WORDPLAY SEPTEMBER 1987

The PUNN Newsletter-Portland, Oregon

What's Inside

Club Officers

President	Keith Fast	777-1531
Vice-President	Dale Kirkwood	646-4354
Secretary	Don Barker	223-1749
Treasurer	Mike King	357-4413

PUNN Staff

Librarians	Ron Mayer	232-7363
	Walt Morey	239-5105
	Jim Thomas	284-2425
Hardcopy	Mike Calkins	636-1839
Program Chairman	Ted Peterson	244-1587
Workshop Chairman	(open)	
Membership Chairman	Terry Priest	649-9583
Newsletter Editor	Charles Ball	639-0466

16576 SW Matador Lane-King City, OR 97224

Ass't Editor	Dan Hawes	620-9725

BBS Committee

Chairman:	Al Kinney	640-5860
	Ron Mayer	232-7363
	Mike King	357-4413

BBS Phone Number 503/233-6804

From the Editor

Each month when I have this opportunity to sound off, I wonder what will I say this month?

As I look back over the years I remember when I had a part in producing the school newspaper. I assure you it was many years ago. We set the type piece by piece and it took the work of many and consumed literally hundreds of hours just to produce a few pages. Later and still a good number of years ago I had the editors job for a local service club. Then I used my trusty typewriter and a mimeograph machine and somehow was able to come up with bulletin that did the basic job of getting the news out to the members.

As some of you know I retired from the printing business a couple of years ago. I watched the computer revolutionize the printing process. I wish some of you could see first hand what is happening. There have been more changes in the printing industry in the last 10 years than from the origination of printing so it seems.

When I look back over the years that I was in the business and compare the relative ease of communication today with the way things were done when I was much younger, I marvel! I wonder sometime if we fully appreciate this fantastic medium we are working with. To be able to print out and communicate accross these many miles is something I didn't even dream of in those earlier years.

Who knows exactly what tomorrow will bring. One thing I think we can be sure of it that tomorrow will make today seem obsolete just as todays modern miracle makes my early years seem very old indeed.

Chuck Ball

News & Views

If you missed the picnic last month, too bad, it was great- - - There was no board meeting in August, but board members should plan on one in September- - - Ted Peterson has a good program planned for the September meeting, it's on word processing-read about it inside this bulletin- - -Bill McCabe is one of our newer users of the BBS-he got his system up and running thanks to Mike King- - -Don Barker is busy painting his house and reports that any one that would like to volunteer some help would be welcome- - -Rich and Sue Hill are organizing the carpooling for the Seattle TI-Faire- - -If you're planning to go and want to share a ride contact them- - -The TI-Faire is being held at the Seatac Holiday Inn in Seattle-the date is Saturday September 26th. from 10am until 5pm-admission to the show is $3.00- - -We are trying to find a bug in the Multi-Column print program promised for WordPlay-we'll try to find it and publish the program next month- - -In the meantime this months issue is filled with goodies- - - We still need your help in producing WordPlay- - -Why not send something in-A program, an article, most anything that would be of interest to the members.

September Program

Our Sept. program will be a complete program on "Word Processing". The speaker will be your program chairman Ted Peterson.

He will show you how to type letters, newsletters and other documents on your computer. He will cover all the different methods of typing from the most simple that do not require a program to the more complex that require intricate programing. Some of these methods only require a computer, a recorder, and some type of printer. Others need a module, computer and some type of printer. Another type only needs a computer and a printer.

He will then explain the types that require a complete system. In addition to word processing programs he will show you how to correct your spelling. One of the programs will let you correct the spelling after you have typed in your document. Another one will correct your spelling continuously while you are typing and will show you all the words that are spelled wrong and a list of words that may work.

Another thing that he will take up is breaking up words so the lines in BA writer, TI Writer, etc do not have large blank sections in some of the lines. Ted is also going to demonstrate a program that will print out copy in 2 columns to the page.

Finally he will show you a way to make your TI Writer module work without any waiting time between sections. There is a way.

Most of the programs that will be shown are in the PUNN libary and we will have them on a disk for sale at this meeting. Some of you may want them.

We need to know what you want to see as programs at our meetings. Let Ted know. Maybe an item that you want demonstrated or the program you want explained is something that other members would also like to see. You can call Ted with a suggestion for a program. You'll find his phone number on the front page of this bulletin.

Club Library

Don't forget your club Library! It is there to serve you and just loaded with programs. You'll find games, utilities and much more.

Our librarians are on duty each meeting night. A list of the various porgrams is displayed and you can even obtain a disk with all the programs so you can look over the various listings at your leisure.

Take advantage of this library. It will make your computing day!

We're bombarded with so much useless information these days. I mean do I really have to know our waiter's name is Bruce?

Tiny Gram

What is a tinygram you may ask? The definition of a tinygram is a program which can be listed on one screen. Here is a musical tinygram by Jim Peterson. It was taken from the Dallas 99 Interface. It's called, 'The Wildwood Flower'.

And, what's more is that you can change the program to play your own musical tune. This program pus a two-octive scale in a 25-element DIMension. Take the tune you want to hear and break it down, note by note. Then use the values below to represent their respective notes.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

C C# D D# E F F# G G# A A# B C C# D

16 17 18 19 20 21 22 23 24 25

D# E F F# G G# A A# B C

Put these numbers into the DATA statements. Each number is approximately one quarter long. To make the note longer, play it twice. Finally, count all your numbers and alter the FOR-TO statement in line 3.

```
1 !#######DULCIMER###########
#########A TINYGRAM###########
#######BY JIM PETERSON########
2 CALL CLEAR :: DIM S(26)::
F=262 :: FOR N=1 TO 25 :: S(
N)=INT(F*1.059463094^(N-1)):
: NEXT N :: READ N :: C=S(N)
:: D=S(N)
3 RESTORE 7 :: FOR J=1 TO 63
 :: GOSUB 5 :: NEXT J
4 U=U+2 :: CALL SOUND(-200,S
(N),U,C,U,D,U):: IF U>27 THE
N U=0 :: GOTO 3 ELSE 4
5 READ N :: CALL SOUND(-500,
S(N),0):: CALL SOUND(-500,S(
N),0,C,9):: CALL SOUND(-500,
S(N),0,C,9,D,19):: D=C
6 C=S(N):: RETURN
7 DATA 5,6,8,8,10,13,5,5,6,5
,3,3,5,3,1,1
8 DATA 5,6,8,8,10,13,5,5,6,5
,3,3,5,3,1,1
9 DATA 8,13,17,17,17,15,13,1
3,8,8,10,10,13,10,8,8
10 DATA 1,1,1,3,5,5,8,5,3,3,
5,3,1,1,1
```

DM 1000 Bug

DM 1000 - - has an annoying little bug if you happen to own a CorComp disk controller. When DM1000 formats disks in double density, it put 16 sectors/track on the header, even though it formats 18 sectors/track. This is all very fine and well if you keep it on a CorComp controller, for the reason the CorComp never even heard of 16 sectors, so it doesn't care what the header says. However, if you send the disk to someone who has a MYARC disk controller, then the MYARC controller looks at the header and sees "16 sectors per track". So it reads the disk based on that information.. But it's 18 sectors per track! So, the MYARC card reports a blank disk. Now a fix has been developed for that situation. If you have version 3.5 of DM1000 edit the first sector of the MGR1 file. At byte 216, you should see (in hex) 10 00 02 D0 00 5A. Change the 10 to 12. Write the sector back to disk, and you should aleviate this problem. Earlier versions of DM1000 may also contain this problem and you should be able to find the sector in MGR1.

How to Load a Disk

We have published information on loading from disk in prior issues but it seems from questions we hear from members that another article on this subject is warranted.

Disk files that can be loaded directly into the computer can be found in the following forms.

PROGRAM
INT/VAR 254
DIS/VAR 163
DIS/VAR 80
DIS/FIX 880

Any other file format represents a data file which cannot be directly loaded, but requires a program already in the computer. For example: INT/FIX 108, INT/VAR 128, DIS/VAR 64.

PROGRAM

This is the most common file and the vast majority are TI Basic or XBasic programs. Many TI Basic Programs load and run correctly from XBasic but not vice-versa. However, if after loading the program file into XBasic and you get a bad value error when you attempt to run the program, you need to reload into Basic. The bad value error is caused by the use of CALL CHAR above 143, which is not allowed in XBasic.

If you attempt to load an XBasic Program into Basic it will seem to load properly, but when you run it you will get a FOR-NEXT error message. Listing the line will produce a screen of unrecognizable characters.

Occasionally a program file will not load in either version of Basic, producing an I/O error 50 when you attempt to do so. These files are likely to be Assembly Language programs that need the Editor/Assembler module to make them load. Other programs are specialized and require the module they work with; such at "Adventure", "Personal Record Keeping", "Tax-Investment Record Keeping", etc.

INT/VAR 254

These files are normally long XBasic programs that load and run in the normal way. They usually exceed 45 sectors and require memory expansion. They must be loaded with XBasic and cannot be saved to tape.

DIS/VAR 163

These files represent an XBasic subroutine in merge format. They can be merged into a program already in memory. To load these files, type merge DSKn.filename and press ENTER. The program that you LOAD will become a part of the program already in memory. One thing to remember is that any line number already in the computer will be replaced if the merged program uses a similar line number. The merge option cannot be used in Basic.

DIS/VAR 80

These are text files which can be read from the screen, edited, and printed using TI-WRITER or one of the clones. This type of file often accompanies a complicated program with documentation on how to use the program. Quite often they will be on the same disk and have a name such as TAX/DOC which would be the instructions for the program TAX. And of course they are used in letter writing, manuscript production and editing articles and programs for FUNN.

DIS/FIX 80

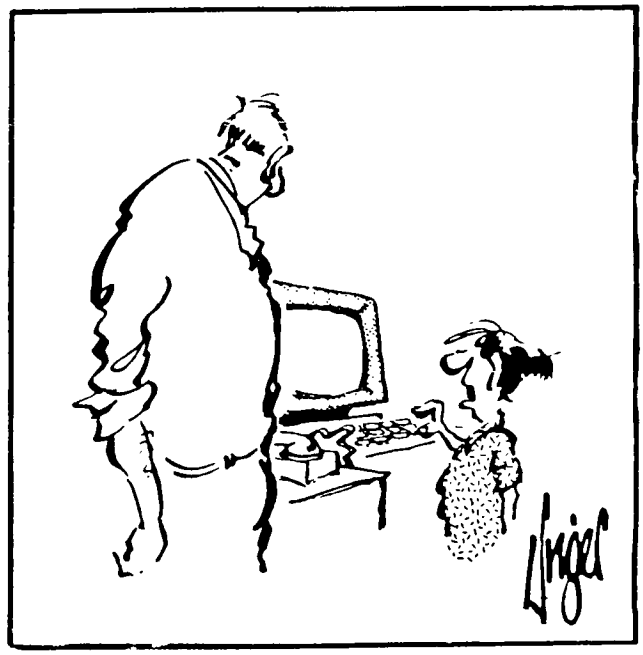
These are assembly language programs which must be loaded and run with the Editor/Assembler or Mini-Memory modules. With the Editor/Assembler try Option #3. If this doesn't work then try Option #5. Use #3 to load from the Mini-Memory module.

Tips for Multiplan

Here are a couple of tips for those of you who use Multiplan.

1. Tired of seeing just the same old white characters on a blue background? Well try this. After you have chosen the option to load Multiplan from the main menu do not press enter. Instead press the space bar to see the 12 choices for screen color. Each time you press the space bar the colors change. Your color choice will remain in effect only for the Multiplan session that follows. When your color choice is on the screen, press Enter. At this point the program disk files will be loaded from your diskette.

2. The time it takes to initialize Multiplan and the response time when it is working with the OVERLAY file is effected by the location of the files on the disk. You can load the files in the desired order by copying them one at a time to a newly initilized disk named TIMP. The best order seems to be: OVERLAY, MPHLP, MPCHAR, MPDATA, MPINTR, and then MPBASE.



When you go to the bank you'll have $50,000!

Sector Editors

Over the past several years a number of different programs have emerged that will allow you to change data on the various sectors of your disk. Advanced Diagnostics, Disk Master and Sector/Edit are but a few of the programs available. What they *all have in* common is the ability to look at data on the disk and change it.

If you decide to use one of these programs on any of the data on one of your files, it's a good idea to do it on a copy so if anything goes wrong you won't loose valuble data.

One example of the use of a sector editor (S/E) is in correcting a problem that many users have had with PRBASE in accessing the default printer - PIO. After looking at the program file PRB:1, I noted that the printer default is PIO/1 - designating the #1 PIO output. That was as expected. However I also noted that there were a number of blank spaces following the PIO/1, indicating that either the printer designator had originally been longer and had been changed with an S/E or the original assembly source code had specified a longer default but had not been changed to reflect the actual one. In any case the byte in hexidecimal (hex) immediately before the PIO/1 (>12) did not agree with the length that it is, >05. Upon changing the >12 to >05 the default printer designation now works for my system.

It is also possible to change the PIO/1 to any other default as long as the name entered does not exceed the existing blanks. For example, the PIO/1 could be overtyped with RS232.BA=1200.PA=E, with the stipulation that the byte immediately preceeding the name is changed to >12 (decimal 18) to tell the system the number of characters to read for the default name. That is the problem with the existing PIO default. The system is reading a PIO/1 name along with the blank characters, however there is not DSR routine with a matching name (with the blanks after the PIO/1). Therefore the system hiccups unless the default is erased, forcing the system to count the actual number of characters that are typed from the keyboard for whatever printer setup is necessary.

Another example is a more general one, and is illustrated again by PRBASE and FastTerm. I run many of my most used utilities on my ramdisk and I prefer to keep my RD designated as DSKx where x is higher than any of my floppy disk drives. (DSK4 for my 3 disk set-up). This causes a problem in several ways. The first is in the case of an interactive system of programs that specify a certain disk drive to access for support files, such as character sets (CHAR type files). In fact, this becomes a double whammay for the RD. Not only is the DSK number incorrect, but many of the character files are still CHAR1 or CHAR2, thus colliding with existing character sets for TIWriter or whatever else is on the specified drive.

With an S/E and some browsing through the program files to locate the existing specific designator, e.g. DSK1.CHARA1, it is possible to use the S/E to change just the CHARA1 to CHARA6. In this case not changing the length of the program name. (It's still 6 characters long). The only thing necessary to do is change the name of the correct CHARA1 file and copy it to the disk that contains the other programs for the utility set. This enables the programs to co-reside on a single disk drive and have the correct character sets selected for the utility in use.

In a similar fashion it is possible to change the disk specifier so the loading program will look at the drive which actually contains the sought for program. PRBASE serves to illustrate the changes which are necessary. PRBASE files are loaded from a floppy which must be named PRBASE as the full file specifier is DSK1.PRBASE,PRB:1. This set-up forces the system to search the attached disk drives until it finds one named PRBASE. Convenient if one uses a changing set-up, but not so if one wants to run the files from a RD or from a utility disk which contains more than just PRBASE files. First the S/E is used to change the full specifier to DSK4.PRB:1. (This assumes that the file name itself is not to be changed). The final change which needs to be made is the byte which immediately preceeds the DSK. In this case it will read >0F (decimal 15) and needs to be changed to >09 (decimal 9) to tell the system the length of the name of the disk specifier and the program.

As long as one keeps in mind the need to change the byte which specifies the length of the name of the disk specifier and the file name, and to not use file names so long that the name exceeds the available space, these changes will work.

This should alleviate those problems that I have experienced. One particular problem comes to mind in which I had formatted a disk and then taken it to another person to get some information copied. That person used a MyARC disk controller. After using it on my system I had a lot of problems with garbage. When I looked at it with a sector editor I found the 17th and 18th sector or each track were not used - no doubt because the MYARC controller looked at the sector information and only used 16 sector per track!

I hope that this will help others who have experienced unusual problems when going from one type of controller to another.

(Rudy Johnson-South. Nevada Users Group)

Design Your Own Screen

```
100 CALL CLEAR
110 CALL SCREEN(5)
120 FOR I=2 TO 8
130 CALL COLOR(I,16,1)
140 NEXT I
150 CALL CHAR(43,"FFFFFFFFFFFFFFFF")
160 CALL CHAR(44,"7E3F1F0F070301")
170 CALL CHAR(45,"80C0E0F0F8FCFEFF")
180 PRINT "     + + +++ + + +++"
190 PRINT "     + + + + + + + +"
200 PRINT "     +++ + + + + +++"
210 PRINT "      +  + + + + +,-"
220 PRINT "      +  +++ +++ + +"
230 PRINT
240 PRINT "     +++ + +++ +  +++"
250 PRINT "      +  + + +  + +"
260 PRINT "      +  + + +  + +++"
270 PRINT "      +  + + +  + +"
280 PRINT "      +  + +  ++ +++"
290 PRINT
300 PRINT
310 PRINT "  YOUR CREDIT LINE HERE": : : :
320 FOR I=1 TO 3000
330 NEXT I
340 CALL CLEAR
350 PRINT "PROGRAM STARTS HERE..."
```

This little program will lets you design your own title screens. You can use it for any new program that you have written or modify the title of an older program.

The program as described changes "+" into a solid character; "," into a ; and "-" into a .

The "," and the "-" enable you to make a slant on the "R" in the title shown.

You could of course modify and find many other patterns and uses with a little work of your own.

This program should be merged in front of your program.

```
*****************************************
*                                       *
*    Murphy's Rule:                     *
*                                       *
*          Leakproof seals will -       *
*      Self starters will not -         *
*        Interchangeble parts won't.    *
*                                       *
*****************************************
```

You can Improve the Quality of this Newsletter. Articles's Needed!

Another Loading Program!

What! Another Loader Program? Well just hold on. We have seen many load programs over these past years. They all do pretty much the same basic thing. You put a load program on your disks and then when you want to load a particular program, you insert the disk and the load program takes over listing the various programs and files on the disk with either numbers or letters for your selection.

That's exactly what this program does -- but more too! It also allows you to catalog the contents of the disk in a condensed print-out that you can paste on your disk jacket. In addition you can delete any unwanted file.

The program takes up only 10 sectors -- many we have seen use from 5 or 6 up to 20. You might want to type in this program and give it a try. It is devoid of fancy graphics to save space but it does work quite nicely.

Like all the programs published in your WORDPLAY it will be available in the FUNN Library if you don't want to take the time to type it in.

```
10 CALL CLEAR :: DISPLAY AT(
3,6):"RUN AUTOBOOT? Y" :: AC
CEPT AT(3,20)BEEP SIZE(-1)VA
LIDATE("YN"):XX$ :: IF XX$<>
"Y" THEN END
100 X$="1"
110 DIM A$(97),H(98),J(97),K
(97)
120 CALL INIT
130 FOR I=1 TO 5 :: READ TYP
E$(I):: NEXT I
140 DATA "DIS/FIX","DIS/VAR"
,"INT/FIX","INT/VAR","PROGRA
M"
150 CALL LOAD(-31806,16)
160 OPEN #1:"DSK1.",INPUT ,R
ELATIVE,INTERNAL
170 INPUT #1:A1$,U,U,V
180 Q,R,S=0
190 GOSUB 440
200 Q=Q+1
210 INPUT #1:A$(Q),H(Q),J(Q)
,K(Q)
220 IF LEN(A$(Q))=0 OR Q=97
THEN 230 ELSE 200
230 S=S+1 :: R=R+1
240 IF ABS(H(S))=5 THEN B$="
  :" ELSE B$=" "&STR$(K(S))
250 T$=TYPE$(ABS(H(S)))&SEG$
(B$,LEN(B$)-2,3)
260 GOSUB 470
270 IF R=15 OR S=Q OR H(S+1)
=0 THEN 280 ELSE 340
280 R=0 :: GOTO 490
290 ACCEPT AT(24,22)BEEP VAL
IDATE(DIGIT," ")SIZE(-3):PRG
300 IF PRG=00 THEN 620
310 IF PRG=97 THEN 330 ELSE
IF PRG=98 THEN 340 ELSE IF P
RG=99 THEN 370 ELSE IF PRG=9
99 THEN 510 ELSE PROG$=A$(PR
G)
320 GOTO 340
330 GOSUB 680 :: DISPLAY AT(
23,1):"INSERT NEW DISK":"PRE
SS ENTER WHEN READY" :: ACCE
PT AT(24,23):AA$ :: CLOSE #1
 :: GOTO 160
340 IF PROG$<>"" THEN 360 EL
SE IF S<Q AND H(S+1)<>0 THEN
 230 ELSE 290
350 CLOSE #1
360 CALL PEEK(-31954,A,B)::
GOSUB 380 :: RUN "DSK1.DOWN/
GEM" :: ::
370 END
380 Z=A*256+B-65536 :: CALL
PEEK(Z,A,B):: Z=A*256+B-6553
6
390 CALL LOAD(Z+29,LEN(PROG$
)+5):: CALL LOAD(Z+33,ASC(X$
))
400 FOR I=1 TO LEN(PROG$)::
P=ASC(SEG$(PROG$,I,1)):: CAL
L LOAD(Z+34+I,P):: NEXT I
410 IF LEN(PROG$)<10 THEN 42
0 ELSE 430
420 FOR I=LEN(PROG$)+1 TO 10
 :: CALL LOAD(Z+34+I,130):: 
NEXT I
430 RETURN
440 DISPLAY AT(1,1)ERASE ALL
:"DSK";X$;" -DISKNAME=";A1$:
"AVAILABLE=";V;" USED=";U-V
450 DISPLAY AT(3,4):"FILENAM
E  SIZE   TYPE":"  --------
  ----  ----"
460 RETURN
470 DISPLAY AT(R+4,1):USING
"##":S :: DISPLAY AT(R+4,4):
A$(S):: DISPLAY AT(R+4,14):J
(S):: DISPLAY AT(R+4,19):T$
480 RETURN
490 DISPLAY AT(21,1):"00 DEL
ETE A FILE":"97 READ NEW CAT
ALOG":"98 NEXT SCREEN 999 PR
INT CAT":"99 END     SELECTI
ON:"
500 GOTO 290
510 GOSUB 680 :: DISPLAY AT(
21,1):"ENTER DATE" :: ACCEPT
 AT(21,12):D$
520 OPEN #2:"PIO" :: PRINT #
2:CHR$(27);CHR$(66);CHR$(3);
CHR$(27);CHR$(51);CHR$(20);
530 PRINT #2:D$
540 PRINT #2:"DSK";X$;".";"-
DISKNAME=";A1$:"AVAILABLE=";
V;" USED=";U-V
550 PRINT #2:"   FILENAME
SIZE   TYPE":"   --------
---- --------"
560 COUNT=S :: S=1
565 FOR I=1 TO Q-1
570 IF ABS(H(S))=5 THEN B$="
   " ELSE B$=" "&STR$(K(S))
580 T$=TYPE$(ABS(H(S)))&SEG$
(B$,LEN(B$)-2,3)
590 PRINT #2:I;TAB(5);A$(I);
TAB(15);J(I);TAB(19);T$ :: S
=S+1
595 NEXT I
610 CLOSE #2 :: S=COUNT :: G
OTO 490
620 GOSUB 680 :: DISPLAY AT(
23,1):"DELETE WHICH FILE NAM
E?"
630 ON ERROR 650
640 ACCEPT AT(24,1)BEEP:DEL$
 :: GOTO 660
650 DISPLAY AT(24,1):"FILE I
S PROTECTED" :: FOR I=1 TO 5
00 :: NEXT I :: GOTO 490660
DELETE "DSK1."&DEL$
670 CLOSE #1 :: GOTO 160
680 CALL HCHAR(21,1,32,125):
: RETURN
```

Change Your Cursor

This program will change your cursor pattern. It is an XBasic loader and will work in XBasic only. It can be used as a program in itself or merged to any of your existing programs. It is not relocatable, so if your program uses assembly language LOADS then caution in using the program is required to prevent writing over the program.

Line 140 contains the new cursor pattern data. Simply determine the data for your pattern and substitute them following the address for start of data in that line (12288). Your editor used the following 255, 128, 128, 128, 128, 128, 128, 255 which produced a C for a cursor.

If you need some assistance in determining how to compute character pattern codes, read pages 56-59 in your Extended Basic manual.

```
110 CALL INIT
120 CALL LOAD(8196,63,248)
130 CALL LOAD(16376,67,85,82
,83,79,82,48,8)
140 CALL LOAD(12288,255,128,
128,128,128,128,128,255)
150 CALL LOAD(12296,2,0,3,24
0,2,1,48,0,2,2,0,8,4,32,32,3
6,4,91)
160 CALL LINK("CURSOR")
170 CALL COLOR(0,16,1)
```

Program Design

How many times have you heard this? "I wish I had a program that would. . . " Even though there are many talented programmers out there writing good programs for your TI, you still might someday need a program that is particularly suited to your need.

You can write that program if you are willing to expend a little effort. The best program for any need is one that works efficiently for that need without giving you information that is unimportant.

The main thing in writing any program is to first sit down and decide what you want the program to do. Supose you wanted to know how much it costs to own and operate an automobile for a year. You would need to start out with the initial cost, determine how much the monthly payments are, find out what the insurance figures are and then consider the trade in value after a stated period of years. Other expenses would include gas, oil, regular maintenance costs, etc.

In programming there a number of ways to approach any problem. In the case of the car you could sequentially add the cost of the car and then the expenses and divide the total by the number of years involved. In other words you could build your program on logical steps one after the other.

If some of the expenses were repeated over and over again you might resort to looping or subroutins to save memory and avoid repeated instructions. If you needed to compare cost to some other vehicle or criteria then branching would come into play. A combination of these processes would produce a program suited to your needs.

The language that you use for your program is up to you but you might consider who has to use and understand the instructions. Good plain instructions would make it easier for a less experienced person to use.

Another technique used by not a few programmers is the modular concept. In other words determine the different tasks that are needed and write a series of small routines that fill each need. This allows you to check and debug each small segment and make sure it runs. Then when you have each routine working you put them all together and you have your completed program.

Now I don't mean to imply that anyone who has never written a line can go right to his keyboard and write an award winning program. What I'm saying is that you start a program first with a need and then plan it in a logical manner.

If you've always wanted to get into programming, try starting out this way. Determine a need and decide just what needs to be accomplished to fill this need. Then in steps you can program each segment in order before combining the whole thing.

Before starting a review of your User Guide might help by refreshing you with what each command accomplishes.

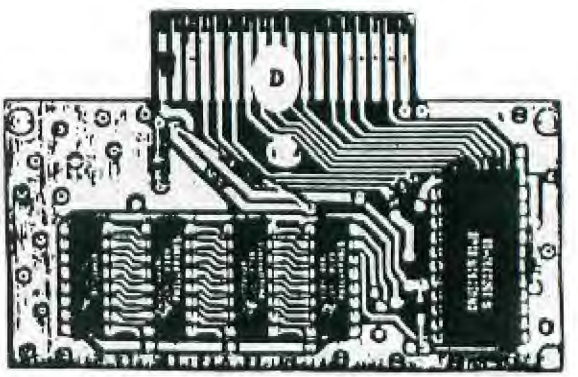
I guarantee that once you design your own program, you'll be on your way to more advanced programming.

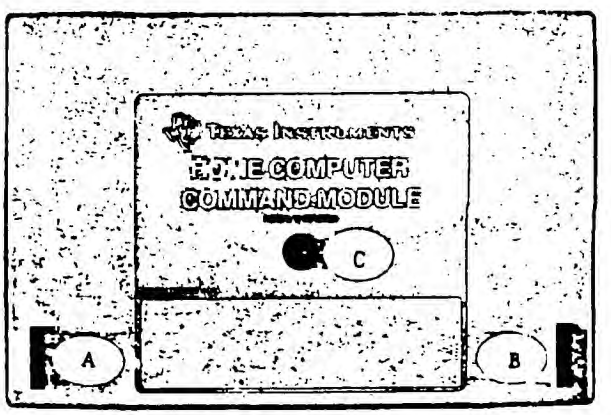
How to Clean Modules

Dirty contacts can screw-up any electrical device and the 4A is not an exception. The only place you are fairly likely to run into this problem is in using command modules. Both the module contacts and the port itself can become dirty but cleaning the port itself is a big job as you have to disassemble the console The good news is that cleaning the cartridge will almost always suffice and can be done quickly without any special tools or cleaners. All you need is a regular screwdriver, some sort of rag, a standard pencil eraser, and in some cases a medium phillips screwdriver

Remove the screw from "C" if there is one. Then pry the clips in slots "A" and "B" outward to pop open the cartridge. If there is a clip in "C" pry it back after "A" and "B" are loose. If it should bend off, don't worry, it won't affect the performance of your module.

The module board can now be removed. Do this carefully and note how the spring-loaded "door" is assembled if there is one so that you can put it back together if it pops out. Once you have the board removed take your rag (a kleenex will work but a cloth is better) and rub off any residue from the contacts "D". Remember to do the contacts on each if the particular module has them. Once the worst is removed take any soft pencil eraser and rub the contsct gently to remove any remaing contanimant. When you have finished reassemble the cartridge and you are back in business. Some symtoms of a dirty contact are the console locking-up, strange errors and display on your screen and a syntax error. Don't jump to clean a cartridge on your first error though, it could be a number of other things. But if you find that you have a continuing problem cleaning the contacts is quick and may correct what was wrong.





Assembly Language

This article was gleaned from the SNUG Newsletter of the Southern Nevada User's Group. Perhaps what is printed here will awaken some interest in those (like myself) who would like to learn something about Assembly Language. There is nothing really awsome about Assembly Language. The most frighting part, I think is the tremendous manual that Texas Instruments published with the disks they furnished with it and expected that you could go right to work and use this great utility.

So hang on and listen to what John Martin of the SNUG group tells us about this useful tool.

John says, locate your EDITOR/-ASSEMBLER manual and dust it off. He will be referring to it and help you understand what many of the passages say.

First turn to page 39 in the manual. This is the beginning of the chapter entitled "General Programming Information". This page refers to the "Registers" that are used by the computer. THIS IS IMPORTANT INFORMATION. There are 3 hardware registers located within the TMS9900 processor chip. None of them are directly accessable (you can't just change them arbitrarily), but there are instructions that automatically take care of it for you.

The first register is the Program Counter (PC). What this register does is hold the address of (points to) the next instruction to be executed. This is normally going to be the next consecutive even address after the instructions being currently processed. This register is affected by several instructions however, so where it actually points to depends on the current instruction. Some examples of instructions that change the PC would be JMP instructions and Branch instructions. The processor looks at the address in the PC and executes that instruction.

The second hardware register is called the Workspace Pointer (WP). This is the one that does all the really neat tricks for us. The WP contains the address of the first software register (R0) in the current workspace. The Workspace consists of 16 consecutive words of RAM. These 16 words can be located virtually anyplace in memory. When your program references one of the registers, it is relative to this address. For example, if you LI R6,15 the computer checks the WP to find out where R0 is and then goes 6 words beyond that to get to register 6. It then places the value 15 into that memory location. The exciting thing about this method of addressing registers is that you can easily set up as many workspaces as you need and branch back and forth between them without loosing or changing the information stored in them. This is called a "context switch" and can allow you to do some very interesting things. There are several ways to change the WP. These would include BLWP,LIMI, and XOP. Most of the time, the BLWP (Branch and Load Workspace Pointer) instruction is the one you would use because it stores the information of all three hardware registers into three of the software registers so that you can easily go back to the old workspace.

The third hardware register is no less important. It is called the Status Register. The Status Register is updated after every instruction. The status register is bit mapped and keeps track of the effect of the last instruction executed. Page 40 describes what each bit indicates, and page 41 tells which bits are affected by each instruction. The Status Register us checked by all Jump instructions except JMP which is an absolute instruction. All the rest make a comparison of some sort to determine whether to jump or not. By looking at the chart, one can see that it would be inappropriate to try to use a JOP instructions based on the results of A (Add words) instruction. One could, however follow and AB (Add Bytes) instruction with a JOP and get some kind of results. because the AB instruction does affect the Odd Parity bit of the Status Register while the A instruction does not. The point is that what determines whether to jump or not is the Status Register and not all instructions affect all bits of it.

The next thing I would like to cover is decoding the Syntax definition of the instructions. For this, lets go to page 79. This is the first page of the first section of actual instructions, the Arithmetic Instructions. This page is reprinted as the first page of each succeeding section, so you won't have to remember where it is. Just look at the first page of whatever section you are looking in to find this information. Near the top of the page, you will see definitions of a number of abbreviations. The most important (at least from my limited experience) are gas, gad, wa, iop, and wad. It would be a good idea to memorize at least these 5 abbreviations. Doing so will save you hours of frustration later on. At the bottom of the screen there a group of symbols that are used to graphically display the execution results. These results are printed for each instruction. By translating them, you can figure out what to expect from each instruction.

Also printed for each instruction is a chart that depicts which bits of the status register are affected by the instruction. By looking at the caret symbols under the chart, you can tell at a glance which bits are affected by each instruction.

Now let's look at how to decode all this information on some actual instructions. Turn to page 80. This is the description of the Add words instruction. Look at the Syntax description of the word. It says:

[<label>] b A b <gas>,<gad> b [<comment>]

Translated, this means that in field 1 there is an optional label. The square brackets indicate that it is optional. Next are one or more spaces (b) to separate the fields. Field 2 is the mnemonic (instruction). In this case the instruction is A. Again we have a space or spaces (again indicated by the b). Now we get to the part that can make us or break us. This part is called the operand field. Notice the abbreviation gas and gad? That means that virtually any type of address is acceptable for both the source operand and the destination operand. There are many instructions that require specific types of operands for either the source, destination or both. The next field is the comment field. This field is optional.

We should now look at another instruction. Please turn to page 85. This is the instruction AI or Add Immediate. Notice that the first 2 fields look similar to the A instruction, but in the third field we have <wa>, <iop>. Looking back on page 79, we discover that wa is Workspace register Address and that iop is Immediate OPerand. That means that this instruction expects (demands is more like it) to have a register (R1, R2, etc) for its first operand and a real number (no addresses, registers, etc) for its second operand. In this case, the first operand is the destination operand. This is typical of immediate instructions. Most other types of instructions require that the second operand be the destination operand. Looking at the execution results we see (wa) + iop => (wa). This translates to "add the immediate operand to the contents of the register and put the result back into the register.

By paying attention to the things I have mentioned here and having LOTS of patience, it is possible to teach yourself how to write ASSEMBLY LANGUAGE programs. Make sure that you check the syntax definition to find out what kind of operands are expected by the particular instruction that you are using. Be aware of the affect of the instruction on the Status Register. Look at the execution results diagram for each instruction. If you follow these simple rules, you will find that writing in ASSEMBLY is not quite as hard or frustrating as you thought. If you ignore these things, you will be spending an awful lot of time going back and forth between the Editor and the Assembler looking for Syntax Errors.

(Editors Note):
PUNN has a group that meets regularly and discusses Assembly Language. You would be welcome at those meetings.

P.O. Box 15037
Portland, OR 97215

DISCLAIMERS: The PUNN User's Group is not affiliated with or sponsored by TI and has no relationship with them, implied or otherwise.
Mention of a company or product is not an endorsement of that company or product.
We are not a subsidiary or branch of any other User's Group and any relationship we may have with other groups is on the basis of equals.

ALL GENERAL MEETINGS ARE HELD ON THE FIRST TUESDAY OF EACH MONTH, AT THE PGE BUILDING 3700 SE 17TH. PORTLAND, OREGON

!! NEXT MEETING DATE !!
SEPTEMBER 1ST. 1987

THE PUNN NEWSLETTER

Wordplay

P.O.BOX 15037
PORTLAND, OREGON 97215

POINTING THE WAY FOR USERS OF TI's 99/4 COMPUTERS

SEPT 1987-VOLUMN VI ISSUE 9