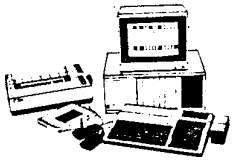


J*U*G*S NEWSLETTER

SEPT/OCT/1985

Our meetings are held at Ken And Cathy's (1401 S.HWY 161 JACKSONVILLE,AR) house on the second and fourth Sunday of each month (SEPT DATES 9/08/85 & 9/22/85, OCT 10/13/85 & 10/27/85) the FORTH SIG starting at about 1:00 PM. & the REG MEETING at about 2 pm.Club dues are $20.00 a year, newsletter only $10.00

J*U*G : President...(501) 985-2739.....Ken Gilbert
: Vice President...(501) 988-1700.....Mark Beck
Jacksonville Users Group : Secretary...(501) 988-2537.....Bill Alcott
P.O.Box 525 : Treasurer...(501) 985-2739.....Cathy Gilbert
Jacksonville, AR. 72076 : Editor.....Cathy Gilbert
* JABBS * : 24hrs 7days (501) 982-9063 Ken Gilbert(SYSOP)

Some new stuff added to the library : Fast-Term includes xmodem, & TEII transfer as well as a timer & text buffer capture...Director99 : an XB catalog program with assembly support...Ironheart : a new Adventure module game (this one was taken from DELPHI not the from Tex-Comp)...32K MATCHBOX SCHEMATICS for inside the console...A POWER SUPPLY MOD to the P-BOX...QUEST new Tunneles of Doom game FREEWARE this one comes fron PA....DM1000 : now everyone can have a disk manager with most of the features of the Cor Comp DM plus some very nice additions (like recovering a deleted file)...TI/PINBALL (game & construction set XB)...JACKET : prints your disk directory on a jacket that you cut out, fold, & glue together...CODER : assembly program that converts your message to morse code...DISK MASTER : another new disk manager...SCREEN/DUMP : this has a lot of options...ET : graphics display BASIC...PLAYBOY LOGO : goes to printer only. Most of these programs are FREEWARE. We also have several new disks of Forth programs, received from Centeral Arkansas Forth Interest Group (CAFIG) in Little Rock.

We have been given to understand from Navarone (in a telephone conversarion) that the user group dealers program has been dumped (sound familuar ?). I also understand the same thing may happen to the HYWAY program as word is some UG are dumping the software modules passing it out to members. Too bad these groups are ruining a great idea. Having user groups test the software was one of the best ideas to ever be put into action. It seems some people are determined to really make the 4A a DEAD machine !!!!

We are attempting to enlarge our newsletter exchange with this issue, which will be going out to some 250+ groups. The real problem is an up to date mailing list. Many of the clubs listed in the Computer Shopper & on TI's UG list are no longer in operation or have different addresses. At any rate maybe we can add a few more with this expansion. Craig Millers latest catalog did supply us with some 100+ addresses that should prove to be current for the most part.

While we are on the subject on the newsletter I want to put out a call for HELPPPPP.....I need someone to be assistant editor. Now don't everone call at once but do give some thought to this. I also could use an assistant librarian. It's time some of you started getting more involved.

Do you know computer language?

With computers a new language has been introduced which is being used increasingly in seemingly unrelated fields. A glossary of some basic computer words has been compiled by the Pacific Area Newspaper Publishers' Association which Farm Computer will run over the next few issues. Keep them. They will be a helpful reference not only for those starting but also for those experienced in computers.

Access time: The time a computer or system takes to get information out of memory, generally for display purposes. Should be of the order of microseconds or milliseconds (which see) but in some overloaded systems can run into seconds or even coffee-break access or response times.

Acoustic coupler: A simple device which accepts the handset of a telephone and transfers data from a terminal at a fairly slow rate using the restricted voice frequencies of the average phone line. Can take the place of a modem (which see), especially for portable use in the field.

Address: In computing, a name or number giving the location of stored data.

ADP: Automatic data processing, the British version of EDP.

Algorithm: A set of rules or procedures for solving a problem.

Alphameric or alphanumeric: Codes which include letters and figures.

Application: The particular problem a computer is asked to solve. Working out the payroll for a company is a typical application for a computer.

Archival system: A dedicated system designed to handle the library or morgue files of a newspaper, either accessible locally as part of a huge database or remotely via telephone.

ASCII: American Standard Code for Information Interchange. A 7-bit (8 if it has a parity check bit) with 96 characters (128 are possible). Pronounced as in Arthur Askey and also called USASCII ("you saskey"); virtually synonymous with the international ISO 7-level code. By international agreement this code is replacing the present teleprinter and telex code and has in many newspaper systems replaced 6-level TTS coding.

Assembler routine: Special computer programs (software) that convert simplified instructions fairly easily understood by humans into the more complex binary language of the computer.

Background: Some computers can process several jobs simultaneously. Lower priority jobs will be pushed into background mode.

Back-up: Where intelligence and information are not distributed, back-up is essential if a newspaper wants to be sure of meeting deadlines. Duplicated equipment, double writing to duplicate memory stores or periodic rewriting to back-up disks are ways of providing back-up.

Bar code reader: A scanning device in the OCR family where each typewritten character features a small code beneath. It is this bar-like code that the machine actually reads. True OCR readers read the typewritten characters themselves. A form of bar coding is now the basis of retail point-of-sale computing reader systems.

Base alignment: The satisfactory state where all type from a photosetter no matter what size or face sits on a common baseline. Not all photosetters achieve this.

Basic: An acronym for Beginner's Assembler Symbolic Instruction Code, a simple program language used in micro-processors and with the growth of mini-computers now the most widely-used of all programming languages.

Batch Processing: A method by which a computer system stores work to be done, handling one job at a time. the user cannot interrupt the processing, but some systems will process priority jobs out of turn. In commercial work, batch processing with priority is considered better than iterative processing since it does not need an operator to order each job processed, and computer time can be maximised.

Baud: A measurement of the rate at which information can be transferred electronically along a wire etc; a baud (pron bawd) is roughly equal to a word a minute. A 1200 baud system would carry 1200 words a minute.

BCD: Binary coded decimal. This is different from pure binary counting. In BCD you take each digit in a decimal number and give it a binary value, using up to four binary digits. Thus pure binary for 33 would be 100001. In BCD it would be 00110011 (made up of 0011 and 0011, equals 3 and 3). BCD also refers to a 6-level code (of 47 characters) compatible with paper tape, cards and magnetic tape. See EBCDIC.

Benchmark: A standard in comparing computers: usually a specific job which can be given to computers under test, their finishing times then being compared.

Binary: A numbering system with a base of 2 instead of 10 (as in decimal), every number is either 0 (zero) or 1, and each number (from right to left) has twice the value the preceding number can have. thus the decimal number 15, for example, is in binary 1111 (from right to left, 1+2+4+8). Decimal 2 in binary is 10 (say one-zero, not ten).

Bit: A single piece of information, represented by 0 (no bit or zero) or 1. Short for binary digit.

Blind keyboard: One which does not produce a hard or visible record of what is being typed.

Blow-up: It didn't work (of a computer program or anything else).

PULSAR ASSEMBLY UTILTIES

NOTE: This article originally appeared in the April Edition of the Ohio New Horizons User Group Newsletter. It is reporinted by permission. Anyone who would like a copy of the PULSAR source and object code should send an initalized SSSD disk along with a self-addressed/stamped return evelope and $5 to: PULSAR UTILITIES, c/o SUBFILE99, POB 533, Bowling Green, Ohio.43402

This article is meant as an introduction to the concept of PULSAR programming and hopefully will interest many inexperienced members into trying out assembly programming as well as spur our more advanced programmers into adding to the list of PULSAR routines.

THE SOBERING ASPECT OF ASSEMBLY : If you are like me, you're probably one of those people who bought the TI Editor/Assembler package because it was "a good deal" and not because you knew everything there was to know about writing in assembly language. You probably were as foolish as I was and assumed that the big fat book that came with the TI Ed/Asm package would "explain it all." Since the TI documentation is less than enlightening, this probably also means that, like me, you're not churning out skads of super-fast bit-byte-ing assembly programs either. You've discovered, like I have that writing assembly programs is a complex, error-prone and frustrating way to get things done. It may be true that a program written in assembly runs fast, but it takes forever to write!

THE HARD PART : This past winter I resigned myself to learn TI Assembler. It took many sleepless nights, but I finally got the hang of it. That's when I realized how much longer each assembly program can be. When I need to qt a string from the keyboard in BASIC I just write INPUT A$. In assembler it takes almost 100 lines of code to do the same thing! Not only that, I also realized that each time I write an assembly program I have to "re-write" the same simple routines! What a waste! What I needed was a set of "portable" utilities that would make writing assembly easier. Ones that would handle the boring stuff like, getting a line of input, printing text on the screen, performing the math factions, etc. Why re-write them every time? That's where the PROGRAMMER'S UTILITY LIBRARY OF SPECIALIZED ASSEMBLY ROUTINES or PULSAR comes in.

PULSAR MAKES IT EASY PULSAR : is a set of 'black box' routines designed to take the drudgery out of assembly programming by performing the tasks usually covered by statements and functions in high-level languages like BASIC, FORTH, PASCAL, etc. By using PULSAR, the programmer can write relatively "bug-free" assembly code in less than half the time it takes to write assembly code from scratch. Of course, this speed in the development stage will be paid for in a slightly slower running time, but in most cases, the delay will be minimal. PULSAR is an attempt to bring the speed of assembly language, the extensibility of FORTH, the modularity of PASCAL and the english-like syntax of BASIC all into one. At present, PULSAR is just a series of routines that are accessed via the BL instruction in TI9900 assembly code. The programs are written using TI Ed/Asm syntax coventions and must follow all the same rules as writing "pure" assembly code. Eventually, PULSAR will have it's own editor and interpreter, thus making a true "programming language." Of course, the interpreter will be written using PULSAR!

WHAT CAN PULSAR DO? : The current version of PULSAR (V 1.2) contains 50 routines covering all the floating point math operations, simple keyboard/screen I/O (print, input, keyboard scan, etc.), graphics commands (CALL COLOR, CALL CHAR, etc.), flow control routines (IF-THEN-ELSE and FOR-NEXT-STEP), and memory manipulation routines (PEEK and POKE, READ-DATA and arrays). In addition, several frequently used values are "pre-defined" to make generating TI9900 code easier and faster.

In most cases, the "syntax" for PULSAR follows the conventions of TI BASIC. This means even those with little assembly experience can learn to write programs very quickly. Advanced programmers may not ever use the more simple routines (the integer math set, for example), but the more complex ones (like the floating point set) will be appreciated by even the skilled TI9900 programmer.

HOW DO I USE PULSAR? : All PULSAR routines are complied into a library file called PULSAR/O. This file contains all fifty routines and the "pre-defined variables." To acess this material the programmer simply uses the TI9900 instruction BL and passes the neccesary data to the routine. Below is an example of the PULSAR version of the TI X-BASIC statement DISPLAY AT:

```
BL   @PRNTAT
DATA ROW,COL,MSG
```

In PULSAR, all data is stored in pre-declared variables. This means that all varaibles are pointers to the data, not the data itself. In the above example the following actual numbers are involved:

```
ROW = >A000      >A000 = >000C
COL = >A002      >A002 = >0010
MSG = >A004      >A004 = (* ASCII chars *)
```

The above informatin tells the PULSAR routine to print the data stored at MSG on the screen row stored at ROW and the screen column stored at COL. This may seem a bit cumbersome at first, but it's easy once you get the hang of it.

It is also important to note that since all PULSAR routines are accessed via the BL instruction (not the BLWP), the routines use the host program reigisters. This means you can't use your workspace reigistes to store values they will most likely be corrupted by the PULSAR routines. It is best to store any important data at a pre-declared address.

Since the PULSAR routines allow for nesting, there is a return stack and a set of loop stacks. These are indexed using R9. Any alteration of R9 could cause the system to crash. Also, R11 is used as the return stack and variable-passing register and must be left alone.

EXAMPLE PROGRAM : The progra we will look at will demostrate many of the routines in PULSAR including converting floating point data to integer, using READ-DATA commands, scanning the keyboard and printing data on the screen. As an aid, I have included a BASIC "translation" of the PULSAR code for those of us who are more familiar with TI BASIC.

Housekeeping : TI9900 code requires that any external references be declared ahead of time. This directs the assembler to search a list in memory of existing routines that may have been written in another program. This is handled with the include file INIT-EA/S. This line *must* be the first actual line in any PULSAR program. This file also contains the workspace and program entry information. All PULSAR programs start with "RUN."

Title Screen : The next lines of code simply clear the screen and print the program title lines. Noe the use of D1, D8, etc. to indicate row and column numbers. the decimal vaules 0 to 40 are "pre-defined" as D0-D40 and are contained in the Housekeeping file called INIT-EA/S.

Read and Print Integers : In PULSAR, the READ-DATA routines are a bit different than in BASIC. You must first position the READ-pointer to the start of the data list. You must also tell the routine the data type it will be accessing. In this case we will be reading integer data starting at the address DLIST1.

PULSAR loops require not only the usual holding variable, start and stop value, but always require a step value (in this case the step value is one).

The PULSAR equivalent of BASIC's READ is GETDAT. Since the info is stored as integer type, we will first convert the data into a floating-point type, then convert the floating-point type ino a string type, then display it on the screen. This is not th fastest way to do it, but it does show off more PULSAR commands!

Read and Display Strings : The next section of code does the same thing with string data. Note that each string starts with a word that contains the length of the string. This is vital to the operation of the PRNTAT routine.

Scan the Keyboard : After printing the data, the keyboard is scanned for any keystroke (BL @KEYCON) and then checks to see which key is hit using the IFTHEN or IFELSE routines. If the QUITKY is hit, the program turns on the interrupts and jumps to the title screen, ending the program. If the REDO key (FCTN8) is hit, the routine is repeated. If some other key is hit, the program simply re-scans the keyboard until a valid key is hit.

FOR THE ADVANCED PROGRAMMER : As you can see, it is relatively easy to write assembly programs using the PULSAR routines. PULSAR is quick and efficient, but does need some imporvements. This is where the advanced assembly programmer comes in.

Routines still needed include disk I/O, string manipulation (SEG$, POS, ASC, CHR$, LEN, etc.), plus access to TI's Bit Map, Multi-color graphics, Sprites, speech and sound capabilities.

The next step in PULSAR development should be the incorporation of the BLWP command in place of the BL. This would allow much easier mix of PULSAR and "pure" assembly language commands.

If you do get interested and begin to write your own routines, be sure to let me know.

Michael Amundsen

```
********************************
*    SIMPLE READ-DATA DEMO     *
*    =====================     *
*     M. Amundsen 3-30-85      *
*                              *
*  Example program using the   *
* "PULSAR" assembly routines.  *
*                              *
* NOTE:                        *
*   The PULSAR master file     *
* MUST be loaded into memory!  *
********************************
* LOAD DEF/REF TABLE
      COPY "DSK2.START-EA/S"
* PUT UP TITLE                          (* "BASIC" VERSION *)
BEGIN  BL   @SCRCLR              100 CALL CLEAR
       BL   @PRNTAT              110 DISPLAY AT(1,8):L1$
       DATA D1,D8,L1
       BL   @PRNTAT              120 DISPLAY AT(2,8):L2$
       DATA D2,D8,L2
       BL   @PRNTAT              130 DISPLAY AT(3,8):L3$
       DATA D8,D2,L3
* READ & DISPLAY INTEGERS
NLOOP  BL   @SETDAT              140 RESTORE "DLIST1" (INTEGER DATA)
       DATA INTDAT,DLIST1
       BL   @FOR                 150 FOR LOOP=10 TO 19 STEP 1
       DATA LOOP,D10,D19,D1
       BL   @GETDAT              160 READ DTEMP
       DATA DTEMP
       BL   @INTFLT              170 FLTEMP=DTEMP (TURN INT TO FL PT )
       DATA DTEMP,FLTEMP
       BL   @NUMSTR              180 STRING$=STR$(FLTEMP)
       DATA FLTEMP,STRING
       BL   @PRNTAT              190 DISPLAY AT(LOOP,8):STRING$
       DATA LOOP,D8,STRING
       BL   @NEXT                200 NEXT LOOP
       DATA LOOP
* READ & DISPLAY STRINGS
SLOOP  BL   @PRNTAT              210 DISPLAY AT(8,15):L4$
       DATA D8,D15,L4
       BL   @SETDAT              220 RESTORE "DLIST3" (STRING DATA)
       DATA STRDAT,DLIST3
       BL   @FOR                 230 FOR LOOP=10 TO 19 STEP 1
       DATA LOOP,D10,D19,D1
       BL   @GETDAT              240 READ STRING$
       DATA STRING
       BL   @PRNTAT              250 DISPLAY AT(LOOP,15):STRING$
       DATA LOOP,D15,STRING
       BL   @NEXT
       DATA LOOP
* WAIT FOR KEYHIT
       BL   @PRNTAT
```

```
       DATA D23,D5,L5
REGET  BL   @KEYCON
       BL   @IFTHEN
       DATA KEYHIT,EQ,QUITKY,EXIT
       BL   @IFELSE
       DATA KEYHIT,EQ,FCTN8,BEGIN,REGET
* END PROGRAM
EXIT   LIMI 2                          310 *TURN ON INTERRUPTS*
       JMP  $                          320 GOTO 320
* VARIABLES
L1     DATA 14
       TEXT 'READ-DATA DEMO'
       EVEN
L2     DATA 14
       TEXT '=============='
       EVEN
L3     DATA 8
       TEXT 'NUMBERS:'
L4     DATA 8
       TEXT 'STRINGS:'
L5     DATA 18
       TEXT 'PRESS REDO OR QUIT'
LOOP   DATA 0
FLTEMP BSS  8
STRING BSS  20
DTEMP  BSS  20
DLIST1 DATA 1,2,3,4,5,6,7,8,9,0
DLIST3 DATA 1
       TEXT 'A'
       DATA 2
       TEXT 'BB'
       DATA 3
       TEXT 'CCC'
       DATA 4
       TEXT 'DDDD'
       DATA 5
       TEXT 'EEEEE'
       DATA 6
       TEXT 'FFFFFF'
       DATA 7
       TEXT 'GGGGGGG'
       DATA 8
       TEXT 'HHHHHHHH'
       DATA 9
       TEXT 'IIIIIIIII'
       DATA 10
       TEXT 'JJJJJJJJJJ'
       EVEN
       END
```

```
100 ! ******************
110 ! *   CATASCRIBE    *
120 ! *   JIM O'NEIL    *
130 ! ******************
140 !
150 ! RUN IN XBASIC
160 !
170 ! PRINT FILE FROM TEXT
180 ! FORMATER TI-WRITER
190 !
200 ! *WARNING* PROGRAM MAY
210 !  RUN FOR SEVERAL HOURS
220 !  BASED ON NUMBER OF
230 !  FILES ON YOUR DISKS;
240 !  LIMIT=625 FILES ON
250 !  50 DISKS.
260 !
270 ! *WARNING* BE SURE THAT
280 !  DISKS CONTAIN NO
290 !  LISTING ERRORS;
300 !  CHECK WITH PROGRAM ON
310 !  PAGE 41, DOC.PHP1240
320 !
330 G$="FIRST" :: CALL CLEAR
340 OPTION BASE 1 :: DIM DNM$(50),DDTA(50,2),PNM$(625),PDTA(625,4),TYPE$(5)
350 FOR X=1 TO 5 :: READ DTA$ :: TYPE$(X)=DTA$ :: NEXT X
360 DATA DIS-FIX,DIS-VAR,INT-FIX,INT-VAR,PROGRAM
370 CALL CLEAR :: FOR NM=1 TO 50 :: DISPLAY AT(12,1):"**PLACE ";G$;" DSK IN DRIV
E" :: DISPLAY AT(14,3):"<ENTER> TO PROCEED"
380 DISPLAY AT(16,3):"<FCTN>6 TO STORE/END" :: DISPLAY AT(20,1):"**";STR$(NM-1);
" DISKS CATALOGED"
390 DISPLAY AT(22,3):STR$(CNTR);" FILES LISTED"
400 CALL KEY(0,K,S):: IF K=13 THEN 410 :: IF K=12 THEN 500 ELSE 400
410 CALL CLEAR :: DISPLAY AT(12,2):"STAND BY.." :: OPEN #1:"DSK1.",INPUT ,RELATI
VE,INTERNAL
420 INPUT #1:DNM$(NM),N,DDTA(NM,1),DDTA(NM,2)
430 CNTR=CNTR+1
440 INPUT #1:PNM$(CNTR),PDTA(CNTR,1),PDTA(CNTR,2),PDTA(CNTR,3)
450 DX$=("  "&STR$(CNTR)):: PNM$(CNTR)=PNM$(CNTR)&SEG$(DX$,LEN(DX$)-2,3)
460 IF LEN(PNM$(CNTR))=3 THEN 490
470 PDTA(CNTR,4)=NM
480 GOTO 430
490 CALL CLEAR :: G$="NEXT" :: CLOSE #1 :: CNTR=CNTR-1 :: NEXT NM
500 CALL CLEAR :: DISPLAY AT(12,2):"INSERT USER DISK..<ENTER>"
510 CALL KEY(0,K,S):: IF K=13 THEN 520 ELSE 510
520 DISPLAY AT(14,2):"DEVISE NAME" :: DISPLAY AT(14,16):"DSK" :: ACCEPT AT(14,19
)SIZE(1)BEEP:Q$
530 DISPLAY AT(16,2):"FILE NAME" :: ACCEPT AT(16,16)SIZE(10)BEEP:QA$
540 TOTQ$="DSK"&Q$&"."&QA$
550 CALL CLEAR :: DISPLAY AT(12,2):"**STAND BY   SORTING.." :: DISPLAY AT(20,1):
"**LOOPS="
560 FOR X=1 TO CNTR-1
570 DISPLAY AT(20,9):STR$(X-1)
580 FOR Y=1 TO CNTR-X
590 IF SEG$(PNM$(Y),1,1)<SEG$(PNM$(Y+1),1,1)THEN 630
600 IF SEG$(PNM$(Y),1,1)=SEG$(PNM$(Y+1),1,1)THEN 630
610 FLAG=FLAG+1
620 SW$=PNM$(Y):: PNM$(Y)=PNM$(Y+1):: PNM$(Y+1)=SW$
```

```
630 NEXT Y
640 IF FLAG=0 THEN 660 :: FLAG=0
650 NEXT X
660 DISPLAY AT(12,15):"STORING.."
670 OPEN #1:TOTQ$,DISPLAY ,VARIABLE 80
680 PRINT #1:".FO PAGE %"
690 PRINT #1:CHR$(14);TAB(22-INT(LEN(QA$)/2));QA$
700 PRINT #1:CHR$(14);TAB(16);"DISKS LISTED"
710 PRINT #1:".HE ";QA$;" CATALOGE PRINTOUT"
715 CALL MBP(DATE$,TIME$)
720 FOR X=1 TO 3 :: PRINT #1:"" :: NEXT X
730 FOR X=1 TO NM-1
740 PRINT #1:TAB(12);STR$(X);TAB(15);DNM$(X);TAB(30);"AVAILABLE=";STR$(DDTA(X,2
);TAB(50);"USED=";STR$(ABS(DDTA(X,2)-DDTA(X,1)))
750 NEXT X
760 FOR X=1 TO 3 :: PRINT #1:"" :: NEXT X
770 PRINT #1:CHR$(14);TAB(15);"PROGRAMS/FILES"
780 FOR X=1 TO 3 :: PRINT #1:"" :: NEXT X
790 PRINT #1:TAB(12);"FILE NAME";TAB(30);"DISK NAME";TAB(49);"SIZE";TAB(55);"TY
E";TAB(70);"P"
800 PRINT #1:TAB(12);"************";"************";"************";"************
;"***********"
810 FOR X=1 TO CNTR
820 Z=VAL(SEG$(PNM$(X),LEN(PNM$(X))-2,3)):: PRGNM$=SEG$(PNM$(X),1,LEN(PNM$(X))-
)
830 PRINT #1:TAB(12);PRGNM$;TAB(30);DNM$(PDTA(Z,4));TAB(49);STR$(PDTA(Z,2));TAB
55);TYPE$(ABS(PDTA(Z,1)));
840 IF ABS(PDTA(Z,1))=5 THEN 860
850 B$=" "&STR$(PDTA(Z,3)):: PRINT #1:TAB(64);SEG$(B$,LEN(B$)-2,3);TAB(70);""
860 IF PDTA(Z,1)>0 THEN 870 :: PRINT #1:TAB(70);"Y"
870 NEXT X
880 CALL CLEAR :: PRINT "**FILE COMPLEATED":"  FORMAT ""DSK";Q$;".";QA$;"""";""
5000 SUB MBP(CA$,IA$)
5010 CALL PEEK(-31156,DATE,DUM,MONTH)
5020 DATE$=STR$(DATE)
5030 IF DATE>9 THEN 5050
5040 DATE$="0"&DATE$
5050 CA$=STR$(MONTH)&"/"&DATE$&"/85"
5060 CALL PEEK(-31162,MINU,DUM,HR)
5070 MIN$=STR$(MINU):: IF MINU<10 THEN MIN$="0"&MIN$
5080 IA$=STR$(HR)&":"&MIN$
5090 SUBEND
```

This came off the Wichata Tibbs. If you look at the coding you"ll see a subroutine for the MBPP card. The program runs file without the card tho.

We downloaded this with the new FREEWARE program Fast-Term now in the library.

PROGRAMMING TIPS : Technique for switching from JOYST to KEY: "Did you ever want to change a game from joystick to keyboard or vice-versa? Here is an example of a change done in the game 'Aardvark' found in June 83's 99'er Home Computer Magazine. Use this section of 'Aardvark' as an example to chang other programs:

Aardvark is like this...

```
630 CALL JOYST(1,X,Y)
640 IF X=-4 THEN FV-FV-1 :: GOTO 780
650 IF X=4 THEN FV=FV+1 :: GOTO 840
660 IF Y=4 THEN FH=FH-1 :: GOTO 900
670 IF Y=-4 THEN FH=FH+1 :: GOTO 970
```

Aardvark can change to this.....

```
630 CALL KEY(0,K,S)
640 IF K=83 THEN FV=FV-1 :: GOTO 780
650 IF K=68 THEN FV=FV+1 :: GOTO 840
660 IF K=69 THEN FH=FH-1 :: GOTO 900
670 IF K=88 THEN FH=FH+1 :: GOTO 970
```

$NOTES$

A. This is in EXTENDED BASIC.
B. In CALL JOYST, the X and Y are the variables for the position of the joystick. 4 or -4 is always returned. (see CALL JOYST in the User's Reference Manual)
C. In CALL KEY, we are checking for the K or Key that is pressed. The numbers 83,68,69, and 88 are the ASCII codes for the S, D, E, and X (arrow keys).
D. Always use the same logic and variables found in the program. It makes it easier. For example, the GOTO's are the same; FV and FH logic are the same; and even the line numbers can be the same.
E. Each identical line number is doing the exact same logic. For example, in line 640, the X=-4 is left on the joystick, while K=83 is S (left) on the keyboard."

To change the screen color for the programming mode: TYPE 10 REM and hold down the CTRL U until the cursor has covered about three lines. Then TYPE 10 and press FCTN X. Now press the space bar once or twice. Next you press the FCTN and 4 keys. This trick is for EXTENDED BASIC. Now you will be able to pogram with a different screen color than cyan. In BASIC, this trick will change the border color only.

JOYSTICKS : Have you ever had difficulty finding which joystick is to be used in a program? If the program is written for joystick #1 then #2 won't work. Or if the program is written for joystick #2 then #1 won't work.

You can use this little trick in your programs to solve the problem. With this program all you do is push the Fire Button on either joystick and the computer will then remember which joystick you are using.

```
100 CALL KEY(1,J1,STATUS)
110 CALL KEY(2,J2,STATUS)
120 IF J1+J2 <>17 THEN 100
130 JS=INT(J1/18+J2/9+1)
```

With this little program prior to the use of CALL JOYST(JS,J,STATUS) in your own program, the computer will respond to whichever joystick you were using when you pressed the fire button!

SHARPER PICTURE : As you know, the screen color while running a built-in BASIC program is green (blue in EXTENDED BASIC). If you have a color TV this is fine. But if you only have black-and-white, then try this trick for a sharper picture. Add the following statement to the beginning of your program: 10 CALL SCREEN(15) This will disable the color-generating circuit in the 99/4A and removes the patern of vertical lines often seen on black-and-white TV. It also increases the sharpness of the characters.

ADVANCED PROGRAM TIP : You must have the MINI MEMORY or the EDITOR/ASSEMBLER to use this programming tip. The program uses some memory addresses that most programmers have not yet discovered.

Using the MINI MEMORY or the EDITOR/ASSEMBLER type in the following program from TI BASIC (not assembly). You will find that pressing the following keys will unviel some secrets: 'N' for NORMAL MODE, 'C' to CLEAR THE SCREEN, 'T' for TEXT MODE, 'M' for MULTICOLOR MODE (each character is a 4X4 block), and 'B' for BIT MAP MODE (99/4A only).

For you advanced programmers, you now know the secret of getting 40 columns on the screen (not the normal 32). You will

also eventually figure out how to address every pixel on the screen. As you can see, contrary to most liturature, it can be done from basic.

```
90 PRINT "PRESS:";"'N' FOR NORMAL MODE : 'C' FOR CLEAR SCREEN : 'T' FOR TEXT MODE" : 'M' FOR MULTICOLOR MODE"
100 PRINT "'B' FOR BIT MAP MODE"
110 CALL KEY(3,G,S)
120 IF G<>78 THEN 140
130 CALL POKEV(-32760,0)
140 IF G<>67 THEN 160
150 CALL POKEV(-32352,0)
160 IF G<>84 THEN 180
170 CALL POKEV (-32272,0,"",-30945,0)
180 IF G<>77 THEN 200
190 CALL POKEV(-32280,0)
200 IF G<>66 THEN 220
210 CALL POKEV(-32766,0)
220 GOTO 90
```

EXTENDED BASIC PRESCAN : Have you ever wondered why it takes so long for a program to run after you type in RUN? The pause is the time the computer takes to pre-scan your program to set up memory space for variables, arrays, data, and subpograms. The computer has to go over each line and reserve memory space. This takes a lot o time because it must proceed through each instruction, perform the appropriate functions, and establish variable values. The time required to pre-scan depends on te length of the program. There is a way to reduce this wait time using EXTENDED BASIC. This is just one of many reasons we recommend EXTENDED BASIC as the best first purchase you can make after buying the 99/4A.

Unlike most of our tips, this one is documented in the EXTENDED BASIC manual, but few seem to take advantage of it. The commands are PRE-SCAN OFF and PRE-SCAN ON. These commands allow you to control which instructions will not be pre-scanned. In a program, only those instructions which contain the first reference to the variables need to be pre-scanned. Therefore, you will find that many program lines don't require a pre-scan. A 23K byte program that would normally take 31.5 seconds to start RUNning will only take 9.5 seconds by using the pre-scan comands. As an added hint, think about the time you could save by having the program reset itself by RUNning itself and using the pre-scan commands. Execution speed is greatly increased.

Unfortunately I have been unable to try any of the above hints. It would be nice to hear from the rest of you out there on how any of the above work or how you used them.(Seeing your name in print is great) Well that should fill my column, the rest of this stuff can wait until next month.

Don't use character sets 15 and 16(ascii codes 144-159) unless you really need to. And if you use multiple colons :: as print separators, put a spece between then : : : Then, when you get Extended Basic, your programs wil run without modification in Extended Basic, and usually faster and better.

```
100 REM - TIGERCUB WIPE FROM OUTSIDE IN
110 CALL HCHAR(1,1,90,768)
120 FOR R=1 TO 12
130 CALL HCHAR(R,R,32,34-(R*2))
140 CALL HCHAR(25-R,R,32,34-(R*2))
150 CALL VCHAR(R+1,R,32,26-(R*2))
160 CALL VCHAR(R+1,33-R,32,26-(R*2))
170 NEXT R
```

To get the computer to read the CALL KEY input as upper case letters, even if the Alpha Lock is up, just use key-unit 3 : CALL KEY(3,K,ST)

This small batch of tips came from any number of unknown places, they have all been floating around for quite awhile now, so many of you are probably saying "why is she printing that old stuff ?" Well not everyone has had their computer as long as you, or maybe not everyone has had access to the amount of information that do have. The point is I'm sure there are as many people who are greatful for this information as there are those who are glad to see the Forth & Assembly articles.

LITTLE BYTES

by

Gene Thomas

Program conversions can be fun. Many programs written for computers other than the TI can be converted to TI basic without driving you mad.

As long as there are not a lot of PEEKS & POKES you can convert many programs written for the C-64,ATARI,APPLE,IBM, and others. If the program has PEEKS & POKES you'll have to see it run so that you know exactly what it is doing. Then you can devise ways to simulate the same actions on the TI. But, in general, stay away from those.

For the most part it is a matter of command alterations. Here is a list of commands which virtually never need to be changed.

FUNCTIONS (ABS, ASC, Etc.)		
CHR$	DATA	DIM
END	GOTO	GOSUB
INT	LET	PRINT
READ	REM	RESTORE
RETURN	SQR	STR$
DEF	IF-THEN	VAL

In general, here are conversions for some other commands.

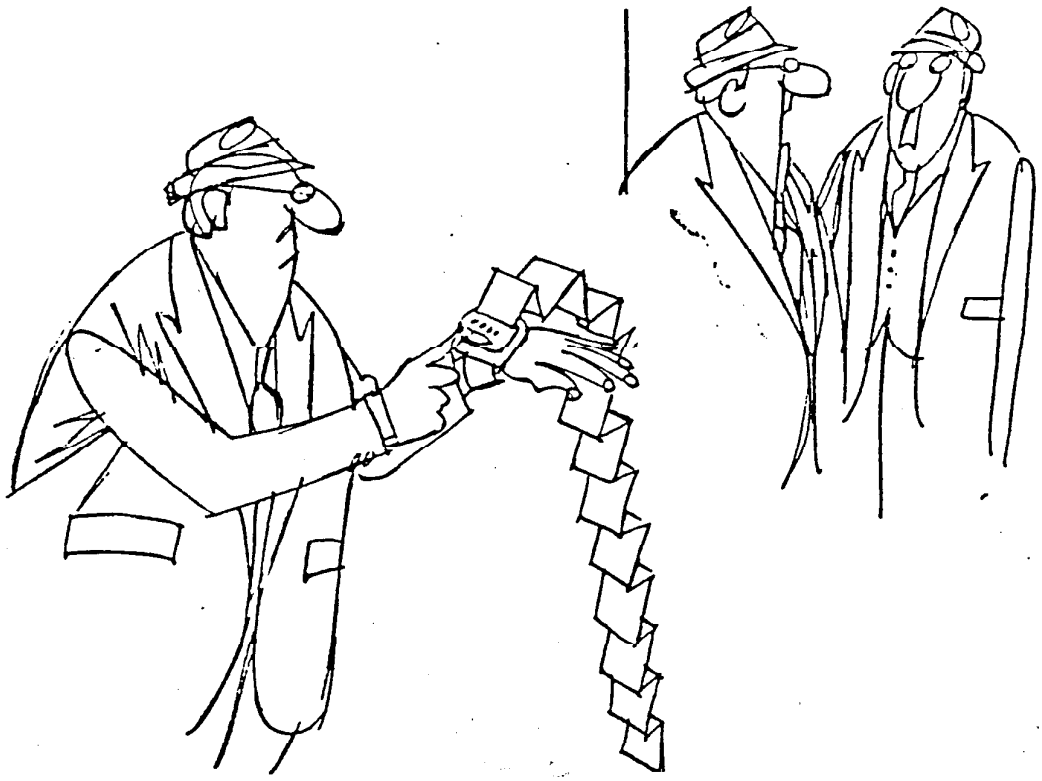
CLS	CALL CLEAR
CLEAR	NEW(Not used within TI programs).
CHANGE	ASC & CHR$
CINT, FIX	INT
CLG	LOG (Base 10).
CLOAD	->OPEN CASSETTE FILE & LOAD
COLOR	CALL COLOR
CSAVE	OPPOSITE OF CLOAD
DEFINT	DEF >INT< (Declare your def statemenet numerals to be integers).
DEFSNG	IGNORE IT
DEFSTR	IGNORE IT
SET, DOT	CALL HCHAR, VCHAR
EQ	= (Equals)
FRE	IGNORE IT
GET, INKEY$	INPUT, CALL KEY
GE	=> (Greater than or equal to)
GOSUB-OF	ON-GOSUB
GOTO-OF	AS ABOVE
INSTR	SEG$
LLIST, LPRINT	LIST, PRINT TO PRINTER
MID$	SEG$ (Exactly)
LEFT$(S$,N)	SEG$(S$,1,N)
RIGHT$(S$,N)	S=LEN(S$/(N+1)) :: SEG$(S$,S,N)
RND(N)	INT(RND*N)+1
PRINT USING	PRINT, DISPLAY USING
SLEEP	FOR-NEXT DELAY LOOP
STRING$(10,65)	S$=RPT$(65,10) :: PRINT S$
WAIT	SEE SLEEP
?	PRINT

With this as a guide you can figure out most of the others. Some are obviously ok: IF-THEN-ELSE, FOR-TO-STEP-NEXT, ECT. HAVE FUN!

This months program is a little bit of music appreciation. As usual, send me a blank tape or disk if you dont want to key it in, or call me if you have a modem.

Gene Thomas
7705 Apache Rd.
Little rock, AR 72205
PH: 225-0625

BYE



"...THAT'S THE LATEST... A WRIST COMPUTER WITH PRINT OUT..."

J*U*G*S NEWSLETTER
Jacksonville TI-99/4a Users Group
P O Box 525
Jacksonville, Ar 72076

LITTLE ROCK AR
PM
SEP 30
1985
72201

USA 22

JACKSONVILLE USERS GROUP
JUG
TI-99/4A

MIAMI COUNTY AREA 99/4A
P.O. BOX 1194
PERU IN.
46970