

THE FORT'S USER GROUP

```

*****
$ President Secretary $
$ Gary Coffee - 747-2891 Denny Przybyla - 432-1228 $
$ $
$ Treasurer Editor/Publisher $
$ Tony Miller - 456-4765 Pat Murphy - 485-2623 $
$ $
$ Vice-president Co-librarians $
$ Jeff York - 625-4546 Bryon Delabarre- 627-5922 $
$ D. Bradtmueller- 483-0896 $
*****

```

March 1987

PRESIDENT'S COMMENTS - BY GARY COFFEE

Last month we had the opportunity to see and feel the long awaited Myarc 9640 computer. Because of technical problems, we really didn't get to see the machine function properly - much to our disappointment. This was no great surprise to those of us that have been close to Myarc for the last several years!

On the west coast camp there soon will be coming the Triton/M6 IBM clone system. Did you ever wonder why all of the NEW machines are entering the TI arena? The target life of the TI 99/4A, projected by TI and their critics after the "pullout" was about 5 years (at best). It appears to me that at least 2 forces are betting that the "good ole" 99/4A owners are tired of their machines and are ready to spend more money to gain power, speed, memory, and enhanced capability - they feel we are ripe to pick! Myarc is after the approximately 200,000 peripheral expansion system owners - in fact, they are after 50,000 of those systems (50,000 times \$400.00 = \$20,000,000 business potential). On the other hand you have Triton/M6 thinking that they can pull you away from the TI machine by "connecting" you with an IBM compatible - Triton sees a way to "capture" a large mass of computer users, hoping that you will eventually want ALL IBM compatible, which they will surely be willing to sell you.

So here we have the TI 99/4A owner - the pawn in a multi-million dollar chess game - a game where the only clear winners are the people selling the products. I see the TI community soon being separated into at least 3 camps - the Myarc 9640 camp, the phoney IBM "connected" camp, and true 99/4A camp. Whichever camp you end up in, you will be in a far smaller group than you are today. The fight, about to happen, between the EAST (MYARC) and the WEST (M6/TRITON) will be the TI "Civil War of 1987"!!!!

After all of this seriousness, I'll leave with a slightly modified popular computer joke going around:
 "Three women are sitting in a bar discussing their lovers. The first says, "mine's a wrestler. He's forceful and aggressive in bed." The second says, "mine's an artist. He's delicate and gentle." The third stares bleakly at her drink and says "mine's from Myarc - he just sits on the edge of the bed and tells me how good it will be when I finally get it."
 (grin)

The next user group meeting will be Saturday March 14, 1987 at 9:30am at the Shawnee Branch Library. Robert Knox will demonstrating the Corcomp expansion system that he recently purchased. Hope to see you there!!

MINUTES

Meeting of February 14, 1987
 Submitted by Dennis B. Przybyla, Secretary

President Gary Coffee called the meeting to order at approximately 9:50 a.m. There were sixteen members and four guests in attendance. The Secretary's minutes of the January meeting and the Treasurer's report for February were published in the February Newsletter. The formal presentation of these reports at the meeting were omitted but submitted as published. There were no additions or corrections to the reports of the Secretary and Treasurer and the reports were approved as published.

There was a discussion about the need for a small black and white TV monitor to compliment our color monitor and the User's group computer system. A motion that "the User's group purchase a small, 5 to 9 inch screen, TV monitor in the \$40 to \$60 price range" was made by Bud Darr, and it was seconded. The motion to purchase the TV monitor carried with no descending votes.

Dave Bradtmueller encouraged members to check out the exchanged newsletters. Also, if any member types a program from a newsletter, please indicate next to the printed program that you have the program recorded to diskette or cassette tape. Through the compliments of Joe Beauchot, the club has a supply of diskette labels and tabs available to the membership.

Pat Murphy related that an updated meeting announcement flyer will be available soon. The update will reflect the date and time change.

Blair MacDermid announced that the Forth User Group meeting will be held on Tuesday, March 10, 7:00 pm, room 138 Neff Hall, on the IPFW campus.

Gary announced that there will be board of directors meeting at his home on Sunday, February 22, 2-4 p.m.

Gary will take the computer system home to install locking hardware on the cabinet. Hopefully, after the March meeting, the group's computer system will be kept at the library.

New members registered at the meeting were John Cains, Jon Hartman, and Rich Baily. Also registered prior to the meeting was Harold Paulson. Our paid membership now stands at twenty-three. Guests in attendance were Bryan Drummond, Ken and Kevin Mahoney, and special guest Charles Good from the Lima Area Users Group of Ohio. Bryon DeLabarre was the door prize winner. He received three blank diskettes.

With the closing of the official meeting and general discussion, a demonstration of the Myarc's 9640 computer was presented. Because of a monitor incompatibility and software problems, the demonstration did not achieve the group's expectation of a new alternative to the TI-99/4A. It was neat, though, to see the 9640 first hand. The group extends their appreciation to Myarc for providing us with a 9640, on loan, to enable us to put on the demonstration. Thank you, Myarc, and the best of success in the production of the 9640.

The next meeting will be held on Saturday, March 14, at the Shawnee Branch Library.

Treasury/Membership - Anthony W. Miller

Well it's that time again. The last meeting sure went well. Since the meeting time was changed to Saturday morning, the turnout has been impressive. I would like to welcome the four new members that signed up at our last meeting. Welcome on board: Harold Poulson, Richmond Bailey, John Cains and Jon Hartman (and son).

This next meeting should be a good one. Bob Knox recently bought the Cor-Comp expansion system. So far he is the only one in this group that has the complete Cor-Comp system. I am looking forward to see what this system has compared to my big PEB sitting out there collecting dust. Since the MYARC computer that we received didn't work, it will be nice to see a good product for a change. This is one person that is looking forward to this demo.

With the approval of the membership and the board of directors, by the next meeting I will have purchased a new black and white TV, for the monitor that is currently being graciously loaned to this user's group by Gary Coffee.

The treasury report for the month of February:

DEBITS	
OLD BALANCE.....	\$337.36

BALANCE.....	\$337.36
CREDITS	
INTEREST.....	\$ 1.35
MEMBERS(S).....	75.00

NEW BALANCE.....	\$413.71

Technical Report -> ** BANK SWITCHING **
by
Anthony W. Miller

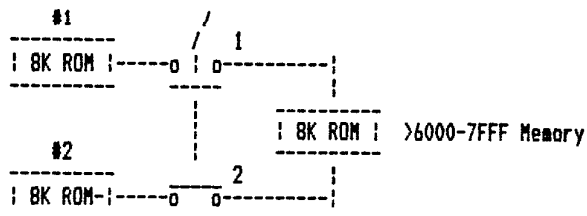
One of the most commonly used terms that we run across in the computer world is the term "Bank Switching". After I got my Gram-Kracker, there was a small explanation on how the Extended Basic module Bank Switched the upper 4K of of memory. To me the analysis that MG gave, confused me even more. In the world of computers most of the fancy things that are done are really not complicated, just explained badly. This is the attitude that I took in trying to find out how the module port was Bank Switched. After opening up the Extended Basic module and looking in, I found all of the interconnections to be more complicated to trace down. Looking through all of the cartridges that I had, I found the Pac-Man cartridge to be the least complicated and the most simple to trace out. The Pac-Man cartridge has only four chips. Two 8K ROMS, one C-MOS 7475 4-BIT Bistable latch and one C-MOS 74LS00 Quad 2-Input NAND Gate. With only four chips, this made some reverse engineering fairly easy. Following this article is the wiring schematic for the Pac-Man module. This schematic will be used to describe the actual software-to-hardware switching method.

I tried to write this article to allow for the basic concept of how Bank Switching is performed. One problem that arises out of writing this kind of article, is how to present this type of technical material. I decided to try to keep the description to a basic ON and OFF type of switching. With this concept, the basic idea should be fairly easy to follow. Just remember, all digital computers are is a set of ON and OFF switches, we just turn the right switches ON and OFF at the right time to make the total system work.

In the consoles cartridge space, only 8K of memory can be put there. So, how do you get 16K out of an 8K memory slot. Let's think about two 8K ROM's. We want to use both 8K's in an 8K slot. Therefore we have:

#1 8K bytes!
 }-- 8Kbytes Memory
 #2 8K bytes!

This is like saying that we have a box that can only hold 8K of material. So let's try to figure out how to fit this into the one 8K box. We could put in one of the 8K into the box, use it, then take it out and replace it with another we when we need the contents of that 8K. This analogy now takes on the picture of what we have in most of our houses. A three-way switch. Here, the switch will turn on our light from one or the other but not both. Now we have our single 8K is represented as our light bulb. Each switch will represent our separate 8K box. In technical terms this is referred to as Exclusive-OR gate. Now we have an idea of how the 8K of ROM can be replaced with two separate 8K ROMs, let me explain. When switch number one is turned on, the first 8K ROM is then activated. Now, when switch number two is activated, this disconnects the first 8K ROM and turns on the second 8K ROM.



This type of circuit has one problem. When either 8K of ROM is switched, we will lose all of our data. Plus the time it takes for us to turn the switch, the computer will lock up since the computer is so much faster than we are. One solution is to put in an electronic switch for us that will switch at the speed that the computer will accept. Then all we have to do is come up with a scheme that will make the computer do the switching of our 8K ROM's for us. Now is when the explanation of how the computer does our switching gets hairy. Just remember, the concept, previously mentioned, is used except we are going to let the computer do the switching for us.

Thanks to TI, we have some features at the cartridge port that will let us do are switching of the 8K ROM's. Following is a list of connections that we can use.

- All 16 address lines -> A0 - A15
- All 8 data lines -> D0 - D7
- A Write Enable line -> WE-
- A ROM select line -> ROMG-

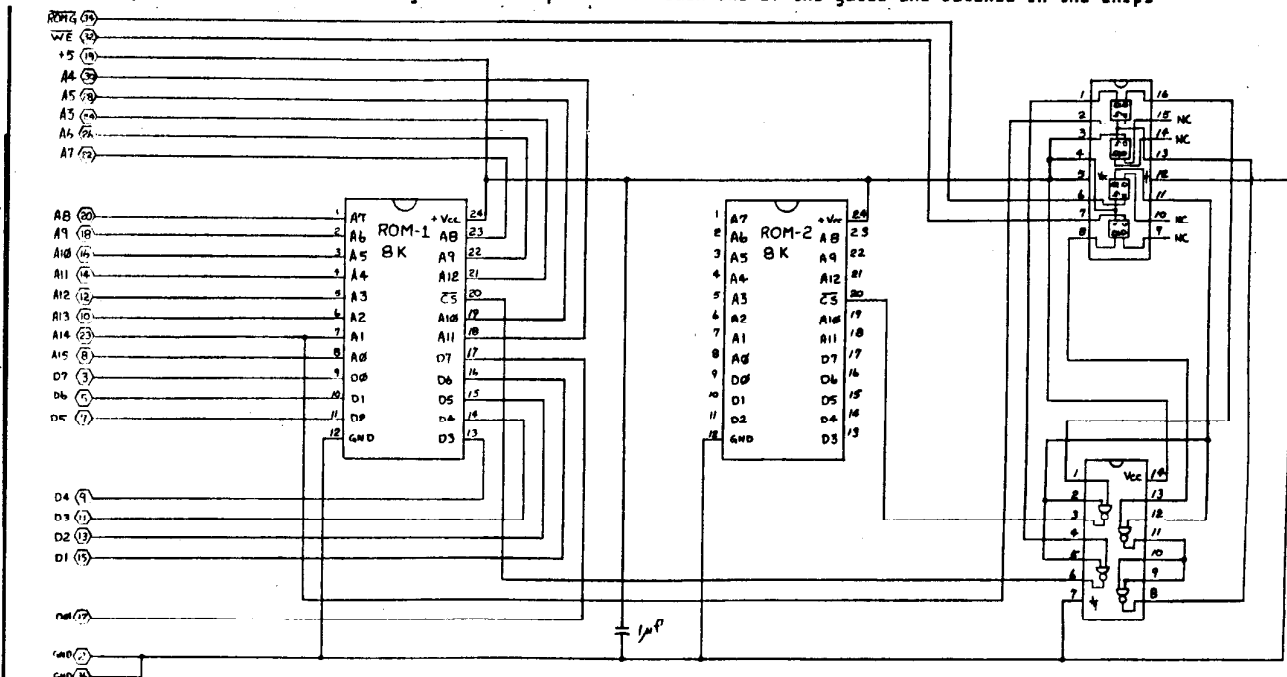
The three lines that we will use will be WE-, ROMG-, A14. With the help of the following schematic I will explain how these are used. First I will explain what each line is used for. The A14 line will be bit number two of the Least Significant Bit. This is the line that will select which ROM that we want to activate. Therefore if we want to select ROM number 2, then we write a BIN xxxx-xx1x to the LSB of the address line in cartridge port. Likewise, if we want to select ROM number 1, then we write a BIN xxxx-xx0x to the LSB of the address line in cartridge port. The ROMG- line is a predecoded line that is enabled only when the addresses of >6000->7FFF are selected. This is a perfect switch to turn on any ROM or RAM in the cartridge port for only the addresses we need. The next line is WE-. This line is activated only when we do a write to any RAM. Notice that this line is NOT predecoded. In other words, when this line is activated, anywhere in the 64K memory can be written to. One special note. The convention of writing computer lines with a (-) means that the actual voltage needed to activate the particular line is a zero voltage or ACTIVE LOW logic. Most of this is already done for us inside of the computer.

Using the convention of top-to-bottom and left-to-right I will explain what each one of the gates and latches in the chips are used for.

BY TONY MILLER 12/13/88 SUBJECT: ATARI PAC-MAN CART... SHEET NO. ... OF ...
 CHECK BY: DATE: ... ENVIRONMENTAL CONCEPTS, INC. JOB NO. ...
 ... FOR BANK SWITCHING

NOTE: ADDRESS AND DATA LINES ARE NOT SHOWN ON ROM-2 FOR SAME OF CLARITY ONLY.

○ = CARTRIDGE PORT PIN NUMBER



DM7475:

This is a bistable latch. This switch has two inputs and two outputs. The D input is the data input. The G input is the switch that is used to enable the transfer of the D input to the output. No data will be transferred until this line is at a HIGH state. The output Q will be the direct state of the input, whereas the Q- output will be the inverse of the input at D.

- #1 This is used as the storage device for the last ROM chip selected. This will stay in the same state until another write-to-ROMG- is selected and the ROM chip selected is different than the previous one. This latch is the workhorse of the bank-switching concept.
- #2 This latch is not used.
- #3 This latch is only used as an inverter for the ROMG- line.
- #4 This latch is only used as an inverter for the WE- line.

74LS00:

This is a quad-NAND gate.

- #1 This NAND gate selects ROM chip number 2.
- #2 This NAND gate selects ROM chip number 1.
- #3 This NAND gate decodes the input from ROMG- and WE-.
- #4 Used as an inverter. The output is then used to enable the #1 latch to the change from the previous state if need be.

Now let's do some Bank-Switching work.

Select ROM 1

In assembly we would write this code:

```
MOV >0000,>6000
```

What this will do is first set the ROMG- line low. Then WE- will go low. Now the #3 NAND gate will be enabled to a LOW state. #4 NAND gate will invert this to a HIGH state allowing the #1 latch to transfer the data from the D input. Since we selected a >6000, the D input will be LOW. The output from the #1 latch is then transferred into #1 and #2 NAND gate. Since Q- will be HIGH, NAND #2 will be set HIGH, setting the CS- LOW on ROM 1 and therefore allowing ROM 1 to be on. The input into NAND gate #2 will have a HIGH and LOW causing an output of HIGH, which will then keep ROM 2 from being selected. After the assembly language instruction is executed, the #1 latch will not change states until another write-to-ROM is enabled. Therefore keeping the #1 latch in it's last state. Since the outputs will be the same, anytime a ROMG- is activated, the ROM2 chip will be selected all the time.

One special note. The address selection is the main switching since the address line is tied to the #1 latch. Therefore any time an odd number address is selected, ROM chip #1 will be selected. This holds true for any odd number address in the >6000-7FFF cartridge slot.

Select ROM 2

In assembly we would write this code:

```
MOV >0000,>6002
```

This command will also set the ROMG- line low and the WE- line low. Now when both of these lines are low, this will then allow the #3 NAND gate to set the #4 NAND gate to an active HIGH state. This will then cause the #1 latch to be enabled. Since we set Bit number 2 to a HIGH state, the D input to #1 latch will then be HIGH. Now outputs Q- and Q will be inverted. These go into the NAND gate #1 and #2. The input to #1 NAND gate will be at two HIGH states, enabling the output to a LOW state, thus setting the CS- on ROM 2 to a LOW state and then enabling it. Since the input to #2 NAND gate has a HIGH and a LOW state, the output to ROM 1 will be HIGH, thus disabling it. Now on this selection we selected an even number address. Since this address was even, any time that we write to an even number address, ROM #2 will be selected. This holds true for any even number address in the >6000-7FFF cartridge slot. This is why the Gram-Kracker has a write protect switch on the bank 1 and bank 2.

I hope that this explanation has helped all of you to understand how the computer switches into one 8K slot two 8K ROM's. All of the information explained, and the schematic is solely based on the understanding that I have of computers. Any inaccuracy is merely coincidental, but the main concept of bank-switching is still intact. Until then-

*** HAPPY COMPUTING ***

>>>EDITOR'S NOTES<<<
by P. Murphy

Last month I typed in and included in this newsletter a program by Tom Freeman of the LA 99'ers. Besides this fine program, there are others very well done by Tom following along the same line of thought. As I mentioned last month, I will include those programs this month (the programs are typed in and in our library...thank Tom Freeman!). I will paraphrase Tom in his comments.

This month's first program entitled ASL/CL, reverse of last month's program...Assembly Language to Call Load. Tom says that you might have an XB program and want to list it in a readable format or have it available for a tape user with expansion. You may list it from memory range or directly from a D/F 80 file. Most object files can be run by loading from command mode by CALL INIT :: CALL LOAD("DSKx.xxx") and this program, then run with the memory range option (this is

considerably faster). Warning - a few files insert the start address into the ISR hook at >83C4, and will thus auto start. You will need to run the program on the DIS/FIX file directly or use a sector editor to change that value (you would find at the end of the file something like 983C4BXXXX where XXXX is the start address - it should be changed to 0000). Please note that the program ends with a statement on the SCREEN that you should type in one or two extra CALL LOAD's. (Please note if the program used auto-start then an additional CALL LOAD(-3184,x,y) will be needed, where x and y are decimal representations of the two bytes following 983C4B above, e.g., if you saw 24F4 then x and y would be 36 and 244).

If memory range option is used after loading the DIS/FIX file, there is an additional program that will help called ORIGINS. Many object files do not load all the bytes in the whole range of addresses used, but leave some blank, to be used after the program (this is signaled by the BSS directive in the source code). ORIGINS will search for these breaks - actually it lists all the origins, and you can see if there are large gaps as normally a single DIS/FIX record can only contain about 22 bytes. Each memory range can be separately specified in the program and not waste a lot of CALL LOAD's.

One more type of assembly file exists. Some authors have written assembly code, and then hidden it in the XB file, using various methods such as Barry Travor's ALSAVE program. You should suspect this when the XB program as listed has more sectors than could be accounted for by the number of lines you see, or if you see a CALL LINK or a CALL LOAD(-31804,x,y) when no assembly was loaded. The program called HIDDEN will search for the area containing the assembly file and inform you of the range. If you save the ASC/CL program in merge format, and then merge it into the hidden program, you can specify the memory range and produce a CALL LOAD file. WARNING - many of these are quite long and will produce a gigantic CALL LOAD file! Would be better to SAVE to produce a separate program image file and then DISKASSEMBLE it! See Tom's article in LA99ers November issue in our library to see how to use SAVE!

The last program listed is PRINTMERGE. It will take a MERGE type file and produce a neat listing in compressed format of each byte of each line and the ASCII representation if possible underneath it. Have fun with this one, but DON'T use it on large files, unless you have lots of paper!

Tom's purpose was to be able to disassemble the CALL LOAD's to understand them. He produced files that DISKASSEMBLER could read. Thanx Tom!

ASSEMBLY TO CALL LOAD

```

100 !CONVERT MEMORY OR DIS/FIX80 FILE TO MERGEABLE CALL LOAD FILE, BY TOM FREEMAN, LA99ERS
110 OC$=CHR$(179)&CHR$(200):: CC$=CHR$(182)&CHR$(0):: H$="0123456789ABCDEF" :: A1,A2=9460 :: DEFADD=16384
120 DISPLAY AT(3,1)ERASE ALL:"CONVERT TO ""CALL LOADS"":" BY TOM FREEMAN" : "CHOOSE:" 1. MEMORY RANGE:" 2.
DIS/FIX 80 FILE 1"
130 ACCEPT AT(7,28)SIZE(-1)VALIDATE("12")BEEP:CH$ :: IF CH$="2" THEN 260
140 GOSUB 560
150 DISPLAY AT(11,1):"NEXT TWO #'S MUST BE DECIMAL LAST WILL BE INCLUDED" : "FIRST ADDRESS ";A1:"LAST ADDRESS ";A2
160 ACCEPT AT(14,15)SIZE(-6)VALIDATE(DIGIT,"-")BEEP:A1 :: A1=2*INT(A1/2)
170 ACCEPT AT(15,15)SIZE(-6)VALIDATE(DIGIT,"-")BEEP:A2
180 DISPLAY AT(16,1):"CORRECT? (Y/N) Y" :: ACCEPT AT(16,16)SIZE(-1)VALIDATE("YN")BEEP:Y$ :: IF Y$="N" THEN 150
190 GOSUB 490
200 DISPLAY AT(24,1):"DO MORE IN THIS FILE?(Y/N) N"
210 ACCEPT AT(24,28)SIZE(-1)VALIDATE("YN")BEEP:H$ :: IF H$="Y" THEN 150
220 DISPLAY AT(24,1):"# BYTES IN DEF TABLE? 0"
230 ACCEPT AT(24,26)SIZE(-3)VALIDATE(DIGIT)BEEP:BY
240 IF BY THEN A2=16384 :: A1=A2-BY :: GOSUB 490 :: DEFADD=16384-BY :: GOTO 540
250 PRINT #1:CHR$(255);CHR$(255):: CLOSE #1 :: STOP
260 GOSUB 570 :: GOSUB 560
270 LINPUT #2:A$ :: RELOC$=SEG$(A$,2,4):: CALL DEC(RELOC$,RELOC):: A$=SEG$(A$,14,80)
280 T$=SEG$(A$,1,1):: IF T$=":" THEN 530 ELSE P=POS(H$,T$):: ON P-1 GOTO
290,290,300,300,310,320,330,330,340,350,360,370,5
290 CALL WARN1 :: A$=SEG$(A$,6,80):: GOTO 280
300 CALL WARN2 :: A$=SEG$(A$,12,80):: GOTO 280
310 RLFLAG=1 :: GOTO 390
320 RLFLAG=0 :: GOTO 390
330 GOSUB 480 :: LINPUT #2:A$ :: GOTO 280
340 RLFLAG=0 :: GOTO 430
350 RLFLAG=1 :: GOTO 430
360 RLFLAG=0 :: GOTO 450
370 RLFLAG=1 :: GOTO 450
380 CALL WARN3
390 DEFADD=DEFADD-8 :: GOSUB 480
400 LN=LN+10 :: CALL START(LN,DEFADD):: GOSUB 460 :: GOSUB 470
410 NAME$=SEG$(A$,6,6):: FOR X=1 TO 6 :: N=ASC(SEG$(NAME$,X,1)):: N$=STR$(N):: PRINT #1:OC$;CHR$(LEN(N$));N$:: NEXT
X :: A$=SEG$(A$,12,80)
420 PRINT #1:OC$;CHR$(U);U$;OC$;CHR$(L);L$;CC$ :: GOTO 280
430 GOSUB 460 :: LN=LN+10 :: GOSUB 480 :: CALL START(LN,ADDR):: PFLAG=1
440 A$=SEG$(A$,6,80):: GOTO 280
450 GOSUB 460 :: GOSUB 470 :: PRINT #1:OC$;CHR$(U);U$;OC$;CHR$(L);L$:: GOTO 440
460 ADDR$=SEG$(A$,2,4):: CALL DEC(ADDR$,ADDR):: ADDR=ADDR+RLFLAG60 :: RETURN
470 ADDR=ADDR-65536*(ADDR<0):: U=INT(ADDR/256):: L=ADDR-256*U :: U$=STR$(U):: L$=STR$(L):: U=LEN(U$):: L=LEN(L$)::
RETURN
480 IF PFLAG THEN PRINT #1:CC$ :: PFLAG=0 :: RETURN ELSE RETURN
490 FOR X=A1 TO A2 STEP 22 :: LN=LN+10 :: CALL START(LN,X)
500 FOR Y=0 TO 21 :: IF X+Y>A2 THEN 520
510 CALL PEEK(X+Y,A):: A$=STR$(A):: L=LEN(A$):: PRINT #1:OC$;CHR$(L);A$:: NEXT Y
520 PRINT #1:CC$ :: NEXT I :: RETURN
530 CLOSE #2 :: IF RELOC THEN ADDR=RELOC+9460 :: GOSUB 470 :: DISPLAY AT(16,1)BEEP:"REMEMBER TO ADD:" CALL

```

```

LOAD(8194,";U$;";L$;")
540 IF DEFADD<16384 THEN ADDR=DEFADD :: GOSUB 470 :: DISPLAY AT(19,1)BEEP:"REMEMBER TO ADD:" CALL
LOAD(8196,";U$;";L$;")
550 GOTO 250
560 DISPLAY AT(9,1):"OUTPUT FILE? DSK1." :: ACCEPT AT(9,14)SIZE(-15)BEEP:0$ :: OPEN #1:0$,VARIABLE 163,OUTPUT ::
RETURN
570 DISPLAY AT(8,1):"INPUT FILE? DSK1." :: ACCEPT AT(8,14)SIZE(-15)BEEP:1$ :: OPEN #2:1$,INPUT ,FIXED :: RETURN
580 SUB START(LN,X)
590 A=INT(LN/256):: B=LN-256*A :: Y=ABS(X):: Y%=STR$(Y):: L=LEN(Y$)
600 PRINT #1:CHR$(A);CHR$(B);CHR$(157);CHR$(200);CHR$(4);"LOAD";CHR$(183);
610 IF X<0 THEN PRINT #1:CHR$(194);
620 PRINT #1:CHR$(200);CHR$(L);Y$
630 SUBEND
640 SUB DEC(A$,A):: A=0
650 L=LEN(A$):: FOR X=1 TO L :: A1=ASC(SEG$(A$,X,1)):: A2=A1-48+7*(A1>57):: A=A+A2*(16)^(L-X):: NEXT X
660 A=A+65536*(A>32767):: SUBEND
670 SUB WARN1 :: DISPLAY AT(22,1)BEEP:"WARNING! FILE CONTAINS AUTO START AND MAY NOT BE XBASIC PRESS ANY KEY TO
CONTINUE"
680 CALL PRESS :: SUBEND
690 SUB WARN2 :: DISPLAY AT(22,1)BEEP:"WARNING! FILE CONTAINS EXT REF'S AND MAY NOT BE XBASIC PRESS ANY KEY TO
CONTINUE"
700 CALL PRESS :: SUBEND
710 SUB WARN3 :: DISPLAY AT(22,1)BEEP:"WARNING! FILE CONTAINS A BADTAG! PROGRAM ABORTED"
720 PRINT #1:CHR$(255);CHR$(255):: CLOSE #1 :: CLOSE #2 :: STOP :: SUBEND
730 SUB PRESS
740 CALL KEY(0,K,S):: IF S=0 THEN 740
750 SUBEND

```

HIDDEN

```

32760 !QUICKLY DETERMINE THE ADDRESS RANGE OF A HIDDEN ASSEMBLY PROGRAM IN AN XB PROGRAM, BY TOM FREEMAN, LA 99ERS
32761 !MERGE THIS FILE INTO THE XB FILE, THEN RUN 32762
32762 CALL PEEK(-31952,Z1,Z2,Z3,Z4):: Z5=Z16+Z2-65536 :: Z6=Z36+Z4-65536
32763 Z4=-32700 :: FOR Z7=Z5+2 TO Z6-1 STEP 4 :: CALL PEEK(Z7,Z1,Z2):: Z3=Z1*(256)+Z2-65536 :: Z4=MAX(Z4,Z3):: NEXT
Z7
32764 CALL PEEK(Z4-1,Z1):: Z4=Z4+Z1-1
32765 PRINT "PROGRAM RUNS FROM ";Z4;"TO -25"

```

ORIGIN

```

100 !DETERMINE ORIGINS OF A D/F 80 FILE, I.E.ADDRESS RANGE LOADED, BY TOM FREEMAN, LA99'ERS
110 DEF$="56" :: ER$="78" :: ORG$="9A" :: DAT$="BC"
120 INPUT "NAME OF DIS/FIX 80 FILE TO BE ANALYZED ":F$ :: OPEN #1:F$,FIXED,INPUT
130 INPUT "PRINTER? (PRESS ENTER FOR SCREEN DISPLAY) ":P$ :: IF P$<>" " THEN P=2 :: OPEN #P:P$,VARIABLE 136 :: PRINT
#P:CHR$(15);
140 LINPUT #1:A$ :: RL$=SEG$(A$,2,4):: CALL DEC(RL$,RL):: PRINT #P:RL;"BYTES OF RELOCATABLE CODE"
150 A$=SEG$(A$,14,80):: GOTO 170
160 LINPUT #1:A$ :: L=L+1 :: PRINT "RECORD";L
170 T$=SEG$(A$,1,1):: IF T$="." THEN 260 ELSE IF POS(ER$,T$,1) THEN 160
180 IF POS(DAT$,T$,1) THEN A$=SEG$(A$,6,80):: BSFLAG=0 :: GOTO 170
190 ORG=POS(ORG$,T$,1):: IF ORG=0 THEN 240
200 R=(ORG=2):: GOSUB 290
210 IF BSFLAG THEN PRINT #P:"BSS";
220 IF ORG=2 THEN PRINT #P:"R";
230 PRINT #P:AD:: BSFLAG=1 :: A$=SEG$(A$,6,80):: GOTO 170
240 RDEF=POS(DEF$,T$,1):: IF RDEF=0 THEN PRINT #P:"ERROR-NOT AN XB DF80 FILE" :: GOTO 270
250 DT=DT+8 :: R=(RDEF=1):: GOSUB 290 :: PRINT #P:"DEF ";SEG$(A$,6,6);AD :: A$=SEG$(A$,12,80):: GOTO 170
260 PRINT #P:" ":DT;"BYTES IN DEF TABLE"
270 CLOSE #1 :: IF P THEN CLOSE #P
280 STOP
290 AD$=SEG$(A$,2,4):: CALL DEC(AD$,AD):: AD=AD-9460*R :: RETURN
300 SUB DEC(A$,A):: A=0
310 FOR X=1 TO 4 :: A1=ASC(SEG$(A$,X,1)):: A2=A1 40+7*(A1>57):: A=A+A2*(16)^(4 X) :: NEXT X
320 A=A+65536*(A>32767):: SUBEND

```

PRINT MERGE

```

100 CALL CLEAR :: INPUT "NAME OF INPUT MERGE FILE? ":F$
110 OPEN #1:F$,VARIABLE 163
120 OPEN #2:"P10",VARIABLE 132 :: PRINT #2:CHR$(15)
130 DIM A$(163)
140 LINPUT #1:B$ :: L=LEN(B$)
150 FOR X=1 TO L :: A$(X)=SEG$(B$,X,1):: NEXT X
160 FOR X=1 TO 33 :: IF X+Y>L THEN 180 ELSE B=ASC(A$(X+Y)):: PRINT #2:STR$(B);TAB(X*4+1);
170 NEXT X
180 PRINT #2
190 FOR X=1 TO 33 :: IF X+Y>L THEN 230
200 B=ASC(A$(X+Y)):: IF B<32 OR B>126 THEN 210 ELSE PRINT #2:CHR$(B);
210 PRINT #2:TAB(X*4+1);
220 NEXT X

```

```
230 PRINT #2 :: IF X+Y<=L THEN Y=Y+33 :: GOTO 160
240 LINPUT #1:B$ :: IF EOF(1)THEN 250 ELSE Y=0 :: L=LEN(B$):: GOTO 150
250 CLOSE #1 :: CLOSE #2
```

ORPHAN PROGRAMMING CORNER --> BY JEFF YORK

To start off this month's column I would like to say that I am going to postpone my C programming section until next month. I have written a program in C that I think anyone interested in the language will enjoy! The major problem is that it has a very peculiar bug in it and I want to solve that puzzle before I produce the program. The bug is minor and may be caused by the compiler itself but I won't know until I have a chance to do some testing. The author doesn't provide much documentation on I/O support or even very much about the basics of sending data to the printer. I have found some solutions by experimenting but have procrastinated in sending the author his just due. (If you are reading this Clint ... don't worry, the proverbial check is in the mail! SERIOUSLY!). When I send the fairware money to Clint I will ask him about more documentation but until I get off my duff... I have found C99 to be an excellent quality software addition to my library. If anyone is serious about programming then C is for you. Before I get on to this month's column I want to say just a little about the program I have been talking about. I took a very good program (which is a recent addition to the library) and converted it from Extended BASIC to C. The program runs so fast it is remarkable when compared with the XB version. That's the major reason that I converted it. I wanted to show programming in C and the benefits of the added speed of the language. Until next month I'll just keep you in suspense.

The majority of my free time has been spent on learning C and translating the program I mentioned above. For this reason I am going to present you with an article that I downloaded from a bulletin board in Texas. I think the article is very interesting and provides a unique way of looking at call load statements. This will also be a nice change in the format of past articles, in case you are not really interested in programming in C. If nothing else, the article will give you a different "view" into our machine.

Downloaded from the Houston Users Group BBS (HUGBBS) an excellent BBS!!!

CALL LOADS

CALL LOADS is a regular column in SUBFILE99 featuring various CALL LOADS and CALL PEEKS that can be used on the TI Home Computer. In order to use these LOADS and PEEKS you must have the ED/ASSM, X-BASIC or MINI-MEMORY Modules and 32K Memory Expansion.

MULTI-PROGRAMMED BASIC

In this issue, we will talk about the TI-BASIC Operating System's use of the Free Memory pointer, how it decides where to load a program and how it knows if a program is in memory. After we learn these few tidbits, we'll learn how to trick the TI99 into holding onto three (3) programs at once. We will also get the computer to run each of the programs on our demand AND save all three to one single disk file ready to run again!

Where Do We Start?

If we were a computer, we'd start at the first free space in memory! And that's just where we will begin our study of the TI.

In TI-BASIC, the First Free address in VDP RAM is 14295. The space from 14296-16383 is taken up by the disk operating system. This free memory marker is stored in the CPU RAM pad at address -31952 and at -31950. Why in both places? It has to do with how the TI can tell if a program is already in memory.

If you'll recall from our article last November on the Line Number Table, TI stores the entire line number set in one place. It keeps track of the START of the Line Number Table (-31952) and the END of the Line Number Table (-31950). Whenever the user types LIST, RUN, or SAVE, the Operating System checks those two addresses. If they are equal then the Line Number Table is empty - there is no program present! If necessary, the TI will honk at you and display the appropriate error message ("CAN'T DO THAT").

NEW Doesn't Erase Programs

Conversely, when the user types NEW, the operating system simply (and ONLY) zeros out the Line Number Table pointer. It checks for the highest available memory (usually 14295) and places that value at the beginning marker (-31952) and the ending marker (-31950). The system does NOT erase your program from memory!

An Unforgettable Example

As an illustration of how the Line Number Table pointers work, let's type in the following lines in TI-BASIC (you will need to have the ED/ASM Module in place for this, too):

First, in command mode type:

```
NEW
CALL INIT
CALL PEEK(-31952,A,B,C,D)
```

```
PRINT A;B;C;D
```

You should get the values: 55,215,55,215

This shows the Line Number Table is empty. If you type LIST TI will say CAN'T DO THAT with a honk.

Now enter the following short program:

```
100 CALL CLEAR
110 CALL SOUND(150,1400,0)
120 PRINT "I'M HERE!"
130 PRINT
140 STOP
```

Now let's check out the Line Number Table markers again. In command mode type:

```
CALL PEEK(-31952,A,B,C,D)
```

```
PRINT A;B;C;D
```

You should get the values: 55,138,55,157

Now the computer knows that there is a program present and that its Line Number Table starts at 14218 and runs to 14237. If you type RUN now, the program will beep and announce its presence.

Now type NEW (yea, go on). If you try to RUN or LIST your program, you are told CAN'T DO THAT and chided with TI's infamous honk tone.

Now type the following:

```
CALL LOAD(-31952,55,138,55,157)
```

and then type LIST. See....

Now TI thinks there's a program again!

Just Put it Anywhere

There's nothing sacred about putting the program high up in the memory. In fact, you can put it anywhere you wish as long as you tell the TI where to find it. Let's try another short example:

- 1) Type NEW
- 2) In command mode type:

```
CALL LOAD(-31952,27,215,27,215)
```

- 3) Enter a short program:

```
100 CALL CLEAR
110 PRINT "I'M SHORT"
120 STOP
```

Now, try SAVE-ing this short program. What happens?

How Big is it?

The TI SAVE command saves your program in "memory image" format. In other words, it simply saves a copy of the program, byte for byte, from the VDP to the disk. Also, the TI SAVE command ALWAYS starts at the top of available memory (14295) no matter where the line number table starts!

This means that in our example above, the TI SAVE-ed every byte from 16383 to around 7100! Quite a few bytes for such a small program!

This may seem like an annoying flaw, but it is, in fact, a great opportunity for TI-Hacking!

Stuffing the TI

Since the computer doesn't care WHERE you start the Line Number Table, and since the system never really erases memory, just changes the table markers, and since the TI can only keep its binary "eyes" on one program at a time, we can use a few tricks to stick more than one program into VDP RAM at the same time. In fact, this is what we'll do:

- 1) Enter two independent programs into the VDP RAM area in different locations.
- 2) Enter a third "Master" program in another location.
- 3) Use the "Master" program to move to and from each of the other two programs on demand
- 4) Save the entire mess to disk under one file name.

Example Program(s)

NOTE: Be sure to enter all programs and other commands EXACTLY as they are here. This is a case where every byte counts!

- 1) Cold start the computer (turn the power off/on)
- 2) Select TI-BASIC with the ED/ASM cartridge in place.
- 3) Type NEW and CALL INIT

4) Enter the following program:

```
100 REM *PROGRAM ONE*
110 REM
120 CALL CLEAR
130 CALL SCREEN(15)
140 CALL SOUND(150,1400,0)
150 PRINT TAB(9);"PROGRAM ONE"
160 FOR L=1 TO 10
170 PRINT
180 NEXT L
190 CALL LOAD(-31952,44,55,44,130)
200 STOP
```

5) Now, let's find the start and end of the Line Number Table for this program. In command mode type:

```
CALL PEEK(-31952,A,B,C,D)
```

```
PRINT A;B;C;D
```

You should get these values: 55 9 55 52

If you didn't check your listing carefully - there's probably a typo, an extra character (or space), etc. Just go back and edit the offending line, you don't have to re-type the entire program.

6) Now let's trick the TI into thinking this program does not exist. Let's also set up a new area in VDP for our second program. In command mode type:

```
CALL LOAD(-31952,50,215,50,215)
```

Just for kicks, try LIST-ing your program... See... Be sure you type these values and not some other ones! If you're not sure, use CALL PEEK at the same address to check those four important values.

If all is OK, we are ready to type in our second program.

7) Now enter the following program...

(yes, start with line 100!):

```
100 REM *PROGRAM TWO*
110 REM
120 CALL CLEAR
130 CALL SOUND(150,700,0)
140 PRINT TAB(9);"PROGRAM TWO"
150 FOR L=1 TO 10
160 PRINT
170 NEXT L
180 CALL LOAD(-31952,44,55,44,130)
190 STOP
```

8) After you have entered program #2, check for the current location of the Line Number Table:

```
CALL PEEK(-31952,A,B,C,D)
```

```
PRINT A;B;C;D
```

If there are no errors, you should get these values: 50 31 50 70

9) We are now ready to write the "Master" program that will allow us to move from one program to the other. First let's set up a new Line Number Table area:

```
CALL LOAD(-31952,45,215,45,215)
```

10) Enter this short program:

```
100 REM *MASTER PROGRAM*
110 REM
120 CALL CLEAR
130 CALL SOUND(150,110,0)
140 CALL SOUND(150,440,0)
```

```

150 CALL SOUND(150,220,0)
160 CALL SOUND(150,110,0)
170 REM
180 PRINT "MASTER POGRAM"
190 PRINT "=====
200 PRINT
210 PRINT
220 PRINT
230 INPUT "WHICH PROGRAM (1 OR 2)? ":P
240 IF (P<>1)*(P<>2) THEN 230
250 IF P=2 THEN 270
260 CALL LOAD(-31952,55,9,55,52)
270 CALL LOAD(-31952,50,31,50,70)
280 STOP

```

11) Check the location of the Line Number Table:

```

CALL PEEK(-31952,A,B,C,D)
PRINT A;B;C;D

```

You should see: 44 55 44 130

12) To save all this together as one file just type SAVE DSK1.MASTER (or whatever). All three programs will be saved to disk and will be available when you load this file next time.

The Critical Test

Now let's see if your programs run. Make sure your Line Number Table is set to the MASTER program - CALL LOAD(-31952,44,55,44,130). Now type RUN.

You will be asked to select program one or two. Select program one.

When the Master program signals it is #DONE#, type LIST. You should now be looking at a listing of PROGRAM ONE - not the MASTER!

Now type RUN again. After PROGRAM ONE is finished immediately type RUN. Yes, you just ran the MASTER program again!

If you are getting screen lockup, wierd error messages like SYNTAX ERROR IN LINE 0, etc. then there is a mistake in the LOADs. The computer will blindly accept ANY values placed into address -31952 as valid Line Number Table markers. When you type LIST, your TI will do it's best to "list" whatever that "table" points to.

So Now What?

There are quite a few possibilities for use of this TI quirk. If someone could figure out how to get these various programs to AUTO-RUN, we'd have a TI-BASIC "CHAIN" command similar to the X-BASIC statement RUN "DSK1.FRED". Of course, you can store more meaningful programs than we have here, too. And it works the same in X-BASIC!

The next step is to locate and SHARE the Symbol Table(s) of these programs.

W.I.S.H. LIST

WISH GAC0686...A terminal emulator program that would transfer a complete disk rather than one file at a time.

WISH TM0686...Would like a ribbon cable connector (female) for a 36 pin .100" card edge connector.

WISH GAC0786...A program that would download different fonts to a dot matrix printer i.e., script, gothic, roman, etc.

WISH BCD0886..Would like a ribbon for a GORILLA BANANA and info on how to access the printer in graphics mode.

WISH JY0886...A program that converts CALL LOAD statements into assembly language source code.

WISH BCD0986..Would like a used expanded system (RS232 Optional), at least one disk drive and 32K memory expansion.

WISH GCC1286..Would like a cassette cable and educational programs or modules (primary level).

ANSWER GAC0686...Use FREEMWARE program "MASS TRANSFER" available from Stuart Olson, 25322 W. Wayside Place, Lake Villa, IL 60046. Program now in user group library.

ANSWER TM0686...Connectors you want are available from PILGRIM'S PRIDE, 5 Williams Lane, Hatboro, PA 19040

ANSWER JY0886...The program to convert CALL LOAD to ASM. LANGUAGE source or object code has been written by Tom Freeman of LA99ers and is in our library.

ANSWER GAC0886...Program called OLDENG prints any TI-WRITER file in old english letters - also Character Sets and Graphics Design III provides 6 full character sets - can be purchased from TEXAMENTS.