ELF II NEWSLETTER
Netronics Research

MARCH, 1979 VOL. I NO. 2 PUBLISHED MONTHLY

The Great Micro Power Contest

by Tom Pittman

Most of you know by now that the February issue of Kilobaud has an article by me entitled "DOTS". In this article I explain how the character generator in Netronics Tiny Basic works. At the end of the article I put out a challenge: "The 1802 is the most powerful 8-bit microprocessor," I said, and it was worth $50 out of my pocket to prove it. Well, I was kind of dumb in laying out the rules for the prize.

Basically it was this: DOTs is a callable subroutine to display characters on the 1861, one character per call. I got it into some 800-odd bytes of code, with an average execution time around 1300 or so memory cycles (i.e. something less than 700 instructions). Multiply these together for a processor power rating of about 1.1 million byte-cycles, or "bykes" as I will call them here. The rules for winning the $50: beat that by 10%.

Well, somebody did. Before I even got my copy of the February Kilobaud, I received a program from someone in Chicago (he said he was stuck in a blizzard) which was written for a Z80 and came to 32% less bykes than mine. I paid him his $50, but not before I looked at his program veeerrrry carefully. You see, he changed the data structure of the program, so he did not prove the Z80 more powerful than the 1802. What he proved is that Mike Amling is a more clever programmer than Tom Pittman.

How do I know? I reworked my program along the same lines and beat his program by another 0.2%. OK, obviously the Z80 is as good as the 1802. But not 30% better.

The reason I mention all this is that all of you who are using the 1802 will be subjected to various forms of bigotry and prejudice, by people who will try to tell you that the 1802 is less powerful than Brand X. Don't you believe it. They just made the mistake of buying into a less powerful CPU and are trying to cover their tails.

When you get right down to it, the differences are slight. The average instruction length in the 1802 is about 1.3 bytes. In the 8080 it is closer to 1.9 bytes. In the 6800 I would guess 1.7 bytes in a typical mix of instructions. That makes the 1802 look a lot better, but you have to realize that because the instructions are smaller it takes some 30% more of them to do the same job, bringing the effective value to about 1.7. Notice that this is still very competitive.

Now look at execution time. But to be fair, let's not measure an 1802 with a 1.7MHz clock against a 6502 with a 4MHz clock. When it was first announced, the 1802 was (and still is) specified at 6.4MHz, for a memory cycle time of 1250ns. The 8080 was specified for a clock speed of 2MHz, giving a memory cycle time of 1500ns. The original Z80s were specified for the same speed, though very shortly they were announcing 4MHz versions. The original 6800 and 6502 spec called for a 1MHz clock and 1000ns memory cycle (these also have speeded up). The RCA people do not seem to understand what sells, so they have not bothered to make a speedup version of the 1802. More's the pity.

But let's consider the original releases of each of these. On the 1802 no instruction is more than three memory cycles, and most are exactly two cycles. That's 2.5 microseconds on a plain-vanilla 1802 running maximum clock. On the 8080 and Z80 there are some instructions that execute in four clock cycles (2us) but most of them require 5, 7, 8, or often 11 and 17 clock cycles to execute. The most common instructions (moves and conditional jumps) generally run four and five microseconds each. In the 6800 the figures are very similar: some instructions execute in only 2 memory cycles; most take three to five cycles, and one takes twelve.

The Z80 has some sexy instructions like block move and block I/O. A two-byte instruction can move any number of bytes at the rate of one byte every 21 clock cycles (10.5us in the plain-vanilla model). The 1802 can do the same thing in a 7-byte loop in 15us; if I get to choose where the data is coming from or going to, the loop is six bytes and only 12.5us per byte moved. The block I/O instructions take the same time in the Z80; in the 1802 I can code a four-byte loop to do the same thing in as little as 7.5 microseconds per byte of output.

Well, then, why isn't the 1802 more popular? "It's the addressing modes," say the armchair critics. "The 6502 has 13 addressing modes and the 1802 has hardly any." Six modes is not exactly "hardly any" (the 8080 has only five), But I can program any addressing mode I want into the 1802. The ELFBUG program uses a (programmed) relative addressing mode. I usually program a base-page addressing mode into most of my programs. You could write 1802 programs with 23 addressing modes if you wanted (and if you could think of a use for them); in the 6502 you are stuck with the 13 they give you.

"But the instruction set is so ridiculous," the critics splutter. I say, who is ridiculing whom? The 1802 can do a subroutine call in 2.5 microseconds; no other micro does it in less than five. The 1802 can receive an interrupt, process it, and be back in the main program in the time it might take the 6800 to notice it got an interrupt. Some people insist that the 6502 "is more like a minicomputer such as the PDP-11." Izzatso? Show me a 6502 program that can do arithmetic on the program counter. Only the PDP-11 and the 1802 can do that. I claim the 1802 also has many of the important features found only in large computers like the DEC-10 and the 360.

I have only begun to list the advantages of the 1802, but I think I had better quit before I get too excited. I just wanted you to know that you have nothing to be ashamed of. Now if only we could convince the folks at RCA to give the 1802 the support it deserves.

The Elusive 256

The basic ELF II comes with 256 bytes of RAM. This is enough to write a large variety of programs, but not quite enough to run bigger things like Tiny Basic. Enter 4K. For a very reasonable price, you can add 4K RAM in the bus connector. Tiny will run OK in 4K, but it does not leave you much room for programs. However, if you leave the 256 bytes in place, you get 4.3K, which is a lot more program space. The problem is, it did not work all the time. I fixed mine by changing two resistors on the memory board: Change R6 from 10K to 1.2K on every memory board; and add R5 (this was left out of my boards) 3.9K on one memory board only. Can you figure out why this works? One of these days I will have to do an article on hardware debugging. Promises, promises.

Multiple Timer Program

by J. H. Hansen

This program was written for use in the photographic darkroom where multiple times of different lengths are required.

Features of the program are:

1. As written the program provides for eight different times. This could be expanded to 18+ if all memory locations are used.
2. The memory locations for desired times are near the beginning of the program, making it very easy to change them.
3. The times are entered and displayed in decimal form, that is, directly in minutes and seconds. Therefore no knowledge of or conversion to hexadecimal is required.
4. Remaining minutes are displayed and when they are zero, remaining seconds are displayed. When remaining seconds are 15 seconds or less a warning beep is sounded each second. At End of Time a tone of different pitch and longer duration is sounded.
5. To start the timer it is only necessary to push the "1" button twice. When time is finished, the next time will start when the "1" button is again pushed twice. The series of times can be reset to the beginning by turning the Run switch off and on.
6. The maximum time limitations are 99 minutes and 99 seconds. If you are willing to work with hexadecimal numbers, the program could be made shorter and the time limits expanded considerably. [But who would want to? —TP]

[Editor's note: This program requires a speaker to produce the sounds. An audible sound can be produced by connecting a cheap speaker across the Q LED. A much louder sound may require an amplifier.]

```
Addr. Hex    Instruction        Comments
 00 F800       LDI 00          Initialize high order
 02 B4         PHI R4          registers 4,5,6,7.
 03 B5         PHI R5
 04 B6         PHI R6
 05 B7         PHI R7
 06 F826       LDI MAIN        Set memory location
 08 A4         PLO R4          for main program
 09 F8AF       LDI DECH        Set memory location for
 0B A5         PLO R5          Decimal to Hex subroutine
 0C F8C6       LDI HEXD        Set memory location for
 0E A6         PLO R6          Hex to Dec subroutine
 0F F8FF       LDI #FF         Set memory location for
 11 A7         PLO R7          decimal data storage
 12 D4         SEP R4          Call main program
 13 MM SS                      Enter times desired
 15 MM SS                      Minutes & seconds in Decimal
 17 MM SS                      max 99 Min. & 99 Sec.
 19 MM SS                      Eight different times
 1B MM SS                      may be entered
 1D MM SS
 1F MM SS
 21 MM SS
 23 F810   NEXT:LDI #10        Set tone for
 25 5D          STR RD         warning "beep"
 26 E0     MAIN:SEX R0
 27 3F27        BN4 *          Press Input button
 29 F0          LDX            Get minutes
 2A 64          OUT 4          Display minutes
 2B E7          SEX R7
 2C D5          SEP R5         Convert decimal to hex
 2D B3          PHI R3         Put minutes in Reg 3
 2E 372E        B4  *          Release button
 30 3F30        BN4 *          Push Button
 32 E0          SEX R0
 33 F0          LDX            Get seconds
 34 64          OUT 4          Display seconds
 35 E7          SEX R7
 36 D5          SEP R5         Convert decimal to hex
 37 A3          PLO R3         Put seconds in Reg 3.0
 38 3738        B4  *          release button
 3A 7B          SEQ            Turn "Q" LED on
          .. One minute loop. Repeat until Min=0
 3B 93     MIN: GHI R3         Get minutes
 3C D6          SEP R6         Convert hex to decimal
 3D 57          STR R7         Put min in loc.FF
 3E 64          OUT 4          Display minutes
 3F 27          DEC R7
 40 93          GHI R3         Get min.
 41 3259        BZ  SEC        If Min=0
 43 FF01        SMI 01
 45 B3          PHI R3
 46 F83C        LDI 60
 48 AE          PLO RE
 49 F88D        LDI (37261).0
 4B AF     ML:  PLO RF
 4C F892        LDI (37261).1
 4E BF          PHI RF
 4F 2F          DEC RF
 50 9F          GHI RF
 51 3A4F        BNZ *-2
 53 2E          DEC RE
 54 8E          GLO RE
 55 3A49        BNZ ML
 57 303B        BR  MIN
          .. 1 second loop. Repeat until sec.≤15
 59 83     SEC: GLO R3         GET SECONDS
 5A D6          SEP R6         Convert hex to dec.
 5B 57          STR R7         Put sec. in loc.FF
 5C 64          OUT 4          Display seconds
 5D 27          DEC R7
```

```
5E 83          GLO R3
5F 32A1        BZ  OUT    If Sec=0
61 FF0F        SMI 15     Subtract 15 from D (sec)
63 3B72        BNF BEEP   If 0 or less
65 23          DEC R3
66 F8A7        LDI (37287).0
68 AF          PLO RF
69 F892        LDI (37287).1
6B BF          PHI RF
6C 2F          DEC RF
6D 9F          GHI RF
6E 3A6C        BNZ *-2
70 3059        BR  SEC
               .. Part of Second timing loop with warning beep
               .. every second for last 15 seconds.
72 83    BEEP: GLO R3     Get sec.
73 D6          SEP R6     convert to sec.
74 57          STR R7     Store in memory loc.FF
75 64          OUT 4      Display sec.
76 27          DEC R7
77 7A          REQ
78 F803  TEND: LDI (521).1
7A BC          PHI RC
7B F809        LDI (521).0
7D AC          PLO RC
7E F810  SL:   LDI #10
80 A8          PLO R8
81 7B          SEQ
82 FF01        SMI 01
84 3A82        BNZ *-2
86 7A          REQ
87 88          GLO R8
88 FF01        SMI 01
8A 3A88        BNZ *-2
8C 2C          DEC RC
8D 9C          GHI RC
8E 3A7E        BNZ SL
90 83          GLO R3
91 3223        BZ  NEXT   If Sec=0 wait for next time
               .. Part of Second timing loop
               .. to make loop = 1 second
93 F851        LDI (25169).0
95 AF          PLO RF
96 F863        LDI (25169).1
98 BF          PHI RF
99 2F          DEC RF
9A 9F          GHI RF
9B 3A99        BNZ *-2
9D 23          DEC R3
9E 83          GLO R3
9F 3A72        BNZ BEEP
A1 57    OUT:  STR R7
A2 64          OUT 4
A3 27          DEC R7
A4 F87F        LDI SL+1   Location of tone for
A6 AD          PLO RD     for signalling end of time
A7 90          GHI R0
A8 BD          PHI RD
A9 F850        LDI #50    End of time note (tone)
AB 5D          STR RD     Go back thru second loop
AC 3078        BR  TEND   to signal end of time
               .. Decimal to Hex Subroutine
AE D4          SEP R4     Go back to main program
AF AA    DECH: PLO RA     Enter byte
B0 FAF0        ANI #F0    Find most sig. Hex digit
B2 F6F6F6      SHR 3      MSHD x2
B5 73          STXD
B6 60          IRX
B7 F4F4F4F4    ADD;ADD;ADD;ADD
BB 73          STXD
BC 60          IRX
BD 8A          GLO RA     Get original byte
BE FA0F        ANI #0F    Get LSD
C0 F4          ADD        D+M(RX)
C1 30AE        BR DECH-1  Exit
C3 XX          .. Hexadecimal to Decimal Subroutine
C4 9F          GHI RF     Get RF.1 into D
C5 D4          SEP R4     Return to main
C6 BF    HEXD: PHI RF     Enter Byte
C7 F800        LDI 00     Initialize
C9 AB          PLO RB
CA AF          PLO RF
CB 9F          GHI RF     Bring back byte
CC FF64        SMI 100    Subtract 100
CE 3BD3        BNF *+5    If less than 0
D0 1F          INC RF
D1 30CC        BR  *-5    Go back to Subtract 100
D3 FC64        ADI 100    Add 100 to get LSD
D5 FF0A        SMI 10     Subtract 10
D7 3BDC        BNF *+5    If less than 0
D9 1B          INC RB
DA 30D5        BR  *-5    Go back to subtract 10
DC FC0A        ADI 10
DE BF    DL:   PHI RF
DF 8B          GLO RB
E0 32C4        BZ HEXD-2  Exit
E2 9F          GHI RF
E3 FC10        ADI #10    Add hex 10
E5 2B          DEC RB
E6 30DE        BR  DL     Loop until D=0
E8 xx                     Temporary storage
.. xx
FF xx
```

Square Roots

The mathematical capabilities of Tiny Basic are really rather limited: no fractions, no trig functions, not even a square root. Who needs them? Maybe you do. So this is a short program to calculate the square root of a number.

If you write out the perfect squares in a row like this

0 1 4 9 16 25 36 49

you can subtract each one from the next square in the series:

1 3 5 7 9 11 13

Notice that the differences are simply the odd numbers in sequence. In other words, if I start with zero and successively add the odd numbers, 1, 3, 5, 7, and so on, I will get all the perfect squares.

We can use this fact to compute the square root of a (small)

number, by subtracting successive odd integers from it until it goes below zero. Here is a Tiny Basic program to do this:

```
100 REM COMPUTE SQUARE ROOT
110 INPUT N
120 IF N<0 THEN GOTO 110
130 PRINT "THE SQUARE ROOT OF ";N;" IS ";
140 F=1
150 IF N<327 THEN F=10
160 N=N*F*F
190 I=-1
200 REM MAIN LOOP
210 I=I+1
220 N=N-I-I-1
230 IF N>=0 THEN GOTO 210
300 REM PRINT THE RESULT
310 PRINT I/F;
320 IF F>1 THEN PRINT "."; I-I/F*F;
330 PRINT
340 GOTO 110
```

This program should fit into the minimum Tiny Basic configuration (see Page 2, this issue), if you take out the REM lines.

Another way to calculate a square root is by successive approximation, a kind of "cut and try" technique known as Newton's Method. In this you make a first guess at a root, then divide it into the original number and average the divisor and quotient for the second guess. This is repeated until you cannot get any closer. Newton's Method is best when the numbers you are working with are large, or if the computer has a good, fast, hardware divide.

We can modify the program to use Newton's method:

```
190 I=1
210 R=I
220 I=(N/I+I)/2
230 IF I<>R THEN GOTO 210
```

Notice that this verson only changes the inner loop. For some numbers this program never finishes. Can you figure out why? In case you cannot, I'll give the answer next month.

You may wonder what the purpose of F is. If the number is small enough, we can get a more precise result by a process known as "scaling". That is, multiplying the number by a scale factor, then dividing the result by (in this case) the square root of that scale factor. The scale factor is 100, and if 100 times the number is still in the Tiny Basic range (i.e. is less than 32768) then we can get one more decimal place of accuracy by scaling.

NETRONICS RESEARCH
AND DEVELOPMENT LIMITED
333 LITCHFIELD ROAD (RTE. 202)
NEW MILFORD, CONNECTICUT 06776

NEW PRODUCTS
Text Editor
Assembler
Disassembler
Video Display Board

PRINTED MATTER