ROCKY MOUNTAIN 99'ers

TIC TALK

VOL III, NO 7 DENVER, COLORADO USA MAR 1985
Non-member Subscription Rate - $7.50 Annually Single Copy Price - 75 cents

FROM THE EDITOR

Well, another month gone by, and I learn more about TI-Writer every day. I appreciate all the hints everyone has given me, and I hope they never stop.

Included in this month's issue is a survey that I hope will be benificial to all of us. If mailed to me at the address on the back, or given to me at the meeting on the 12th, I can guarantee privacy as far as the information is concerned. I want the information so that it can be combined and used to decide what we put in the newsletter as well as what we schedule for the monthly meetings. If you put your name, etc. on them, I will act as a clearinghouse of sorts for people that are interested in the same things. If someone wants assistance or to buy something, I will contact the party with items to sell or help to give and relay the information. I do not want this information to be the cause of a theft any more than any of you do. Be assured that I will not give out any information I am not specifically told to.

Enough of that! I hope you all respond to the survey, and I'll see you on the 12th.

MARCH MEETING

MARCH 12

Jefferson County Fairgrounds

Auditorium 7:00 PM

6th Ave. West to Indiana Ave.

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	[✓ 12]	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

EXTENDING THE eXtended OPeration
by Tom Frerichs

The designers of the Texas Instruments TMS9900 central processing unit, anticipating that they could not provide all of the opcodes that a programmer would need, implemented an opcode named eXtended OPeration (XOP). This opcode was provided to extend the normal instruction set of the processor by providing for sixteen user implemented subroutines that permit parameter passing. These XOP's are similar to the supervisor calls of the IBM 360/370.
The XOP instruction operates similarly to the BLWP instruction, except that the transfer vectors are located at specified memory addresses, there is a general parameter passing protocol, and there are only sixteen vectors available.
The general format of the instruction is:

```
label XOP  <g>,<xop #>
```

where <g> is a general address form - register, register indirect, register indirect autoincrement, symbolic, or symbolic indexed - and <xop #> is the XOP number ranging from 0 to 15.
The XOP number specifies the vectors for the call. The new workspace pointer is located at the word at >40 + 4 * <xop #>. The new program counter value is in the word at >42 + 4 * <xop #>, i. e. the new program counter is in the word following the new workspace pointer. For example, if XOP R1,1 were called, the workspace pointer would be loaded with the word located at >0044 and the program counter would be loaded with the word at >0046.
The return linkage of the XOP is identical to the BLWP call. The old workspace pointer is stored in the new R13; the old program counter is stored in the new R14; and the status register is stored in the new R15. The is functionally similar to BLWP, except that XOP vectors are located in a particular memory area.
The added feature is the <g> operand. It is this operand that makes the XOP useful for system calls. The address of the <g> operand is placed in R11 of the new workspace. For example, the call XOP R0,0 would take the address of R0 and place it into R11 of the new workspace. If the calling routine used >2BA0 as its workspace pointer, then R11 of the XOP workspace (after the call) would contain >2BA0 in the above example.
As another example, consider the following code:

```
WS     EQU  >2000
XOPWS  EQU  >2020

ENTRY  LWPI WS              ESTABLISH WORKSPACE
       LI   R0,>0040        LOAD XOP 0 VECTORS
       LI   R1,XOPWS
       MOV  R1,*R0+
       LI   R1,XOP0
```

```
          MOV  R1,*R0
                                   PROGRAM GOODIES

          XOP  R1,0                SAMPLE CALLS
          XOP  *R2,0
          XOP  *R3+,0
          XOP  @MEMADD,0
          XOP  @MEMADD(R4),0

                                   PROGRAM CONTINUES

   XOP0   A     @THREE,*R11        ADD THE CONSTANT AT THREE
                                   TO THE ADDRESS CONTAINED IN
                                   R11.
          RTWP                     RETURN TO CALLING PROGRAM
   THREE DATA 3                    CONSTANT TO ADD
```

After defining the workspaces that we will use, and then establishing the main workspace, the program loads the XOP 0 vectors at the required memory addresses. The program continues doing something (we would hope that it is doing something) when the first XOP 0 call is encountered. When this call (XOP R1,0) is executed, the old workspace value is stored in the new R13. In otherwords, the value >2000 is stored at >2020 + 26. The program counter is stored in R14, and the status register in R15. The workspace pointer is loaded with the word at >0040 (>2020) and the program counter is loaded with the word at >0042 (XOP0). Execution begins at XOP0.
When XOP 0 is entered, the address of the operand is stored in the new R11. The call XOP R1,0 would store the address of the old R1 in to R11 (>2002). The instruction A @THREE,*R11 would add the constant at THREE to the address pointed to by R11. The value at >2002 would be incremented by 3. XOP0 concludes by returning to the calling program.
The effect of this XOP call is to add three to the operand. The other examples would add three to the address pointed to by R2; add three to the address pointed to by R3 and then increment R3 by two; add three to @MEMADD; and add three to the address at @MEMADD + the contents of R4. We have created a new opcode that adds three to any operand we can name.
In most systems that use the TMS9900, some XOP's are defined as utility programs. In TIBUG, a monitor program that runs on the TI-99/4 computer, XOP R3,12 would write the ASCII character in the left byte of R3 to the terminal and XOP R5,13 would read one ASCII character from the terminal to the left byte of R5. Other similar utilities are written as part of TIBUG for XOP's 9, 10, 11, and 14. In the TI-990/10 bit slice computer, some of the XOP's are implemented in hardware to provide floating point functions.
In the Texas Instruments TI-99/4A Home Computer, the XOP vectors are fixed in ROM. We cannot change them to implement our own functions. Further, only XOP 2, and on some versions, XOP 1 have transfer vectors

that are usable. XOP 2 loads the workspace pointer with >83A0 and the program counter with >8300; XOP 1 vectors are >FFD8 and >FFF8. XOP 1 vectors are relatively safe from interference by BASIC, but beware of GPL routines and CALL LINK when using XOP 2.
You can see that the use of XOP can make some programming tasks easier and faster, but the availability of only one or perhaps two XOP's would seem to limit their usefulness. However, with a little programming, we can create an XOP that will let us keep the parameter passing ability, can be used as a general call, and will let us use BLWP-like calls without having to worry about workspace allocation conflicts. We will be able to have recursion to a depth limited only by available memory.
First, the Editor/Assembler lets us rename XOP's to whatever we want. The syntax is found on page 233 of the manual and is: label DXOP symbol,<xop #>. We will rename XOP 1 as "CALL" by entering:
    DXOP CALL,1. Our new syntax will be CALL @TEST rather than XOP @TEXT,1. "CALL" is a new opcode.
Second, most popular microprocessors are stack oriented machines. A stack is necessary to maintain subroutine linkage. When a proceedure is called, the saved address is stored on the stack. On return, the address is popped from the stack back into the program counter. The TMS9900, on the other hand, implements subroutine linkage using a linked list. Stacks can be created and used, but are not necessary.
In our new XOP, CALL, we are going to implement a stack. We will not be stacking return addresses, but will be stacking workspaces. When we use call, we will always use a new workspace located on the stack. When we return, the workspace that we were using will be returned to the stack ready for the next call. To begin our code, we need to allocate memory for workspace stack.

```
WS      BSS  160         SAVE SPACE FOR 5 WORKSPACES
MWS     EQU  WS+128      SET MAIN WORKSPACE AT TOP OF
                         STACK
SETUP   LWPI MWS         SETUP WORKSPACE AND XOP 1

                         SINCE THERE IS NO SPACE AT THE
                         ADDRESS POINTED TO BY THE XOP 1
                         VECTOR, WE WILL PUT A BRANCH
                         TO OUR NEW ADDRESS
        LI   R0,>FFF8
        LI   R1,>0460    BRANCH OPCODE
        MOV  R1,*R0+     PUT BRANCH CODE AT >FFF8
        LI   R1,XOP1     THEN PUT ADDRESS AT >FFFA
        MOV  R1,*R0


                         CONTINUE WITH THE PROGRAM


        CALL R2          OUR FIRST CALL USING "CALL"
        DATA RDKEY       THE FORM OF THE CALL FOR OUR
                         ROUTINE "RDKEY", A ROUTINE TO
                         READ THE KEYBOARD AND RETURN
```

```
                                THE KEY PRESSED VALUE TO THE
                                LEFT BYTE OF THE OPERAND. IS
                                     CALL <OPERAND>
                                     DATA <ROUTINE ADDRESS>
                                THE KEY VALUE WOULD BE STORED
                                IN THE LEFT BYTE OF R2 THIS
                                TIME.
                                THIS COMPARES WITH
                                     BLWP @<ROUTINE ADDRESS>

                                PROCESSING CONTINUES


        CALL @BUFFER            OUR NEXT CALL WILL ALSO READ
        DATA RDKEY              THE KEYBOARD. BUT THIS TIME
                                THE KEY VALUE IS RETURNED TO
                                THE BYTE AT @BUFFER.

                                REST OF PROGRAM


RDKEY   SB   *R11.*R11          OUR RDKEY PROGRAM BEGINS BY
                                CLEARING THE BYTE POINTED TO
                                BY R11.  R11 CONTAINS THE
                                ADDRESS OF THE OPERAND OF THE
                                CALL STATEMENT.
        MOVB @M,@>8374          SELECT KEY SCAN MODE
        CLR  R0                 CLEAR STATUS REGISTER COPY
                                (TO SEE IF KEY PRESSED)
        BLWP @KSCAN             GO DO KEY SCAN ROUTINE
        MOVB @>837C,R0          COPY STATUS REGISTER TO R0
        COC  @MASK,R0           SEE IF A KEY PRESSED
        JNE  $-12               NO, GO SCAN AGAIN
        CLR  R0                 YES, CLEAR STATUS (NO ERROR)
        MOVB @>8375,*R11        RETURN KEY VALUE TO ADDRESS
                                POINTED TO BY R11.
        RTWP                    RETURN

M       BYTE >04                USE PASCAL SCAN MODE
MASK    DATA >2000              KEY PRESSED MASK


                                NOW FOR THE CALL ROUTINE...

XOP1    LIMI 0                  DISABLE INTERRUPTS (FOR THIS
                                SECTION OF THE PROGRAM ONLY)
                                REMEMBER - R13 CONTAINS THE
                                OLD WORKSPACE POINTER OF THE
                                CALLING ROUTINE.

        MOV  R11,@-10(R13)       MOVE THE PARAMETER ADDRESS
                                 STORED IN R11 BY THE XOP
                                CALL TO THE ADDRESS OF THE
                                OLD WORKSPACE R0 LESS 10.
                                THIS WILL BE THE NEW WORK-
                                SPACE R11.
       MOV  R13,@-6(R13)        COPY THE OLD WORKSPACE
                                POINTER TO NEW R13.
       MOV  R14,@-4(R13)        DO THE SAME FOR THE OLD
                                PROGRAM COUNTER, BUT
       INCT @-4(R13)            INCREMENT IT BY 2 TO GET
                                IT PASSED THE DATA AFTER
                                THE CALL.
       MOV  R15,@-2(R13)        DO THE SAME TO THE OLD
                                STATUS REGISTER

                              NOW ALL OF THE VECTORS ARE
                              STORED AT OUR NEW WORKSPACE.

       AI   R13,-32           SET WORKSPACE POINTER TO NEW
                              LOCATION 32 BYTES LESS THAN
                              IT WAS ON ENTRY.
       MOV  *R14,R14          GET DATA WORD FOLLOWING CALL
                              TO PROGRAM COUNTER VECTOR
       RTWP                   NOW LOAD NEW WORKSPACE AND
                              PROGRAM COUNTER WITH RTWP.
```

The CALL routine allows us to call the routine that we are using from the routine itself as long as we have space on the workspace stack. Recursion, a powerful programming technique, is automatically supported. Further, we do not have to worry about knowing which routine calls which other routine. There is no workspace contention. Our only limit is that we have a depth of only four subroutine calls. This is generally sufficient and could be increased by changing our workspace space reservations at the beginning of the program.
Aside from the small amount of space necessary for the XOP1 program, the use of this technique will not add to the length of a program. The XOP call is either one or two words long, and the routine address is one word long. The variation in the length of the XOP instruction is dependant upon the operand. If a register or register indirect operand is used, the call is one word long. If symbolic or symbolic indexed addressing is used for the operand, then the call is two words long. The added word in this instance is often much less than the instructions necessary to store our parameter before calling with BLWP and then finding it later with our called routine.
Of course, in addition to the parameter passed by using the operand, we can also use the standard techniques such as using @14(R13) to point to the calling routine's R7. Anyother method is to use the operand pass of the CALL opcode to point to a parameter list. Several such lists could be kept in memory. However, the real value of the CALL routine is automatically allocating workspaces and in passing one value.

Although the TMS9900 does not use a stack to provide subroutine linkage and the TI-99/4A does not have the full XOP instruction set available, the combination of the stack data structure and the XOP opcode permits the programmer to define new methods of using his machine.

AMA LINK

A Review of a Terminal Emulator.
For the TI-99/4A
By Jerry Stachowski

A terminal emulator is typically recognized as a program that will make a computer simulate the actions of a terminal. Most terminal emulator programs allow you to have functions that your terminal by itself, cannot do. The most common feature is file transfer from the host to the slave, and back in the other direction. That's why many businesses today are not buying terminals for their mainframe computer systems but desktop workstations or home computers running a terminal emulator program. However the only terminal emulator program that can survive today, is one that does it's job well. Some terminal programs have the sad tendency to lose characters from the host computer, lock up when a strange control character comes down the line, lock up when a sequence of keys are pressed, or exhibit poor file transfer. These poorly thought out programs will be forced out of the marketplace but before they do, users of these programs will be burned.

AMA-LINK is a terminal emulator program for the TI-99/4A by AMA software. It is currently selling through Tenex for $34.95. It requires in addition to the console, 32k expansion memory, disk memory system, RS232 interface, and one of the following modules, extended basic, mini-memory or editor/assembler. It will operate at the following baud rates, 110, 300, 600, 1200, 2400, 4800. It can route text to an alternate output device, to a disk file, or to a ram buffer. ASCII uploads and downloads with X-ON, X-OFF protocols are supported. Caution, X-ON, X-OFF protocols don't have any error checking. Terminal Emulator II protocols, which do support error checking, are not supported. You can set parity, and word length. AMA-LINK does not actually emulate any commercial terminal.

AMA-LINK is not error free. If you are not using any of it's special features, it loses lines at 2400 and 4800 baud. It will lose characters at 1200 baud about the second or third page. This may not seem too bad unless you are thinking about downloading programs. The loss of one character can cause a program not to run. This can be diffcult to detect in object or machine code files. This gets much worse if you are using the alternate output device. The AMA-LINK manual states that usually if your printer has a transfer rate of 300 baud or faster will work just fine. My printer, an Epson RX-80 with a parallel interface, runs at an effective baud rate of 860. Using the alternate output is useless at 300 baud. It badly loses characters. At 1200 baud it loses 4 out of every 6 lines. I've found that turning my printer off is the only reasonable solution. My only problem with this is that whenever you enter the setup menu, to change baud rate or parity, is that the alternate output switch toggles on. So I suffer through several unreadable lines until I get it turned off.

AMA-LINK does have a few advantages. The X-ON, X-OFF file transfer does work well with systems That support it. Most computer bulletin boards and Compuserve do support it. The only problem with XON XOFF is that it makes 1200 baud seem like 300 baud. AMA-LINK's setup menu does work well. Unlike Terminal Emulator II you do not have to reset the machine in order to use a different bulletin board. My only complaint about it is that it defaults to 300 baud, 7 data bits, and even parity. Which is not the default for most bulletin boards. It does work better than Terminal Emulator II for file transfers on boards that don't support TE2 protocols. But I wouldn't sell my Terminal Emulator II cartridge.

In summary, this program has serious flaws that would make me think more than twice about buying it. The alternate output device is totally useless. I can't see using it at 1200 baud, as most subscription time share services like Compuserve, the Connection, and the Source charge higher fees for the use of 1200 baud service. The only reliable method of using it slows the 1200 baud down to 300. I wouldn't buy it for 300 baud use because it isn't enough of an improvement over Terminal Emulator II.

DBM
Data Base Management from Navarone Industries.
By Perry Patton

At last it's here -- a Data Base Management System for the TI99. How many times have we all wished for a data base management system with greater speed and capabilities than the TI Personal Report Generator or the Personal Record Keeping Modules. In other words big system capabilities for our small system computer. Data Base Management (DBM) will allow its users to build a data base file of multiple records of up to 255 characters in length from user created input screens, to index information by multiple keys and to sort files in any sequence by up to six keys.

DBM requires at least one disk drive and is both command module and disk operated. DBM is composed of four parts: 1. DBM entry, 2. DBM Setup, 3. DBM Reports and 4. DBM sorts. Documentation is poor but don't let that detract from the a well thought out system.

DBM setup provides a means to set up your data files. Screen formats are created here for customizing data entry requirements. Up to 32000 records with up to 25 fields may be created using these screens.

DBM entry uses the screen format for entering data. Records may be selected, displayed, changed, added or deleted using user defined unique key fields. DBM sort uses assembly language with high speed

sort algorithms to sort Data Base files using up to size nested sort keys either ascending or descending. A special select code feature allows sorting of only selected records.
DBM report allows the user to customize reports, total fields as required, and to store the report setup for use at a later time.
At its listed price of $65.00 DBM from Navarone is well worth the investment. It is one of the most powerful productivity tools available to date for the TI-99.

```
********************
*                  *
* WANT ADS         *
*                  *
********************
```

FOR SALE

3 each TI-Writer NEW! Packaged in the original shrink wrap.

$60.00 each
ENHANCEMENTS available from the Rocky Mountain 99'ers

call Dan Baugh 699-1940

SIG Meetings!

The EDITOR/ASSEMBLER SIG will meet on Wednesday, March 6th, 7:00 PM at Unique Systems located at Bates and Broadway behind the Oak and Pine store.

<<<<< DISPLAY ADS >>>>>

RATES:	10 in X 7.5 in - $15.00
	5.5 in X 7.5 in - $8.00
	3 in X 7.5 in - $4.50

ALL DISPLAY ADDS must be camera ready and must be received before the 15th of the month and accompanied by a check made out to the ROCKY MOUNTAIN 99ers P.O. Box 12605 Denver, CO 80212. Since the Club is a non-profit organization all money collected for advertizing goes toward the publishing costs of this newsletter.

==

<<<<< WANT AD RATES >>>>>

MEMBERS - FREE (25 word max) We must have your add by the 15th of the month to assure insertion in the next issue. Call 458-7315 or mail to BOX 12605 Denver, CO 80212. NON-MEMBERS must use DISPLAY ADS!

Rocky Mountain 99'ers

SURVEY

In order to better serve your needs with both the newsletter and the meetings please fill this out and either bring it to the next meeting or mail it to me at Box 12605 Denver,Co. 80212

NAME:____________________ADDRESS:______________________________________

CITY____________________STATE____________ZIP__________PHONE______________

Put an "O" in the blank if you OWN an item. Put a "P" if you plan to PURCHASE, put an "S" if you wish to SELL, and put an "L" in the blank if you will LOAN that item

99/4______99/4A______OTHER____

P.E.BOX______DISK______(make)____________HOW MANY?______________________

RS232______32K______128K_____ _CASSETTE______(make)___________________

PRINTER______(make)_______________MODEM______(make)_____________________

PASCAL______FORTH______TI-WRITER______LOGO II______X-BASIC____________

MULTIPLAN______EDITOR/ASSEMBLER______MINI MEMORY______OTHER____________

____

____

SPECIAL INTERESTS: Education____Business____Games____Graphics_________

Word Processing____Programming____Communications____Music______________

Voice Synthesis____Other____

Please list any topics you would like discussed and/or programs you would like to see demonstrated at our meetings.______________________

____

____

____

LIBRARY SUGGESTIONS:____

____

NEWSLETTER SUGGESTIONS:____

____

ARE YOU WILLING TO HELP WITH THE LIBRARY?________NEWSLETTER?__________

A SPECIAL INTEREST GROUP OR AT THE MEETINGS?______________________

Rocky Mountain 99'ers

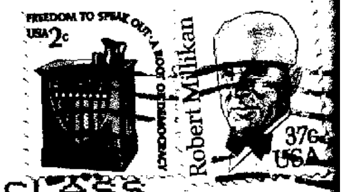
TIC TALK

This publication is printed monthly for the benifit of the membership of the Rocky Mountain 99'ers Computer Club. The Club and the paper are not for the benifit nor backed by any commercial enterprize. Both are non-profit in nature and are for the sole purpose of computer education. Any fees collected are used to defray any cost to maintain the organization. Neither the paper nor the Club have any affiliation with Texas Instruments. Any statements published in this paper are not necessarily the opinion of the membership.

\`\`\` OFFICERS and CHAIRMEN '''

PRESIDENT	TED MICHELSEN	986-3513
VICE PRESIDENT	MIKE HOLMES	751-7945
SECRETARY	MARTHA WEEG	320-5589
TREASURER	KEN MONSON	233-1788
EDITOR	DAVID OWEN	458-7315
LIBRARIAN	PETE CROWELL	750-5949
MEMBERSHIP	MARTHA WEEG	320-5589
PROGRAM CHAIRMAN	MIKE HOLMES	751-7945
EDITOR/ASSEMBLER..SIG	MIKE HOLMES	751-7945
TI FORTH..........SIG	PETE CROWELL	750-5949
MULTIPLAN.........SIG	BEN KRAMER	297-1056
THE STAR BOARD....BBS		455-3113

* * ROCKY MOUNTAIN 99ers * *
P.O. Box 12605
Denver, CO 80212



FIRST CLASS

****
* Do you see stars on the label *
* this means your membership is *
* now due. Send in your renew- *
* al today so you don't miss a *
* single issue of TIC-TALK!!! *
****

Dallas TI Home Computer Grp
1221 Mosswood
Irving TX 75061