basic and "PRINT USING" as in "PRINT USING "###.###########":A " Which should print the current value of variable 'A' with more than 10 significant digits, IF it HAS more than that many, and IF it is not larger than 999.99999999999 or smaller than 0.10000000000000 . The Value to be displayed MUST MATCH the 'frame' created by the 'PRINT USING' in order for this to provide these extra displayed digits.
Page III-13 of the "Users Reference Guide" contains more detailed information on Radix-100 operations.

ASCII CODES:

Certain of the codes which are used in computers represent Characters which we intend to display or print, and a standardized set of Code Numbers was developed so that different computing systems would all "recognize" the same code for the same character. The most commonly used of these is the "ASCII" (ask-key) system, which includes codes from 000 to 127. This does not mean that other codes are not used-- the "IBM Extended Code Set" uses codes from 128 to 255 for other characters for printing purposes, and the C/64 does not "Print" with a true ASCII set of codes, which is why it requires special printers or interfaces. MOST systems also use these same codes internally to represent these same characters, in both "Literal" or character-strings situations, and also within part or all of the "Program" lines or steps, as the 4A does. Even in these cases, some or all of the codes may be shifted or "offset" by some numeric amount, such as by +96 (HEX >60) in the 4A video processor. Regardless of these uses, ASCII Coding IS THE STANDARD for transmitting information between computers via modems, etc, and are in common usage in other contexts also.

Keep in mind that 'Computers' can only deal with ONE THING, and that is the BIT VALUES in 'bytes' and 'words'. All instructions, all addresses, all data values, all character codes and names, all coding values for the display screen, EVERYTHING is in the form of byte values from 000 to 255, (or word values from 00000 to 65535). Virtually every bit (bad pun) of information within, going in or out of, or stored by a computer is in this form.

Of course, the actual "computer" (the CPU or processor) actually never knows what those binary bytes represent: it only carries out the activities as directed by the Program and the Operating System, or in our case, the GPL Interpreter also. The CPU itself doesn't know a decimal from a hiccup, and THE ONLY 'MATH' IT KNOWS HOW TO DO IS ADDITION, SHIFTING, and COMPARISON OF BITS. ALL higher level math activities are performed by reiterative procedures in the GPL and Basic Interpreters, or the Program, since the Processor uses Integer Math only. We are lucky in that the 4A DOES have a very good math system built into the GPL environment, and that BASIC allows easy structuring of fairly complicated calculations series.

I hope that all of this helps to put the concept of the NUMBERS used or found in a computer into perspective. Just remember, for any particular "number" encountered by the processor in a memory "byte", the processor has to know what the value is supposed to represent in order to know what action to take. The same "Numbers" or values are used for many different purposes, and can be written in several different forms.

---------------BASICS-IV---r.lumpkin, Houston---May 1989.///

FEBRUARY 13, 1990 HAPPY VALENTINE'S DAY!!

MUNCH OFFICERS AND NUMBERS (all in 508 area unless noted)

| | | |
|---|---|---|
| President | W.C. Wyman | 839-4134 |
| Vice President | Bruce Willard | 852/3250 |
| Secretary | Jim Cox | |
| Treasurer | Jim Cox | 869-2704 |
| Acting Editor | Jim Cox | |
| Adv.Prog. Chair | Dan Rogers | 248-5502 |
| Library | Al/Lisa Cecchini | |
| Disk Librarian | Lou Holmes | 617 965/3584 |
| Tape Librarian | Walter Nowak | 413 436/7675 |
| +++++++ | Jack Sughrue | 476/7630 |

JANUARY MEETING. The January meeting featured Jack Sughrue and a demo of his disk of the month ccalled "SPACE". Everyone liked the disk and we did not have enough to sell to everyone who wanted one. More copies will be available at the next meeting. Jack also showed the TIPS disks whicch are views of T.I. Print Shop. We will be selling the TIPS disks in a package of 10 SSSD disks for $10.00, this is a fairware program. Jack had a print-out of all of the pictures on the disks and they were impressive.

FEBRUARY MEETING. If Lou is present he will get a chance to demo his disk of the month #3 which we never got around to at the December meeting. All disks of the month will be available for purchase, they only cost $1.50.

BOYCOTT COMPUTER SHOPPER. I call on everyone in the TI community to boycott the Computer Shopper. Brian O'Brien talked to the new editor in New York and he reports that this man said that there will no longer be a TI Column in his magazine. This editor didn't seem to mind that his magazine will lose subscriptions. If you want to look up something in the magazine go to the library and read it for free.

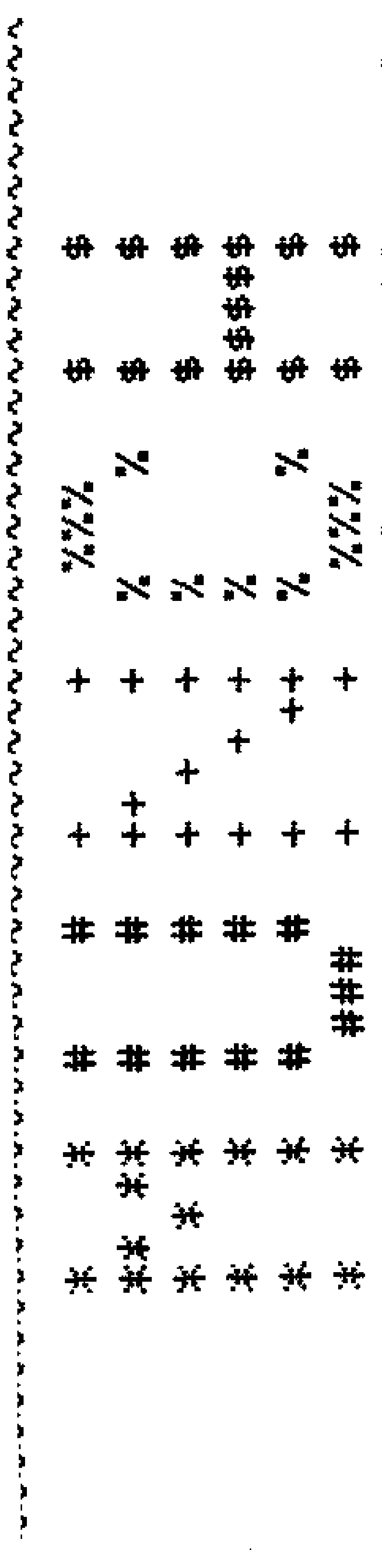
RAFFLE. Every month we have a raffle to help defer the cost of the monthly hall rental. The number of prizes awarded depends on thee number of tickets sold. This month we have a number of Norton Software games and utilities for prizes. If you have some old things you no longer usee how about some donations for the raffle.

MONTHLY SALES. At each meeting you have the opportunity to buy and/or sell new or used hardware, software, books and original programs. Please have prices marked on any items you have to sell.

LIBRARY NOTICE. Please return any items borrowed from our library. If you can not come to a meeting or give these items to someone who will be at the meting, please mail any library items to the group address which is listed on the cover of this newsletter. There are no late fees, we don't care how long they have been out, please return these items. Corson has donated a copy of Tony Lewis' boklet Interface Standard & Design Guide for the TI and periphals to thee library, and it will be available at the next meeting.

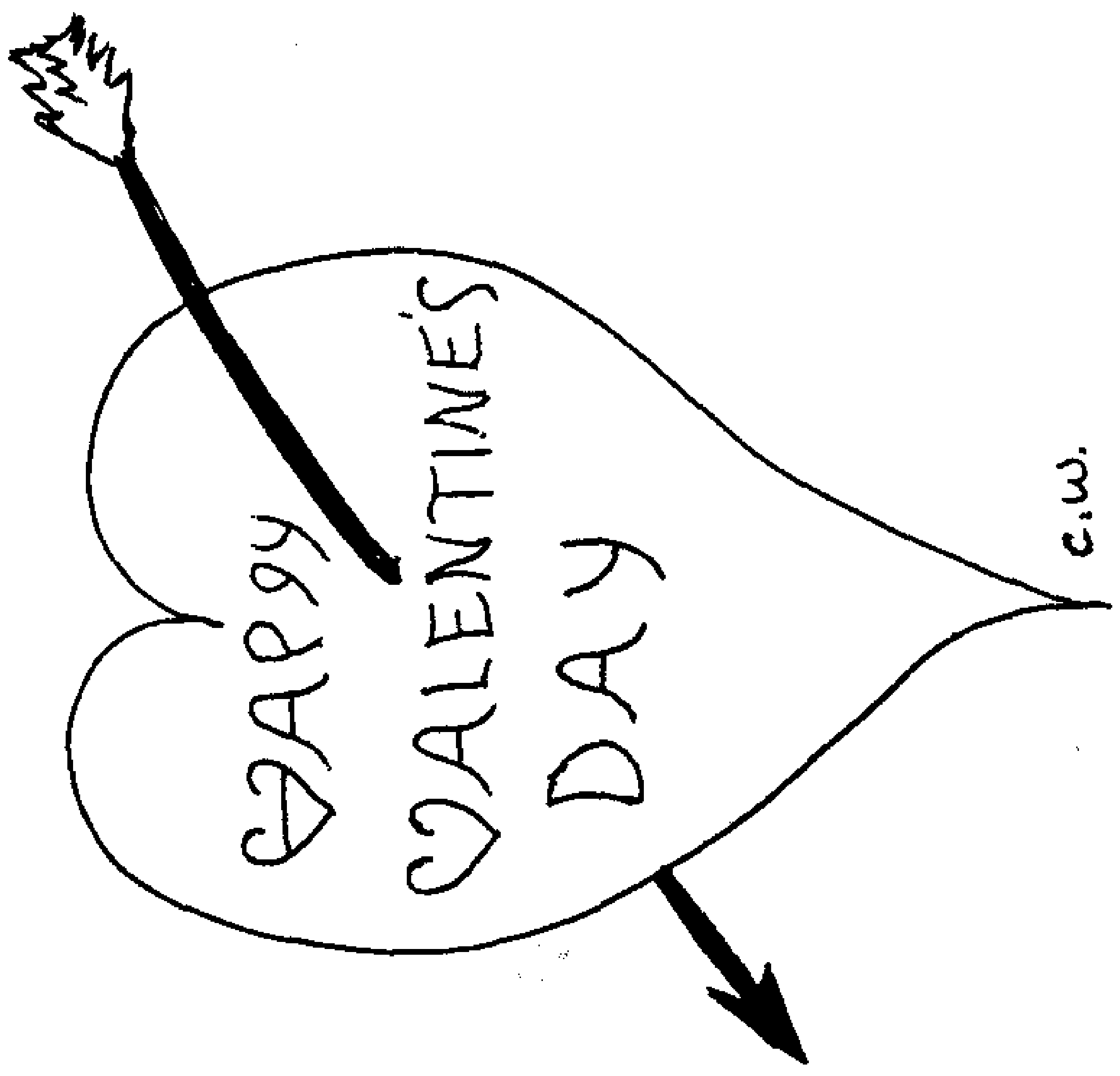
REPRINTS. Reprints are permitted as long as credit is given to M.U.N.C.H.

ARTICLES. I am always looking for articles for this newsletter, anything which interests you will probably interest other members of the TI community, so please share your ideas and opinions with all of us.



Mass Users of the Ninety-nine and Computer Hobbyis

FEBRUARY 1990 Monthly Newsletter Version 9.02



### JOIN THE CROWD AT OUR FEBRUARY 13th MEETING##

M.U.N.C.H.
560 LINCOLN ST.
P.O. BOX 7193
WORCESTER, MA. 01605-7193

WORCESTER, MA 016 PM 6 FEB 1990

USA 25

FIRST CLASS

!BOYCOTT COMPUTER SHOPPER!

Next Meeting FEBRUARY 13th.

POSTMASTER: Forwarding and Address Correction Requested.