Sorcerer Users' Group (Toronto) Newsletter
200 Balsam Ave., Toronto, Ont., M4E 3C3
Volume 1 Number 6
August 1980

EDITOR'S TURN

The way that I presently feel towards a computer is vastly different from the way I have ever felt before or even thought that I would feel. Having done some programming in Fortran at the University of Toronto and worked for a while as an operator/programmer trainee for IBM, I had come to feel the same way about computers as I feel about a dry cleaner. You send the stuff out, a few days later it comes back and either you're pleased with the results or you're not. I never touched it, it never touched me.

When I got my Sorcerer, it was much the same in many ways. There it sat on the table in front of me. I typed in a few programmes from a magazine, ran them, fixed up the bugs and felt pleased that the programme was (seemingly) running correctly. I didn't touch it (emotionally speaking) and it didn't touch me. Here's this digital electronic machine sitting in front of me, touted by the media as being capeable of acting as an extension of my mind, but I was completely cut off from it both intellectually and emotionally. There was no relationship, no rapport, no communication. AHA! No communication.

In effort to resolve this vacuum, I started working extensively with the thing, trying to get to know it better. A rash of BASIC programmes started to break out. Wonderful. Now my computer seems to be talking back to me, giving me information that I'm interested in, even the smart ass remarks that I taught it. Something was still missing, however. It was now soewhat like a tape recording, just a carbon copy of my thought processes, no personality. I want to know where its eyes are, its ears, its mouth, its brain.

Months later, after having disected the monitor ROM like a Forensic scientist, I finally started to feel more like I was communicating with an intelligence. Using Z-80 assembly language, monitor subroutines, port I/O and the video DMA, I could really get into things. Put this in this part of the brain, store that away there, diddle this thing across the screen, filter input, synthesize output, live, breathe, think!!! (Dramatic eh!)

Now I have to deal with problem number two, tedium. In getting all this to happen, I've become totally exhausted. LD this, EXX that, PUSH off, POP out of view. There's got to be an easier way. Interpreters are too slow and restrictive, assemblers, while faster, more efficient and conservative of memory are too tedious to programme. Isn't there something that strikes a happy medium? YES! COMPILERS!

Wait a minute now. I've worked with a compiler before. Fortran. Hate it. Too much is handled internally. This thing was written by engineers. It only

does what engineers want it to do. It doesn't understand "personal" computing. It only wants to do Fourier transforms, Simpson's method of derivation, dx/dy, split that graph, take smaller sample areas, extrapolate that curve, mean deviation, linear regression, least squares fit....

COBOL! Come. Join our team and we'll show you how you can become a chartered accountant through the wonderful world of computer programming. Well, next time I have a ledger balance sheet I can't balance, I'll give it a try, until then....

APL! ?* '?))((': -*/)).½@½-. HUH???

PASCAL. Well everybody's talking about it these days. Let's get out those old magazines that had a course on Pascal. This looks good. Straight-forward, structured programming. Input this, print that, add these, BEGIN here, END there, state this PROCEDURE. This is starting to look more like a fancy interpretive language every time I read it.

Get it together now. I only have 32K of RAM and CP/M. There must be something up and running for CP/M users. Its the software bus didn't you know. Yes, there must be something that runs on CP/M, that's affordable, that runs in 32K, that doesn't require you to sign a license giving the implementer all your rights and every second male child for using the damn thing (definitely an upcoming editorial). There's one. Let's give this thing a try. HMMM...."C".

(to be continued)

SOFTWARE

As I mentioned last month, you can greatly increase the power and flexibility of BASIC programmes by interfacing them with machine language subroutines using the USR function. A slight problem exists, however, on where to place these routines in memory so that they don't get trampled on by BASIC during execution. Remember, by default, BASIC will use all the memory it desires of whats available as a big scratchpad. It's really a pig. Steve Dicker's article in last month's issue is one very good way to handle this problem. Another exists which is a little less cumbersome in some ways and more in others. It's just a matter of which is more comfortable for you to work with.

The top of the BASIC scratchpad area (which is also the top of the memory area that BASIC uses excepting a 256 byte stack) is pointed to in the BCA - BASIC Control Area. Recall from the April issue of PORT FE that location 145H (325 dec) points to the bottom of the scratchpad space, location 192H (402 dec) points to the top of the scratchpad space, and location 1A6H (422 dec) points to the current location in the scratchpad for work. When you turn on the machine, this is set so that 1A6H=192H and 145H=192H-50. Location 192H (top of memory for BASIC) is set just below the BASIC stack which is just below the monitor stack which is just below the MWA. That is, in a 16K machine, the memory map looks like:

Top of RAM	3FFFH	
	3F91H - 3FFFH	MWA
	3F00H - 3F90H	Monitor Stack
	3E00H - 3EFFH	BASIC Stack
	xxxxH - 3DFFH	BASIC scratchpad area

When you start up a BASIC programme, the very first statements can alter the BCA to start the scratchpad lower down in memory. This will leave an n-sized buffer area of memory free for you to put in your machine language subroutines. The map will now look like:

Top of RAM	3FFFH	
	3F91H - 3FFFH	MWA
	3F00H - 3F90H	Monitor Stack
	3E00H - 3EFFH	BASIC Stack
	yyyyH - 3DFFH	your M/L routines
	xxxxH - yyyyH-1	BASIC scratchpad area

All you have to do is POKE the three locations with the address yyyyH - 1 (before any CLEAR statements are made) and put your M/L routines at location yyyyH and up. You can reserve the exact space you need for these routines since you know how much memory they will take up.

The drawback to this method is that addresses will have to change to accomodate different machine sizes. This can be overcome with a few statement lines which can calculate the offset addresses for any size machine. When you turn the Sorcerer on, it seeks out the top of RAM in order to place the MWA in the proper location. This location (top of RAM) is stored for reference at locations F000H and F001H. Therefore, your programme can find the top of RAM in in any Sorcerer with the single statement:

```
TOR = PEEK(-4096) + 256 * PEEK(-4095)
```

Since the MWA and the two stack areas are always a constant 512 bytes, then the location to put your routines at (ie. yyyyH) is given by the statement:

```
YYYY = TOR - 512 - (the size of RAM needed for your routines)
```

Now POKE the three addresses in the BCA with YYYY - 1, use your CLEAR command as the LAST command of the section just before the main programme and you're almost away.

Another problem exists and that is that your M/L routines have to be relocatable. Otherwise, all the JP and CALL addresses will be wrong. In order to have a fully self-contained BASIC programme, you should store the M/L subroutine in DATA statements and on start-up, POKE them into memory. In order to properly relocate the code, I use the following method.

Any JP and CALL addresses are spacified as an offset from the beginning of the M/L block area. That is, the address of the CALL or JP is stored in the DATA statement as:

```
--(JP or CALL address - beginning of block)
```

Yes, that is a big "MINUS" sign in front. That is so that when BASIC is reading data to poke into memory, it can determine offset addresses from the rest of the regular code. Since the only thing that is different from one machine to another is the start block address of the subroutine area, this can be added back in at execution time.

A "dummy" programme to do this would look like:

```
XX10 RESTORE XX90:MLP=YYYY
XX20 READ CODE:IF CODE=9999 THEN X100
XX30 IF SGN(CODE)=-1 THEN CODE =ABS(CODE)+YYYY
XX40 POKE MLP,CODE:MLP=MLP+1
XX50 GOTO XX20
XX90 DATA 196,45,110,-67,222,9999
X100 (start of main programme)
```

then when calling the routine in your programme using the USR function, the calling address is YYYYH for any machine.

If it looks a little troublesome, belive me, its not really. Just do a few small test programmes for practise and you will see that it is quite straight forward once you get used to it.

GENERAL NEWS

- Paul and Robert are still working on the modems. They are presently getting sporadic readings. They hope to have the bugs ironed out soon.

- however, Universal Data Systems just announced a new modem in the last issue of BYTE which is FCC approved and operates at 1200 baud via a direct connection for $295.00 US. It may be worth closer investigation.

- next meeting is THURSDAY AUGUST 14, 1980 at my place, 200 BALSAM AVE. TORONTO at 7:30 pm.

  By car:-
  take the GARDINER EXPRESSWAY Eastbound to the end which turns into LAKESHORE BLVD. Continue on LAKESHORE to the end which turns into WOODBINE AVE. Turn right (EAST) at the first light which is QUEEN ST. Take QUEEN ST East about three or four stoplights to BEECH AVE. Turn left (north) on BEECH and go up the hill for 2 blocks which is the intersection of BEECH and BALSAM. My house is on the NORTHWEST CORNER of that intersection.

  By TTC:-
  take the SUBWAY to COXWELL STN and get the COXWELL SOUTH BUS. The bus will go down to QUEEN ST and come back up KINGSTON RD. Get off at BEECH and walk 1 block south to BALSAM AVE. My house is on the NORTHWEST corner of that intersection.