

Mr. B. Woods #32
9 Thirlmere Pde.
TARRO NSW
2322

ALLEY



99ERS NEWS

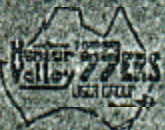
TI99/4A

HOME COMPUTER NEWSLETTER

AUGUST
1986



REGISTERED BY AUSTRALIA POST
PUBLICATION NO 1008023



The Secretary - HV 99ERS
E Arcot Close, TARRO - NSW
Australia - 2322

TEXAS
INSTRUMENTS
Newcastle
& The Hunter Region

Home Computer
USERS' GROUP

FOUN COMMITTEE

PRESIDENT

Allen Wright
77 Andrew Rd.,
VALENTINE 2280
Ph. 448120

VICE PRESIDENT

Jim Grimmond
31 Jarrett St.,
TORONTO 2283
Ph. 595751

SECRETARY

Albert Anderson
4 Arcot Close,
TARRO 2322
Ph. 662602

TREASURER

Brian Rutherford
9 Bombala St.,
REDHEAD 2301
Ph. 498184

SOFTWARE LIBRARIAN

Alan Lawrence
35 Bayview St.,
WARNERS BAY 2282
Ph. 486509

PUBLICATIONS LIBRARIAN

Paul Mulvaney
26 Macmohs St.,
HARMONG POINT 2284
Ph. 589623

EDITOR

Brian Woods
9 Thirlemere Pde.,
TARRO 2322
Ph. 662307

COMMITTEE

Bob MacGlure
75 Deborah St.,
KOTARA SOUTH 2288
Ph. 437431

Gary Jones
53 Janet St.,
JESMOND 2299
Ph. 573744

Tony McGovern
215 Grinsell St.,
KOTARA 2285
Ph. 523162

Peter Coxon
25 Reserve Rd.,
WANGI WANGI 2267
Ph. 751930

CONTRIBUTIONS

Members and non members are invited to contribute articles for publication in HV99 NEWS.

Any copy intended for publication may be typed, hand written, or submitted on tape/disc media as files suitable for use with TI Writer (ie. DIS/PIX 80 or DIS/VAR 80). A suitable Public Domain word processor program will be supplied if required by the club librarian Al Lawrence.

Please include along with your article sufficient information to enable the file to be read by the EDITOR eg. File Name etc.

The preferred format is 35 columns and page length 66 lines, right justified.

All articles printed in HV99 NEWS (unless notified otherwise) are considered to be PUBLIC DOMAIN. Other user groups wishing to reproduce material from HV99 NEWS may feel free to do so as long as the source and author are recognised.

Articles for publication can be submitted to:

THE EDITOR
HV99 NEWS
9 THIRLEMERE PDE.,
TARRO 2322

General address for ALL other club related correspondence:

THE SECRETARY
HV99 USER GROUP
4 ARCOT CLOSE,
TARRO 2322

DISCLAIMER

The HV99 NEWS is the official newsletter of the HUNTER VALLEY NINETY NINE USER GROUP. Whilst every effort is made to ensure the correctness and accuracy of the information contained therein, be it of general, technical, or programming nature, no responsibility can be accepted by HV99 NEWS as a result of applying such information.

TEXAS INSTRUMENTS trademarks, names and logos are all copyright to TEXAS INSTRUMENTS.

HV99 is a non profit group of TI99/4A computer users, not affiliated in any way with TEXAS INSTRUMENTS.

bu
th
mc
'C
ha
pl
re
we
re
of
fr

ex
Mc
de
a
al
pl
re
(
he
of
fo
th
re
th
af
fr
ex
pr
wi
is
Fu
wo
de
wi

be-
asj
ret
the
col
int
col
Dos
Use
acc
muc
nev
Car
out
nev
qua
are
fri
Pat

SECRETARYS

NOTES

Hello again and welcome from the business end of HV99. Thanks to all those that contributed to last month's newsletter - the message on 'user input' from the Editor must have struck home and has produced pleasing results. This has been reflected in the positive feedback we are receiving from our growing readership and the increasing number of exchange requests, especially from the USA groups.

Special congratulations are extended to both Tony and Will McGovern as their FUNLWRITER deservedly gains rapid acceptance as a first class product available to all 4A users worldwide. It is also pleasing to note that some monetary reward to the authors also flows in (mainly from the USA) and this helps to demonstrate the character of the 'real' 4A user, by his/her follow up of the financial part of the FAIRWARE concept. I would recommend that you have a look at the JULY issue of MICROpendium, and after you finish smiling at the front cover, have a read of an excellent review of an excellent product - FUNLWRITER. Couple this with the article by Tony in this issue on Customising Your Funlwriter, and I feel sure that the work that has gone into it's development by both Tony and Will will be much more appreciated.

As I have probably mentioned before, one of the most pleasing aspects of the Secretarys' job is receiving mail from groups all over the world and having the pleasure of contact with people with similar interests. A letter of congratulations to our group from Al Doss, Editor of the Mid-South 99 User Group in Tennessee, USA, accompanied by a fine newsletter, is much appreciated (as are all the newsletters we receive from the US, Canada and the UK). Not to be out-done, the Australian groups' newsletters continue to abound in quality articles and all of these are available for loan from our friendly Publications Librarian, Paul Mulvaney.

From the group in Melbourne comes word that they have 'adopted' a sister group in the USA - None other than the LA 99ers Users Group in California. Congratulations to you both and I hope your association flourishes as I am sure it will. An interesting idea for groups elsewhere to consider! While I am at it, congratulations to the new office bearers of the Melbourne group. Keep up the good work.

Whilst on the subject of interesting ideas, I have just received a letter from Mark Beck of the Jacksonville Users Group in Arizona, USA, with a request to review and distribute a FAIRWARE program from that group. The program is a data based filing system called CREATIVE FILING SYSTEM and seems quite interesting. I shall follow this up on behalf of HV99. The interesting part of this letter involves the method of distribution (and hopefully for the Jacksonville group, the method of payment) for the program via the user groups. A kind of insurance is also attached with only purchasers of the program receiving updates as necessary. I might add that the proceeds of this program are going to support the newsletter costs of the Jacksonville group and not just the author - real user type stuff - good work, Mark Beck!

Also from the States comes another FAIRWARE author payment idea of a different kind. A suggestion for a "fairware author of the month", with the group purchasing and distributing the program selected and collecting the proceeds which are then sent on to the author. This would be a real test of character for some user groups, but I think that the idea of getting the monetary reward to the author are well worth serious consideration for possible use by all groups.

Back to the local scene, and members will be happy to know that membership records and mailing details are now up to date thanks to Database 1 (a great database program from Asgard Software in the States), Brian Woods, Brian Rutherford and yours truly mixed with the usual blood, sweat and ale (or two). If there is anyone that needs their details (change of address etc.) updated could you please get them to

me so it can be done.

There are still some groups sending mail to the previous secretary's address and hence it takes much longer for me to get it and take action on it. Please note that the address of the Secretary (me) is :-

6 Arcot Close,
Tarro, NSW.
2322
Australia.

MISSING PERSONS

Edwards Hall
NEWCASTLE UNIVERSITY

Could JOHN DIXON (or anyone that knows him) of the above address, please contact me with your new address as the newsletters I assume are not getting to you.

Albert Anderson
4a4me

HARDWARE MODIFICATION

**** FAST CONSOLE ACCESS ****

FIT A DOLLAR SWITCH.
For 60 cents.

Ron Kleinschafer HV99ers.

This modification is ideal for an afternoon workshop project and our hard working software librarian MR. AL LAWRENCE, who spends countless hours sifting, sorting, checking and shuffling programs around, (and has often been heard to mutter, "THE *&##%#@* THING TAKES TO LONG!!") would probably be first in line. (Whack one of these in AL !!).

As everyone with a TI 99/4a and Disk Drive(s) knows when the machine is Powered Up, Quit or whatever and XB selection is then made, the computer goes off searching Drive No 1 for a program file called LOAD. If that specific file is required, fine, but when you want to access the console only without using the Drive(s) as yet, RUN another program

stored on disk with a file named LOAD, or use DRIVE 2 or 3 first, then the only options until now have been to,

- 1 LEAVE THE DISK OUT.
- 2 FLIP THE DRIVE DOOR OPEN.
(certainly not recommended)
- 3 AND WAIT.!!!

Now whichever option is selected it takes, with SS SD Drives, approx 10 to 14 seconds to return to the console, (I believe that DS Drives takes longer). Well now you can access the console in approx 2 SECONDS, disks in or not, with or without a LOAD file.

What is a DOLLAR Switch ???

Well that's an acronym for DRIVE OR LOAD LOCKOUT AND RESET Switch !!! This is a small push button switch connected to the disk controller card that forces the controller to return to the console software without starting the Drive.

All that is required is a small N/O push button switch (Dick Smith CAT. No S-1102) and two short lengths of hookup wire. This is where the usual disclaimer part comes in.

DISCLAIMER

IF YOU CARRY OUT THIS MODIFICATION
AND YOU WRECK SOMETHING,* ROUGH.*

Power down the PEU and wait several minutes. Remove the disk controller card and with precautions against static damage, remove the card pull up clips, cut through the label at the case join with a sharp knife, then open the card's plastic casing by levering in the retaining clips and open the case, remove the PCB.

The TI card uses an FD1771 CONTROLLER IC. But this modification also applies to the FD179X designs and the WESTERN DIGITAL WD279X Family and other Disk controller IC'S which perform the functions of Formatter/ Controller, in single chip implementation. (Using the correct pin connection).

On the TI card locate the 1771 IC. (Its the largest chip on the board). Locate pin No 17, solder a length of hookup wire onto the pad on the BOTTOM of the board of that pin, taking antistatic precautions. Solder another wire to the ground

et
is
fe
in
fi
wh
th
an
it

a
by
se
to
ap
bee
wit
BUT
the
the
hav

Pin
RES
LOG
ing
HEX
Thi
to
rel
LOG
the
the
rea

ther
list
Expa

120
LOAD
X<>Z

140
LOAD
X<>Z

named first, have
is SD
ds to lieve Well e in not,
for RESET push disk the nsole rive.
small Smith short is is part
ION H.*
wait disk tions the h the sharp astic lining e the
D1771 this the STERN Disk m the ller, Using
1771 on the er a e pad that ions. ground

etch (somewhere near the heat sink is ideal), then refit the casing feeding the wires out through a hole in the direction you are going to fit the PB switch. Drill a hole where you want to fit the switch then solder the wires to the switch and fit it. Refit everything and it's ready to go.

In operation after Power Up hit a key to select the Menu Screen then by holding in the button and selecting XB, the software returns to the familiar * READY * within approx 2 SECONDS. The DISK DSR has been set and any program can be run with the usual RUN DSKx.xxxx, etc. without any loss of functionality, BUT if you press the button LATER than 2 SECONDS AFTER selecting XB then the computer locks up and you have to RESET and start again.

WHAT HAPPENS INSIDE THE IC ?, Pin 19 is the controllers MASTER RESET connection, it is held high LOGIC 1. A LOGIC 0 (low) on this input resets the device and loads HEX 03 into its command register. This causes the NOT READY status bit to be set. When the button is released the pin is brought to a LOGIC 1 (high), a RESTORE command is then executed, and at the same time the sector register is reloaded ready for the next operation.

Total cost 60 CENTS !!!.

DOOPS

In last month's newsletter, there was an error in the programme listing for "Testing the 32K Expansion".

Line 120 should read:-

```
120 FOR J=0200 TO 16383 :: CALL  
LOAD(J,Z) :: CALL PEEK(J,X) :: IF  
X<>Z THEN PRINT "ERROR AT";J
```

and line 140 should read:-

```
140 FOR J=-24576 TO -1 :: CALL  
LOAD(J,Z) :: CALL PEEK(J,X) :: IF  
X<>Z THEN PRINT "ERROR AT";J
```

Sorry about that Chief.
The Editor

COMMENT

FROM

"COCKROACH
COTTAGE"

BY RICHARD TERRY
H. V. 99ERS

This will be my last article for some time, if one can call it an article, for this month I am going to make a special plea to all you TI users out there, especially in the Hunter Valley club, to embrace Forth with the same fervour you would embrace the one you love the most. Not that I am attempting to drag you away from your better halves, nor create some sort of fetish for a computer language, but just entice you a little to savour the joys that Forth has to offer in your computing life.

I hear reported on the grapevine that Keith Bruce is champing at the bit to take over the role of resident Forth writer. Keith's approach to Forth tends to be somewhat different to mine, in that he is interested in graphics/mathematics etc so he can probably give us a different slant.

I probably resuscitate myself in the New Year, when hopefully if pressures of business are less, and my extensive terrace house rebuilding project is over I will have more time. Perhaps I might set up a rival name to Tony's & name the environs my work emanates from, Cockroach Cottage, at the seamier end of town!!!

MESSAGE TO ALL HV99'ERS.

YOU WILL GET AS MUCH OUT OF YOUR COMPUTER AS YOU PUT INTO IT. MANY MEMBERS OF THE CLUB WORK HARD FOR LONG HOURS TO PRODUCE ARTICLES,

AND RUN TUTORIALS. WHEREAS THEY ARE NOT DOING IT FOR THANKS, RATHER FOR THE LOVE OF IT, A LITTLE ENTHUSIASM ON THE PART OF CLUB MEMBERS WILL ENSURE A CONTINUATION OF THE HOPEFULLY HIGH QUALITY OF MATERIAL WE HAVE PUBLISHED IN OUR JOURNAL.

REGARDING FORTH, IF A FEW MORE OF YOU SHOWED A LITTLE ENTHUSIASM IT MAY EVEN BE WORTH WHILE ONE OF US SETTING UP SOME SIMPLE FORTH DEMONSTRATIONS FOR YOU, IN THE FLESH, ON A FULL SYSTEM, AT ONE OF THE MEETINGS.

TAKE A FRESH LOOK AT FORTH.

You may ask, "Why bother, its a crazy language, the code is unreadable, its too hard....". If you do its unlikely life will ever unfold its mysteries to you, its unlikely you'll ever stand atop of Mt Kilimanjaro on the roof of the world at 18000 ft, gaze in awe at the Tombs in the Valley of the Kings, or be transfixed by the eternal beauty of the Taj Mahal by moonlight. . Your life WILL be boring.

However, if you decide to have an open mind, your life and experience will expand before you. You may think I'm waxing lyrically, being too philosophical. Perhaps I am, but I don't believe so. Let me give you a personal history of my excursion into Forth.

At the end of 1984, Xmas time, I purchased my trusty 99/4a. After spending 3 hours trying to turn it on, and suffering the indignity of my girlfriend (who by the way was in no way electrically or mathematically oriented) figuring it out in a couple of minutes, my life in computing began. I fairly quickly mastered the elements of Basic, and I thought, Extended Basic. In reality I had not scratched the surface of the potential of Extended Basic. I'd flipped past user defined subprograms as a command in the manual I just didn't understand so therefore, as no-one had ever explained their potential. I assumed it was just one of many commands. It wasn't until I met Tony McGovern and his enthusiasm, and reading his articles I started to see the light.

Though I had written some very useful programs I still use to this day, such as a proper auto-incrementing 12 column general ledger, I suddenly realised I had not scratched the surface of the potential of the language, and probably still haven't.

Forth arrived, was given a glowing write up in the Sydney News digest by someone who impressed us all by his apparent knowledge of the language, using baffling terms like "threaded interpretative code" etc (apologies if I'm misquoting), so I eagerly tried to boot it up and play with it.

The result : TOTAL DISAPPOINTMENT.

I found it frustratingly difficult. The accompanying manual, or rather excuse for it, (perhaps we should call it a CLAYTONS manual) seemed to be written in a dialectical tongue more suited for communication between beings from another universe, rather than mere mortals. Of course, had I have had a computing background it might have made a glimmer of sense. But I didn't. I was totally thrown by the jargon, and infuriated by such sentences ending in words like "..... will be obvious to the user". Having no concept of internal computer architecture, or even what that meant, even the glossary wasn't much help. Reading and cross referencing the words seemed to go round and round in circles.

I tried a few things such as DO LOOPS and was suitably amazed by their speed compared to EXTENDED BASIC. I fumbled a few definitions, drew a few lines, but got frustrated by the system locking up all the time. I very nearly got to the point of sending my computer to the repairman as I was becoming totally convinced it was breaking down, having faulty electrical circuits.

I bought Brodie. He waxed lyrical about the value of the language. It helped a bit, I did some exercises, got totally confused about double numbers, tripped up on differences between 83 Forth and Fig Forth and with a great sense of disappointment packed the whole thing in. Despite reading the whole book and doing the exercises I still couldn't see the application.

esc
Zor
Pla

up
I'd
ext
pro
the
bec
mem
nee
hol
lot
exc
rou
Add
ava
lim
and
lumb
some

comp
perc
Agai
extc
lang
by c
thin
hint
" a
is
weir
few
yet
mast

givi
chan
plac
Had
that
off
eati
Funl
in A
brie
unti
over
caus
to
Besic
under
them
since
occa
know
know
later

very
this
proper
general
I had
of the
, and

ven a
y News
ed us
of the
like
" etc
, so I
d play

TOTAL

tingly
manual,
perhaps
manual)
in a
ed for
from
mere
ve had
t have
But I
by the
such
like
to the
pt of
re, or
the
reading
words
ound in

as DO
azed by
XTENDED
tions,
strated
all the
to the
to the
totally
down,
its.

waxed
of the
I did
nfused
up on
and Fig
se of
whole
whole
still

Bloody stupid language!!!!

12 months passed..... I escaped into fantasy, finally solved Zork, and was deeply into Planetfall.

I forget what made me take it up again.... no wait, I remember. I'd being writing some quite extensive and complicated business programs having relatively mastered the use of subprograms. I was becoming frustrated by lack of memory space, as some of my programs needed a lot of space dimensioned to hold string arrays and involved a lot of sorting of data using the excellent Assembly language sort routine that's floating around. Additionally, the 28/32 columns available on the screen were limiting, and I was becoming less and less impressed with the TI's lumbering speed problem compared to some of the quicker Basics, around.

I wandered over to the yearly computer show at the lni and found perchance several books on Forth. Again the authors, not just Brodie extolled the virtues of the language. I was particularly struck by one whose name I forget because I think Keith Bruce still has it (hint hint). A comment was made that "a beginners description of Forth is likely to contain words such as weird, strange, unreadable, plus a few undeleted expletives right, and yet they all rambled on how, once mastered, it was a great language.

Well, perhaps it warranted giving it a second glance, or second chance. I wandered over to Tony's place and sounded him out on Forth. Had he tried it? He didn't sound all that impressed. Too busy beating off funnel webs or killing those eating holes in his nascent Funlwriter. Will was just dabbling in Assembler and had just written a brief disk catalog which ran great until the end of the process when it overwrote some part of memory which caused something strange to happen to the display. No help there. Besides I felt inadequate not understanding what seemed obvious to them both. (He's come along way since then, and has helped me out on occasions in Forth, not by his knowledge of the language but by his knowledge of the machine and his lateral thinking.)

So, I'd just have to go it alone. The first useful articles were coming out from the States in Millers Graphics etc. One could learn how to save the options in binary code, which made the eternal re-booting after crashing due to my stupidity a little easier.

I decided on a rather ambitious project, which paradoxically was quite easy: A FORTH disk manager. No, not another basic disk manager written in Forth, but a FORTH MANAGER as I called it. It was modelled after the DM1000 style which didn't exist in the country at the time. It wasn't a planned program, it just evolved. I first experimented just writing things on the screen using the GOTOXY command, to make neat menus. Joe, Bruce and I worked out a few basic words for accepting strings and numbers. Once the visual framework was written and I'd figured out how to flick back and forth through menus I wrote the working guts of the thing. This was quite easy as most of the disk managing words exist already in Forth, and one can by changing the definitions slightly make them easier to use. Unlike a conventional disk manager which does disk initialising/copying etc, in Forth one has to cope with copying one of more screens of code or moving one or more from one disk to another. The Basic Disk catalog is really replaced by an Index of the title lines of each of the 90 or more screens on a disk (360 if your Keith Bruce and want to be make communication with your fellow Forth users difficult!)

Anyway, the point it writing a Forth Program was actually a lot easier than I thought. I didn't do what the purists said I should, ie design top down and write bottom up. It was much more exciting quickly seeing the screen displays flash up so fast. I still didn't understand the guts of the machine, but I could write and binary save useful source and object code.

As an aside I want to hark back to the concept of a computer language as a philosophy, particularly Forth. There is an excellent book out by Brodie called THINKING FORTH, which although I havn't read it properly, what I've gleaned through skimming it makes a lot of sense.

AS I progressed with Forth I found it was starting to rub off on my day to day life in quite interesting ways. Because you have total control over your computer in Forth you have to be accurate. Putting the wrong thing at a memory address, or not putting what is expected on the stack often causes computer lockup and an end to your computing session. Unlike Basic in which you always have a way out with those cop-out GOTO statements, Forth is unforgiving. If you do not design your program correctly it will crash. Forth carries the concept of subprograms to the ultimate extreme, because virtually everything in Forth is a subprogram/program which will run entirely independently of everything else, provided you put its starting data on the stack. Each word passes data on to the next or changes the values in a memory location. Paradoxically, but because the units are so separate, this feature also gives Forth its amazing flexibility, because you can change the workings of a part, with no effect on the whole. (an oversimplification, but true in essence).

Anyway what I meant to say was my day to day thinking became clearer, my assessment of non-computing problems seemed to become more accurate. My passage in my personal life when faced with problems about myself or others changed. It was easier to see the wood for the trees.

THINKING IN FORTH.

I know you'll probably think I'm joshing but believe it or not, after a while one does think easily in Forth, especially once you've mastered the stack and its basic operators. I can actually sit down now and write source code as you may in basic, but with the added advantage I'm not restricted to line numbers or rigid commands. I create my own computer language as the program evolves, for in fact each Forth program one writes is a separate dialect of Forth itself. Its like having a tailor made language for each application you choose.

I was faced the other day with having to modify one of my older

Extended Basic programmes. Would you believe I couldn't even remember the editing function keys. When I re-mastered them I groaned at the abysmal slowness of the editing process. I groaned even louder when I examined my code. Surely I couldn't have written anything this jumbled. I found my code difficult to understand and follow. Sure its my fault, but had it been in Forth at least as we make our own words up I would have had a head start by just reading the English words which describe exactly what they do.

I could continue ad-nauseum, but rather than make you, (by the tedium of this article) take the Latin seriously I'll end with a few reasons why you should re-consider Forth.

WHY YOU SHOULD LEARN FORTH.

1. It's something new.
2. It will expand your thinking.
3. It will enhance the quality of your life.
4. Its not all that difficult.
5. Its quicker than Basic
6. You have access to all the resources in your computer without the tedium of Assembler.
7. You have several keen Forth programmers in the HV99ers.
8. Plus many more!!!!!! you will only discover if you learn, as the unique elegance of this language unfolds to you.

Finally this month I have included a useful series of wall charts which summarise all the commands of Ti-Forth, which will save you wrecking your bound pages (that is if you haven't already burned it or stuffed it in the bin) like I have. I would like to acknowledge the source of this, it comes from one of our fellow user groups in Canada....CIM 99 in Montreal.

and has been enlarged to page size by Joe Wright.

ADDRESS FOR CORRESPONDENCE.

RICHARD TERRY.
141 DUDLEY RD
WHITEBRIDGE 2290
NSW AUSTRALIA.
PHONES (049) 436061 (WK)
22450 (HOME)

18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

TI FORTH QUICK REFERENCE CARD Resident Words

: Par Maria Jonathan Stephanie & Liz Beaulieu (1985) 1 of 2

<p>STORE <i>n</i> <i>addr</i> — MEMORY Store 16 bits of <i>n</i> at address. Pronounced "STORE".</p> <p>CSP — STACK Save the stack position in CSP. Used as part of the compiler security.</p> <p>CHAR <i>d1</i> — <i>d2</i> CONVERSION Generate from a double number <i>d1</i>, the next ASCII character which is placed in an output string. Result <i>d2</i> is the quotient after division by BASE, and is maintained for further processing. Used between $\langle \& \rangle$ and $\langle \> \rangle$. See $\langle \& \rangle$.</p> <p>CR <i>d</i> — <i>addr</i> <i>cnt</i> CONVERSION Terminates numeric output conversion by dropping <i>d</i>, leaving the text address and character count suitable for TYPE.</p> <p>AS <i>d1</i> — <i>d2</i> CONVERSION Generates ASCII text in the text output buffer, by the use of $\langle \& \rangle$, until a zero double number <i>d2</i> results. Used between $\langle \& \rangle$ and $\langle \> \rangle$.</p> <p>Used in the format: — <i>pfa</i> DICTIONARY <i>nnnn</i> Leaves the parameter field address of dictionary word <i>nnnn</i>. As a compiler directive, executes in a colon definition to compile the address of a literal. If the word is not found after a search of CONTEXT and CURRENT, an appropriate error message is given. Pronounced "TICK".</p> <p>Used in the format: — COMMENT <i>(cccc)</i> Ignore a comment that will be delimited by a right parenthesis on the same screen. May occur during execution or in a colon definition. A blank after the leading parenthesis is required.</p> <p>(+LOOP) <i>n</i> — STRUCTURE The run-time procedure compiled by +LOOP, which increments the loop index by <i>n</i> and tests for loop completion. See +LOOP.</p> <p>(.) — PRINT The run-time procedure, compiled by $\langle \cdot \rangle$ which transmits the following in-line text to the selected output device. See $\langle \cdot \rangle$.</p> <p>(:CODE) — ASSEMBLER The run-time procedure, compiled by :CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence. See :CODE.</p> <p>(ABORT) — ERROR Executes after an error when WARNING is -1. This word normally executes ABORT, but may be altered (with care) to a user's alternative procedure.</p> <p>(DO) — STRUCTURE The run-time procedure compiled by DO which moves the loop control parameters to the return stack. See DO.</p> <p>(DOES) — DEFINING WORDS The run-time procedure compiled by DOES).</p> <p>(FIND) <i>addr1</i> <i>addr2</i> — <i>pfa</i> <i>b</i> <i>if</i> (ok) DICTIONARY <i>addr1</i> <i>addr2</i> — <i>ff</i> (bad) Searches the dictionary starting at the name field address <i>addr2</i>, matching to the text at <i>addr1</i>. Returns parameter field address, length byte of name field, and boolean true for a good match. If no match is found, only a boolean false is left.</p> <p>(LINE) <i>n</i> <i>scr#</i> — <i>addr</i> <i>cnt</i> EDITOR Convert the line number <i>n</i> and the screen <i>scr#</i> to the dist buffer address containing the data. A count of E_4 indicates the full line text length.</p> <p>(LOOP) — STRUCTURE The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP.</p> <p>(NUMBER) <i>d1</i> <i>addr1</i> — <i>d2</i> <i>addr2</i> CONVERSION Convert the ASCII text beginning at <i>addr1</i>+1 with regard to BASE. The new value is accumulated into double number <i>d1</i>, being left as <i>d2</i>. <i>Addr2</i> is the address of the first unconvertable digit. Used by NUMBER.</p> <p>(OF) — STRUCTURE The run-time procedure compiled by OF.</p> <p>* <i>n1</i> <i>n2</i> — <i>n3</i> ARITHMETIC Leave the signed product of two signed numbers.</p> <p>/ <i>n1</i> <i>n2</i> <i>n3</i> — <i>n4</i> ARITHMETIC Leave the ratio $n4 = n1n2/n3$ where all are signed numbers. Retention of an intermediate 31 bit product permits greater accuracy than would be available with the sequence: $n1 n2 * n3 /$</p> <p>MOD <i>n1</i> <i>n2</i> <i>n3</i> — <i>n4</i> <i>n5</i> ARITHMETIC Leave the quotient <i>n5</i> and remainder <i>n4</i> of the operation $n1n2/n3$. A 31 bit intermediate product is used as for $\langle \& \rangle$.</p> <p>+ <i>n1</i> <i>n2</i> — <i>n3</i> ARITHMETIC Leave the sum of <i>n1</i> + <i>n2</i>.</p> <p>+ <i>n</i> <i>addr</i> — ARITHMETIC Add <i>n</i> to the value at the address. Pronounced "PLUS STORE".</p> <p>- <i>n1</i> <i>n2</i> — <i>n3</i> ARITHMETIC Apply the sign of <i>n2</i> to <i>n1</i>, which is left as <i>n3</i>.</p> <p>+BUF <i>addr1</i> — <i>addr2</i> <i>f</i> DISK Advance the dist buffer address <i>addr1</i> to the address of the next buffer <i>addr2</i>. Boolean <i>f</i> is false when <i>addr2</i> is the buffer presently pointed to by variable PREU.</p>	<p>+LOOP <i>addr</i> <i>n1</i> — (run) STRUCTURE <i>addr</i> <i>n2</i> — (compile) Used in a colon-definition in the form: DO ... <i>n1</i> +LOOP At run time, +LOOP selectively controls branching back to the corresponding DO based on <i>n1</i>, the loop index and the loop limit. The signed increment <i>n1</i> is added to the index and the total compared to the limit. The branch back to DO occurs until the new index is equal to or greater than the limit (n1>0), or until the new index is equal to or less than the limit (n1<0). Upon exiting the loop, the parameters are discarded and execution continues ahead. At compile time, +LOOP compiles the run-time word (+LOOP) and the branch offset computed from HERE to the address left on the stack by DO. <i>n2</i> is used for compile time error checking.</p> <p>* <i>n</i> — DICTIONARY Store <i>n</i> into the next available dictionary cell, advancing the dictionary pointer. (comma)</p> <p>- <i>n1</i> <i>n2</i> — <i>n3</i> ARITHMETIC Leave the difference of <i>n1</i> - <i>n2</i>.</p> <p>-> — EDITOR Continue interpretation with the next dist screen. Pronounced "NEXT SCREEN".</p> <p>-DUP <i>n1</i> — <i>n1</i> (if zero) STACK <i>n1</i> — <i>n1</i> (non-zero) Reproduce <i>n1</i> only if it is non-zero. This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.</p> <p>-FIND — <i>pfa</i> <i>cnt</i> <i>f</i> (ok) DICTIONARY <i>ff</i> (bad) Accepts the next text word (delimited by blanks) in the input stream to HERE, and searches the CONTEXT and then CURRENT vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte, and a boolean true are left. Otherwise, only a boolean false is left.</p> <p>-TRAILING <i>addr</i> <i>n1</i> — <i>addr</i> <i>n2</i> MEMORY Adjusts the character count <i>n1</i> of a text string beginning at <i>addr</i> to suppress the output of trailing blanks, i.e. the characters at <i>addr</i>+<i>n1</i> to <i>addr</i>+<i>n2</i> are blanks.</p> <p>. <i>n</i> — PRINT Print a number from a signed 16 bit two's complement value, converted according to the numeric BASE. A trailing blank follows. Pronounced "DOT".</p> <p>Used in the format: — PRINT <i>cccc</i> Compiles an in-line string <i>cccc</i> (delimited by the trailing $\langle \cdot \rangle$) with an execution procedure to transmit the text to the selected output device. If executed outside a definition, $\langle \cdot \rangle$ will immediately print the text until the final $\langle \cdot \rangle$. See $\langle \cdot \rangle$.</p> <p>.LINE <i>n</i> <i>scr#</i> — PRINT Print on the terminal device, a line of text from the dist by its line (<i>n</i>) and screen number. Trailing blanks are suppressed.</p> <p>-R <i>n1</i> <i>n2</i> — PRINT Print the number <i>n1</i> right aligned in a field whose width is <i>n2</i>. No following blank is printed.</p> <p>/ <i>n1</i> <i>n2</i> — <i>n3</i> ARITHMETIC Leave the signed quotient of <i>n1/n2</i>.</p> <p>MOD <i>n1</i> <i>n2</i> — <i>rem</i> <i>n3</i> ARITHMETIC Leave the remainder and signed quotient of <i>n1/n2</i>. The remainder has the sign of the dividend.</p> <p>0 1 2 3 — <i>n</i> ARITHMETIC These small numbers are used so often that it is attractive to define them by name in the dictionary as constants.</p> <p>0< <i>n</i> — <i>f</i> STRUCTURE Leave a true flag if the number is less than zero (negative), otherwise leave a false flag.</p> <p>0= <i>n</i> — <i>f</i> STRUCTURE Leave a true flag if the number is equal to zero, otherwise leave a false flag.</p> <p>ORANCH <i>f</i> — STRUCTURE The run-time procedure to conditionally branch. If <i>f</i> is false (zero), the following in-line parameter is added to the interpretive pointer to branch ahead or back. Compiled by IF, UNTIL, and WHILE.</p> <p>1+ <i>n1</i> — <i>n2</i> ARITHMETIC Increment <i>n1</i> by 1.</p> <p>1- <i>n1</i> — <i>n2</i> ARITHMETIC Decrement <i>n1</i> by 1.</p> <p>2+ <i>n1</i> — <i>n2</i> ARITHMETIC Leave <i>n1</i> incremented by 2.</p> <p>2- <i>n1</i> — <i>n2</i> ARITHMETIC Leave <i>n1</i> decremented by 2.</p> <p>: — INTERPRET Used in the form called a colon-definition: <i>f</i> <i>ccc</i> Creates a dictionary entry defining <i>ccc</i> as equivalent to the following sequence of FORTH word definitions "... until the next $\langle \cdot \rangle$ or $\langle \& \rangle$". The compiling process is done by the text interpreter as long as STATE is non-zero. Other details are that the CONTEXT vocabulary is set to the CURRENT vocabulary and that words with the precedence bit set (P) are executed rather than being compiled. When colon definitions are compiled under the TRACE option, $\langle \cdot \rangle$ takes on an alternate definition which allows the colon definition to be traced.</p> <p>: — INTERPRET Terminates a colon-definition and stops further compilation. Compiles the run-time IS.</p> <p>IS — INTERPRET Stop interpretation of a screen. IS is also the run-time word compiled at the end of a colon-definition which returns execution to the calling procedure.</p>
---	---

Would remember When I at the editing er when couldn't this difficult re its n Forth words up start by s which .

auseum, by the take the a few consider

inking. quality cult. all the without n Forth you will n, as the language I have of wall all the h will nd pages already the bin) ke to this, it ow user 99 in to page

NCE. ---

← **n1 n2** — **f** **STRUCTURE**
Leave a true flag if n1 is less than n2 otherwise leave a false flag.

← **CONVERSION**
Setup for pictured numeric output formatting using the words: **<# # #B SIGN #>**
The conversion is done on a double number producing text at PRU.

← **INTERPRET**
<BUILDS Used within a colon-definition:
cccc <BUILDS ... DOES> ...
Each time **cccc** is executed, **<BUILDS** defines a new word with a high level execution procedure. Executing **cccc** in the form:
cccc nnnn
uses **<BUILD** to create a dictionary entry for **nnnn**. When **nnnn** is later executed, it has the address of its parameter area on the stack and executes the words after **DOES** in **cccc**. **<BUILDS** and **DOES** allow run-time procedures to be written in high-level rather than in assembler code (as required by **<CODE>**).

← **n1 n2** — **f** **STRUCTURE**
Leave a true flag if n1<n2 otherwise leave a false flag.

← **STACK**
<CELLS **addr** — **n2**
This instruction expects an address or an offset to be on the stack. If this number is odd, it is incremented by 1 to put it on the next even word boundary. Otherwise, it remains unchanged.

← **n1 n2** — **f** **STRUCTURE**
Leave a true flag if n1 is greater than n2 otherwise leave a false flag.

← **STACK**
<R **n**
Remove a number from the computation stack and place as the most accessible on the return stack. Use should be balanced with **<R>** in the same definition.

← **addr** — **PRINT**
<P Print the value contained at the address in free format according to the current **BASE**. This word must precede the address.

← **ERROR**
<?COMP Issue error message if non-compiling.

← **ERROR**
<?CSP Issue error message if stack position differs from value saved in **CSP**.

← **ERROR**
<?ERR Issue an error message number n. If the halt-on flag is true.

← **ERROR**
<?EXEC Issue an error message if not executing.

← **KEYBOARD**
<?KEY **ch**
Scan the keyboard for input. If no key is pressed, a 0 is left on the stack. Else, the ascii code of the key pressed is left on the stack.

← **KEYBOARD**
<?KEYS **n**
Scan the keyboard for input. If no key is pressed, a 0 is left on the stack. Else, the 3-bit code of the key pressed is left on the stack.

← **ERROR**
<?LOADING Issue an error message if not loading.

← **ERROR**
<?PAIRS **n1 n2**
Issue an error message if n1 does not equal n2. The message indicates that compiled conditionals do not match.

← **ERROR**
<?STACK Issue an error message if the stack is out of bounds.

← **KEYBOARD**
<?TERMINAL **f**
Perform a test on the terminal keyboard for actuation of the break key. A true flag indicates actuation. On the TI 99/4A, the CLEAR key is used as the BREAK key.

← **MEMORY**
<B **addr** — **n**
Leave the 16 bit contents of **addr**.

← **INTERPRET**
<ABORT Clears the stack and enter the execution state. Return control to the operator's terminal, printing an appropriate message.

← **ARITHMETIC**
<ABS **n1** — **n2**
Leave the absolute value of n1 as n2.

← **STRUCTURE**
<AGAIN **addr n** — (compiling)
Used in a colon-definition in the format:
BEGIN ... AGAIN
At run-time, **AGAIN** forces execution to return to corresponding **BEGIN**. There is no effect on the stack. Execution cannot leave this loop (unless **<R>** **<DROP** is executed one level below).
At compile time, **AGAIN** enables **BRANCH** with an offset from **HERE** to **addr**. **n** is used for compile-time error checking.

← **DICTIONARY**
<ALLOT **n**
Add the signed number to the dictionary pointer **DP**. May be used to reserve dictionary space or re-align memory.

← **KEYBOARD**
<ALTN **addr**
A user variable whose value is 0 if input is coming from the keyboard else its value is a pointer to the UOP address where the PRB for the alternate input device is located.

← **PRINT**
<ALTOUT **addr**
A user variable whose value is 0 if output is going to the monitor else its value is a pointer to the UOP address where the PRB for the alternate output device is located.

← **LOGICAL**
<AND **n1 n2** — **n3**
Leave the bitwise logical AND of n1 and n2 as n3.

← **DISK**
<B/BUF **n**
This constant leaves the number of bytes per disk buffer.

← **DISK**
<B/BUF **addr**
A user variable which contains the number of bytes per buffer.

← **EDITOR**
<B/SCR **n**
This constant leaves the number of blocks per editing screen. By convention, an editing screen is 1024 bytes organized as 16 lines of 64 characters each.

← **EDITOR**
<B/SCRN **addr**
A user variable which contains the number of blocks per SCREEN.

← **DICTIONARY**
<BACK **addr**
Calculate the backward branch offset from **HERE** to **addr** and compile into the next available dictionary memory address.

← **CONVERSION**
<BASE **addr**
A user variable containing the current number base used for input and output conversion.

← **STACK**
<BASE->R
Place the current base on the return stack. See **R->BASE**.

← **STRUCTURE**
<BEGIN **addr n** (compiling)
Occurs in a colon-definition in the form:
BEGIN ... UNTIL
BEGIN ... AGAIN
BEGIN ... WHILE ... REPEAT
At run-time, **BEGIN** marks the start of a sequence that may be repetitively executed. It serves as a return point from the corresponding **UNTIL**, **AGAIN**, or **REPEAT**. When executing **UNTIL**, a return to **BEGIN** will occur if the top of the stack is false; for **AGAIN** and **REPEAT** a return to **BEGIN** always occurs.
At compile time, **BEGIN** leaves its return address and n for compiler error checking.

← **CONVERSION**
<BL **ch**
A constant that leaves the ASCII value for "blank".

← **MEMORY**
<BLANKS **addr cnt**
Fill an area of memory beginning at **addr** with **cnt** blanks.

← **INTERPRET**
<BLK **addr**
A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.

← **DISK**
<BLOAD **scr#** — **f**
Loads the binary image at **scr#** which was created by **BSAVE**. **BLOAD** returns a true flag (1) if the load was NOT successful and a false flag (0) if the load WAS successful.

← **DISK**
<BLOCK **n** — **addr**
Leave the memory address of the block buffer containing block n. If the block is not already in memory, it is transferred from disk to whichever buffer was least recently written. If the block occupying that buffer has been updated, it is rewritten to disk before block n is read into the buffer. See also **BUFFER**, **R/W**, **UPDATE**, and **FLUSH**.

← **INTERPRET**
<BOOT
Examines the SCREEN designated as the booting SCREEN (**SCR 3**). If it contains only displayable characters (32-127) it performs a **LOAD** on that SCREEN.

← **STRUCTURE**
<BRANCH
The run-time procedure to unconditionally branch. An in-line offset is added to the interpretive pointer **IP** to branch ahead or back. **BRANCH** is compiled by **ELSE**, **AGAIN**, and **REPEAT**.

← **EDITOR**
<BUFFER **n** — **addr**
Obtain the next memory buffer, assigning it to block n. If the contents of the buffer is marked as updated, it is written to disk. The block is not read from the disk. The address left is the first cell within the buffer for data storage.

← **MEMORY**
<C1 **b** **addr**
Store 8 bits at **addr**. Bytes always occupy the low order bits when on the stack.

← **DICTIONARY**
<C **b**
Store 8 bits of b into the next available dictionary byte, advancing the dictionary pointer. This instruction should be used with caution on byte addressing, word oriented computers such as the TI 9900.

← **EDITOR**
<CL **n**
Returns on the stack the number of characters per line.

← **EDITOR**
<CLs **addr**
A user variable whose value is the number of characters per line.

← **MEMORY**
<CB **addr** — **b**
Leave the 8 bit contents of the memory address on the stack.

← **STRUCTURE**
<CASE **n**
Initiates the construct:
CASE...OF...ENDOF...ENDCASE

← **DICTIONARY**
<CFA **pfa** — **cfa**
Convert the parameter field address of a definition to its code field address.

← **EDITOR**
<CLEAR **scr#**
Fills the designated screen with blanks.

← **MEMORY**
<CHOKE **addr1 addr2 cnt**
Move the specified quantity of bytes beginning at **addr1** to **addr2**. The contents of **addr1** is moved first proceeding toward high memory.

← **INTERPRET**
<COLD
The **COLD** start procedure to adjust the dictionary pointer to the minimum standard and restart via **ABORT**. May be called from the terminal to remove application programs and restart. **COLD** calls **BOOT** prior to calling **ABORT**.

← **INTERPRET**
<COMPILE
When the word containing **COMPILE** executes, the execution address of the word following **COMPILE** is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in addition to simple compiling an execution address (which the interpreter already does).

CONSTANT A set

to cr When the

CONTEXT A user

COUNT Leave

CR Trans

CREATE A def

CSP A use

CURPOS A use

CURRENT A use

D+ Leave

D- Appl

D. Print

D.R Print

DABS Leave

DECIMAL Set

DEFINITE Used

Set the

CONTE spect

DIBIT Conv

DISK_MN A use

DISK_MI A use

DISK_LO A use

DISK_BIT A use

DLITERAL If ca

DO Occur

At pu contr

LOOP equal

exec run-1

a led stack

comp for

DOES A use

defin para

comp in

word area

asse gener

CONSTANT n --- MEMORY
A defining word used in the form `CONSTANT cccc` to create word `cccc`, with its parameter field containing `n`. When `cccc` is later executed, it will push the value of `n` to the stack.

CONTEXT $addr$ --- INTERPRET
A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.

COUNT $addr1$ $addr2$ n --- MEMORY
Leave the byte address (`addr2`) and byte count (`n`) of a message text beginning at `addr1`. It is presumed that the first byte at `addr1` contains the text byte count and the actual text starts with the second byte. Typically, **COUNT** is followed by **TYPE**.

CR --- PRINT
Transmit a carriage return and a line feed to the selected output device.

CREATE --- DICTIONARY
A defining word used in the form `CREATE cccc` by such words as **CONSTANT** and **CONTEXT** to create a dictionary header for a FORTH definition. The code field contains the address of the word's parameter field. The new word is created in the **CURRENT** vocabulary.

CSP $addr$ --- STACK
A user variable temporarily storing the stack pointer position, for compilation error checking.

CURPOS $addr$ --- UDP
A user variable that stores the current UDP cursor position.

CURRENT $addr$ --- INTERPRET
A user variable pointing to the vocabulary into which new definitions will be compiled.

D+ $d1$ $d2$ --- $d3$ --- ARITHMETIC
Leave a double number sum of two double numbers.

D- $d1$ n --- $d2$ --- ARITHMETIC
Apply the sign of `n` to the double number `d1`, leaving it as `d2`.

D. d --- PRINT
Print a signed double number from a 32 bit two's complement value. The high order 16 bits are most accessible on the stack. Conversion is performed according to the current **BASE**. A blank follows. Pronounced "D DOT".

D.A d n --- PRINT
Print a signed double number `d` right aligned in a field `n` characters wide.

DABS $d1$ --- $d2$ --- ARITHMETIC
Leave the absolute value of a double number.

DECIMAL --- CONVERSION
Set the numeric conversion **BASE** for decimal input/output.

DEFINITIONS --- INTERPRET
Used in the form `cccc DEFINITIONS` to set the **CURRENT** vocabulary to the **CONTEXT** vocabulary. In the example, executing vocabulary name `cccc` made it the **CONTEXT** vocabulary and executing **DEFINITIONS** made both specify vocabulary `cccc`.

DIBIT ch $n1$ --- $n2$ ff (ok) --- CONVERSION
Convert the ascii character `ch` (using **BASE** `n1`) to its binary equivalent `n2`, accompanied by a `ff` flag. If the conversion is invalid, leave only a false flag.

DISK_BUF $addr$ --- DISK
A user variable that points to the first byte in UDP RAM of the 1K disk buffer.

DISK_HI $addr$ --- DISK
A user variable which contains the **SCREEN** number immediately above the **SCREEN** range wherein **SCREEN** writes are permitted.

DISK_LO $addr$ --- DISK
A user variable which contains the first **SCREEN** number of the range wherein **disk** writes are permitted.

DISK_SIZE $addr$ --- DISK
A user variable whose value is the number of **SCREENS** logically assigned to a diskette.

DLITERAL d d (executing) --- INTERPRET
If compiling, compile a stack double number into a literal. Later execution of the definition containing the literal will push it on the stack. If executing, the number will remain on the stack.

DMINUS $d1$ --- $d2$ --- ARITHMETIC
Convert `d1` to its double number two's complement.

DO $n1$ $n2$ --- (execute) --- STRUCTURE
Occurs in a colon-definition in the form:
`DO ... LOOP` or `DO ... +LOOP`
At run-time, **DO** begins a sequence with repetitive execution controlled by a loop limit `n1` and an index with initial value `n2`. **DO** removes these from the stack. Upon reaching **LOOP**, the index is incremented by 1. Until the new index equals or exceeds the limit, execution loops back to just after **DO**; otherwise the loop parameters are discarded and execution continues ahead. Both `n1` and `n2` are determined at run-time and may be the result of other operations. Within a loop, **I** will copy the current value of the index to the stack. When compiling within the colon-definition, **DO** compiles **(DO)**, leaving the following address (`addr`) and `n` for later error checking. See **I**, **LOOP**, **+LOOP** and **LEAVE**.

DOES --- INTERPRET
A word which defines the run-time action within a high-level defining word. **DOES** alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following **DOES**. It is always used in combination with **(BUILDS)**. When the **DOES** part executes, it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area of its contents. Typical uses include the FORTH assembler, multi-dimensional arrays, and compiler generation.

DP $addr$ --- DICTIONARY
A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by **HERE** and altered by **RLIMIT**.

DPL $addr$ --- CONVERSION
A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used to hold output custom location of a decimal point. In user generated formatting, the default value on single number input is -1.

DRS OR1 R2 --- DISK
Command to select disk drives by pre-setting **OFFSET**. The contents of **OFFSET** is added to the block number in **BLOCK** to allow for this selection. **OFFSET** is suppressed for error text to that it may always originate from drive 0.

DRIVE n --- DISK
Adjusts **OFFSET** so that the drive number on the stack becomes the first drive in the system.

DROP n --- STACK
Drop the top number from the stack.

DUP a --- a --- STACK
Duplicate the value on the stack.

ECOUNT $addr$ --- ERROR
A user variable which contains an error count. This is used to prevent error recursion.

ELSE $addr1$ $n1$ --- $addr2$ $n2$ (comp) --- STRUCTURE
Occurs within a colon-definition in the form:
`IF ... ELSE ... ENDF`
At run-time, **ELSE** executes after the true part following **IF**. **ELSE** forces execution to skip over the following false part and resume execution after **ENDIF**. It has no stack effect.
At compile-time, **ELSE** emplaces **BRANCH** reserving a branch offset and leaves the address `addr2` and `n2` for error testing. **ELSE** also resolves the pending forward branch from **IF** by calculating the offset from `addr1` to **HERE** and storing it at `addr1`.

EMIT ch --- PRINT
Transmit ascii character `ch` to the selected output device. **OUT** is incremented for each character output.

EMITB ch --- PRINT
Transmit an 8-bit character to the selected output device. **OUT** is incremented for each character output.

EMPTY-BUFFERS --- DISK
Mark all block-buffers as empty, not necessarily affecting the contents. Updated blocks are not written to disk. This is also an initialization procedure before first use of the disk.

ENCLOSE $addr1$ ch --- $addr1$ $n1$ $n2$ $n3$ --- MEMORY
The text scanning primitive used by **WORD**. From the text address `addr1` and an ascii delimiting character `ch`, is determined the byte offset to the first non-delimiter character `n1`, the offset to the delimiter after the text `n2`, and the offset to the first character not included. This procedure will not process past an ascii 'null', treating it as an unconditional delimiter.

ENO f --- STRUCTURE
This is an 'alias' or duplicate definition for **UNTIL**.

ENDCASE --- STRUCTURE
Terminates the **CASE** construct.

ENDIF $addr$ n --- (compile) --- STRUCTURE
Occurs in a colon-definition in the form:
`IF ... ENDF`
At run-time, **ENDIF** serves only as the destination of a forward branch from **IF** or **ELSE**. It marks the conclusion of the conditional structure. **THEN** is another name for **ENDIF**. Both names are supported in fig-FORTH. See also **IF** and **ELSE**. At compile-time, **ENDIF** computes the forward branch offset from `addr` to **HERE** and stores it at `addr`. `n` is used for error tests.

ENEOF --- STRUCTURE
Terminates the **OF** construct within the **CASE** construct.

ERASE $addr$ n --- MEMORY
Clear a region of memory to zero from `addr` over `n` bytes.

ERROR $n1$ --- $n2$ $n3$ --- ERROR
Execute error notification and restart of system. **WARNING** is first examined. If 1, the text of line `n1`, relative to screen 4 of drive 0, is printed. This line number may be positive or negative, and beyond just screen 4. If **WARNING**=0, `n1` is just printed as a message number (non-disk installation). If **WARNING** is -1, the definition **(ABORT)** is executed, which executes the system **ABORT**. The user may cautiously modify this execution by altering **(ABORT)**. fig-FORTH saves the contents of **IN** (`n2`) and **BLK** (`n3`) to assist in determining the location of the error. Final action is execution of **QUIT**.

EXECUTE cfa --- INTERPRET
Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.

EXPECT $addr$ cnt --- MEMORY
Transfer characters from the terminal to `addr`, until a "ENTER" or the count of characters has been received. One or more nulls are added at the end of the text.

FENCE $addr$ --- DICTIONARY
A user variable containing an address below which **FORGET** is trapped. To **FORGET** below this point the user must alter the contents of **FENCE**.

FILL $addr$ cnt b --- MEMORY
Fill memory beginning at `addr` with the specified number (`cnt`) of bytes `b`.

FIRST $addr$ --- DISK
A constant that leaves the address of the first (lowest) block buffer.

FIRSTB $addr$ --- DISK
A user variable which contains the first byte of the disk buffer area.

PRINT **addr**
A user variable for control of number output field width. Presently used in `FIG FORTH` and `YI FORTH`.

FLUSH **EDITOR**
Rewrites to the disk all disk buffers that have been updated.

FORGET **DICTIONARY**
Executed in the form: `FORGET cccc`
Deletes definition named `cccc` from the dictionary with all entries physically following it.

FORTH **DICTIONARY**
The name of the primary vocabulary. Execution makes `FORTH` the `CONTEXT` vocabulary. Until additional user vocabularies are defined, new user definitions become a part of `FORTH`. `FORTH` is immediate, so it will execute during the creation of a colon-definition to select this vocabulary at compile time.

FORTH-LINK **addr** **DICTIONARY**
A user variable used for vocabulary linkage.

GOTOXY **c r** **UDP**
Places the cursor at the designated column and row position. NOTE: Rows and columns are numbered from 0.

HERE **addr** **DICTIONARY**
Leave the address of the next available dictionary location.

HEX **CONVERSION**
Set the numeric conversion base to sixteen (hexadecimal).

HLD **addr** **CONVERSION**
A user variable that holds the address of the latest character of text during numeric output conversion.

HOLD **ch** **CONVERSION**
Used between `(0` and `)` to insert an ASCII character into a pictured numeric output string. e.g. `2E HOLD` will place a decimal point.

I **n** **STRUCTURE**
Used within a `DO-LOOP` to copy the loop index to the stack. Other use is implementation dependent. See `R`.

ID. **nfa** **DICTIONARY**
Print a definition's name from its name field address.

IF **f** **(run-time)** **STRUCTURE**
Occurs in a colon-definition in the form:
`IF (fp) ... ENDIF`
`IF (fp) ... ELSE (fp) ... ENDIF`
At run-time, `IF` selects execution based on a boolean flag. If `f` is true (non-zero), execution continues ahead thru the true part. If `f` is false (zero), execution skips to just after `ELSE` to execute the false part. After each part, execution resumes after `ENDIF`. `ELSE` and its false part are optional; if missing, false execution skips to just after `ENDIF`.
At compile time, `IF` compiles `BRANCH` and reserves space for an offset of `addr`. `addr` and `n` are used later for resolution of the offset and error checking.

IMMEDIATE **INTERPRET**
Mark the most recently made definition so that when encountered at compile time, it will be executed rather than being compiled. I.e. the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by preceding it with `COMPILE!`.

IN **addr** **KEYBOARD**
A user variable containing the byte offset within the current input text buffer (terminal or disk) from which the next text will be accepted. `WORD` uses and moves the value of `IN`.

INTERPRET **INTERPRET**
The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or disk) depending on `STATE`. If the word name cannot be found after a search of `CONTEXT` and then `CURRENT` it is converted into a number according to the current base. That also failing, an error message echoing the name with a `?` will be given. Text input will be taken according to the convention for `WORD`. If a decimal point is found as part of a number, a double number value will be left. The decimal point has no other purpose than to force this action. See `NUMBER`.

INTLNK **addr** **ISR**
A user variable which is a pointer to the Interrupt Service linkage.

ISR **addr** **ISR**
A user variable that initially contains the address of the interrupt service linkage code to install an Interrupt Service Routine. The user must modify `ISR` to contain the CFA of the routine to be executed each 1/60 second. Next, the contents of `HEX 83C4` must be modified to point to this address. Note, the interrupt service linkage code address is also available in `INTLNK`.

J **n** **STRUCTURE**
Copies the loop index of the second innermost loop to the stack.

KEY **ch** **KEYBOARD**
Leave the ASCII value of the next terminal key struck.

KEYS **ch** **KEYBOARD**
Leave the 8-bit value of the next terminal key struck.

L/SCR **n** **EDITOR**
Returns on the stack the number of lines per screen.

LATEST **nfa** **DICTIONARY**
Leave the name field address of the topmost word in the `CURRENT` vocabulary.

LEAVE **STRUCTURE**
Force termination of a `DO-LOOP` at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution proceeds normally until `LOOP` or `+LOOP` is encountered.

LFA **pfa** **ifa** **DICTIONARY**
Convert the parameter field address of a dictionary definition to its link field address.

LIMIT **addr** **DISK**
A constant which leaves the address just above the highest memory available for a disk buffer.

LIMITS **addr** **DISK**
A user variable which contains the address just above the highest memory available for a disk buffer.

LIST **scrn** **PRINT**
Lists the specified `SCREEN` to the output device. See `PAUSE`.

LIT **n** **INTERPRET**
Within a colon-definition, `LIT` is automatically compiled before each 16 bit literal number encountered in input text. Later execution of `LIT` causes the contents of the next dictionary address to be pushed on the stack.

LITERAL **n** **(compiling)** **INTERPRET**
If compiling, then compile the stack value `n` as a 16 bit literal. This will execute during a colon-definition. The intended use is:
`xxx [calculate] LITERAL ;`
Compilation is suspended for the compile-time calculation of a value. Compilation is resumed and `LITERAL` compiles this value.

LOAD **n** **INTERPRET**
Begin interpretation of `SCREEN n`. Loading will terminate at `R` the end of the `SCREEN` or at `IS`. See `IS` and `—`.

LOOP **addr n** **(compiling)** **STRUCTURE**
Occurs in a colon-definition in the form:
`DO ... LOOP`
At run-time, `LOOP` selectively controls branching back to the corresponding `DO` based on the loop index and limit. The loop index is incremented by one and compared to the limit. The branch back to `DO` occurs until the index equals or exceeds the limit; at that time, the parameters are discarded and execution continues ahead.
At compile-time, `LOOP` compiles `(LOOP)` and uses `addr` to calculate an offset to `DO`. `n` is used for error testing.

M* **n1 n2** **d** **ARITHMETIC**
A mixed magnitude math operation which leaves the double number signed product of two signed numbers.

M/ **d n1** **n2 n3** **ARITHMETIC**
A mixed magnitude math operation which leaves the signed remainder `n2` and signed quotient `n3`, from a double number dividend and divisor, `n1`. The remainder takes its sign from the dividend.

M/MOD **u1 u2** **u3 u4** **ARITHMETIC**
An unsigned mixed magnitude math operation which leaves a double quotient `u4` and remainder `u3`, from a double dividend `u1` and a single divisor `u2`.

MAX **n1 n2** **n3** **ARITHMETIC**
Leave the greater of the two numbers.

MESSAGE **n** **PRINT**
Print on the selected output device the text of line `n` relative to screen 4 of drive 0. `n` may be positive or negative. `MESSAGE` may be used to print incidental text such as report headers. If `WARNING` is zero, the message will simply be printed as a number (disk un-available).

MIN **n1 n2** **n3** **ARITHMETIC**
Leave the smaller of the two numbers.

MINUS **n1** **n2** **ARITHMETIC**
Leave the two's complement of a number.

MOD **n1 n2** **mod** **ARITHMETIC**
Leave the remainder of `n1/n2`, with the same sign as `n1`.

MOVE **addr1 addr2 n** **MEMORY**
Move the contents of `n` memory cells (16 bit contents) beginning at `addr1` into `n` cells beginning at `addr2`. The contents of `addr1` is moved first.

MYSELF **INTERPRET**
Used in a colon definition. Places the CFA of a routine into itself. This permits recursion.

NFA **pfa** **nfa** **DICTIONARY**
Convert the parameter field address of a definition to its name field address.

NOP **INTERPRET**
A do nothing instruction. `NOP` is useful for patching as in assembly code.

NUMBER **addr** **d** **CONVERSION**
Convert a character string left at `addr` with a preceding count, to a signed double number, using the current numeric base. If a decimal point is encountered in the text, its position will be given in `DPL`, but no other effect occurs. If numeric conversion is not possible, an error message will be given.

OF **n** **STRUCTURE**
Initiates the `OF ... ENDOF` construct inside of the `CASE` construct. `n` is compared to the value which was on top of the stack when `CASE` was executed. If the numbers are identical, the words between `OF` and `ENDOF` will be executed.

OFFSET **addr** **DISK**
A user variable which may contain a block offset to disk drives. The contents of `OFFSET` is added to the stack number by `BLOCK`, messages issued by `MESSAGE` are independent of `OFFSET`. See `BLOCK`, `DRG`, `MESSAGE`.

OR **n1 n2** **n3** **LOGICAL**
Leave the bit-wise logical OR of two 16 bit values.

OUT **addr** **PRINT**
A user defined variable that contains a value incremented `EMIT` and `EMITB`. The user may alter and examine `OUT` to control display formatting.

OVER **n1 n2** **n1 n2 n1** **STACK**
Copy the second stack value, placing it as the new top.

PAGE **addr** **UDP**
A user variable which points to a region in `UDP RAM` which has been set aside for creating `PAGEs`.

PAUSE **PRG**
Leave fixed

PAUSE **PRG**
The `PAUSE` by `PRG` can't press

PFA **CONV**
Its `P`

PREU **A var**
reca to be

QUERY **Input**
opera conte

QUIT **Clear**
to th

Copy

R0 **A user**
cursor

R-BASE **Reasi**
BASE-
BASE-

R/W **The f**
speci sequ for f
mass check

R0 **A user**
RETURN

R) **Remove**
the `pr`

ROISK **the pr**
address the `bl`

R) **n3 is**

REPEAT **Used**
t

At pur
just s

At sed
HERE

ROT **Rotate**
to the

RPI **A proc**
variab

6-XD **Sign**
d

8-XF **Conver**
float

8-XFC **Takes**
to file

sg **Pronou**

SCR **A user**
refer

SCRN_LEN0 **A user**
immedi tabl

SCRN_STAR **A user**
the `sc`

SCRN_WID0 **A user**
which screen

SIGN **Stores**
conver when a
signed

SLA **Arithm**
the `16`
modula
shifts
could
`SLA EM`

SHUDGE **Used**
d
defin
defin
until

SP1 **A proc**

CTIONARY
 (empty)

the highest

ASK
 at above the

PRINT
 See

INTERPRET
 by compiled
 to input
 bits of the
 stack.

INTERPRET
 as a 16 bit
 initiation. The

calculation of
 applies this

INTERPRET
 terminate at

STRUCTURE

ing back to th
 (limit. The
 to the limit.
 equals or
 are

is addr to
 or testing.

ARITHMETIC
 the double

ARITHMETIC
 the signed
 double number
 its sign from

ARITHMETIC
 which leaves a
 double dividen

ARITHMETIC

PRINT
 of line n
 blank or
 until text such
 as msg will
 (while).

ARITHMETIC

ARITHMETIC

ARITHMETIC
 sign as n1.

MEMORY
 contents)
 addr-2. The

INTERPRET
 a routine

DICTIONARY
 addition to its

INTERPRET
 catching as in

CONVERSION
 a preceding
 current numeric
 the text. Its
 effect occurs.
 error message will

STRUCTURE
 of the CASE
 cases on top of
 bars are
 will be executed

DISK
 fast to dist
 the stack number
 dependent of

LOGICAL
 values.

PRINT
 as incremented
 line OUT to

STACK
 the new top.

TOP
 TOP RAM which

PA0 — addr — **DICTIONARY**
 Leave the address of the text output buffer, which is a fixed offset above HERE.

PAUSE — f — **PRINT**
 The words LIST, INDEX, DUP and ULIST all call the word PAUSE. Pause allows the user to temporarily halt the output by pressing any key. Pressing another key will allow continuation. To exit one of these routines prematurely, press BREAK.

PFA nfa — pfa — **DICTIONARY**
 Convert the name field address of a compiled definition to its parameter field address.

PREV — addr — **DISK**
 A variable containing the address of the disk buffer most recently referenced. The UPDATE command marks this buffer to be later written to disk.

QUERY — **KEYBOARD**
 Input 80 characters of text (or until a "enter") from the operator's terminal. Text is positioned at the address contained in TIB with IN set to zero.

QUIT — **INTERPRET**
 Clear the return stack, stop compilation, and return control to the operator's terminal. No message is given.

R — n — **STACK**
 Copy the top of the return stack to the parameter stack.

RA — addr — **PRINT**
 A user variable which may contain the location of an editing cursor, or other file related function.

R->BASE — **STACK**
 Restores the current base from the return stack. See BASE->R.

R/W addr n1 f — **DISK**
 The fig-FORTH standard disk read-write linkage. addr specifies the source or destination block buffer, n1 is the requested number of the referenced blocks, and f is a flag for r/w write and r/l read. R/W determines the location on disk storage, performs the read-writes and performs error checking.

RA — addr — **STACK**
 A user variable containing the initial location of the return stack. Pronounced "R zero". See RPI.

R> — n — **STACK**
 Remove the top value from the return stack and leave it on the parameter stack. See >R and R.

ROISK addr n1 n2 — n3 — **DISK**
 The primitive routine that performs disk reads. addr is the address where the block is to be written in CPU RAM, n1 is the block number, n2 is the number of bytes per block, and n3 is the returned error code.

REPEAT addr n — (compiling) — **STRUCTURE**
 Used within a colon-definition in the form: BEGIN ... WHILE ... REPEAT
 At run-time, REPEAT forces an unconditional branch back to just after the corresponding BEGIN.
 At compile-time, REPEAT compiles BRANCH and the offset from HERE to addr. n is used for error testing.

ROT n1 n2 n3 — n2 n3 n1 — **STACK**
 Rotates the top three values on the stack, bringing the third to the top.

RPI — **STACK**
 A procedure to initialize the return stack pointer from user variable RA.

S->D n — c — **CONVERSION**
 Sign extend a single number, to form a double number.

S->F n — f1 — **CONVERSION**
 Converts a single precision number on the stack to a floating point number.

S->FAC n — **CONVERSION**
 Takes a single precision number from the stack, converts it to floating point, and leaves it in FAC.

SG — addr — **STACK**
 Pronounced "S zero". See SP1.

SCR — addr — **EDITOR**
 A user variable containing the screen number most recently referenced by LIST or EDIT.

SCR_LEN — addr — **UP**
 A user variable containing the address of the byte immediately following the last byte of the screen image table to be used as the logical screen.

SCR_START — addr — **UP**
 A user variable containing the address of the first byte of the screen image table to be used as the logical screen.

SCR_WIDTH — addr — **UP**
 A user variable which contains the number of characters which will fit across the screen. (32 or 48) Used by the screen scroller.

SIGN n d — d — **CONVERSION**
 Stores an ascii "-" sign at the current location in a converted numeric output string in the text output buffer when n is negative, n is discarded, but double number d is maintained. Must be used between (0 and 0).

SLA n1 cnt — n2 — **ARITHMETIC**
 Arithmetically shifts the number on the stack cnt bits to the left, leaving the result on the stack. cnt will be modulo 16, except when cnt=0, causing 16 bits to be shifted. To create a word which permits shifts when cnt could be zero, use the following definition: SLA0 -DUP IF SLA ENDOIF ;

SMUDGE — **DICTIONARY**
 Used during word definition to toggle the "smudge bit" in a definition's name field. This prevents an uncompiled definition from being found during dictionary searches, until compiling is completed without error.

SP1 — **STACK**
 A procedure to initialize the stack pointer from SG.

SP0 — **STACK**
 A procedure to return the address of the stack position to the top of the stack, as it was before SP0 was executed. (e.g. 1 2 SP0 0 . . . would type 2 2 1).

SPACE — **PRINT**
 Transmit an ascii blank to the output device.

SPACES n — **PRINT**
 Transmit n ascii blanks to the output device.

SRA n1 cnt — n2 — **ARITHMETIC**
 Arithmetically shifts n1 cnt bits to the right and leaves the result on the stack. cnt will be modulo 16, except when cnt=0, when 16 bits will be shifted. To create a word which permits shifts when cnt could be zero, use the following definition: SRA0 -DUP IF SRA ENDOIF ;

SRC n1 cnt — n2 — **ARITHMETIC**
 Performs a circular right shift of cnt bits on n1 leaving the result on the stack.

SRL n1 cnt — n2 — **LOGICAL**
 Performs a logical right shift of cnt bits and leaves the result on the stack. cnt will be modulo 16, except when cnt=0, when 16 bits will be shifted. To create a word which permits shifts when cnt could be zero, use the following definition: SRL0 -DUP IF SRL ENDOIF ;

STATE — addr — **INTERPRET**
 A user variable containing the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent.

SWAP n1 n2 — n2 n1 — **STACK**
 Exchange the top two values on the stack.

SWP n1 — n2 — **MEMORY**
 Reverses the order of the two bytes in n1 and leaves the new number as n2.

SYSE — addr — **INTERPRET**
 A user variable that contains the address of the system support entry point.

SYSTEM n — **INTERPRET**
 Calls a system synonym. You must specify an offset n into a jump table for the routine you wish to call. n must be one of the predefined even numbers.

TASK — **DICTIONARY**
 A no-operation word which can mark the boundary between applications. By forgetting TASK and re-compiling, an application can be discarded in its entirety.

THEN — **STRUCTURE**
 An alias for ENDOIF.

TIB — addr — **KEYBOARD**
 A user variable containing the address of the terminal input buffer.

TOGGLE addr b — **MEMORY**
 Complement the contents of the byte at addr by the bit pattern b.

TRAVERSE addr1 n — addr2 — **DICTIONARY**
 Move across the name field of a fig-FORTH variable length name field. addr1 is the address of either the length byte or the last letter. If n=1, the motion is toward high memory; if n=-1, the motion is toward low memory. The addr2 resulting is the address of the other end of the name.

TRIA0 scr0 — **PRINT**
 Display on the RS232 the three SCREENS which include that number scr0, beginning with a SCREEN evenly divisible by three. Output is suitable for source text records and includes a reference line at the bottom taken from line 15 of screen 4.

TRIA0S scr0 scr0 — **PRINT**
 May be thought of as a multiple TRIA0. You must specify a SCREEN range. TRIA0S will perform as many TRIA0's as necessary to cover that range.

TYPE addr cnt — **PRINT**
 Transmit count characters from addr to the selected output device.

U — n — **STACK**
 Places the contents of register U on the stack. Register U contains the base address of the user variable area.

U* u1 u2 — ud — **ARITHMETIC**
 Leave the unsigned double number product of two unsigned numbers.

U. u — **PRINT**
 Prints an unsigned number to the output device.

U.R u n — **PRINT**
 Prints an unsigned number right justified in a field of width n.

U/ ud u1 — u2 u3 — **ARITHMETIC**
 Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.

U@ — addr — **INTERPRET**
 A user variable that points to the junction between the user variable area and the return stack.

UK u1 u2 — f — **STRUCTURE**
 Leaves a true flag if u1 is less than u2, else leaves a false flag.

UCONS — addr — **INTERPRET**
 A user variable which contains the base address of the user variable default area which is used to initialize the user variables at COLD.

UD. ud — **PRINT**
 Prints an unsigned double number to the output device.

UD.R ud n — **PRINT**
 Prints an unsigned double number right justified in a field of length n.

YOUR FORTH NOTES

FORGETTABLE `addr` `f` **DICTIONARY**
Decides whether or not a word can be forgotten. A true flag is returned if the address is not located between **HERE** and **HERE**.

UNTIL `f` `addr n` **STRUCTURE**
`f` --- (run-time)
`addr n` --- (compile)
Occurs within a colon-definition in the form:
`BEGIN ... UNTIL`
At run-time, **UNTIL** controls the conditional branch back to the corresponding **BEGIN**. If `f` is false, execution returns to just after **BEGIN**; if true, execution continues ahead.
At compile-time, **UNTIL** compiles (**BRANCH**) and an offset from **HERE** to `addr. n` is used for error tests.

UPDATE `---` **DISK**
Marks the most recently referenced block (pointed to by **PREV**) as altered. The block will subsequently be transferred automatically to disk should its buffer be required for storage of a different block.

USE `---` `addr` **EDITOR**
A variable containing the address of the block buffer to use next, as the least recently written.

USER `n` **MEMORY**
A defining word used in the form:
`n USER cccc`
which creates a user variable `cccc`. The parameter field of `cccc` contains `n` as a fixed offset relative to the user pointer register **UP** for this user variable. When `cccc` is later executed, it places the sum of its offset and the user area base address on the stack as the storage address of that particular variable.

VARIABLE `n` **MEMORY**
A defining word used in the form:
`n VARIABLE cccc`
When **VARIABLE** is executed, it creates the definition `cccc` with its parameter field initialized to `n`. When `cccc` is later executed, the address of its parameter field (containing `n`) is left on the stack, so that a fetch or store may access this location.

VOC-LINK `---` `addr` **DICTIONARY**
A user variable containing the address of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control for **FORGETTING** thru multiple vocabularies.

VOCABULARY `---` **DICTIONARY**
A defining word used in the form:
`VOCABULARY cccc`
to create a vocabulary definition `cccc`. Subsequent use of `cccc` will make it the **CONTEXT** vocabulary which is searched first by **INTERPRET**. The sequence "cccc DEFINITIONS" will also make `cccc` the **CURRENT** vocabulary into which new definitions are placed.
`cccc` will be so chained as to include all definitions of the vocabulary in which `cccc` is itself defined. All vocabularies ultimately chain to **FORTH**. By convention, vocabulary names are to be declared **IMMEDIATE**. See **VOC-LINK**.

WARNING `---` `addr` **ERROR**
A user variable containing a value controlling messages. If a disk is present, and screen 4 of drive 0 is the base location for messages, if `addr`, no disk is present and messages will be presented by number. If `-1`, execute (**ABORT**) for a user specified procedure. See **MESSAGE**, **ERROR**.

WRITE `addr n1 n2` `---` `n3` **DISK**
The primitive routine which performs a disk write. `addr` is the CPU RAM location of the block to be written. `n1` is the block number, `n2` is the number of bytes per block, and `n3` is the returned error code.

WHILE `f` `---` (run-time) **STRUCTURE**
`addr1 n1` `---` `addr1 n1` `addr2 n2`
Occurs in a colon-definition in the form:
`BEGIN ... WHILE(ip) ... REPEAT`
At run-time, **WHILE** selects conditional execution based on boolean flag `f`. If `f` is true (non-zero), **WHILE** continues execution of the true part thru to **REPEAT**, which then branches back to **BEGIN**. If `f` is false (zero), execution skips to just after **REPEAT**, exiting the structure.
At compile-time, **WHILE** replaces (**BRANCH**) and leaves `addr2` of the reserved offset. The stack values will be resolved by **REPEAT**.

WIDTH `---` `addr` **DICTIONARY**
A user variable containing the maximum number of letters saved in the compilation of a definition's name. It must be 1 thru 31, with a default value of 31. The name character count and its natural characters are saved, up to the value in **WIDTH**. The value may be changed at any time within the above limits.

WORD `ch` **MEMORY**
Read the text characters from the input stream being interpreted, until a delimiter `ch` is found, storing the packed character string beginning at the dictionary buffer **HERE**. **WORD** leaves the character count in the first byte, the characters, and ends with two or more blanks. Leading occurrences of `ch` are ignored. If **BLK** is zero, text is taken from the terminal input buffer otherwise from the disk block stored in **BLK**. See **BLK**, **IN**.

XOR `n1 n2` `---` `n3` **LOGICAL**
Leave the bitwise logical **EXCLUSIVE OR** of two values.

C `---` **INTERPRET**
Used in a colon-definition in the form:
`xxxx [words] more`
Suspend compilation. The words after **C** are executed, not compiled. This allows calculation or compilation exceptions before resuming compilation with **J**. See **LITERAL**, **J**.

COMPILED `---` **INTERPRET**
Used in a colon-definition in the form:
`xxxx [COMPILED] FORTH`
COMPILED will force the compilation of an immediate definition, that would otherwise execute during compilation. The above example will select the **FORTH** vocabulary when `xxxx` executes, rather than at compile time.

J `---` **INTERPRET**
Resume compilation, to the completion of a colon-definition. See **C**.

As
&
re
to
La
ex
to
ch
in
an
be
ou
ma
We
pr
wi
re
co
to
ev
pr
wo
be
po
fo
99
me
an
in
if
ou
ma
ar
30
10
20
30
40
50
RC
10
to
01
2


```

1370 FOR I=1 TO 3
1380 CALL SOUND(200,550,2)
1390 CALL SOUND
1400 NEXT I
1410 CALL HCHAR
1420 CALL HCHAR
1430 GOTO 1870
1440 TOTAL=TOTAL
145
146
147
1480 CALL SOUND(200,440,2)
1490 CALL HCHAR(17,23,51)
1500 CALL HCHAR(17,24,48)
1510 GOTO 1870

```

```

1890 CALL HCHAR(19,22+I,ASC(
SEG$(STR$(TOTAL),I,1)))
TO LEN(STR$(PLA
HAR(21,22+I,ASC(
AYS),I,1)))

```

EXPLORING BASIC WITH

THE H. V. 99ERS BASIC GROUP

```

1970 IF K=78 THEN 1120
1980 IF K<>83 THEN 1940
1990 CALL CLEAR
2000 END

```

As yet another month has drifted by & it's time to put something readable together again - hopefully to be of interest to the majority. Last month we printed a couple of example problems and our solutions to them, and a problem issued as a challenge to anyone who was interested in putting forward their answer. However as our Editor can't be kept waiting we will have to show our answer to it this month and maybe yours next month (we hope). We also include another couple of problems for you to tax your brain with and hopefully share your results with us. However we are confident our problems and solutions to them will be beneficial to you even if you just study the programming logic in them. Also we would like to add that TI Basic being slow doesn't mean its not a powerful and interesting language for all ages. The simple fact that 99.9% of us only learn it as a hobby means we can learn at our own pace and the more we learn the more interesting it can become. Anyway if you studied or attempted to solve our problem 1-1 in last months magazine here's the solution we arrived at.

SOLUTION 1-1.

```

10 CALL CLEAR
20 FOR X=0 TO 50 STEP 2
30 PRINT A,
40 LET A=A+2
50 NEXT X

```

PROBLEM 2-1.

Now the first problem this month is to write a program to evaluate the following equation

$$B^2+2*B-3$$

Given that $B=0,0.1,0.2,$ etc to 1.9,2.0. Print the values of B and the value of the equation on the same line.

PROBLEM 2-2.

Write a program as short as possible to select six random numbers under 30 not repeating any of the six.

If you wish to contribute to this article please send input to this address:

ALAN FRANKS C/O THE SECRETARY
HV99 USER GROUP
6 ARCOT CLOSE,
TARRO 2322

Finally we would appreciate any constructive criticism or advice. (even praise won't fall on deaf ears.)

Thank you.
Alan Franks

A CURLY ONE?

From Rodney Gainsford and the BASIC Class.

This is a program which we hope will baffle you as much as it did us. Try running this in Basic then try it in Extended. WHY DOES IT DO WHAT IT DOES?.....

```

100 CALL SCREEN(2)
110 CALL SCREEN(15)
120 GOTO 100

```

FORMATTER DEFAULT REVISIONS

Rick Cosmano, Vice-President of the Southern California Computer Group (S.C.C.G), at address PO Box 21181, El Cajon, CA 92021, USA is circulating a proposal to make changes in the default characters used for overstrike and underscore in the Formatter, with the intention that some uniformity be obtained around the TI-99 world.

His proposal is to replace the "at" sign (Shift-2) that causes heavy printing of the following characters, to be replaced by "TICK" (Fctn-C). The ampersand (Shift-7) for underscoring is also proposed for change to "BACKSLASH" (Fctn-Z). This would seem to be a very good change at least for the kind of material that we write for User Group Newsletters, where I know that you consciously avoid the use of the existing ones in any article that is likely to be Formatted, at the cost of some circumlocution, whereas I cannot ever remember having used the proposed alternatives in text writings. I will add the detailed instructions to the FWDOC/TIWR document file.

If you have reasons why the change should not become standard practice, or further suggestions on how the change can be made universal, contact Rick at the above address.

This change can be made in the original TI-Writer file FORM1 or the Funlwriter modified version FORMB1 by using a disk sector Editor on the offending bytes. These are located at addresses >A06D and >A06E in the first sector of the file (hex bytes 40 26). On the sector display these will be displaced along 6 bytes because of the file header. Change these bytes to 60 5C.

Rick gives further information for

those who are using the Miller's Gramcracker and its TIWGRANDSK utility. I can only repeat these here as we have no way of verifying it. Search for string 23 21 40 26 in FORM1 and replace the 40 26 as above. Rick's gramcracker location was 9A571.

Rick also gives information for changing default screen colors (the Gramcracker locations quoted are 9B2A5 and 9B2B4 for the existing color bytes, value >F5) This is of little value to the Funlwriter user as the program takes care of this automatically and far more conveniently.

DOES THIS MEAN YOU

Would the person who borrowed my instruction books for 'Milliken' Subtraction and Multiplication please return them as soon as possible please.

Gary Jones

*
* UPDATE - UPDATE - UPDATE - UPDATE *
*

*YOUR module library has expanded yet again we now have a total of 14 modules 1 tape and 1 set of joysticks:

- 1 EXTENDED BASIC
- 2 THE ATTACK
- 3 MIND CHALLENGERS
- 5 DRAGON MIX
- 6 CAR WARS
- 7 ALLIGATOR MIX
- 8 TYPING TUTOR
- 10 TOMBSTONE CITY
- 11 ADDITION AND SUBTRACTION
- 12 DIVISION 1
- 13 READING ON
- 14 BLASTO

MISSING PERSONS!!!!

A.HART M.PENMAN see me as soon as possible.....

RODNEY GAINSFORD

This
peop
Ext
recc

The
and

SEL
the

the
mon

or
for

tap

pre

cos
nee

ins
It
for

ex

If
the

so
BR

HOME RECORD KEEPER

This programme is designed for people who only have the console, Extended Basic and a cassette recorder to save their files on.

The first menu gives eight choices, and they are:-

- 1 Add data
- 2 Display month
- 3 Change month
- 4 Save file
- 5 Load file
- 6 Yearly Total
- 7 Sorted
- 8 Exit

SELECT:-

1 Add data - to add new data to the file

2 Display month - to display on the screen the data for a particular month.

3 Change month - for changing or deleting any data already input for a month.

4 Save file - saves the data to tape.

5 Load file - loads data previously saved on tape.

6 Yearly Total - gives total cost for the year and the total needed for next year.

7 Sorted - displays all instances of a selected Category or Item and the total of the selection for this year and next year.

8 Exit - the neater way of exiting the programme.

If you select 1 2 or 3, a list of the months is displayed. Just select the required month and press ENTER. When entering data you do not have to select the months in order.

Press 1 at the first menu and then any month you wish and press ENTER. And the screen changes to look like this:-

Category (Item)	Used Value	Th_Year (N_Year)
1 ()		()
2 ()		()
3 ()		()
4 ()		()
5 ()		()

The cursor appears next to the 1, ready for you to type in the category. In this instance lets call it HOUSE. You may notice at this point that all the inputs are in upper case regardless of whether the alpha lock is up or down, and you are restricted to five letters. The cursor then drops down a line ready to accept the (Item), type in RATES, which is immediately surrounded by brackets to identify it. The cursor then moves over under the Used Value column. Actually in this column you may input any identifying name or number you like, this time type in 35000 for the amount you are rated on. Now we move over to the last column, the Th_Year (N_Year) one. There we type in the cost of the rates, say 475, and you will notice that if no cents are inputed the decimal point and two zeros are added to the input. Next, at the bottom of the screen you will be asked to input the expected inflation on this item for the year is. Let's type in 7.3 this time, and then under the 475.00 is displayed (509.68), the brackets around it to remind you that that is (N_Year) figure, that is 475 plus 7.3%. The cursor drops down next to the two, make the category CAR the item INS short for insurance, and over in the next column you could type in the amount the car is insured for or if you have more than one car, which car it is. But to be clever type in F6000 to show that it is the Ford and it is insured for \$6000. Over the next column input the cost of the insurance, say 275.06, and again you are prompted for an inflation factor. Type in 5 and 288.81 is placed in the brackets to show next years cost. If that is all you have to input for that month, just press ENTER in the first

column to enter a null string, the total cost for the month for this year "Month tot th_year " and the monthly total for next year "Month tot n_year " is displayed on the bottom of the screen and the message "PRESS ANY KEY TO CONTINUE". Press a key and you go back to the first menu. From there you may select to add data to another month or go back to the month you were just at to add more data.

When you select 3 Change month, you are prompted for the month. The selected month is displayed and you are prompted for which row and are given the choice of 1=Delete 2=Change. If you press 1 the previously selected row is deleted and all the other lines are moved up one row. If 2 is pressed then the cursor is placed at the start of the previously selected row, ready for the changes, with things generally proceeding the same as when you are inputting new data. After you have made your changes to the selected row you are asked if you have any more changes to make. If the answer is no then the monthly totals are displayed, you press a key and you go back to the main menu.

When you select to save a file or read a file, the programme saves or reads in data for every month of the year even if some months do not have any data in them. It also tells you which month it is up to as it does it. This may seem as though it would be the slowest way, but it turns out to be the fastest in the long run.

When you select 7 Sorted from the main menu, you are prompted as to whether you want a Category sorted out or an (Item), and then the name of it. The programme goes through and displays everything with that name, and finally the total cost of it for this year and next year. For example you could sort all the thing in Category HOUSE, for total house costs for the year, or say an (item) named INS for insurance, which would give you a total of all your insurance costs for the year.

That should be enough information to get you going with the programme for now, but if you have any questions please ask. don't

be like some adventure freak who shall remain nameless, who gave trying to edit a line using the micro word processor; when he wrote his column for the magazine he just retyped it. So again if there is something you do not understand ASK

Brian Rutherford

DISKETTES DE-MYSTIFIED

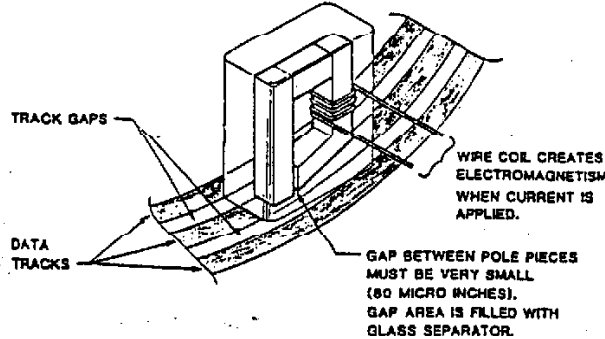
BY PAUL MULVANEY
H. V. 99ERS

The humble 5.25" mini diskette first appeared in 1976 and made recording data and programs for computers much easier and more reliable and a quicker task. It reduced saving and loading time from minutes to seconds for even small programs.

The heart of a diskette is a thin sheet of Mylar coated with a magnetic material such as iron oxide, capable of storing information which can be used at a later time. The information is imprinted on the magnetic material by a read/write head which works on the same principle as the head on a tape or cassette recorder.

The head on a disk drive performs the read, write and erase functions on a diskette. The outer covering of the head consists of a barium titanate ceramic material which is magnetically inert ie, it will not conduct magnetic pulses or signals. This supports the ferrite core and wire coil which serve as the electromagnet that rearranges the iron oxide coating on the Mylar diskette into meaningful information.

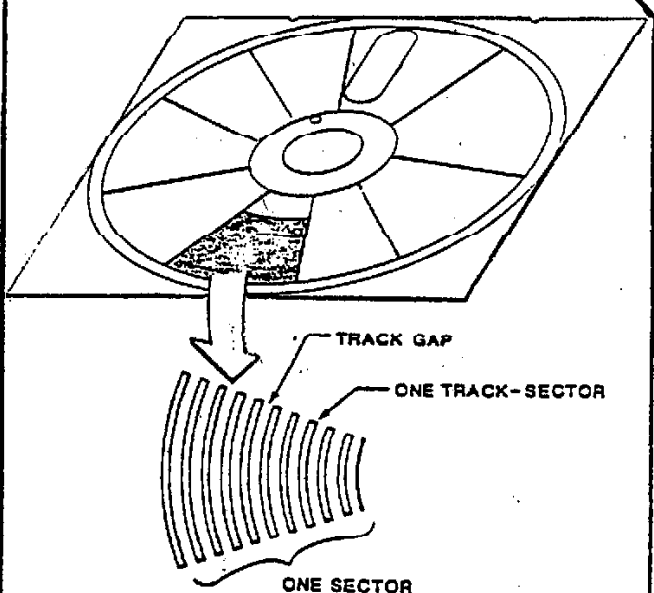
ask wh
ave u
ne min
te hi
jus
re i
nd ASK



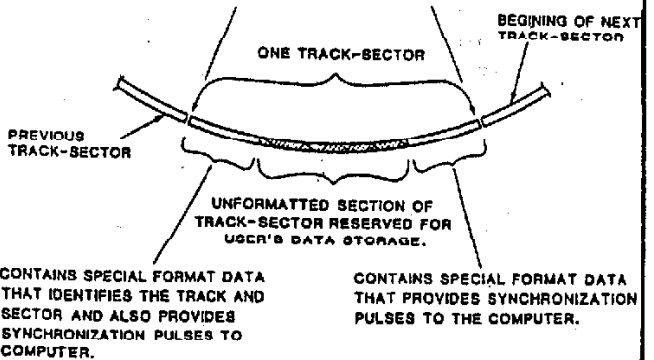
The computer sends the information to the disk controller card which sends electrical impulses to the head where they are converted into magnetic pulses by the coil wound around the ferrite core. The ferrite core is positioned such that the magnetic field produced rearranges the material on the diskette surface, the information will remain on the diskette until removed by another magnetic source. For this reason diskettes should be stored away from any source of magnetism, eg TV's, telephones, and any electric motor including the fan in the PE box.

Before a diskette can be used it must be formatted or initialized. This is because computer systems are not compatible with each other and therefore each diskette must be set up by the computer to its own requirements. These include single or double sided, single or double density and number of tracks. Single or double sided depends on the disk drive installed, whether it has one or two heads. Most diskettes will support double sided recording, however some are certified for single sided use only. Single or double density is determined by the disk drive controller, the II controller will not support double density but the CorComp will. If you wish to use double density it is better to use diskettes designed for that purpose as they have a better coating than the cheaper diskettes.

A single sided diskette is divided into 40 concentric circles called tracks which are numbered from 0 on the outside to 39 on the inside. Each track is divided into 9 sectors around the diskette. The sectors and tracks can not be seen with the human eye because they represent a magnetically defined area rather than a physically defined area on the diskette.



Having 9 sectors with 40 tracks gives the user 360 track-sectors per diskette side, however they are referred to simply as sectors. Each sector is capable of storing up to 256 bytes or 2048 bits of information. Each sector has additional space used only by the controller to verify identification and data accuracy. The sectors are numbered from 0 on the first track through to 359 on the last track for both single and double sided, with double sided having an extra 360 sectors on the second side which are numbered from 360 on the last track to 719 on the first track.



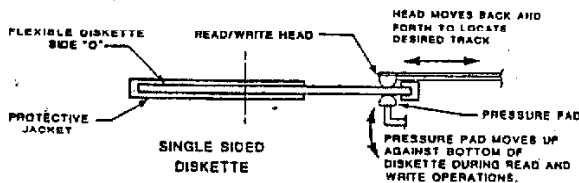
The disk controller uses sectors 0 and 1 of the first track to form an index of file locations. If these sectors become damaged or unreadable the computer cannot locate any files on the diskette. To locate the first sector on a diskette the small hole beside the large central hole is used. A light beam is shone through the hole and when the hole in the actual diskette lines up the beam is detected and registers the position, this is called 'soft sectoring'.

first
ording
ers a
te and
g and
conds

thin
th a
iron
oring
at a
h is
erial
ks on
on a

forms
tions
ering
arium
ch is
not
nals.
and
the
the
Mylar
ngful

Single sided drives have a read/write head on one side and a pressure pad on the other side. Unlike the head on a hard disc system the head makes actual contact with the diskette during read/write operation. Double sided drives have a read/write head on both sides. A stepper motor moves the head in and out across the surface of the diskette, accurately positioning the head above the tracks. A separate motor rotates the diskette at 300 R.P.M.

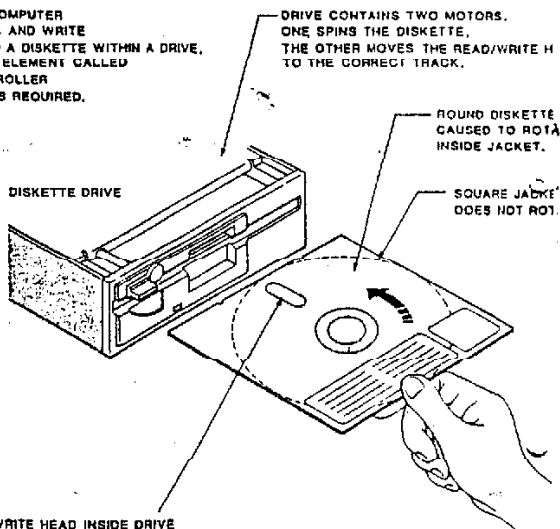


(ON TWO SIDED DISKETTES, BOTTOM PRESSURE PAD IS REPLACED WITH A SECOND READ/WRITE HEAD. NOTE: ONLY ONE HEAD READS OR WRITES AT A TIME)

To prevent accidental erasure of a program the small notch on the diskette envelope can be covered. When the notch is covered the write function of the head is disabled, allowing the head to only read information from the diskette to the computer. The cover must be removed if any information is to be written to the diskette.

If after initializing a diskette you come up with sectors used there is probably a build up a oxide on the head. To clean the head obtain a cleaning diskette and apply the fluid supplied to the surface, insert the diskette and call up a program. This will cause the diskette to spin and clean the head.

NOTE:
FOR A COMPUTER TO READ AND WRITE DATA TO A DISKETTE WITHIN A DRIVE, A THIRD ELEMENT CALLED A CONTROLLER BOARD IS REQUIRED.



READ/WRITE HEAD INSIDE DRIVE BOX MAKES CONTACT WITH ROTATING DISKETTE THROUGH THIS SLOT IN JACKET

ASSEMBLY FOR THE LAYMAN

BY ALLEN WRIGHT
H. V. BEERS

The references used for these articles are;

INTRODUCTION TO ASS. LANG. FOR THE T.I. 99/4A. By Ralph Molesworth.

The Series of Tutorials. By Mack Macormack

What ever I could lay my hands on and read.

These articles are aimed directly at the person who has a flutter in the stomach at the thought of Assembly language, let alone having done anything with it. To my mind there are two main reasons why one would want to learn Assembly:

1) To write fantastic programmes and market them.

2) To use learning the Assembly language as a means of learning more about computers in general and the T.I. in particular.

I think anybody who has 1 in mind is going to be bitterly disappointed, you will not get very rich on the T.I. community.

Something of a disclaimer!! Not being an expert assembly programmer I no doubt will make errors, either by honest mistake or due to the lack on a full understanding of the language as applied to the T.I. I will not make any apology for this but simply ask for the assistance of people who do know the correct answers to come forward and help, either by personal comment or by mail. BUT! please don't allow yourself the luxury of being an armchair critic, get in and participate!! I am doing these articles because I see a need for this type of article.

EASY??.

I have to admit that I do not find Assembly easy to write. A lot of the materials I have read try to give the impression that it isn't all that bad, but I find that these articles etc. having saying that, go off into a discussion of registers, addressing, branching etc. These are all unknown or difficult concepts for the raw beginner to grasp. My approach, I hope, will be somewhat different.

THE EDITOR.

An assembly language programme is written using the Editor in the Editor Assembler package. If you have Funnel Writer the Editor in that can be used. (NOTE!! NOT the word processor Editor.)

GETTING INTO EDITOR.

If you are going to use the E/A module the following applies:

PLACE THE MODULE IN THE CART SLOT.
PLACE E/A DISC PART A IN DRIVE 1.
SELECT E/A.

This Menu should be on screen:

- 1) TO EDIT
- 2) ASSEMBLE
- 3) LOAD AND RUN
- 4) RUN
- 5) RUN PROGRAMME FILE

Select option 1.

The Editor sub Menu now appears.

- 1) TO LOAD.
- 2) EDIT.
- 3) SAVE.
- 4) PRINT.
- 5) PURGE.

Select option 2. The Editor will load from the disk in drive 1.

When the Editor is loaded from the E/A part A disk the programme starts in the edit mode. The screen looks like this:

```
END OF FILE VERSION X
```

Press Func. 9 to get back to the mode select Menu. The Screen now has a list of options displayed at the top of the screen, this is the list:

EDIT, FIND, REPLACE, MOVE, INSERT, COPY, SHOW, DELETE, ADJUST, TAB, HOME.

As time progresses we will look at and use all of these features which are available. The Menu loaded with the Funnel Writer Version of the Editor looks a bit different but all the features are available by entering the first letter of the required options.

Now press E to get into the Edit mode. (For Funnel Writer users the Cursor will appear on line 0001 with the END OF FILE marker on the next line). For E/A version users press enter once, this moves the E.O.F. marker down one line.

E/A version users now press FUNC. S ONCE!, this puts line number 0001 on the left of the screen. Pressing FUNC S at the left margin will make the line numbers come and go.

Now type the following exactly as shown then PRESS ENTER;

```
123456S1234S1234567890123456
```

The screen looks like this;

```
0001 123456S1234S1234567890123456
0002
                                END OF FILE VERSION X
```

Now type the following onto line 0002 and PRESS ENTER to make the screen look like this;

```
0001 123456S1234S1234567890123456
0002 LABELS CODE OPERAND COMM
0003
                                END OF FILE VERSION X
```

The cursor should now be on line 3 under the L of LABELS.

Now operate the Tab function, FUNC 7. You will notice that the cursor steps to the beginning of each word. This feature is used to set out the Assembly language programme you are typing in.

The screen is now indicating the General Syntax for T.I. Assembly.

The LABEL which is optional, starts in the first position on the line & can be up to 6 characters long. The first character must be alphabetic.

The code, the correct name for which is OP-CODE, is separated from the LABEL by at least one space. The

these
THE
Mack
as on
y at
the
mbly
done
there
ould
and
mbly
more
the
d is
ted,
the
Not
mer
ther
lack
the
I
this
of
ect
elp,
by
llow
a
and
ese
for

OP-CODE is the instruction which tells the computer which operation to perform.

The OPERAND is separated from the OP-CODE by at least one space. The OPERAND can consist of one or two operands. If two operands are used they must be separated by a comma. The operand contains the information that the OP-CODE is to use & can be a Register or some other address.

The Comment field is again separated from the OPERAND by at least one space. When typing your programme it is a good idea to start your comment at the same character position on each line. This makes the programme easy to read and looks neat and tidy. A final point about comments is that you can't have enough of them.

TYPING IN A PROGRAMME.

Now press FUNC 9 to get back to the mode selection and purge the file. Select Edit again after purging. Now we can type in our first programme.

The following Programme fills the screen with the letter A. Well!! you might say that isn't so clever, and right you are, it isn't. The programme is here to teach you how to TYPE IN, SAVE and then ASSEMBLE a programme. You could no doubt write a programme in Basic or Ex Basic to do this job without the time, saving and assembling involved with Assembly language & be up and running in no time at all. However wait until you see how quickly the screen is filled with the character A, this makes it all worthwhile!.

One other point!! These articles are not going to be a blow by blow description of how each and every Op-Code used in the Source Statement works. I will put a list of references at the end of each article to direct you to the E/A Manual pages you must read. If you are not going to read them GO NO FURTHER than this point. The programmes are going to be used to illustrate ways in which to use the language & when required, a description of how a feature of an instruction etc. is being used will be included.

```
0001 * FIRST TUTORIAL PROGRAMME
0002 * 18-7-86 FILENAME SOURCE=EA/PROG/1
0003 * OBJECT=EA/P/108J
0004      DEF START PROGRAMME NAME
0005      REF VSWW REF TO VSWW
0006 RETURN DATA >0000      SAVE RETURN ADDRESS
0007 CHARA DATA >0041      CHARACTER CODE FOR A
0008 W^9E6 BSS >20          SET ASIDE BLOCK FOR W/SPACE
0009 START MOV R11,0RETURN.  SAVE RETURN ADDRESS
0010      LNPI WSREG          CREATE MY W/SPACE
0011      LI RO,>2FF          ADDRESS LAST SCN POSIT.
0012      MOV @CHARA,R1       CHAR CODE INTO R1
0013      SNPB R1             MOV DATA TO LEFT BYTE.
0014 LOOP BLWP @VSWW.        BRANCH TO SCREEN WRITE
0015      DEC RO              DECREASE RO BY ONE
0016      JNE LOOP           EQUAL TO ZERO? GOTO LOOP.
0017      MOV @RETURN,R11    RETURN ADDR INTO MY R11.
0018      RT
0019      END.
```

So to typing;
Get line 0001 on screen with the cursor in the first or left-most character position. The first three lines to be typed in are comment lines. A comment line starts with the character *. I always make practice of using the first few comment lines to enter some useful data. In this case I have included what the programme is, the date, and the filenames I have used for the Source Code and Object Code on disk. The Source Code is what you are about to type in, the Object Code is the Assembled output of the Assembler. For the purpose of this exercise type in the first three lines exactly as shown below. (You must do this so the programme you create can be used in the next Tutorial).

The cursor should be now on the start of line 0004. In the following discussion these abbreviations have been used.

[CR]= PRESS ENTER.
[T] = PRESS TAB (FUNC 7) ONCE

Now type the following;

On Line 0004

```
[T] DEF [T] START [T] PROGRAMME
START NAME [CR]
```

On Line 0005

```
[T] REF [T] VSWW [T] REF TO VSWW
[CR]
```

On Line 0006

RETURN [T] DATA [T] >0000 [T] SAVE
RETURN ADDRESS [CR]

On Line 0007

CHARA [T] DATA [T] >0041 [T]
CHARACTER CODE FOR A [CR]

Continue through to line 18.

On Line 0018

[T] RT [CR]

On Line 0019

[T] END [CR]

GETTING IT ASSEMBLED!

Now that you have finished typing in
the programme, DOUBLE CHECK FOR
ERRORS!!

All clear? The next step is to GAVE
the SOURCE CODE to disk;

PRESS FUNC.9 twice & you will return
to the EDIT MENU.

- 1 TO LOAD
- 2 EDIT
- 3 SAVE
- 4 PRINT
- 5 PURGE

If you have one disk drive, remove
the ASSEMBLER PART A disk and place
an initialized disk into the drive,
making sure that it is a disk which
has a bit of room on it. If you are
lucky enough to own more than one
drive then DON'T remove the
Assembler disk, place your DATA disk
into drive 2. I have 2 disks set
aside which I use for all my
Assembly hacking, one for SOURCE
CODE and one for OBJECT CODE. If
you have only one drive you do not
have this luxury. But do have at
least one disk for the purpose
mentioned. Sometimes when a
programme or routine goes missing I
can invariably find it or an earlier
version of it on my hacking disk.

With your "hackers" disk in the
drive select OPTION 3 SAVE.

Answer Y to then next prompt (asking
if you want to save in DIS/VAR 80 or
not).

Now enter the filename for the
Source Code;

DSK1.EA/PROG/1
or
DSK2.EA/PROG/1

PRESS ENTER and your SOURCE CODE
will be saved to disk.

When SAVING is complete PRESS FUNC 9
to return to the E/A MAIN MENU.

If you have ONE drive remove the
DATA disk which contains the SOURCE
CODE and place ASSEMBLER PART A back
into the drive.

Now Select OPTION 2 ASSEMBLE.

Answer YES to LOAD ASSEMBLER? The
Assembler programme is then loaded
from the disk.

ONE DRIVE OPERATION.

When the Assembler has finished
loading remove the PART A disk and
place the disk containing the SOURCE
CODE into the drive.

TWO DRIVE OPERATION.

If like me you decide to have
separate SOURCE CODE and OBJECT CODE
disks, remove the ASSEMBLER PART A
disk and place the OBJECT CODE disk
into drive 1.

The Screen should now have the
prompt showing;

SOURCE FILENAME
Enter DSK1. or DSK2.EA/PROG/1

OBJECT FILENAME
ENTER DSK1.EA/P/10BJ

The Third Prompt is;

LIST FILENAME

If you have a printer then enter the
Printer filename here. If your
printer is a parallel interface
enter;

PIO.

Include the period!!

The SOURCE LISTING can be LOADED to
disk instead of directly to the
printer if you want. If you don't
have a printer saving the SOURCE
LISTING to disk can make finding
ERRORS a bit easier. Of course if

you want a more permanent copy of the LISTING then disk is the best option to take. The LISTING can be dumped to PRINTER from disk through the EDITOR MENU at some later time.

The fourth and last prompt is OPTIONS;

The five available are;

R
Indicates to the Assembler that you have used R to prefix your REGISTER numbers. (As we have done in our small programme.)

L
Indicates that you required a Source Code Listing. (This listing is a little different than the Code you typed in. We will talk about this listing next month and how to use it.)

S
Prints the Symbols and Registers used in your Source Code. It is printed after the Source Code Listing.

C
Will cause the Assembler to produce Compressed Object Code.

T
Prints full text string in the Source Listing.

For this exercise use RL only. If you are not going to have a listing only type R.

So!! Under the OPTIONS heading type;

RL [CR]

The Assembler will start Executing immediately. The Source Code will be printed on your printer as the Assembler executes. The Object Code produced is placed onto disk under the Filename you used.

ASSEMBLER EXECUTING will appear on screen.

When the Assembler is completed; ASSEMBLER EXECUTING- will be followed by

0000 ERRORS
PRESS ENTER

HOPEFULLY (!!) your screen will be

indicating 0000 ERRORS. If so PRESS ENTER which returns to the E/A Menu screen. If there was an ERROR and you did not request a SOURCE CODE LISTING then make a note of the ERROR on a piece of paper. Then skip this section for the time being until you correct the ERROR. Go to the heading ERRORS.

RUNNING THE PROGRAMME

The Programme can now be RUN;

Select OPTION 3 LOAD And RUN

At the FILENAME prompt enter your OBJECT CODE filename;

DSK1. or DSK2.EA/P/10BJ

The programme will load and when completed PRESS ENTER once. The PROGRAMME NAME prompt appears, enter the programme name START and PRESS ENTER.

The programme will RUN and momentarily fill the screen with the character A then return control of the computer back to the E/A module

To RUN the programme again return to the E/A MAIN MENU and select OPTION 4 RUN

Enter the programme name START then PRESS ENTER to RUN it again.

ERRORS.

If after the ASSEMBLER had completed it indicated that there was an ERROR or even worse ERRORS there has been some problem in the SOURCE CODE you have entered.

If you obtained a listing of the SOURCE CODE on your printer then the line on which the ERROR occurred will be clearly indicated.

The SOURCE CODE will have to be RELOADED into the EDITOR for correction. Place E/A PART A disk back into drive 1.

Select OPTION 1 TO EDIT from the E/A MAIN MENU.

Then from the EDITOR MENU select. OPTION 1 TO LOAD

When loading is completed, for on

drive operation, replace E/A DISK PART A with the disk which contains your SOURCE CODE. Enter the drive and FILENAME of the SOURCE CODE;

DSK1. or DSK2.EA/PROG/1 [CR]

When loading is completed select;
OPTION 2 EDIT

Locate and correct the ERROR indicated on the SOURCE LISTING. When that has been done repeat the SAVING and ASSEMBLING described previously.

THE MISSING A!!!

If you were quick enough you will have noticed that the screen is not completely full of that character A when the programme RUNS. To have a look at this insert the following line into your SOURCE CODE after line 17. ie before RT.

DEC R11

Then save, and ASSEMBLE the programme again. This time when it is RUN the programme does not return control to the E/A module but "locks up" the computer. This "freezes" the screen. Now the missing A can clearly be seen. After reading the References in the E/A MANUAL at the end of the article try to determine why the first A is missing. By the way, the reason is not a clever trick on my part. It is one of the small programming problems which do crop up in any language.

GETTING OUT!!!

To regain control of the computer you will have to use QUIT. This returns the computer to the colour bar screen - this isn't gracious but it works! While we are in this neck of the woods it is timely to mention that the programme you write in ASSEMBLY has total control of the computer when it is running. There are no nice error messages as in Basic or EX Basic. A programme error invariably means a "lock up" with no way out except to turn off and start again. The error messages you require must be written into the programme. I have mentioned this here to illustrate the control you have over the computer. It is a bit like purchasing a FERRARI in pieces, you know what it can do all you have to do is get it together correctly!

FINISH.

That will have to do for this month. If you feel inclined to go further before next month I would strongly recommend you get hold of MOLESWORTH and start working through it.

Next month will be about the SOURCE CODE LISTING and what it tells us, a quick description of what is happening when this months programme runs and SUPER BUG 11, and if it will fit in, a start on a CURSOR routine.

Finally I would appreciate any comments about the article, good or bad (I am a big boy now), and any specific request for articles, remembering of course that WILL and TONY look after the complex end of things.

Until next month SEE YA MATE!

REFERENCE LIST.
E/A MANUAL.

PAGE
28/36

86
119
163
166
171
212
225
227
228
248

TOPIC
USING E/A CARTRIDGE
DECREMENT
JUMP IF NOT EQUAL
LOAD IMMEDIATE
MOV WORD
SWAP BYTES
BLOCK STARTING SYMBOL
DATA
DEF
REF
VSBW



SPECIAL CHARACTER CODES FOR T. I. WRITER

BY JENNIE WATKINS
H. V. 99ERS

After trying to use the transliteration program "CODE 2", which was described in the special magazine issue on T.I. Writer, without much success, I decided to experiment with some of the commands that can be input to send special signals to the printer. "CODE 2" was designed for use with an AMUST 100 printer and does not work satisfactorily with a B.M.C. or an AMUST 80 printer. Hence a lengthy investigation of my printer manual and some help from club members has produced the following aids when fancy typing is required.

I began by using The Special Character Mode which can be operated both by the Editor and the Formatter. The table below sets out the various controls for the different printing styles.

TYPING COMMANDS FOR THE B.M.C. PRINTER.

(AMUST 80 and EPSON)

NOTE: CTRL = (CTRL-U) (SHIFT-?) (CTRL-U)
ESC = (CTRL-U) (FCTN-R) (CTRL-U) (?)

FUNCTION	KEY STROKES		EXAMPLES
	ON	OFF	
ENLARGED	CTRL N	CTRL T	ENLARGED
CONDENSED	CTRL O	CTRL R	CONDENSED
EMPHASIZED	ESC E	ESC F	EMPHASIZED
DOUBLE STRIKE	ESC G	ESC H	DOUBLE STRIKE
ITALIC	ESC 4	ESC 5	ITALIC
DOUBLE WIDTH	ESC W CTRL A	ESC W CTRL 2	DOUBLE WIDTH
SUPERSCRIPIT	ESC S CTRL 2	ESC T ESC H	12345 23001
SUBSCRIPT	ESC S CTRL A	ESC T ESC H	12345 x10
UNDERLINE	ESC - CTRL A	ESC - CTRL 2	<u>CONTINUOUS UNDERLINE</u>
LINE SPACE 1/8	ESC 0	ESC 2	THE DOG THE CAT
LINE SPACE 7/72	ESC 1	ESC 2	the dog the cat
CARRIAGE RETURN	CTRL M		

ich
uch
be
ned
ith
my
the

ted
the

DT

This mode of operation is ideal if only an occasional print style is required but can become rather lengthy if many controls have to be executed. For example, superscript and subscript typing styles are automatically double struck. Hence, this also has to be turned off after using these typing styles -- a total of 8 keys to press to perform such an operation!! Rather cumbersome.

So I then turned to transliteration for the B.M.C. Printer. This can only be used with the Formatter. I have created a separate file for these codes which I have called "TL/CODES".

TL/CODES

- .TL 17:27,69
- .TL 23:27,70
- .TL 5:27,52
- .TL 18:27,53
- .TL 20:27,45,1
- .TL 25:27,45,0
- .TL 1:15
- .TL 19:18
- .TL 4:27,87,1
- .TL 6:27,87,0
- .TL 7:27,71
- .TL 8:27,72
- .TL 26:27,83,0
- .TL 24:27,84,27,72
- .TL 3:27,83,1
- .TL 22:27,14
- .TL 2:27,20

Now, whenever I wish to use different printing styles, I type .IF DSK1.TL/CODES at the beginning of the text. This really simplifies the number of key strokes required to send commands to the printer. This can be seen by looking at the following table.

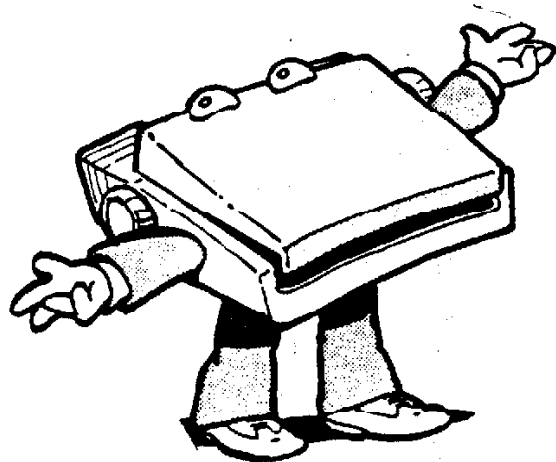
SPECIAL CHARACTER CODES
USING "TL/CODES"
(CTRL U, LETTER, CTRL U)

PRINT TYPES	ON	OFF
ENLARGED	V	B
CONDENSED	A	S
EMPHASIZED	Q	W
DOUBLE STRIKE	G	H
ITALIC	E	R
DOUBLE WIDTH	D	F
SUPERSCRIPT	Z	X
SUBSCRIPT	C	X
UNDERLINE	T	Y

In both modes of operation, the necessary controls are inserted just before, and just after, the word, phrase or paragraph which is to be printed in that style.

I am still experimenting with combinations of the controls and other commands which can be sent to the B.M.C. Printer in order to perform more interesting functions. So I may follow this article with another in the future.

Meanwhile --- Good Printing!



ADVENTURERS INFO CORNER

WITH RODNEY GRINSFORD
H. V. SSERS

Hi this month we will have something new in the corner. We now have a full and complete map to ADVENTURELAND. You will notice the crosses on the map. These are there to show you where to find the 13 treasures needed to complete the adventure. Please send any maps or hints you have to the address below. If you need any maps or hints send a stamped self addressed envelope to:
RODNEY G.
56 SEDGEWICK AVENUE
EDGEWORTH 2285

ADVENTURELAND

- * In the swamp climb the tree
- * The rug is another way out of the maze
- * Oh no! A bear. I think I am going to SCREAM"
- * Hot lava - maybe I will BUILD a DAM
- * Mud is a great cure for chigger bite

PIRATE ADVENTURE

- * Use a hammer to remove nails
- * One hole is good but two is better
- * The pirate can sail the boat

H.H.G.T.T.G.

- * The bug latter beast of Traal thinks that towels on heads are unseeable
- * Forget about your intelligence till much later
- * Feed the dog the cheese sandwich you bought in the pub
- * Put the junk mail on top of the satchel
- * The sauna is good after ten or twelve times

MISSION IMPOSSIBLE

- * Sit in the chair
- * RED-ARM BOMB
- * WHITE-DISARM BOMB AND TAKE PICTURE
- * YELLOW-MAINTENANCE * BLUE-SECURITY
- * Break the glass
- * Unlock the button with a key of the corresponding colour
- * Shake the mop
- * Wet the bomb
- * Insert the cartridge then watch the movie

DEADLINE

- * Rub pencil on notepad
- * Read second section of newspaper
- * The envelope is useless*
- * Ask McNabb about the roses *
- Attend the reading of the will
- * Ask Dunbar about loblio bottle

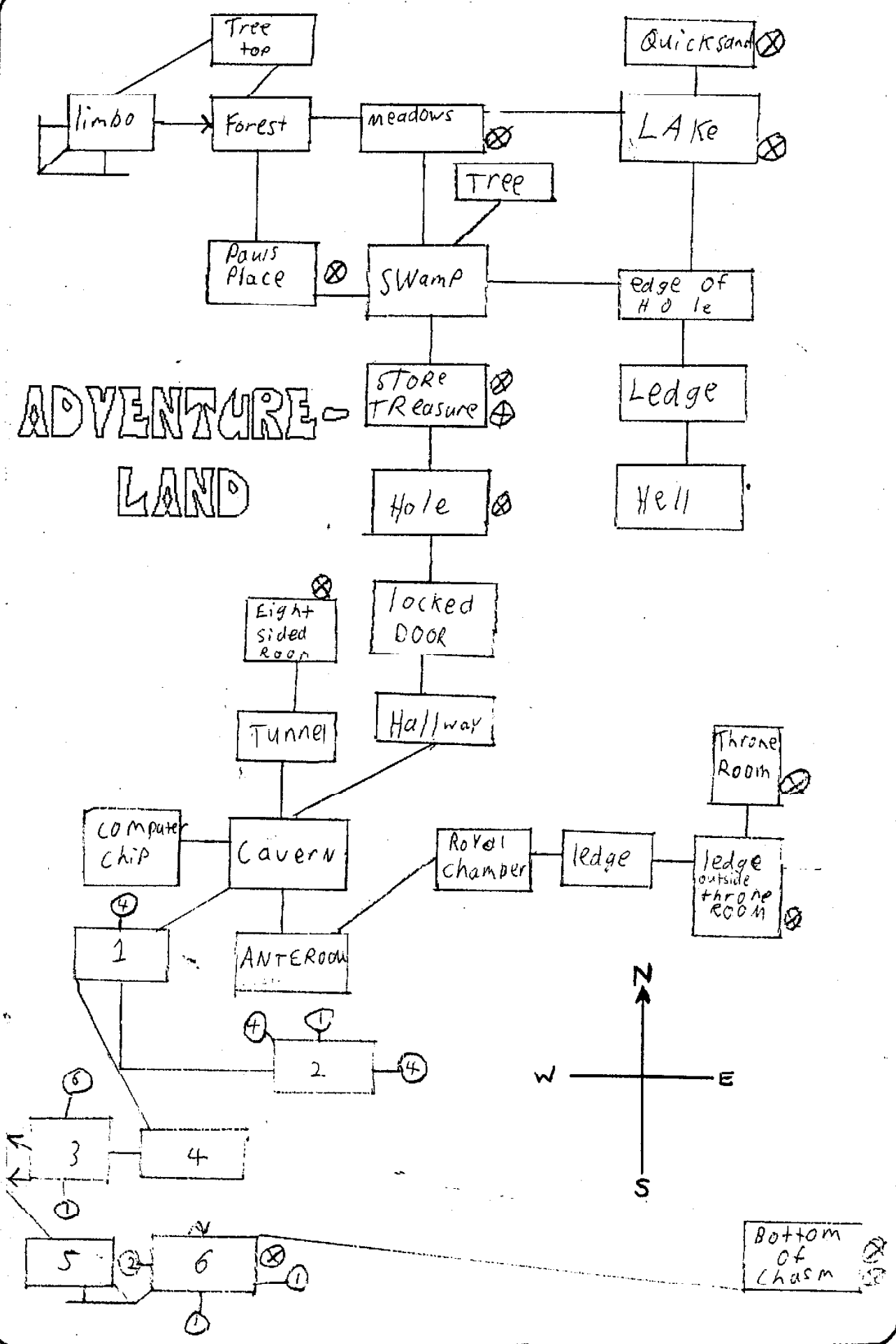
ZORK I

- * A trophy case is for trophies
- * SAVE THE GAME when in the CYCLOPS ROOM
- * Put your torch in the bucket in the SHAFT ROOM to light the MACHINE ROOM
- * DON'T take a flame into the GAS ROOM
- * If you are sent to Hades as a ghost make your way to the altar for a reincarnation
- * The scepter has a direct relationship with the rainbow. Try waving it
- * Even a deranged vampire bat hates garlic*
- * A sound, a light and a prayer all found in the temple area can dispel the spirits
- * You need a boat to travel the frigid river*
- * To cross the reservoir, you must drain the dam. To open the dam, press one of the buttons and turn the bolt.
- * The CYCLOPS is hungry. To avoid becoming a snack say ODYSSEUS

That's about all for this month but next month there will be a MAP to PIRATE ISLAND and hopefully some more hints for
ZORK.....

Rodney G.

ADVENTURE- LAND



ENTOMOLOGY CORNER

7

BY TONY MCGOVERN
H. V. BEERS

Sometimes I get people asking why I almost always write letters out long-hand. The answer is that after I write articles like this and do all the programming my fingers have had enough. And unlike real typists who can spread the load around 10 fingers, I can usually only manage to share it amongst two of them. On a good day I get to hit the space bar with my thumb too. Also we have only one machine which is under heavy pressure all the time. Where did the mini-PE-box project disappear to? So what's for this issue? I thought that after the general gossip I might devote it to a run through of what needs to be done to customize Funlwriter Vn 3.3 for your particular requirements.

First there should be a little correction to the discussion of GPLLNKs that I went into with regard to the Myarc disk controller card in the last column. There is in fact another suitable XML instruction in console GROM #0 which occurs earlier than the one I mentioned. This is at the end of the routine that builds the selection list that appears after the title screen and awaits your choice. This is the one that is found first by GROM searches (it's trickier to find with DEBUG and similar programs because these search for a whole word say >OFF0 only on even address boundaries. That's OK in CPU RAM but less than fully useful in byte oriented VDP or GROMs). It doesn't actually make any difference to the discussion of GPLLNK because it doesn't really matter where the XML instruction is, only that you know where it is and where it points to. Incidentally, Funlwriter finds this address when it first puts up the

central menu screen, stores it, and then uses on all later occasions. So don't pull the E/A or TI-Writer module out of its socket and expect GPLLNKs to work. This way GPLLNK doesn't have to do a GROM search every time it is called. Even when loaded from Myarc or Corcomp it searches GROM #3 first in case an E/A module is present. And talking of Corcomp I have found that the card's loader in the Disk Manager program is for object files, not program files as I had vaguely and incorrectly remembered. No worry, I'm told LDFW works perfectly well here too. What may be a problem with Corcomp is that I have a report of a Corcomp card which does not support Funlwriter's automatic boot disk tracking. I don't know how many variations there have been in their disk DSR.

We have recently received from Edgar Dohmann of JSC/UG the heavily revised Vn 2.0 of SUPERBUG II. This is a fairware disk available from the HV99 club library. A booklet of documentation is available from Edgar on receipt of the suggested fairware contribution of US\$10. Maybe that's the way I should have handled the documents for F'Wr. Talking of fairware returns not a single contribution (and just one telephone call) has been received from the whole of the enormous Sydney User's group. Earlier versions of the program have been in Sydney for some time now, and the HV99 contingent at the Melbourne TI-Fair in June presented all the groups represented there with a complimentary distribution copy of Vn 3.3. I hope they don't treat overseas fairware authors similarly, as I wouldn't like our TI-friends elsewhere to get that kind of impression of Australia as a whole. It would seem to be much the same elsewhere in that it is the small to medium sized groups that are working effectively to prolong the viability of our orphan, both in production and encouragement of good software.

It is probably a good idea to be cautious about the physical quality of disks received from overseas. They may be cheap in the USA but there is some reason to believe Gresham's Law is at work. I have just received a second disk

which is binding in its jacket so that it can't be read, and does heavens knows what damage to the disk drive. In newer models with direct motor drive of the disk spindle there probably wouldn't be too much problem, unless in the motor driver electronics, but I would be more concerned about the drive belt on the TI/Shugart original drives. The first one was from the Boston area if I remember correctly. In a soft-hearted moment I sent a good disk back but I've not heard a thing back in reply. As a result of this and other such experiences I'm getting a lot harder nosed about sending out disks. The latest dud was from Chicago, home of another large group that from the empirical evidence operates like Sydney when it comes to fairware support. The letter file was all I got off, with difficulty, before I later realized the problems were due to its seizing up. The sources of letters from the US seem to follow no discernably predictable pattern. California is a state which I know quite well, and I never would have guessed ahead of time that F'Wr would gather in numerous TI-computing friends in San Diego, a small number in the Bay area, and not a single response from the whole of the greater Los Angeles area, which is the part I know best of all. Maybe it's the group size effect.

The Myarc disk card still functions as described last time. It gets to stay in the machine now. Why take it out I hear you ask? Well, William has been doing all sorts of programming involving track reading, and he had only worked it out for the TI original. This was leading to continuing conflict because the F'Wr source code is on DS/DD disks now. Maybe my agonized complaints finally got through because last Sunday afternoon he sat down at the computer and by evening had it reading 3D tracks, and by Monday he was reading DD tracks with complete success, even from an MS/DOS sample disk that was lying around here. All he needs to do is improve his Maths and English grades as fast.

I cannot recommend Myarc's customer service however because it appears to be non-existent. I

wrote to Myarc shortly after we received the card via our TI-99-computing friends in the US, asking about bugs I had uncovered and for other information. Several months later (mid-July) there has still been no communication of any sort from Myarc about this or anything else. Maybe the only way to stir them up is to send them a copy of Will's document file summarizing how to talk to a Myarc controller, just before it gets published in the HV99 news. It would have been interesting to see Myarc's new computer at the Melbourne TI-Fair, though I gather PAL/NTSC and/or connector problems foiled any real demo. I can't see us ever buying one of these, nice machine as it may be on paper. There are three reasons, the first being the lack of service already apparent on their PE-box cards if anything were to go wrong with it, a problem which would be immensely compounded by distance from Australia. The second is the feeling that the money would be better put towards a machine with a wider future when the time comes to upgrade. The third is that the card computer would tie up our single PE box and I do want to keep the TI-99 system as a second computer when we move on. The full system with DSDD is just too good a machine to dismember, but console only operation is worthless to us. It will in many ways be a wrench to move on as we have a lot of time and effort invested in our mastery of the TI-99/4a and the elegant TMS9900 assembly language. The way the A\$ is going these days the 99/4a may remain our sole computer for some while.

Since the first release of Funlwriter Vn 3.3 at the TI-Fair in Melbourne there has been a steady process of fixing various minor bugs that have been noticed and several enhancements. The last set was called to my attention by our TI-computing friends, Woody Wilson and David Allen in San Diego and concerned the use of LIST option with the Assembler. In the process I averted introduction of a potentially disastrous bug, fixed another one that affected only the Myarc card, and smoothed off a couple of other rough edges. The

only one I haven't been able to reproduce is the one they mentioned. Maybe it was just a printer problem. With a program as complex, tightly coded, and wide ranging as Funlwriter it is very difficult to catch all the bugs in the first place, and even more difficult to make sure that new ones are not inadvertently introduced. Feedback is always welcome.

While at it I should mention that Rick Cosmano of the SCCG in San Diego is circulating a proposal for an alteration in the standard usage in the TI-Writer Formatter. The details will be elsewhere in this newsletter.

Now on to the promised substance of this article. It will refer to the state of the program as it is now in late-July / 86 and serve as the text for the promised demo session at the August HV99 group meeting. The program updates will be available from the club librarian. Most of the enhancements have been in the form of extra loading possibilities, but there has been a change in LOAD as well, intended to facilitate use with RAMdisks. I don't yet know how useful this really is and I can't demonstrate it because we don't have a RAMdisk of any description. Also the LOAD program has been RESequenced to keep tidy minded people happy. So it will be worth updating your copy just for ease of finding your way around LOAD.

The first set of choices I will look at is that of screen colors. The set provided reflects our personal preferences here with the display we use, a console color TV with excellent convergence, up on the desk behind the PE Box. Apart from sheer color preference the strongest reason for change is if you are using a monochrome monitor, where improved display is obtained when no color is present. The screen colors are cycled through a set of 8 choices by entering the highest acceptable key value on any Funlwriter selection screen which expects choice by number. This is usually one higher than the highest number shown on the screen. The last 3 color

selections are built into the program and can only be changed by sector edit of the disk file, but if anyone wants to change them I am quite happy to tell them where to look. The first 5 are written into the Editor when it is loaded, overwriting any already in the Editor. The Editor will always start out in the current Funlwriter screen color, even if it is one of the last 3. In the latest revision the Editor's internal color list pointer is set to match the F'Wr screen color except for the last 3 colors when it remains on the first color of the 5 and in this case if SD is invoked it will revert to this. The colors outside the short list cannot be retrieved easily because text mode colors reside in a read only register in VDP.

Your preferred colors in order of choice may be installed in your own working copy by altering a CALL COLOR statement in the LOAD program. Place the Funlwriter disk in drive #1, and select Extended Basic, which will then autoload the LOAD program. Hold down CLEAR (fctn-4) until the program BREAKs execution in its first line. It is now ready for LISTing or Editing in the usual XB fashion. If you are careless and let go so the Funlwriter title screen comes up, then start again. First though a word of caution.

EDIT and reSAVE only a freshly loaded copy of LOAD. Do not attempt to RESequence and do not use, edit or save the program if it is STOPped after a re-entry from the User's List. Leave your master copy untouched.

The syntax of XB's CALL COLOR subprogram is treated on p66-7 of the XB manual. If you are rusty or uncertain of how it works try it in a XB program first. The color choice is in line #120 of LOAD and starts with color group 10. In each triplet the first number is the color group, the second is the color of the lettering and the third is the background screen color. To experiment with colors make your entry and then RUN the program, and cycle through the

choices by pressing 4 on the first selection screen. Reload and enter different choices until satisfactory. When you are satisfied enter the final list and SAVE off LOAD to disk.

The next two lines allow you to enter your choice of device name for the printer option. EDPR\$ in line #130 is the printer name for either Editor and FRPR\$ in line #140 is the printer name for the Formatter. Normally the Formatter name will be the same as for the Editor except for the addition of a you would have typed in as response to the program PF prompt in the days before F'Wr. The program allows for names up to 23 characters long.

Lines #160 - #230 are the descriptions that F'Wr writes into the User's List screen that comes up from that choice on the first selection screen. Each line has the form

```
160 OP$(0)="1 .. "
```

You assign the 9 elements of the array OP\$(), defined with OPTION BASE 0. The last one, OP\$(8), in line #240 should be left as it is. For best appearance on the screen enter each descriptive name starting over the dots. Any more than 10 characters will cause the program to halt with an error.

Line #250 hands over to the imbedded assembly code, passing in the string values already defined and a couple of other parameters as well. The first of these K has useful values 0-3. This number is written as digit "x" into the default disk/file name DSKx, which pops up at various times if no other file name can be found.

The second more anonymous parameter immediately following this is a string of at most 1 character length. This parameter controls the auto boot disk tracking feature of F'Wr. If this is left as a null string the program attempts to find which disk it was loaded from, and this drive number is written into all the system utility filenames, and into any User List program loads from disk unless flagged otherwise (see

later in this article). This works from the TI disk controller where it checks the filename left at the top of VDP memory, and if it can't find anything there proceeds to check in the Myarc card's onboard RAM before giving up the attempt. It is not clear whether this works for all Corcomp disk controllers. If the string is given a single character value the boot disk search is bypassed and this character written into the disk name. Any character may be used so you may have DSK6, or DSKR, as the disk drive name if desired. This means that only the original loader programs need be kept on the disk in drive #1 and the system utilities and User List programs may be kept permanently in drive #3 or in a RAMdisk. I expect this would be especially useful if this is battery backed. As an example our system has two Chinon DS/DD drives in the PE Box and the TI original external drive as DSK3. This drive is only SSDD so disks are not fully interchangeable with the PE box drives, but a SSDD disk is plenty big enough to hold all the utility files. Something you may notice is that if you copy LOAD onto a disk in drive #2 using SAVE DSK2.LOAD after the program was first loaded from drive #1, then when it is RUN, unless it has been set for a fixed drive, it will try to find its system files on disk #2 because this is where it thinks the load came from. This causes no problems in normal usage.

Lines #250-#270 contain the actual details for the User's List loading process. The lines have tail REMarks to remind you which selection each refers to. From here it is possible to RUN another XB program if desired. Assembly program choices are passed back into the program via line #350 with a parameter K that is set for each individual choice. There are some programs which must be loaded this way because it just isn't reasonable to provide for all possibilities in the load environment screen. The best known of these is TI-Forth and its load parameter illustrates all the possibilities.

The primary parameter needed is the one that would be entered

from the load environment screen. Remember to read FWDOC/EASM for all the details. That is general advice too - if all else fails read the manual. The Forth boot program is an object file which normally is run from E/A LOAD and RUN, and needs load parameter 4. The Forth disk is by nature a separate disk so we add 8 to the parameter otherwise necessary. This signals Funlwriter to pause with a reminder to insert the correct disk before loading the utility. For reasons known only to TI, since internally it does allow for XB, the Forth boot program also assumes some very particular details of the E/A loader and adding a further 16 flags this for Funlwriter. I don't quite know why TI did it this way but they did so we are stuck with it. The alternative is to do like Myarc and supply a specially adapted Forth boot program. As a last little problem Forth disks are also very fussy about which drive they are in so adding a further 32 tells Funlwriter not to assume the program is being loaded from the boot drive. The total entered as the parameter is 60. Each of these additions sets a bit in the byte that the program extracts from this number. See the XB manual discussion of the AND, OR, XOR operations in XB for further illumination. Just in passing it should be noted that TI-Forth will not load successfully from other than XB or E/A. More modifications than are provided for in Funlwriter are then needed. Loads from the disk cards should use the programs provided with these, which at least in the case of Myarc, involve modification of the Forth disk. We have not provided for a Minimem load of Forth either.

This is also a good place to enter other utilities which by design or necessity return to the title screen on exit. One that we use this way is the Myarc DM program, of course useful only with a Myarc disk controller. Assembly programs loaded this way return as if they had been loaded by name from within F'Wr, and the XB environment is destroyed as soon as the F'Wr assembly code is re-entered via the UTILA link. Loading errors also return to the main body of the Funlwriter program.

That about exhausts the user inputs to the XB LOAD program itself. Apart from the User's List these may be frozen in the UTIL1 (or RELOAD) program for reloading after using DM1000, or for auto-loading from E/A or TI-Writer, or the other loading possibilities. The process for doing this is described in FWDOC/REFC and I won't repeat the details here. UTIL1 also performs a search for the boot disk drive number unless the load method specifically asks for a drive number. It is possible to doctor a copy of UTIL1 so that it bypasses the boot drive search in the same way that LOAD does if instructed to. Let me know if you need to do this and I'll give you the details. It takes sector editing of only one byte in the UTIL1 disk file. This property is NOT carried over from LOAD to UTIL1 by UPATCH.

The F'Wr program also now has a second User List function which may be reached from the Central Menu Screen. When you keep pressing 5 for Switch the User List entry comes up. This function is provided by a separate file UL as there was just not enough room left in the main program to accommodate it. Once loaded it survives through most system utilities but may need to be reloaded after other programs have been loaded. The reloading is handled automatically. This user list is entirely separate from the previous one and there is no reason why the entries can't be duplicated so they are always available. The only exception to this statement is that XB RUN is not available, as the XB environment has by then been destroyed or was never present in the first place.

At present entries are most easily made in this list by using a disk sector editor on the UL file. This is quicker and easier than re-assembly and resaving of the original source code in any event. Use Disk Edit or your preferred program of this type to locate file UL. The body of the file is only 2 sectors long. Entry is little more complicated than that in the XB program. The first block of names for the screen list starts towards the end of the first sector following the words SCREEN ENTRY

and the arrow. Each entry starts with a digit 1-8, followed by two spaces, then a screen message of up to 10 characters. The next byte after this 10 character block is the load parameter that you would have entered for an XB UTILA Link entry (remember decimal to hex conversion). The next word, already in place, points to the file-name entry in the following block. This block has the file names preceded by a length byte which you have to enter along with the file name. There is no entry corresponding to "9 BACK".

Once again I should remind you that the TI-Forth loader option only works with either XB or E/A present, which means that its principal use will be as a XB loader for Forth. Loading from other modules or the Myarc disk card will need more extensive modifications to the Forth boot program, while F'Wr was always intended just to set up conditions that would allow the original Forth disk to work. Internally TI allowed only for E/A and XB operation. The modified Forth disk seems to work perfectly well with F'Wr. We may look into a more universal loader for Forth but I suspect this will involve writing yet another short special purpose file, and I feel that F'Wr already has enough of these. So please let me know if there is any demand for this, and as usual if there is I'll think about it. Also when using the switch option the choice MODEM comes up. The F'Wr package does NOT include a modem program, and MD is there merely as a dummy file to suggest how one can be called up easily and explicitly from F'Wr. There are plenty of modem programs around to use already, and little incentive in Newcastle to use one and even less to write yet another. There is now enough information in the FWDOC/REFC file to allow a modem program author to build in a return to F'Wr if desired.

Incidentally one of the more heart stopping exercises that can be done with Vn 3.3 is to press AID while the Formatter is in the middle of printing out a file. The Assembler on the other hand doesn't take any notice of it once it has started executing. The Quick Directory function is always

available from AID immediately after the program is loaded, unless file QD was absent from the boot disk, and is the easiest way of getting a disk directory. You don't have to load the Editor or DM1000 just for that purpose any more.

My next personal project here is to get stuck into c99 and Pascal. I have been trying to get the p-system going with all of the utilities on a single DSDD disk. So far this has not been successful, as it always seems to hang up on one thing or another. I suppose there is more than enough room on a DSDD disk to carry the necessary system files on each program disk as specified by TI for two drive operation, but I'm still curious to find out what is really going wrong. There must be a lot of utilities out there for making p-code easier to live with - eg to allow use of TI-Writer, so I would be very happy to get copies if anyone out there with experience of p-code is prepared to share them. I keep hearing rumours of such things but I haven't yet seen any in captivity. I'm sure that after the experience of doing F'Wr, re-inventing those particular wheels would be no great big deal, but it does seem a waste of precious time re-inventing any. So thanks in anticipation.

Tony McGovern
Funnelweb Farm

LATE NEWS!!

The cost of manufacturing Printed Circuit Boards (bare board only) for the 'Kleinschafer PE Box', as described in a previous newsletter, is being investigated at the moment. The likely cost should be \$20-\$30, depending on the number we order. If you are interested in purchasing one (or more) when they become available, either see me or drop me a line. Please understand that it will be a tentative order only. DO NOT SEND MONEY. Watch the newsletter for availability date and the final cost.

Albert Anderson,
Secretary

THE INFORMATION PAGE

COMING EVENTS

Next Committee meeting: Tuesday, 2nd September at 6.45pm
Next General meeting: Tuesday, 9th September at 7.00pm

AGENDA for SEPTEMBER GENERAL MEETING

Monthly disk/tape demonstration with Paul Mulvaney

GARY JONES' EXTENDED BASIC GROUP

Concluding Sprites

AL WRIGHT'S BASIC GROUP

Matrix and Arrays

All meetings are held at the Warners Bay High School

Annual subscriptions to the Group cover the period 1st July to 30th June following year.

People interested in joining our Group are asked to contact:

The Secretary,
Hunter Valley 99'ers Users Group,
6 Arcot Close,
TARRO. N.S.W. 2322
AUSTRALIA
phone (049)662602

The annual subscription is as follows:
Residents of Australia...\$20
Overseas residents.....\$40 (Airmail)
 \$30 (Surface Mail)

Back issues of our Newsletter are available for \$1.00 plus postage