SOONER 99ERS

This newsletter is the official publication of the
SOONER 99ers
POB 61061
Oklahoma City, OK 73146

Sooner 99ers BBS#:
(405) 672-8270 300/1200 8N1

Club Officers:

Dave Lewis	President (405) 329-2274
Robert Stepp	Vice President
Garth Potts	Acting Secretary
Mark Kitchell	Treasurer
Jerry Robertson	Librarian
Barry Peterson	Newsletter Editor (405) 721-6930

November, 1989

Greetings, friends! Welcome to the November edition. I sure am in a THANKFUL mood and appreciate all of you and what you do to make my editorial job easier.

By the way, I am very thankful for the editors and members of the users group we exchange with since that's where I get much of my input.

Many things need mentioning now, so I'll just dig right in: first, a BIG welcome to new member Mike Longstreath! And a request; he has a zapped 32K card (I have it now) and needs either repair, replacement, Zeno board or some other memory expansion. Speaking of the Zeno board, Dick Farrah has one and is in the process of getting it together. (The Zeno board, I mean) I hope to see it and its' documentation soon and order one for myself. I have extracted some Zeno board info from the West Penn 99'ers and reprinted it here for those who have or plan to have a Zero board.

On another, slightly related, subject I have reprinted an article from the Boston Computer Society's TI-99/4A's fine newsletter about some high-quality music software from Harrison Software. It includes more than just a sales pitch and sounds promising, pun intended.

In looking through some OLD (5 years) newsletters from the Washington D.C. area, I came across an article on assembly language that impressed me. Although written on a complex subject, I think that even the 'Bozo' level readers can learn from this fine tutorial. Garth and friends, READ IT! Please!

In order to fill up this newsletter and avoid sending out a page with printing on one side only, I have included a copy of a page from the PUG Peripheral of the Pittsburgh Users Group. They reprinted the page from VAST News/Vast users Group; Tempe, Arizona. They copied info from the Cedar Valley 99'er Users Group via the QB 99'ers and from the Net99er News of the Hurst (Texas) Computer Users Group via the SNUGGLEter of the Southern Nevada Users Group.

The older stuff is great but.... 'Twould be nice if we can generate some NEW stuff!

Barry Peterson, Editor

STYDKACBWATA[1]

by Barry Peterson

Have you ever wondered what is going on inside your computer while you use your database, word processor, spreadsheet or game??? You might have heard some rumors/guesses but now if you want to know more, read on! (Don't be intimidated, please, this will be at what Garth likes to call the Bozo level.)

Each computer contains the same general types of components but the manufacturer assembles them differently using combinations of parts. Often, the company uses parts they manufacture (TI) but often the name on the case is the assembler rather than manufacturer.

Every computer contains a CPU (central processing unit) that is the brains of the outfit. As in ever-day life, brains are not enough to produce success, you also need to communicate. When the CPU, (often a single integrated circuit called a microprocessor), does something, (such as add 2 numbers), it requires data (input) and produces data (output). To move these numbers around inside the machine requires a path for the data, normally called the DATA BUS. This group of wires defines an important characteristic of the machine, the size of the number which can be moved with one operation, the TI-99 has a 16 BInary digiT (16-bit) data bus. Some other (big blue) computers also have the ability to add two 16-bit numbers together but must move them in two steps, 8 bits at a time, since the data bus is ONLY 8 bits wide. The data bus is bi-directional, which means data can be moved to or from any point in the machine.

NOTE: you are learning the special terms involved with the field of computers which make others believe you know what you are talking about. Add a few BUZZWORDS to the conversation and people will treat you with increased respect! (Is BUZZWORD a buzzword?)

If it is necessary to move some piece of data or information, the microprocessor must know WHERE to go for the data. The group of wires used for this purpose is the unidirectional address bus. Addresses always originate at the microprocessor and are sent to all devices (memory, CRU, peripheral, controller, etc.)

Another set of wires(bus) which is involved in controlling the operation of the computer is called, naturally, the control bus. This category involves timing, data direction, etc. Since the address bus connects many circuits, the control bus can contain enable/disable lines to CONTROL those actions.

During the execution of a program, the microprocessor sends an address to memory and waits for the contents of that address (an instruction). The memory unit sends the instruction to the CPU, which examines the instruction and does the operation. These pieces of data are transmitted on the DATA BUS, using the ADDRESS BUS for directions, under control of the CONTROL BUS.
Makes sense, doesn't it? Well, with any luck at all, it will.
Barry Peterson Editor

1. Some Things You Didn't Know About Computers But Were Afraid to Ask

AN INTRODUCTION TO 9900 ASSEMBLY CODE

David L. Ramsey

(Reprinted from the Washington DC area
TI Home Computer Users Group Newsletter 4/84)

This is the first in a series of articles on how to use assembly language routines in your Extended Basic Programs. In it I will cover the major utilities at the disposal of the programmer and how to interface these with TI's Extended Basic. Finally, I will show those interested in assembly language programming how to construct routines to read joysticks and move sprites, all at speeds far faster than those available in Extended Basic.

First, let's discuss the unusual features of the TMS 9900 micro-processor. The 9900, unlike most microprocessors used in home computers today, is a 16 bit processor. It has a 64 kilobyte direct memory address range unlike its 8 bit cousins who use various paging techniques to achieve the same memory address range.

Another important feature of the 9900 microprocessor is its lack of built in hardware registers. Only the workspace pointer, the program counter and the status register are built into the hardware. The 16 working registers are defined by the user. In addition, each subroutine can have its own set of registers thus eliminating the need to save any but the three hardware registers. The constant "pushing" and "popping" of values onto and off of the stack is not necessary with the 9900 chip.

Another feature of the 9900 instruction set is its memory to memory architecture. This allows the programmer to perform many operations on data in memory without ever moving it into a register first. For instance, I can use the instruction CB (compare bytes) and reference the two bytes of data being compared in the symbolic addressing mode. If I have previously defined the locations LABEL and CHECK, I could use the following line of code and never move the data into the workspace registers: CB @LABEL,@CHECK

Yet another feature of this versatile chip is its ability to extend its own instruction set by using the XOP instruction. This gives the programmer the capability to define up to 16 of his own operations and use them in his assembly language programs. Related to this is the ability of the 9900 chip to build and use "macros". If I needed to use a type of operation where a stack became necessary, I could define the two instructions PUSH and POP and then designate a specific area or memory as the stack with the BSS (Block Starting with Symbol) directive.

The final important feature of the TI 99/4A's assembly language instruction set is its large number of built in machine language subroutines. These make it much easier for the beginning programmer to develop application programs. As an example, the TI sees the screen as a memory mapped device and cannot access it directly. Instead, values must be read into certain registers and the VDP must be given an instruction on what it is to do with these values. The necessary instructions to access the VDP RAM can be written by the programmer but we need not bother since, with a simple BLWP (Branch and Load Workspace Pointer) instruction he can access any of a number of utilities to do precisely that. To write a single byte of data to the VDP RAM you could simply write BLWP @VSBW (Video Single Byte Write) or you could write the necessary routine yourself which would be made up of from 5 to 10 separate instructions depending on how you decided to do it.

A SIMPLE TUTORIAL

To get a feel for using the 9900 instruction set, let's put together a simple routine to read the keyboard for input and then to output that data on the screen.

```
        DEF READS
        REF VSBW,KSCAN
*
HEXFF   BYTE    >FF
*
STATUS  EQU     >837C
*
BUFF1   EQU     >8375
*
READS   NOP
*
RESET   LI      R5,>0000
        LI      R6,>0300
*
KEYBD   LI      R0,>0000
        MOVB    R0,>8374
        BLWP    @KSCAN
        CB      @HEXFF,@BUFF1
        JNE     WRITE
        JMP     KEYBD
*
WRITE   MOVB    @BUFF1,R1
        MOV     R5,R0
        BLWP    @VSBW
        INC     R5
*
LOOP    LI      R1,0
        LI      R2,6300
LOOP1   DEC     R2
        C       R1,R2
        JNE     LOOP1
        C       R5,R6
        JNE     KEYBD
        JMP     RESET
*
        END
```

This listing shows a simple program to scan the keyboard and display the input on the screen. It was put together on the Editor/Assembler package and uses some of the unique features of that package. Those of you who have the Mini Memory module and the Line-by-Line Assembler should note the following differences. First, the Editor/Assembler package supports 6 character label names; the MM module supports only 2 character label names. Next, the E/A package supports the DEF directive; the MM module does not. What this means is that to use the above routine, MM module users will need to shorten label

names to 2 characters and they will need to delete the DEF directive. Also, after they have completed entering the program they will need to make an entry in the REF/DEF table. This is what the DEF directive accomplishes for the E/A user. Finally, the MM module user cannot use the names of the utilities such as VSBW, instead they must use the address where the utility entry point is located. The BLWP @VSBW example given before becomes BLWP @>6024 with the MM module. (Please note that the > symbol indicates a hexadecimal value.)

At this point we can begin to examine the routine that is listed above.

```
RESET   LI      R5,>0000
        LI      R6,>0300
```

This portion of the program is given the name RESET. In it we simply load the values for the first position on the screen and the last position on the screen. In this way, we can compare our present screen location with register 6 and determine when it is time to return to the top of the screen. Register 5 is used by the program to indicate the first screen position to which we must write.

Next is the principal routine of the program, the keyboard scanning routine. It is listed below.

```
KEYBD   LI      R0,>0000
        MOVB    R0,>8374
        BLWP    @KSCAN
        CB      @HEXFF,@BUFF1
        JNE     WRITE
        JMP     KEYBD
```

The label that I gave to this portion of the program is KEYBD. First, I placed all zeros in register with the clear (CLR) instruction. Before we can access the KSCAN utility, it must know what device to scan. A value of >00 placed in memory location >8374 tells the utility to scan the entire keyboard. A value of >01 tells it to scan the left side of the keyboard and joystick #1. A value of >02 tells it to scan the right side of the keyboard and joystick #2. Since we want to scan the entire keyboard for this tutorial we need to place a value of >00 in memory address >8374. Now that we have a value of >0000 in register 0, all we have to do is move the most significant byte in that register to >8374. We do this with the move byte (MOVB) instruction. We place the byte from register 0 (R0) at (@) memory address >8374. Now that we have set this single necessary parameter we can access the utility with the single instruction BLWP @KSCAN.

At this point we need to check if the utility actually found a key depressed during the keyboard scan. There are two ways to accomplish this. The first method is to check the status byte and see if there has been a change since before we accessed the KSCAN utility. The other, which is simpler, is to simply check the value at >8375 with a value of >FF. This value (>FF) is placed at that location by the KSCAN utility when it finds that no key was depressed during the scan.

Since we have used the compare bytes (CB) instruction to make this check, we can now use any of the conditional jump statements to transfer program control. I have chosen to use the jump if not equal command because the value I am testing against is an indicator of no input. What I want to do is jump if the two values are not equal to the WRITE routine. For this I use the JNE instruction and tell the computer to jump if not equal to WRITE. The actual instruction looks like this:

```
        JNE     WRITE
```

If the value is equal to >FF then the jump will not occur so I need to cover those cases where no input is received. To do this I follow the conditional jump instruction with an unconditional jump instruction of

```
        JMP     KEYBD
```

This means that if input is found the computer goes to the WRITE routine; if no input is found, then the computer goes back to the KEYBD routine and scans the keyboard again.

This segment of our program is the heart of it. All of the other routines are built around it and with it in mind. It drives our program and we will always come back to it for more input.

The second most important routine is the WRITE routine that I mentioned above. It is listed below.

```
WRITE   MOVB    @BUFF1,R1
        MOV     R5,R0
        BLWP    @VSBW
        INC     R5
```

The WRITE routine will write the character of the ASCII value that the computer detected during the scanning routine. To do this we use the VSBW utility. There are two things that the VSBW utility must know before it can accomplish what we want. First, in register 0 we need to have the location in the video RAM that we wish to write to and secondly, we must place in register 1 the value we wish to write to the chosen location. The value we wish to write is obviously at BUFF1 (>8375) which is the value we detected during the keyboard scan. We can move it into the register with the move byte instruction. Next, we need a screen location. If you think back, when the program was just beginning we loaded register 5 with just such a value. So we can use the move byte instruction to move our location into register 0. Now we can access the VSBW utility. We do this with the branch and load workspace pointer instruction and the character will appear on the screen. At this point we increment the location counter by one with the INC instruction and we fall into our delay loop.

Now, the delay loop is not a mandatory part of this program but to not use it can mean that a single press of a key could fill one half to two thirds of the screen with the same character. Those of you who are unfamiliar with the speed of machine language may like to delete the lines from the label LOOP to the line with the instruction JNE LOOP1. If you assemble the program without the delay loop you will get a vivid indication of the real speed of machine language.

After you have it assembled, go ahead and run it with the LOAD AND RUN option on the E/A module.

TI-ARTIST PLUS

CTRL · HORIZONTAL SIZE

FCTN · VERTICAL SIZE

Disc Q · Alpha Num E · Rays R · Invert I · Circle O · Point P

Clear Image A · Store S · Draw D · Fill F · H or V H · K-line K · Line L · F-; Cursor Speed

Zoom Z · Box X · Hard Copy C · Frame V · Clear Color B · Swap N · Mirror M · F/Ground Color F-< · Plot/Erase F->

CTRL · SPACE BAR – Abort · Most functions toggle on/off by re-pressing letter · FCTN

PRINT UPRIGHT · ROTATE PRINT (R)

HIGH DENSITY · LOW DENSITY (D)

SPREAD APART · SCRUNCH THEM (S)

GO TO PRINTER · GO TO WRITER FILE (F)

SELECT DEVICES & PRINT (P)

SCALING · VECTOR (V)

H-90 (A) · H-180 (B) · H-360 (C) · HTILT (D)

V-90 (E) · V-180 (F) · V-360 (G) · VTILT (H)

Move without color (A) · Move with color (B) · Copy without color (C) · Copy with color (D) · SLIDES (S) · INSTANCES (I)

COLOR CURSOR · PATTERN INDICATOR · PLOT/ERASE · FOREGROUND/BACKGROUND

THESE REQUIRE FUNCTION KEY:

Up · Left · Right · Down · Enter · Fire-button · Speed · Color · Plot/erase

Helpful Hint #38

Make a mental note of political maneuverings in your company.

Look.. you're only six. Don't you think you should wish for something else besides a Princess Fax Machine for your room?

Sooner 99ers
PO Box 61061
Oklahoma City, OK 73146

TI-99/4A HOME COMPUTER

TAKE THIS "ORPHAN" OUT OF THE CLOSET AND JOIN THE SOONER 99ERS USERS GROUP!

WITH THE TI-99/4A YOU CAN DO—

*WORD PROCESSING *GRAPHICS *GAMES
*SPREADSHEET *MUSIC & SPEECH
*PROGRAMMING IN 6 LANGUAGES

```
QUEST3
RESET
QUEST4
FINISH
END
```

Subprocedures are:

```
TO SKIP :NUMBER
REPEAT :NUMBER [PRINT []]
END

TO RESET
CLEARSCREEN
MAKE "X 0
MAKE "COUNT 1
END

TO QUEST1
PRINT [WHO SAID "I AM THE GREATEST."]
SKIP 1
IF READLINE = [MOHAMMED ALI] THEN MESSAGE
IF :X=1 THEN STOP
WRONG
QUEST1
END

TO WRONG
SKIP 1
PRINT [NO, TRY AGAIN]
WAIT 150
SKIP 2
MAKE "COUNT :COUNT + 1
END

TO MESSAGE
SKIP 1
IF :COUNT = 1 PRINT [EXCELLENT! YOU GOT IT.]
IF :COUNT = 2 PRINT [THAT IS VERY GOOD]
IF :COUNT > 2 PRINT [CORRECT]
SKIP 1
IF :COUNT > 1 PRINT (SENTENCE [YOU GOT IT IN] :COUNT [TRIES])
WAIT 100
MAKE "X 1
END
```

QUEST2, QUEST3, QUEST4 etc. are done in the same format as QUEST1 (Choose your own questions).

Once the program works, examine its logic and make changes to make it more efficient or interesting. For example, you could keep track of the total number of tries used to give a score at the end. The heading could be written as a procedure and the program easily changed to another type of quiz. We can consider some changes and other approaches later. Because TI Logo is much more than turtle graphics, it has real program potential. It has color, graphics, sound, sprites and powerful text and list processing features. Share your Logo ideas with us.

Charles R. Midkiff
7303 Longbranch Drive
New Carrollton, Maryland 20784

"99er-NOTE"

Just for your Enjoyment
from

Jack Price

Subject:

To: Barry Peterson
From: Jack T. Price..
Date: 11/06/89

Dear Barry

This week while I was in ABQ N.Mex I picked up some disks for TIB and I will bring some copies to the club meeting next time I attend. Football season will soon be completed......AND we are winning again.!!!!!!

The main reason I am sending this letter to you is to **BRAG** on our little 'ORPHAN as it is soooooo often referred. This is a tale about the fantastic **IBM** machines that are so plentiful and becoming more plentiful in the TI arenas.

So many of us **strive** to get programs that are similar to the IBM formats, and I read from time to time that the new age of program for the 'Orphan will be similar to the IBM and will require more and more Memory to operate.....

Let me share with you some of the IRONY of trying to emulate IBM from personal experience.

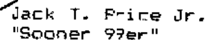
My Brother-in-law had to have an IBM so that he could rob from his company (pirate would be a better term) high dollar IBM programs costing several hundred dollars. He took his machine to one of our Club members as per my recommendation to have some programing done for his convenience. I am not talking in a negative sence toward the club member with reguard to his IBM's or his IBM knowledge. I am only relating to you the fact that my Brother-in-law's "High-dollar' IBM compatible machine now has a menu to choose from "thanks to the programing he had done" and his famous wordstar 5.0 is sooo large that it takes several disks to Boot. Thank goodness for his Hard drive! However, if he is to run the program that he originally got the IBM for he will have to purchase another hard-drive just to have it resident with his wordprocessor program and some other relatively simple programs. I am very proud that my little 'Orphan does not require so much expence to do what his high dollar IBM compatible can do with the

exception of the Auto-Cad. TIW is a real GIFT. With the large amount of space it occupies 66 sectors plus the loader (module space) I can do anything that the very expensive Wordprocessors can do. It is fast with the RAM or 8K as I have and with the exception of the Thesaurus, and Dictionary being resident I can do what they can and almost as fast. Even with the 40 colum handicap, once you get used to it and learn how to use it to its fuliest extent, I will put it up against any of the $500.00 plus word processors on the market. I am glad that I have a TI and TIW. I hope that more can learn about the powers of TIW so they can profit as I do every day in my office. If anyone has any questions concerning TIW give them my number. I don't know it all, and I still learn something new each time I use it. But if I can give some advice to a member or a user I will be more than happy to help anyone learn the power of TIW. The club has helped me alot and if I can offer anything back I want to.

If you could extract some of this info concerning MY brother in law's experience I would like to see it in the news letter.

Thanks for your time Barry.

Respectfully,



Jack T. Price Jr.
"Sooner 99er"

P.S. Here is a real 99erNote: I recently discovered when using the (.IF) command to include several commands in a document if you put a "New Page" (Ctrl 9) command at the end of the file when it is included it will begin at the start of a new page. If you do not include the "New Page" command, the file will be inserted where the end of the first file leaves off. This is particularly useful when using the (.FO) command for page numbers.

cc: File

While BASIC is a line oriented language with references to line numbers, Logo is procedure oriented. A procedure is defined as a word and then used in subsequent procedure definitions. This is a boon to the lazy programmer or poor typist. Once defined a procedure can be used much like a subroutine in BASIC. The Logo word has the advantage of being readily recognized as to function whereas in BASIC line 50 GOSUB 3000 is not very informative. You may or may not remember what the subroutine at 3000 does. If, in Logo, you want to use three blank lines on the display, a procedure could be written as:

```
TO SPACE3
PRINT []
PRINT []
PRINT []
END
```

This procedure can then be incorporated into others whenever three blank lines are needed. A more compact version makes use of the REPEAT command:

```
TO SPACE3
REPEAT 3 [PRINT []]
END
```

This can be generalized to allow varying the number of blank lines as needed:

```
TO SKIP :NUMBER
REPEAT :NUMBER [PRINT []]
END
```

Then for a different number of blank lines use: SKIP 1, SKIP 2, etc. in your other procedures.

Logo has other features familiar to the BASIC programmer such as the conditional IF --- THEN --- ELSE ---. Use of THEN is optional in Logo but the statement is otherwise the same as its counterpart in BASIC. To illustrate some of the text oriented features in Logo, let's construct a simple quiz program. The main program, LOGOQUIZ is as follows:

```
TO LOGOQUIZ
CLEARSCREEN
SKIP 3
PRINT [===SPORTS QUIZ ====]
SKIP 3
PRINT [---FEBRUARY 1985]
SKIP 3
PRINT [- A DEMONSTRATION OF LOGO]
SKIP 1
PRINT [- PROGRAMMING AND LOGIC.]
WAIT 450
RESET
QUEST1
RESET
QUEST2
RESET
```