President Rowland Buckwalter 264-5790
Vice-pres Joe Birchak 759-4052
Secretary Ann Halko 262-8206
Treasurer Mark DeNardo 791-1015
one-leg Pete at the IO Port
III, no. 2 February 1985
or Frederick Hawkins 432-5913
R GROUP

Next meeting: Monday, Mar 18 ity Room, First Nat'l Bank
7:30 PM as usual. 7th and Hamilton, Allentown

at the io port

Now hear this, now read this: your editor

reclaims his voice from the ubiquitous third-parties, er, make that third person. After an 'obtuse' issue we'll at least make an attempt at friendly and forget the users... Editorial stalwarts are in short supply this month. Our usual motley band of stevedores have balked at loading the great maw. All save the Chief Petty Officer Schreiber -- (note the raise in rank) -- who submits two XBASIC printing programs.

Faced with this near-mutiny, your editor, no Cap'n Bligh nor Ahab either, braves the high seas from a lifeboat. So, a 'letter is proposed, containing less news than views.

Meanwhile, March promises to be different.

A sea cable has arrived, announcing the first of several UCSD PASCAL tutorials by Ron Hartranft. Tentatively, he'll be in charge of four pages per month, worthy of an Admireship. Pascal, by the way, can be used to design programs that are NOT written in Pascal. From TI's TM990 FAMILY SOFTWARE DEVELOPMENT HANDBOOK (MPA29, $8.30 or so):

"A design language can be regarded as a generalised programming language, with the following characteristics:

(1) Syntax need not be completely rigid, as long as the logic is clearly defined and unambiguous.

(2) Operations can be identified by verbal description to start with, and later described precisely -- eg 'calculate mean'.

(3) Only standard, 'universal' constructs -- sequence, selection, iteration and standard data structures -- are used. Language-dependent constructs are not included.

"The aim of the design language is to establish the precise logical structure of the application before proceeding to implementation.... PASCAL WAS DEVELOPED AS A LANGUAGE THAT WOULD IMPLEMENT, MORE OR LESS DIRECTLY, THE FEATURES REQUIRED FOR SOFTWARE DESIGN. It was not designed for any particular machine architecture and hence has a 'universal' structure.

"IT IS POSSIBLE TO USE PASCAL ITSELF AS A DESIGN LANGUAGE. THE ADVANTAGE OF THIS IS THAT A DESIGN CAN BE CHECKED AUTOMATICALLY FOR LOGICAL CORRECTNESS, EVEN IF PARTS OF THE DESIGN ARE INCOMPLETE." (caps mine -ed.)

bug41 & threading the straits

caught redfaced, he toes the mark...

First off, CHECK41 is ok and better than it deserved. Its fate was sealed by a DUMB MURPHY'S law violation. Guilty, your Honor: Doubtless, the more energetic reader may have tried the sample base 41 loads. And herewithin lies the trap: If you did them in sequence, the first didn't work. No snickering in the gallery, please! "Your Honor, 'tweren't ALL my fault -- I was just tryin' to fix a bug. It's like this, see--" (Murphy has 'im fer sure)

The whole routine as printed in MID SOUTH's newsletter went like this:

from GARY NOEL CIS #75166,324

```
10 CALL CLEAR::CALL INIT:: CALL LOAD(8196,63,248)::
CALL LOAD(16376,84,32,32,32,32,32,48,0)
20 CALL LOAD(12288,2,224,131,224,2,1,240,129,216,1,
131,212,216,1,140,2,6,193,216,1)
30 CALL LOAD(12308,140,2,2,1,244,135,216,1,140,2,6,
193,216,1,140,2,6,155)
40 CALL LINK("T")
50 INPUT A$::IF A$="C" THEN CALL CLEAR:: GOTO 50 EL
SE 50
```

Careful consideration of the code above -- well, within limits! None but the most obsessed can read AL in decimal bytes -- reveals that the following is done:

At decimal 8196 (>2004, XBASIC's REF/DEF table pointer) is loaded with 63,248 which converts to >3F,>F8.

>3FF8, in turn, translates to an address value (word) of 16376; that's the next LOAD-- the string "T   " and the address where "T" begins.

Decimal 48,0 is equal to >3000 or 12288 in decimal. So, there's no surprise in lines 20 and 30: they're just putting the routine in place.

So where's the bug? Well, firstly the First Free Available Memory (FFAM) pointer in XBASIC is at 8194 (>2002). It usually starts out with >24F4 (9460 or 36,244). Between >24F4 and >3000 is 2828 bytes. The routine above barely uses 1/100th of that space -- one could even count it to check. As AL routines go, to even consider a count means SHORT bordering on TINY.

{continued on page 3}

```
60 !!!!  NOTEPAD  !!!!!
70 !************************
*  1985 FOR LEHIGH 99'ers   *
*    BY JACK SCHREIBER      *
****************************
80 ! Press REDO to print
     CLEAR to quit
     ERASE clears screen
90 CALL CLEAR
100 T=2 :: L=1
110 CALL KEY(0,K,S):: CALL H
CHAR(L,T,30):: IF S>-1 THEN
110 :: IF K>13 THEN 140
120 IF K=8 THEN 170 :: IF K=
9 THEN 180 :: IF K=10 THEN 1
90 :: IF K=13 THEN 210
130 IF K=11 THEN 200 :: IF K
=7 THEN 90 :: IF K=6 THEN GO
SUB 250 :: IF L<1 THEN 100
140 CALL HCHAR(L,T,K):: Q=T+
1 :: IF T>31 THEN 210 :: IF
L=24 THEN 100
160 CALL HCHAR(L,Q,30):: CAL
L HCHAR(L,Q,32):: T=T+1 :: G
OTO 110
170 IF T<2 THEN 220 :: CALL
HCHAR(L,T,32):: T=T-1 :: IF
T<2 THEN 220 :: CALL HCHAR(L,
T,30):: CALL HCHAR(L,T,32)::
GOTO 110
180 CALL HCHAR(L,T,32):: T=T
+1 :: IF T>31 THEN 210 :: CA
LL HCHAR(L,T,30):: CALL HCHAR
(L,T,32):: GOTO 110
190 CALL HCHAR(L,T,32):: L=L
+1 :: IF L>23 THEN 100 :: CA
LL HCHAR(L,T,30):: CALL HCHAR
(L,T,32):: GOTO 110
200 CALL HCHAR(L,T,32):: L=L
-1 :: IF L<1 THEN 100 :: CAL
L HCHAR(L,T,30):: CALL HCHAR(
L,T,32):: GOTO 110
210 CALL HCHAR(L,T,32):: T=2
 :: L=L+1 :: IF L>24 THEN 10
0 :: GOTO 110
220 T=31 :: IF L<=1 THEN L=2
5 :: L=L-1 :: GOTO 110 ELSE
110
250 CALL HCHAR(L,T,32):: OPE
N #1:"RS232.BA=4800.CR"
260 ! PRINT #1:CHR$(27);CHR$
(49) !a printer control
270 FOR R=1 TO 24 :: P$="" :
: FOR C=2 TO 32 :: CALL GCHA
R(R,C,Z):: P$=P$&CHR$(Z):: N
EXT C
280 PRINT #1:P$;CHR$(10);CHR
$(13);:: NEXT R
290 CLOSE #1 :: RETURN
```

xbasic: Notepad

Notes about notepad: A man of few words, Jack programs and I get to document them. It's not a bad methodology for putting together an article. In other words, you too can become famous.... Anyway, here is the discursive material--

NOTEPAD lets you use the XBASIC system as a poor man's word processor. It simulates a full screen editor, allowing the cursor to move both vertically and horizontally. You move the cursor around the screen with the FCTN arrow keys, depositing text where-you-will. When you're satisfied, you may then send the screen to the printer. On line 110, 'S>-1' works best on some XBASICs; on others, try 'S=0'. (The second doubled most keys on the faster version of XBASIC. Line 250 has the printer's device name. At line 260, when unREMarked, a printer escape sequence may be sent with each screen-full.

Interestingly, the last COMPUTER SHOPPER has a piece about RealType for KAYPRO systems. The idea is remarkably similiar -- essentially, both turn a computer into a typewriter.

Jack admits that this program is dated B.G. --Before Good programming. NOTEPAD should not be viewed as either engraved in stone or the epitome of great XBASIC. Both Jack and I made some quick going-to-press changes to NOTEPAD and we don't warrant anything. Simple enhancements might include:

- Try to calculate the value K so you can use the ON K GOTO or ON K GOSUB constructs. Advantages include speed, size and clarity: K would get evaluated just once and all of the IF's reduce to one statement.
- Work up some editing functions: insert, delete, a non-erasing cursor.

PS: CTRL N,J,M & L may be used with Epson and compatible printers to get the following results: N=emphasized, J=linefeed, M=carriage return and L=form feed.

xbasic: Howide?

As a follow-up to last month's 28 character columnizer, Jack sends this control program. This technique can be much more direct than last month's LIST to disk program, particularly when all one needs is a LISTing. Again, this sequence works with Epson compatibles; check your printer's documentation. TI WRITER users may duplicate the control sequence from the Editor:
SET COLUMN WIDTH: CTRL U, FCTN R, CTRL U, Q, (character for ASCII #)
(in our example program 28 would be CTRL U, FCTN Z, CTRL U)

program by Jack Schreiber

```
90 ! Set Printer width for s
maller than 80 character col
umns.
100 OPEN #1:"RS232.BA=4800"
105 PRINT #1:CHR$(27);"E"
110 W=28 :: PRINT #1:CHR$(27
);"Q";CHR$(W)
120 ! Change W in line 110 t
o the width you want the pri
nt-out to be.
```

bug41 continued & detailing XBASIC's AL environment --

Another and more significant bug

is that the FFAM isn't changed. Thus, a legitimately LOADed AL program could collide with "T", which wouldn't bother its REF entry at all, but it would bother you when LINK("T") crashed.

The base41 version took care of the location problem: none of the code was absolute (I disassembled it to see) so a quick fix was good enough. You can fix it too--

in line 10: change the 8196 load to catch 8194 (the FFAM) and LOAD the next address after the program: That would be 38 more than the load address and broken up into 2 decimal bytes.
Thus: CALL LOAD (8194,37,0,63,248)

Then change the REF table LOAD (16376) from ,48,0) to ,36,244. Now you've a new start address.

in line 20: change 12288 to 9460
& line 30: change 12308 to 9480

And that's what the base41 conversion used. But I wasn't so bright as this month's hindsight implies. I had LOADed the FFAM.with the start address plus the length of code. So my first available memory address was also my BL *R11 word. "T" could get really long if another routine was normally LOADed.

So as the bug caught my eye, Murphy tripped me up. Since I knew how the base41 conversion worked and how the checksum was calculated, I figured, "well I'll add 2 to both the checksum and the base41 number for the FFAM." THIS WAS A MAJOR INFRACTION OF MURPHY'S LAW. Given a foolproof method (checksums, automatic DATA statement generation, let-the-computer-do-the-calculating-of-base41) I found the one way to wreak havoc. So I added two to the REF table pointer 9UH to make 9UJ and NOBODY could LINK to "T".

Avoiding Murphy: USE the long way around. SKIP shortcuts. DOUBLE CHECK that it still works.

The bug41 correction is to change line 1's 5QR,9UJ to 5QT,9UH. Go ahead and do it. If you typed it in, likely it's saved someplace.

One omission was purposeful, though: Those three routines were not accompanied by their assembly language. I did it to force the curious to try base41 notation. Henceforth, the IO PORT will try to print at least a disassembly, and commented code with labels whenever we've the space and time.

An editorial aside: BARE CODE

is better than none. I realize that XBASIC buffs are proud of their AL, but let's not get into an elitist trap. COMPUTE!, notorious for their lousy taste in TI literature, is equally a pain among Commodore users for using a decimal load notation for C-64 ML routines and hardly ever published the code. This habit leads to an ML-fluent elite that don't really share... and readers that get tired of the same old one-up-manship. Comment your code, explain it in text and people will thank you for it.

(continued page 4 →)

XBASIC LOW MEMORY EXPANSION ADDRESSES

address hex	decimal	bytes	contents after CALL INIT
>2000	8192	32,0	>205A

used by XMLLNK(?) — this points to the CALL LINK routine.

address hex	decimal	bytes	contents after CALL INIT
>2002	8194	32,2	>24F4

FFAM: First Free Address in low Memory.
Points to next available byte AFTER last LOADed AL routine/program.

address hex	decimal	bytes	contents after CALL INIT
>2004	8196	32,4	>4000

LFAM: Last Free Address in low Memory (plus 1)
Points to REF/DEF table's start. >4000 is end of lo mem, thus no programs are LINKable.

address hex	decimal	bytes	contents after CALL INIT
>2006	8198	32,6	>AA55 170,85

Valid byte >AA is tested by CALL LOAD, LINK, INIT. Not >AA will get you a * SYNTAX ERROR *.

------ START of BLWP vectors for AL subprograms

	WS	PC	Name of subprogram	
>2008 =	>2038	>2096	NUMASG	NUMeric ASsiGnment
>200C =	>2038	>217E	NUMREF	NUMeric REFerence
>2010 =	>2038	>21E2	STRASG	STRing ASsiGnment
>2014 =	>2038	>234C	STRREF	STRing REFerence
>2018 =	>2038	>2432	XMLLNK	eXecute MLang LiNK
>201C =	>2038	>246E	KSCAN	Key SCAN
>2020 =	>2038	>2484	VSBW	Vdp Single Byte Write
>2024 =	>2038	>2490	VMBW	" Multi. " "
>2028 =	>2038	>249E	VSBR	" Single " Read
>202C =	>2038	>24AA	VMBR	" Multi " "
>2030 =	>2038	>24B8	VWTR	" WriTe to Registers
>2034 =	>2038	>2090	ERR	ERRor report to BASIC

Notice that XBASIC omits these standard utilities:
DSRLNK - Device Service Routine LiNK
GPLLMK - Graphics Programming Language Link
LOADER - assembly language tagged object code loader.

>2038 = UTLWS UTiLity WorkSpace Registers used by all of the above routines.

>2058 DATA >6520 ("A " if it's a string) Use unknown.
>205A CALL LINK routine; checks LINKname with REF/DEF table, and performs a simple Branch to the routine's start or when no match is found falls through to ERR routine with >2500 error in >8322.

XBASIC: AL subprogram LINKages --

(from page 3)

Aha! Another PAD location deciphered: -31966 is XBASIC's ERRor report.

Simply astonishing!

As your typical jack of all and master of none, this month I FINALLY started to write some assembly routines for BASIC. The writing of same turned out to be so simple and seemingly effortless that I was at loss to explain why I never did it before. (I had a slight advantage over XBASIC systems in that I was using the MINI MEMORY system, a truly underrated cartridge -- if you see it on sale, get it! A very good price is $60. This cartridge can put the console and cassette user at a par with XBASIC in power. There are some irritating aspects of the MINIMEM: It has only a byte dump and your readable AL code disappears when you go back to BASIC from the Line-by-line assembler. But the MINIMEM lets you watch the assembly process happen and illegal opcodes and rookie operand errors are simpler to correct than the big-time ED/ASM.)

In truth, FORTH made the difference.

If I've an explanation for why assembly language has gotten easy it must be FORTH. It gives one the familiarity with the system's details that BASIC users can only read about. Nearly all of the system utilities are FORTH words -- VSBW, VSBR, DSRLNK, GPLLNK, etc. A week with FORTH can pay (and has paid) dividends that two years with XBASIC and all the assembly language texts you'd care to read can't match.

As a sample of how powerful AND easy utility subprograms can be used from XBASIC, take a look at the adjacent program. This routine accepts from BASIC an address, a string of machine language (we'll get to that), and a program name. It LOADs the ML at the address plus one and updates the REF table with the program name. It happens much faster than you read this word: FAST!

Think about it: NO CALL LOAD.

Not from the program. Not from the disk. Further, your routines take up no more space than the code itself. They come in from the disk WITH THE XBASIC PROGRAM. That means they're already there when your program is RUN, skipping the tedious uncompressed XBASIC LOAD.

Obviously, just short routines are possible. But these can be ones that do such niceties as change the entire XBASIC color set, display a form on the screen instaneously, check a coincidence, or convert decimal to hex or even base41. There are two limits on length: first, DATA statements hit a maximum at about 155 characters and second, strings truncate after 255. With a slight modifications the routine could load two or three strings, but after a point there are better ways to skin the cat.

QUICKLOAD needs some AL to LOAD, so we'll offer up an XBASIC FASVID -- a go-anywhere VMBW, a ED/ASM cursor and color routine -- CURCLR, and a MINIMEM multiple line VDP write. And of course, a couple BASIC utilities that drive the whole works. Read on...

QUICKLOAD: DATA Load r

```
*   program by Fred Hawkins
*
* XBASIC version for ED/ASM assembly
*   loads a STR$ of ML into memory.
*
*used: 300 RESTORE 305:: READ ADDR,AL$
*       303 CALL LINK("QL",ADDR,AL$,"FASVID")
*       305 DATA 9763,machine_language_routine_here
*                    ^this location must be odd!

NUMRF  EQU >200C    names are changed to avoid interfering
STRRF  EQU >2014     with ED/ASM predefined references.
XMLLK  EQU >2018

FFAM   EQU >2004  First Free Address- not used by this pgm.
LFAM   EQU >2006  Last Free Address- ref table pointer
FACC   EQU >834A   Floating Point ACCumulator

QLOAD  CLR R0        NUMREF expects an array element # here
       LI R1,1       which param in CALL LINK: here ADDR
       BLWP @NUMRF   go and get ADDR from BASIC

       BLWP @XMLLK Convert BASIC's floating point to an
       DATA >1200    (hex) integer value. (returned in FAC)

       MOV @FACC,R2  ADDR is now in FAC; and copied into R2


       SETO *R2      STRREF expects a max string length. we
                     will take up to BASIC's max of 255.
       INC R1        point to the second param: AL$
       BLWP @STRRF   STRREF puts AL$ at ADDR+1, and ADDR
                     (the actual memory location, not the
                     BASIC variable) will get AL$'s actual
                     length.


       LI R3,LFAM    point to REF table pointer
       MOV *R3,R4    get REF table pointer
       DECT R4       point to AL$'s routine vector
       INC R2        R2 now contains ADDR+1
       MOV R2,*R4    put ADDR+1 into next routine vector

       LI R5,6
       S R5,R4       change R4 to point where AL$'s name
                     will be placed.
       MOV R4,*R3    It's also our new REF table pointer.
       DEC R4        Point just in front, remember what
                     happened to ADDR.
       MOV R5,*R4    This time we set the maximum STR$ to 6
       MOV R4,R2     STRREF expects R2 to have starting loc

       INC R1        Now we get third parameter: "FASVID"
       BLWP @STRRF   put it in.

       B *R11        return to XBASIC
```

AL: the rules of the road

Very likely, much of this is obscure to the unstudied user. So what follows is an attempt to demystify assembly language and its notation as practiced by the TI-taught.

Fundamentally, PROGRAMMERS read and write ASSEMBLY LANGUAGE. AL consists of a set of commands, often called mnemonics that correspond one for one with the computer's instruction set. They're called mnemonics because they are memory aids, and maybe partly because they seldom approach enough letters to warrant readability. The computer's instruction set is fortunately limited, so it's possible to learn all of the phrases.

An assembly language isn't read directly by the COMPUTER but by an program that constructs from the AL text the MACHINE LANGUAGE version of the AL. This ML version can be loaded into the computer, and finally be RUN or executed. It is important to realize that in both parts of the translation process, just exactly WHICH assembler and WHICH loader determine in a very large extent how the ASSEMBLY language will be written. However, once the process is complete and the program is in memory, there is very little difference, if any, between the different assemblers and loaders output. In other words, MACHINE language is pretty well fixed and assembly is pretty flexible. This is because an assembler is a PROGRAM -- software, and a CPU is hardware -- a fixed machine.

So it's possible to write assemblers that do more or less and loaders that do an equally variable job. A common enough item is a cross-assembler, a program that writes the machine language for a different computer. Equally common is the lack of an assembler for an enviroment. XBASIC lacks an assembler, not because it CAN'T but because no one has bothered to write one. The QUICKLOADer at left is a homemade loader. It's not very complicated and it can't load output from, say, the ED/ASM but in the end analysis what either DO is the same. (Well nearly, QUICKLOAD doesn't handle the FFAM. It CAN however, if we bother to write the code.)

Assembly languages have 'conventions', or a way of expressing what a particular instruction will do. In this, an assembler is no different than BASIC: one can't expect "HI THERE" PRINT to work better than one could CLR BOTH R0 AND R2. In both cases, we have to know what either program expects. Learning an assembly language places an additional burden on its student -- because you're controlling the machine directly. One has to learn the ins and outs of that as well. Until FORTH, TI 99/4A users had been at a handicap compared with other systems in how much we knew about the system. Much of it was purposely boxed in by TI, and the rest an effect of TI's withdrawal making it difficult to find a forum for system level information. In this respect, the Chesire cat HCM hurt us grieviously.

A thumbnail about the commands used in QUICKLOAD.

Certain commands operate only on workspace registers. Several of these are called Immediate; they all use the next memory word as a value that modifies the register. Thus,

LI (Load Immediate) Put the value into the specified register. Operand order in AL's convention mimics memory contents-- value follows the ML instruction (which specifies the register).

(continued on page 6)

the yellow brick road

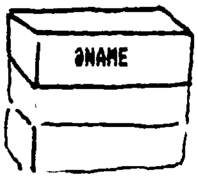
A simple analogy for a computer's memory is to view it as if each location is a brick. Just as you can build walls, houses and doorways with a pile of bricks, you may equally build 'data structures' with memory. The point, of course, is that the bricks don't change from the wall to house to street, and neither does RAM when it's a number or a character or a program or a register. It's all in ~~your~~ how you organize and view them.

The bricklayer's bag of tricks includes five different ways by which the memory locations can be named. (Unlike bricks, every location has its own and separate name.) Assembly language adds a layer of ease, by which you can tell the assembler a name that it'll remember for any location. This is done with the EQUate directive. A 'directive' isn't a machine language command but merely a control signal to the assembler program. This name is termed 'Symbolic'; however when you get down to the ML level what you'll find is merely a hex number. (PAD EQU >8300 will generate the same code no matter whether you LI R1,@PAD or LI R1,@>8300. The @ sign specifies that you're using a symbolic or named memory reference.) That's the only trick, the rest we can do with pictures:

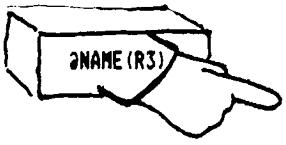
workspace register
(what's in the reg?)

WS register indirect
(what does it point at?)
(not where)

symbolic
(what's in a name?)



indexed symbolic
(reg offsets a name)



WS reg ind. auto-increment
(point and point past)

AL: pilgram's progress

(LI is easy to spot when PEEKing memory. Look for 2 followed by 0 through 15. Immediate commands always start with 2 except for LIMI which is 3,0. All are followed by two bytes that are used Immediately.)

More common than the WR operand commands are those that can refer to any location in memory. What's unusal about these commands is that they are variable in length; they may be one, two and sometimes three memory words long. In this respect, the TMS9900 MPU chip was in 1975 quite revolutionary.

> **"The TMS 9900 product line has for some time been one of the enigmas of the microprocessor industry. Even a casual examination of the TMS 9900 instruction set shows that the programmer's viewpoint, this microprocessor was at least two years ahead of its time....It was the certainly the microprocessor of choice for data processing-type, program-intensive applications, yet it was not widely used in these markets.**
>
> **"The reason for this lack of acceptance has been poor support from TI."**

From Osborne 16-bit Microprocessor Handbook, 1981, Osborne/McGraw-Hill. Though strictly speaking, it's stretching the context to say that variable instruction size was all Osborne and Kane were refering to.

Anyway, two bits in these instructions are used to signal what sort of memory location is meant. (Two bits can be 00, 01, 10, and 11; or four different flags. 10 is used twice in conjuction with four more bits that can specify either 0 or an INDEX register 1 through 15. Since 0 is used to determine which mode (Indexed or Symbolic) register 0 can not be used as an index register. The point to remember here is there are five modes of memory reference. All the different modes are diagrammed at left.) Since there are so many General memory referencing commands, it's simpler to categorize them by their common operations. So,

arithmetic:
DEC (DECrement) Subtract one from memory location. (one operand)
DECT (DECrement by Two) Subtract two. "
INC (INCrement) Add one to memory location. "
S (Subtract) Subtract first location from second and store result in second.
The first is unchanged. (two operands)

memory manipulation:
CLR (CLeaR) Puts 0's into 16 bits of memory; (one operand)
SETO (SET to Ones) Puts F's into 16 bits of memory. " The inverse of CLR.
MOV (MOVe) Transfer memory contents to memory location. (two operands)
This may transfer a location to itself; in that case the programmer is usually checking memory against zero.

program control:
B (Branch) Unconditionally branches to new program location. (one operand)
BLWP (Branch and Load Workspace Pointer) Unconditionally branches to new program location. Additionally, change the workspace and place in
Register 13: the old workspace pointer
Register 14: the old program counter (points just past the BLWP
Register 15: the old status register.

Two quick points -- RTWP (Return w/Workspace Pointer) is the exact inverse of BLWP; those saved values in R13,14 and 15 are loaded back in. Of course if you change them before the RTWP, you'll end up someplace else, doing something else. More interesting, these three values are the only internal CPU registers the program can manipulate -- the rest is STORED in the RAM registers.

big_al
XBASIC: AL to DATA
pgm by Fred Hawkins

Automatic LOAD of an assembly language routine into a DATA statement. Adjust L and DATA until routine's length is correct; remove tail REMark in line 190.

```
100 L=35 !ADJUST THIS
110 CALL PEEK(-31952,A,B)
120 A=A*256+B+2
130 CALL PEEK(A,B,C)
140 B=B*256+C
150 FOR A=0 TO L STEP 2
160 CALL PEEK(9460+A,C,D)
170 CALL PEEK(B+A+3,E,F)
180 PRINT C;D,E;F
190 !CALL LOAD(B+A+3,C,D)
200 NEXT A
300 DATA ABC4567890123456789O
1234567890123CBA
```

Program shows set up for a 18 word routine length. BIG_AL will always LOAD into the last DATA statement of the program. By changing the offset in lines 170 and 190 greater than 3, you may LOAD into a second field in the DATA statement.

XBASIC will not permit this line to be edited -- the system can't handle the curious material in it. In other words, try to use the right line number in the first place.

MERGE will perform without any problems. BIG_AL can be resequenced and dropped into any program. (Programs with SUB's have not been tried; let me know how it goes.)

BIG_AL assumes the assembly language routine was LOADed after a CALL INIT, and therefore doesn't bother to INIT itself. Adjust the 9460 in line 160 to pick up from another memory location.

To use BIG_AL with MINI_MEM and ED/ASM BASICs, change the PEEK in line 130 to PEEKV and the LOAD in line 190 to POKEV. Otherwise, the routine works identically with the XBASIC version, although CALL LOAD pt must be determined for line 160.

BIG_AL can be very likely be made into a two-line XBASIC program. (One caution, BASIC doesn't keep its programs on word boundaries. If you attempt to LINK to these DATA statements directly, you run the risk of LINKing to a program that can not work. In other words, look before you leap.

AL: grist for the mill

XBASIC: FASVID

pgm by Fred Hawkins

FASVID is written for ED/ASM assembly and XBASIC.
To use: CALL LINK("FASVID",VADD,STRING$)
VADD should be within 0 to 767; additionally the length of STRING$ and VADD should not exceed 767 -- No range checking is performed. It's possible to write the string (offset by BASIC >60) anyplace in the VDP RAM. Works like a DISPLAY AT.

```
MW  EQU >2024  VMBW     * FASVID uses the next 256 bytes of
XM  EQU >2018  XMLLNK   * memory.  Overwriting of the next
SR  EQU >2014  STRREF   * LOADed program is possible.
NR  EQU >200C  NUMREF   *
FC  EQU >834A  Floating point aCcumulator

FV  MOV R11,R6     save BASIC return
    LI  R10,>045B  Put B *R11 code in R10
    BL  R10        Put address of HR in R11;

* This is used to make the pgm 'position independent'
* which can be POKEd (instead of legitimately LOADed)
* anywhere and still work.
HR  CLR  R0
    LI   R1,2   take second parameter first
    MOV  R11,R2 get HR's location
    AI   R2,>3E offset that to word after pgm's end
    SETO *R2    We'll take up to 255 len BASIC string
    BLWP @SR    Get string from BASIC

    DEC  R1     Now we'll get the first parameter.
    BLWP @NR    Get a number from BASIC
    BLWP @XM      and convert to
    DATA >1200       an integer.

* SET up for a VMBW
    MOV  @FC,R0  Get Vaddress
    MOV  R2,R1   Get address of str$ len byte (@SR)
    INC  R1      Point past to string
    MOVB *R2,R3  Get the length byte
    SRL  R3,8    Limit to just the byte
    MOV  R3,R2   Specifies how much to write to VDP
    LI   R5,>6000  Offset of 96 for BASIC char. set
    MOV  R1,R4   Point to beginning of string

AD  AB   R5,*R4+ *
    DEC  R3      * add constant to every character of
    JNE  AD      *       the BASIC string

    BLWP @MW     write string to screen
    B    *R6     return to BASIC
    END
```

QLD!

A brief and absolute addressed QUICKLOAD, designed for a CALL LOAD. Use this to LOAD the longer QUICKLOAD. (MINIMEM version below)

```
CALL LOAD(32000,4,192,2,1,0,1,2,2,125,23,4,32,96,76,5
,130,200,2,127,254,4,91,0,255) REM that's the routine.
CALL LOAD(32760,81,76,68,33,32,32,125,0,"",28702,127,254)
```

QLD! continued

The code looks like this: (compare with page 4)

```
      DEF QLD!     for ED/ASM BASIC!!!!!
      REF STRREF   ->>QLD! MUST BE YOUR FIRST LOADed program

 QLD! CLR  R0        *to use: READ your AL$ from the DATA
      LI   R1,1      *  string you've created with BIG_AL
      LI   R2,>A001  * then CALL LINK("QLD!",AL$)
      BLWP @STRREF   *
      INC  R2        * AL$ will be put into memory and QLD!
      MOV  R2,@>3F36 *  will point to it.  (It modifies its
      B    *R11      *  own REF table vector to point to your
      DATA >FF       *  routine.) To use AL$, CALL LINK to
                     * "QLD!" with the appropriate parameters
```

By the way, there's no reason why you can't reuse the CALL LOAD sequence (like the MINIMEM version) over and over. Your AL$'s will then always go into the same spot and all be LINKed as QLD!.

CURCOL: ED/ASM BASIC

```
       DEF CURCOL    *
       REF VMBW      * There's nothing particularly new about
CURCOL LI R0,1008    * this program.  It does, however, serve
       LI R1,CU      * as an object of contemplation:  once it
       LI R2,8       * is LOADed into memory CURCOL becomes
       BLWP @VMBW    * 'position dependent'.
       LI R0,783     *
       LI R1,CL      * Compare this with FASVID.  Can you see
       LI R2,13      * how to write it so it may be put another
       BLWP @MW      * place than originally LOADed and work?
       B  *R11
CU     DATA >FF,>8181,>8181,>81FF
CL     DATA >7171,>7171,>71C1,>C1C1,>C1F1,>F1F1,>F100
```

Noel's program disassembly

(from page 1)

```
LWPI  >83E0     Interesting.  Unnecessary.  (note B)
LI    R1,>F081
MOVB  R1,@>83D4 System copies to VdpR1 on keypress
MOVB  R1,@>8C02
SWAP  R1          * Otherwise this is a pretty
MOVB  R1,@>8C02  * standard VDP reg setting
LI    R1,>F487   * program.  Note that the low byte
MOVB  R1,@>8C02  * is sent first.  Note that the high
SWPB  R1          * nybble sent is >8: write-to-V REGs
MOVB  R1,@>8C02  * second nybble has # of register.

BL    *R11      Bug here? see note:
```

Or/dinarily it would be at least bad form. Remarkably, this routine doesn't work correctly without the Branch and Link, even if one bothers to clear the GPL STATUS byte at >837C.

NOTE B: BASICs always start out your routines and programs with >83E0 as your WS. R11 always has the return to BASIC.
CORRECT: Just skip this instruction.
Lastly, consider using LIMI 0 when using the VDP. As it stands, an interrupt can muddle the system. eg: Where's the VDP addr?

(continued on page 8)

ALs spring flours

Noel annotated

As brief as this program is, it can be very easily be shortened by nearly a third. One overlooked aspect of the GPLWS @>83E0 is that register 15 always contains the VDP Write Address. Not surprisingly that's >8C02 and by using workspace indirect addressing we can save four words of memory. The routine speeds up as well. Two possible versions:

```
LI R1,>F081                  MOV R11,R10
MOVB R1,@>83D4               BL @ST
MOVB R1,*R15                 DATA 4,>F081,>F487
SWPB R1                 ST   MOV *R11+,R0
MOVB R1,*R15                 MOVB *R11,@>83D4
LI R1,>F487             LP   MOVB *R11+,*R15
MOVB R1,*R15                 DEC R0
SWPB R1                      JNE LP
MOVB R1,*R15                 B *R10
BL *R11
```

The first can go anyplace in memory; the second will only work in the place it is first LOADed. It's a fair trade, though, because by adjusting the DATA statements one can easily set all of the VDP registers with a minimum of fuss. Note especially the passing of a data pointer in register 11.

Confessions and caveats.

FASVID, for sure and perhaps some of the other routines seem not to work as this issue goes to press. Humbug! I haven't the time nor the sharpness of mind to quickly catch the bug. Simply put the routines were OK for MINIMEM. But FASVID hangs up someplace, though I must admit I direct-assembled it in the correct (>24F4) location with MINIMEM's Line by Line.

If my assembly process is the culprit, then a standard ED/ASM sequence should work. However, I suspect that XMLLNK works differently from ED/ASM and MM BASICs. The ED/ASM manual has a curious pointer for XBASIC's CFI: EQU >12B8. On the other hand, this may be a red herring, and perhaps I've transposed a register. Arrrrggggghhh. So be warned FASVID, ain't. How about a sharper mind than mine rendering assistance? EI, HELP!

Otherwise, ensure that you don't stack these routines together, as most if not all of them use a block of memory following themselves. They can overwrite the next routine with remarkable ease.

>Frederick Hawkins

MEMBERS' ADVERTISEMENTS

(Send a postcard and we'll print the details. Buy, swap and sell.

Al Notak, phone 433-6001 work and 439-0483 has been in the hospital. He's out now but hasn't the same drive to learn about computers, so he's selling in a bundle the following:

TI 99/4A console, program recorder and manuals
Amdek COLOR 1 monitor
Digital double disk drives (SS but there's two. Not converted to TI as yet, but Mike Mattas knows how.)

$550 for the lot.

Two members have cloned on to IBM PC land. So they're offering the works. One, Jeff Albert, has just a couple items left:

TI MULTIPLAN $75, Parsec $10, TI cassette recorder $35, cassette cable $5, Personal Report Generator $10.
Also a monitor cable $5, and lastly an Epson MX-80 Dot Matrix printer with Centronics interface, extra ribbon, and paper, all for $225.

Call Jeff after 6:00 M-F, anytime weekends at 691-5756.

Robert Wenger, proud possessor of a new Panasonic SR. Partner, has the following:

Anchor Signalman modem $65
TI Joysticks $12
connector for ATARI joysticks $4
monitor interface cable $8
keyboard cover $4
TI-WRITER $70, MULTIPLAN $70, EXTENDED BASIC $70
A-MAZE-ING cartridge $5, Household Budget Management $10
And two books:
-TI99/4A in Bits and Bytes $10
-COMPUTE! Programmer's Reference Guide $10

Contact Bob between 6 and 10 pm at 717-421-5475 or write: Robert Wenger * 61 South Green St. * East Stroudsburg, PA 18301.

We're late--

And how. In case no one has noticed the IO PORT has been pretty stale of late. I sure would like some new (and old) writers. I've a push down stack approach to things and in the last two months, the stack hasn't cleared. In FORTH, stack underflow is OK but overflow is always fatal. As a partial remedy, the IO PORT presents:

The nekkid Steffen-

In our first reprint from the other newsletters, I'll grab George Steffen (in TOPICs -LA 99'ers). He's found a way 'round a problem I mentioned in Dec.....

reprinted from TOPICS -LA 99'ers January 1985

SUBROUTINE EXTRACTOR by George F. Steffen

In the latest issue of the Lehigh Computer Group's Newsletter, there was an article by Frank Hawkins commenting on the use of "Translator" programs to convert TI Writer files to programs. Since I wrote such a program (which got garbled in printing by the TI Writer Formatter), I was quite interested. The use on which he was commenting had been suggested by many writers of such programs, but I had never thought of putting the program to such use since I do have TI's Programming Aids III.

The comments concerned the use of TI Writer to delete a large number of unwanted lines from a program. The problem arises when you have a long program which contains a subroutine you wish to save. If you do not have the PA-III, you must delete the lines one at a time (quite time consuming as well as boring), list to a printer and retype only the lines you want to save (subject to error), or list to a disk and edit out the unwanted lines with TI Writer then translate back to a program. The point of Frank Hawkins' article was a little routine which used PEEK to find the beginning and end of the Line Pointer Table and then calculated the changes necessary to restrict its range to the routine which you are interested in saving. However, this program took two passes and required that you count the number of lines before and after the routine so that it could calculate the corrections to be posted to the Table.

As usual, when I see a program, I always try to find out how to make it better. I see no need for you to count lines when you have a computer available to do the job. The attached program will request beginning and ending line numbers of the routine which you wish to save and will do the job automatically. Its speed of operation depends solely on the distance from the highest line in the program to the lowest line to be saved. It checks about ten lines per second.

This program can be used in Extended Basic only. It should be stored in MERGE format and then merged with the program you wish to dismember. If there is a line below 7 which you wish to save, your program should be resequenced before merging this. RUN the program. When it stops, LIST. You will have only the lines you asked to be saved. You should save the shortened program using MERGE so that the unused lines, which are still in memory, are not saved.

How can this program wipe itself out while running? The Operating System refers to the Line Pointer Table only when it is looking for another line of Basic. Regardless of what else is happening, it continues on to the end of the current line unless directed elsewhere. So, the entire routine which makes the change is in Line 6, the rest of the program is just calculations.

```
1 CALL CLEAR :: CALL INIT ::
 INPUT "Line numbers of rout
ine to  be saved:  First, La
st?     ":L,M :: G=256 :: CA
LL PEEK(-31952,H,I,J,K)
2 C=INT(M/G):: D=M-C*G :: F=
(J-G)*G+K :: FOR E=(H-G)*G+I
 TO F STEP 4 :: CALL PEEK(E,
A,B):: IF A=C AND B=D THEN 4
3 NEXT E :: PRINT :"Line";O;
"not found!" :: STOP !@P-
```

```
4 H=INT(E/G):: I=E-(G*H):: H
=H+G :: C=INT(L/G):: D=L-C*G
 :: FOR E=E+4 TO F STEP 4 ::
 CALL PEEK(E,A,B):: IF A=C A
ND B=D THEN 6 !@P-
5 NEXT E :: PRINT :"Line";N;
"not found!" :: STOP !@P-
6 E=E+3 :: J=INT(E/G):: K=E-
(G*J):: J=J+G :: CALL LOAD(-
31952,H,I,J,K):: STOP !@P-
```

the asynchronous seive

Following hot on the heels of last month's newsletter article on bulletin boards comes a tour through several boards across the country. Our travel starts to the southwest in the nearby town of Reading 215-929-5348. This board is one of many TIBBS(tm) systems around the country. The program is copyrighted and was written by Ralph Fowler of Kennesaw, Ga. The Reading board features messages, a newsletter section and upload-downloads. The featured program in the download section was a terminal emulator named COMM99.

Leaving Reading and traveling southeast we arrive in the City of Brotherly Love, Philadelphia 215-927-6432. Another TIBBS(tm) system, Philly's security is so tight that you must request an application by mail (SASE) before you will be allowed access. If you do not use the board for more than a month then you will have to re-apply.

We hightail it out of town and follow I-95 south to Washington 301-434-0117. Finding another TIBBS(tm) system at least leaves us feeling comfortable about the sign in procedures. The 99ers Bull Board has a two week inactivity period so be sure to use it often or face re-applying. This board has some great features like Assembly, Pascal, and Forth columns and tips on mini-memory, disk mapping and peek & pokes. To access some of these will require a higher access level, which you must request.

We decide to leave our nation's capital and head for warmer territory. Our next stop is sunny Tampa,Florida 813-677-0718. Again faced with a TIBBS(tm) we find the going easy. This board features a Forth download section and an X-Basic TI-WRITER program for those people with access levels high enough to permit downloading. A twenty minute time limit is placed on calls.

From Tampa it's Westward Ho! until we reach the scourge of last month's newsletter Houston, home of the HUG TIBBS. After giving HUG a second chance I managed to download the music to the movie Ghostbusters and a print art file of a "Peanuts" calendar. There were also several tutorials and the usual message section. HUG is constantly updating the download files and I have since downloaded the Beatles song "If I fell in love with you". Although this program is very good, the Ghostbusters music is the finest music I have ever heard come from my TI. Keep up the good work HUG, I'll be back!!!!

As we head for home we pause briefly in Indiana to check in (see Jan. 85 I/O PORT). We fail to stop in Allentown and end up visiting our friends in northern New Jersey 201-929-8161. The Dragon's Lair has many downloads on-line. On my last trip there it contained three music , three game and three utility programs plus information on Pascal, Forth and Assembly. To gain full access requires a $10 donation to the C.J. 99 Computer Club. Details are available upon initial log on.

We finally end our journey and sit back in our easy chairs to review and digest the vast wealth of information we have collected during our little trip through the phone lines.

>Dave Hendricks

aha! A late arrival. DH was locked in the brig or was that asleep in the galley?

P.O.Box 4837 * 1501 Lehigh St.
Allentown, Penna. 18103

! samp target !
! put it !
! here, pal !
Allentown, PA.18102
! PERMIT NO.2018 !

The Real Programmer finds that deadlines and bugs are in direct proportion. The strength of coffee, however, is in an inverse relation to other two.