commodore

comments and bulletins
concerning your
COMMODORE PET™

The Transactor

Vol. 2
BULLETIN #6
November 30, 1979

PET™ is a registered Trademark of Commodore Inc.

Inside the 2040 Disk Drive

Jim Butterfield, Toronto

Yes, you can look at the programs inside the 2040. But unless you're strong in machine language - and have a bit of hardware background - it won't make much sense.

There are two processors in there. One looks out toward the PET .. I'll call it the IEEE processor; the other looks in toward the disk mechanics .. this one I'll call the disk processor. Each processor has a completely different set of programs. The two processors talk to each other by sharing a little memory space: about 4K of RAM is common to both micfoprocessors.

The IEEE processor is relatively easy to look into. You have the M-R, or memory read, command which allows you to look at the whole 64K memory space of this processor. Not all of this is actually fitted with memory, of course. As far as I can tell, ROM occupies hex locations E000 to FFFF. There's RAM in zero page; and the RAM which is shared with the disk microprocessor is in hex 1000 to 1FFF. The 6532 PIA chips seem to be in ~~the~~ addresses $0200 to $03FF.

To analyze a completely unknown 650X program, you must start by inspecting locations $FFFA to $FFFF. This gives you the three main vectors, for NMI, Reset, and INT. As far as I can tell, NMI isn't used - the vector points at non-existent memory. Reset is of course used; in my 2040 it points at F480, and that's where the main code for initialization begins. It looks to me as if the interrupt line must be kicked by the IEEE ATN (attention) line: when I follow the vector (FDDE) in my machine, it looks like an IEEE handshake is taking place.

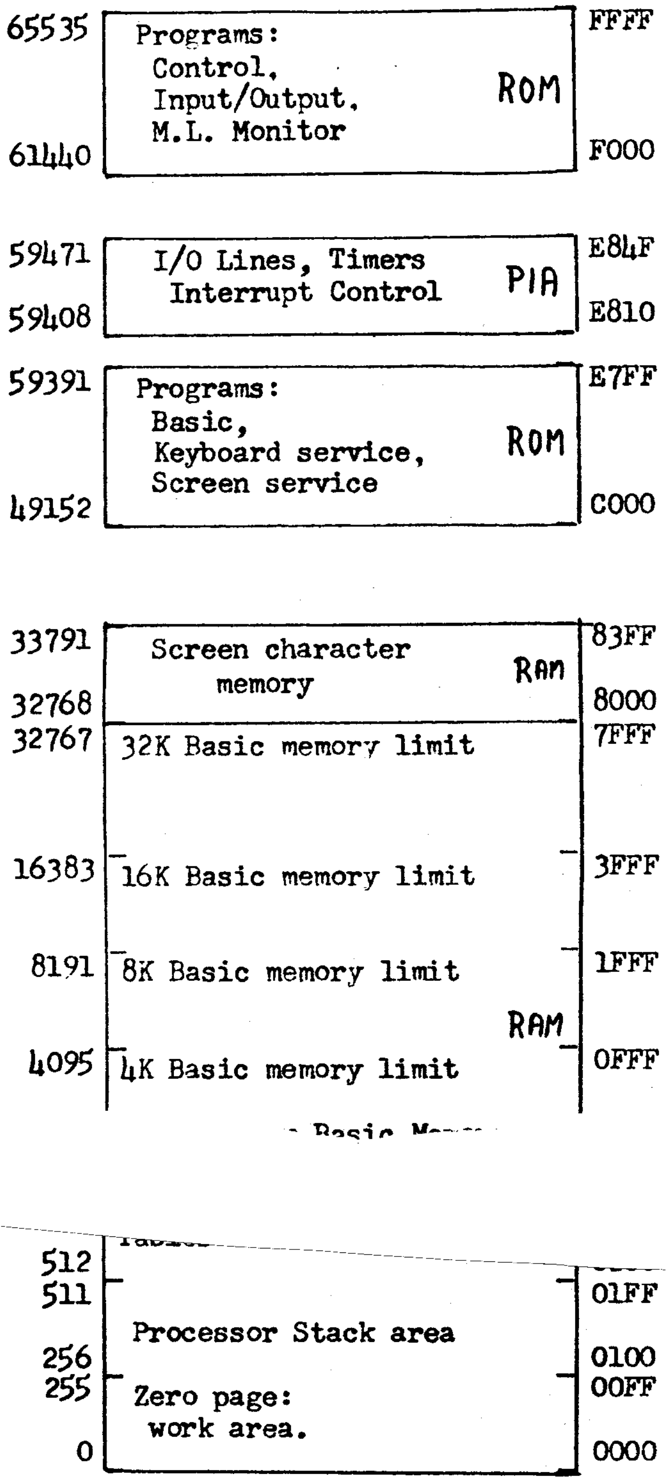
That's all vefy well for the IEEE processor, but how can you get a look at the inner, disk processor? I had trouble with this one, until one day I discovered that the IEEE processor can download the disk processor - via the shared RAM - and make it execute this new code! So all that's needed is a little program to tell the disk processor to copy part of its memory to the shared RAM space, where it can be examined by using the M-R command.

I couldn't get this to work, however, until I discovered the vital missing link. The shared RAM, which is seen at locations 1000 to 1FFF by the IEEE processor, is seen in a completely different location by the disk processor! .. in this case, hex 0400 to 13FF. The hardware just "maps" the memory into a different location. I might never have spotted this if the memories had not overlapped; but a little rummaging around and tearing of hair showed that my early programs seemed to be putting data into the wrong buffer. Eventually, the penny dropped, and the system became clear.

I'm far from being able to give details about the inner secrets of the 2040. But with the enclosed DISK PEEK program, you too can rummage around in there - in either processor's memory space - and come up with interesting data.

```
100 PRINT"▯DISK MEMORY DISPLAY    JIM BUTTERFIELD"
110 DATA77,45,87,0,18,16,162,0,189
120 DATA157,64,06,232,224,16,208,245,76,193,254
130 FORJ=1TO9:READX:C$=C$+CHR$(X):NEXTJ
140 FORJ=1TO11:READX:D$=D$+CHR$(X):NEXTJ
150 PRINT"▮THERE ARE TWO PROCESSORS:"
160 PRINT"  1) THE IEEE PROCESSOR;"
170 PRINT"  2) THE DISK PROCESSOR;"
180 INPUT"WHICH DO YOU WANT TO PEEK (1 OR 2)";D
190 PRINT"INPUT MEMORY ADDRESS"
200 PRINT"IN HEXADECIMAL:":OPEN1,8,15
210 PRINT"   ▮▮▮▮                           ⌝"
220 INPUTZ$
230 PRINT"⌝";:IFLEN(Z$)<>4THENGOTO210
240 FORJ=1TO4:Y=ASC(MID$(Z$,J))
250 IFY<58THENY=Y-48
260 IFY>64THENY=Y-55
270 IFY<0ORY>16GOTO210
280 Y(J)=Y:NEXTJ:K=0:PRINT"▮▮▮▮▮▮";
290 ONDGOTO300,320:GOTO180
300 U=Y(3)*16+Y(4):V=Y(1)*16+Y(2)
310 GOSUB360:GOTO210
320 PRINT#1,C$;CHR$(Y(3)*16+Y(4));CHR$(Y(1)*16+Y(2));D$
330 PRINT#1,"M-W";CHR$(4);CHR$(16);CHR$(1);CHR$(224)
340 PRINT#1,"M-R";CHR$(4);CHR$(16);CHR$(1);CHR$(224)
345 GET#1,X$:IFX$=CHR$(224)GOTO340
350 U=64:V=18:GOSUB360:GOTO210
360 PRINT#1,"M-R";CHR$(U);CHR$(V)
370 GET#1,X$:IFX$=""THENX$=CHR$(0)
380 PRINT" ";:X=ASC(X$)/16
390 FORJ=1TO2:X%=X:X=(X-X%)*16:IFX%>9THENX%=X%+7
400 PRINTCHR$(X%+48);:NEXTJ
410 U=U+1:IFU=256THENU=0:V=V+1
420 K=K+1:IFK<8GOTO360
430 Y(0)=0:Y(4)=Y(4)+8:J=4
440 IFY(J)>15THENY(J)=Y(J)-16:J=J-1:Y(J)=Y(J)+1:GOTO440
450 PRINT:PRINT"   ";:FORJ=1TO4:Y=Y(J):IFY>9THENY=Y+7
460 PRINTCHR$(Y+48);:NEXTJ:PRINT"⌝":RETURN
```

```
**** THE LAST THREE ITEMS IN LINE 120 (76,193,254) MAY BE CHANGED
    IF NECESSARY TO A RESET SEQUENCE OF 108,252,255      ****
```



65535	Programs: Control, Input/Output, M.L. Monitor	ROM	FFFF
61440			F000
59471	I/O Lines, Timers Interrupt Control	PIA	E84F
59408			E810
59391	Programs: Basic, Keyboard service, Screen service	ROM	E7FF
49152			C000
33791	Screen character memory	RAM	83FF
32768			8000
32767	32K Basic memory limit		7FFF
16383	16K Basic memory limit		3FFF
8191	8K Basic memory limit		1FFF
		RAM	
4095	4K Basic memory limit		0FFF
	- Basic Memo		
	Tab		
512			
511	Processor Stack area		01FF
256			0100
255	Zero page: work area.		00FF
0			0000

Commodore PET and CBM memory organization.

Printer Formatting

There has been a bug detected with the formatting feature of the 2022 and 2023 Printers but fortunately, Kim Lantz of North Sydney, Nova Scotia, has found the fix.

It seemed that setting up the first format was no problem, but changing to a second format was. When PRINTing to the printer, the last character to be sent to a line is a CRLF. This is done for obvious reasons but, the Carriage Return is printed on the current line and the Line Feed is printed on the next line. The Line Feed character is of course not printed on the paper but the printer "sees" it as the first character of the new line and when the printer is anywhere but the absolute beginning of a line, it doesn't like changing the format.

Therefore, anything that is output to secondary address 1 of the printer should be followed by...

```
;CHR$(13);
```

For e.g.

```
OPEN 1,4,1
PRINT #1, X;CHR$(13);
PRINT #1, "PET";CHR$(13);
```

...especially when the format string is about to be changed. This is also true for secondary address 0.

The above can of course be shortened by first equating R$ to CHR$(13) and using R$ in place of CHR$(13). Also the first semi-colon is not necessary when preceeded by a closing quote or another string variable but is necessary when following numeric variables.

However, the general idea is to keep the printer in the 0'th position after a carriage return when the format string is to be changed.

Bits and Pieces

The IF..THEN statement can be very useful in avoiding certain unexpected hazards. Two in particular are 1) argument outside range and 2) dividing by zero.

The ON..GOTO statement has a limited range on its argument; 1 to 255. Zero causes execution to drop through to the next line but values negative or over 255 will cause an error and a forced break. Protecting against this is easy and often a good idea.

```
500 IF X > -1 AND X < 256 THEN ON X GOTO...   (GOSUB)
501 REM -CODE FOR X = 0
```

Executing a 'THEN' causes PET to interpret the code following as a "new line". A 'THEN' can therefore be followed by any BASIC statement including another 'IF..THEN'.

Dividing by zero will fail for obvious reasons. Preceeding a possible trouble spot with a denominator test will protect against ?DIVISION BY ZERO ERROR.

```
600 IF D <> 0 THEN IF N/D <> 0 THEN
    IF N2/(N/D) > 1 GOTO 880
```

Another hidden gotcha that has been known to cause bald spots is the peculiar behavior of the FOR..NEXT loop. Code within a FOR..NEXT loop will always execute at least once regardless of the initial loop counter values.

```
700 IF J > 0 THEN FOR X = 1 TO J:...: NEXT
```

...will guard against unwanted looping. Only one problem; the entire loop must be squeezed into one line otherwise GOTOs must be used.

One further note; a STEP size of zero will cause endless looping. Depending on the extent of STEP use, testing of STEP variables might be advisable.

Bullet-Proof INPUT

As you know, INPUT allows the cursor control characters to be typed which can really foul up a program especially when user infallibility is of importance. The following subroutine could substitute for INPUT:

```
5000 POKE  167 , 0
5010 A$ = ""
5020 GET B$ : IF B$ = "" THEN 5020
5030 IF ( ASC ( B$ ) AND 127 ) > 31 THEN
     PRINT B$; : A$ = A$ + B$
5040 IF B$ = CHR$( 13 ) THEN POKE 167 , 1 : RETURN
5050 GOTO 5020
```

Line Explanation

5000 The only drawback using GET over INPUT was that a simulated cursor was required. POKE 167 , 0 (548 in old ROM) conveniently turns the PETs cursor on.

5010 Sets A$ (the input string) to null string.

5020 Standard "GET loop".

5030 This test masks out all of the cursor control keys, allowing only numeric, alpha and graphics to PRINT.

5040 Test for 'RETURN' key, yes...turn cursor off, exit.

Extra tests could be inserted between 5030 and 5040 to include cursor left/right and/or delete. Also, a character counter might be incorporated to limit the input string length.

Floating Binary

The following program by Jim Butterfield shows the true value of a decimal floating point number as stored by PET in floating binary. The program illustrates how some decimal values cannot be represented in binary exactly. Try values of 1.1, 1.2 and 1.7.

```
100 PRINT : INPUT V
110 PRINT INT(V);".";
120 V = (V - INT(V)) * 10 : IF V=0 GOTO 100
130 PRINT CHR$(V + 48);
140 GOTO 120
```

The following reference table shows the screen memory POKE locations. Note the start and end locations and that the most significant digit (3) has been dropped throughout the table. Reprinted from the Commodore Japan Newsletter.

Vol 2 #6

32768

2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799	2800	2801	2802	2803	2804	2805	2806	2807
2808	2809	2810	2811	2812	2813	2814	2815	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879	2880	2881	2882	2883	2884	2885	2886	2887
2888	2889	2890	2891	2892	2893	2894	2895	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959	2960	2961	2962	2963	2964	2965	2966	2967
2968	2969	2970	2971	2972	2973	2974	2975	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039	3040	3041	3042	3043	3044	3045	3046	3047
3048	3049	3050	3051	3052	3053	3054	3055	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119	3120	3121	3122	3123	3124	3125	3126	3127
3128	3129	3130	3131	3132	3133	3134	3135	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199	3200	3201	3202	3203	3204	3205	3206	3207
3208	3209	3210	3211	3212	3213	3214	3215	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279	3280	3281	3282	3283	3284	3285	3286	3287
3288	3289	3290	3291	3292	3293	3294	3295	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359	3360	3361	3362	3363	3364	3365	3366	3367
3368	3369	3370	3371	3372	3373	3374	3375	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439	3440	3441	3442	3443	3444	3445	3446	3447
3448	3449	3450	3451	3452	3453	3454	3455	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519	3520	3521	3522	3523	3524	3525	3526	3527
3528	3529	3530	3531	3532	3533	3534	3535	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599	3600	3601	3602	3603	3604	3605	3606	3607
3608	3609	3610	3611	3612	3613	3614	3615	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679	3680	3681	3682	3683	3684	3685	3686	3687
3688	3689	3690	3691	3692	3693	3694	3695	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759	3760	3761	3752	3763	3764	3765	3766	3767

33767

Screen I/O

Some of you may have experienced problems PRINTing characters to the screen over top of characters that are already there. Try, for example, the following program:

```
        100 ?"home";
        110 FOR J = 1 TO 10
        120 ?"+++++++++++++++++++++++++++++++++
        +++++++++++++++++++++++++++++"   (approx 60)
        130 NEXT
        140 ?"home";
        150 FOR J = 1 TO 10
        160 ?"*******************"
        170 NEXT
        180 END
```

So why the extra line feeds? PET maintains a "line wrap" table in RAM which determines whether the line is a single or a double line or more precisely, over or under 40 characters. This is done for things like INPUT and for entering or altering BASIC.

For upgrade ROMs the wrap table is kept in RAM from 00E0 to 00F8 (decimal 224 - 248). 0229 to 0241 (dec 553 - 577) for old ROMs.

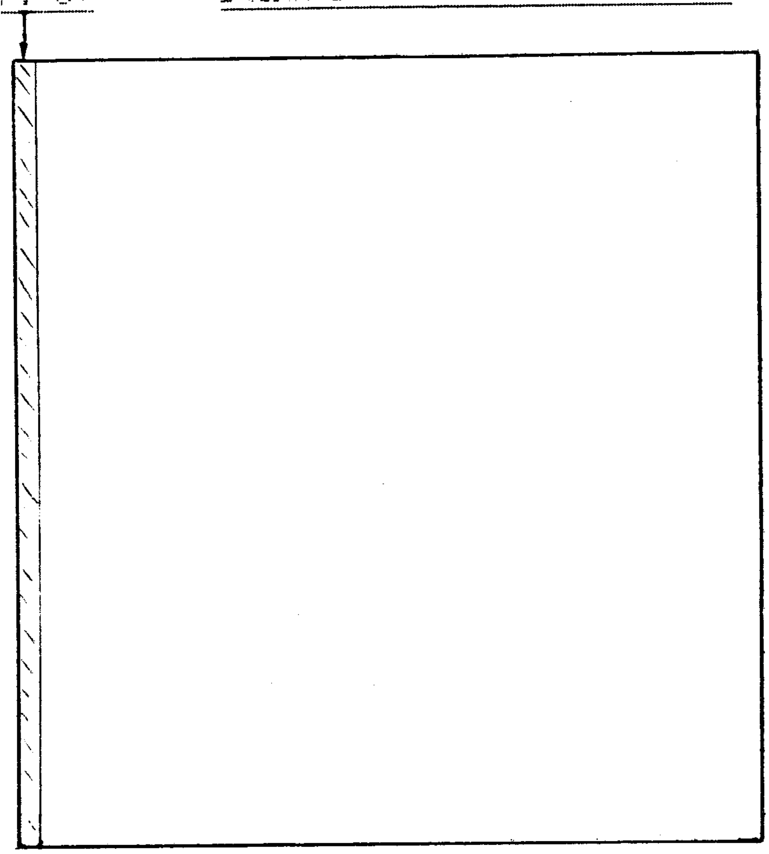
So how do we eliminate these dastardly line feeds? You could play with "cursor ups" but if some lines are double and others single this can be somewhat cumbersome especially if your PRINT strings end at column 40. The alternative is to alter the information held in the line wrap table.

The table consumes 25 bytes of RAM, one byte for each line on the screen. These bytes will contain the lines high order memory address. As you know, screen memory starts at hex 8000 and continues to hex 8FFF (see memory map). The home position of the screen is therefore at hex 8000. Since the address of a line is taken from the beginning of that line, the address of the top line will be $8000 ($ = hex). The high order address is simply $80 and the decimal equivalent of $80 is 128. The PEEK of the first location of the wrap table will return a 128 which is of course decimal.

The following relates wrap table decimal values (PEEK values) to the hex address of the first character space of each screen line. Remember, only the high order part of the address is of any concern to the wrap table. Also, the table resides in different locations for old and new ROMs so for now we'll call them locations 1 through 25.

	Wrap Table	Hex addr. of
1.	128	8000
2:	128	8028
3.	128	8050
4:	128	8078
5:	128	80A0
6.	128	80C8
7.	128	80F0
8.	129	8118
9.	129	8140
10:	129	8168
11.	129	8190
12.	129	81B8
13.	129	81E0
14:	130	8208
15.	130	8230
16.	130	8258
17.	130	8280
18.	130	82A8
19.	130	82D0
20.	130	82F8
21.	131	8320
22.	131	8348
23.	131	8370
24:	131	8398
25.	131	83C0

Blank Screen (single lines)



If the wrap table PEEK values were represented in binary, the eighth bit would be set to 1 in each case:

```
128 = 1 0 0 0  0 0 0 0
131 = 1 0 0 0  0 0 1 1
```

This means that the corresponding line is single or has less than 40 characters on it.

When characters outputing to the screen wrap around the right side, PET considers these characters as part of the above line. Take, for example, the top two lines (lines 1 & 2). The screen is cleared and a string of 52 characters are PRINTed from the home position, past column 40 and onto line 2. Line 2 is now considered part of a double line but more importantly, line 1 is considered a single line of double length. The wrap table records this by setting the eighth bit of the value corresponding to line 2 to zero. The top two lines are now treated by PET as a single line hence the extra line feeds. This is most noticeable when using the screen editor on program lines of length greater than 40.

The wrap table values for the example program would be:

Wrap Table		Hex addr. of
1	128	8000
2	0	8028
3	128	8050
4	0	8078
5	128	80A0
6	0	80C8
7	128	80F0
8	0	8118
9	129	8140
10	1	8168
11	129	8190
12	1	81B8
13	129	81E0
14	2	8208
15	130	8230
16	2	8258
17	130	8280
18	2	82A8
19	130	82D0
20	2	82F8
21	131	8320
22	131	8348
23	131	8370
24	131	8398
25	131	83C0

Program Example

```
******************++++++++++++++++++++++
+++++++++++++++++++++++
******************++++++++++++++++++++++
+++++++++++++++++++++++
******************++++++++++++++++++++++
+++++++++++++++++++++++
******************++++++++++++++++++++++
+++++++++++++++++++++++
******************++++++++++++++++++++++
+++++++++++++++++++++++
******************++++++++++++++++++++++
+++++++++++++++++++++++
******************++++++++++++++++++++++
+++++++++++++++++++++++
******************++++++++++++++++++++++
+++++++++++++++++++++++
******************++++++++++++++++++++++
+++++++++++++++++++++++
******************++++++++++++++++++++++
+++++++++++++++++++++++
```

The Solution

If PRINTing on double lines has thrown a wrench into your program, the easiest solution is make all lines single. Insert the following lines into the example program and RUN it

```
New ROM      143 FOR J = 224 TO 248 : X = PEEK (J)
             145 POKE J,X OR 128   : NEXT

Old ROM      143 FOR J = 553 TO 577 : X = PEEK (J)
             145 POKE J,X OR 128   : NEXT
```

The "OR" function in line 145 is used to set the eighth bit to 1, thus altering the wrap table such that PET considers all lines as single.

A few entry points, original/upgrade ROM   Jim Butterfield

Entry points seen in various programmer's machine language programs. The user is cautioned to check out the various routines carefully for proper setup before calling, registers used, etc.

ORIG	UPGR	DESCRIPTION
C357	C355	?OUT OF MEMORY
C359	C357	Send Basic error message
C38B	C389	Warm start, Basic
C3AC	C3AB	Crunch & insert line
C430	C439	Fix chaining & READY.
C433	C442	Fix chaining
C48D	C495	Crunch tokens
C522	C52C	Find line in Basic
C553	C55D	Do NEW
C56A	C572	Do CLR
C59A	C5A7	Reset Basic to start
C6B5	C6C4	Continue Basic execution
C863	C873	Get fixed-point number from Basic.
C9CE	C9DE	Send Return,LF if in screen mode
C9D2	C9E2	Send Return, Linefeed
CA27	CA1C	Print string
CA2D	CA22	Print precomputed string
CA49	CA45	Print character
CE11	CDF8	Check for comma
CE13	CDFA	Check for specific character
CE1C	CE03	'SYNTAX ERROR'
D079	D069	Bump Variable Address by 2
D0A7	D09A	Float to Fixed conversion
D278	D26D	Fixed to Float conversion
D679	D67B	Get byte to X reg
D68D	D68F	Evaluate String
D6C4	D6C6	Get two parameters
D73C	D773	Add (from memory)
D8FD	D934	Multiply by memory location
D9B4	D9EE	Multiply by ten
DA74	DAAE	Unpack memory variable to Accum #1
DB1B	DB55	Completion of Fixed to Float conversion
DC9F	DCD9	Print fixed-point value
DCA9	DCE3	Print floating-point value
DCAF	DCE9	Convert number to ASCII string
E3EA	E3D8	Print a character
na	E775	Output byte as 2 hex digits
na	E7A7	Input 2 hex digits to A
na	E7B6	Input 1 hex digit to A
F0B6	F0B6	Send 'talk' to IEEE
F0BA	F0BA	Send 'listen' to IEEE
F12C	F128	Send Secondary Address
E7DE	F156	Send canned message
F167	F16F	Send character to IEEE
F17A	F17F	Send 'untalk'
F17E	F183	Send 'unlisten'
F187	F18C	Input from IEEE
F2C8	F2A9	Close logical file
F2CD	F2AE	Close logical file in A

F32A	F301	Check for Stop key
F33F	F315	Send message if Direct mode
na	F322	LOAD subroutine
F3DB	F3E6	?LOAD ERROR
F3E5	F3EF	Print READY & reset Basic to start
F3FF	F40A	Print SEARCHING...
F411	F41D	Print file name
F43F	F447	Get LOAD/SAVE type parameters
F462	F466	Open IEEE channel for output.
F495	F494	Find specific tape header block
F504	F4FD	Get string
F52A	F521	Open logical file from input parameters
F52D	F524	Open logical file
F579	F56E	?FILE NOT FOUND, clear I/O
F57B	F570	Send error message
F5AE	F5A6	Find any tape header block
F64D	F63C	Get pointers for tape LOAD
F667	F656	Set tape buffer start address
F67D	F66C	Set cassette buffer pointers
F6E6	F6F0	Close IEEE channel
F78B	F770	Set input device from logical file number
F7DC	F7BC	Set output device from LFN.
F83B	F812	PRESS PLAY..; wait
F87F	F855	Read tape to buffer
F88A	F85E	Read tape
F8B9	F886	Write tape from buffer
F8C1	F88E	Write tape, leader length in A
F913	F8E6	Wait for I/O complete or Stop key
FBDC	FB76	Reset tape I/O pointer
FD1B	FC9B	Set interrupt vector
FFC6	FFC6	Set input device
FFC9	FFC9	Set output device
FFCC	FFCC	Restore default I/O devices
FFCF	FFCF	Input character
FFD2	FFD2	Output character
FFE4	FFE4	Get character

Infinitely Long PET Programs

Henry Troup, Diemaster Tool

Even with a 32K PET, it is desirable to have a means of handling programs in sections, to be loaded as necessary. The PET implementation of the load command from a program does not reset any pointers, so that variables are preserved. However, any new program must be the same length or shorter than the first of the series.

In order to make certain details such as filenames and the disk commands transparent to the end user, it may be desired to have a small front end or menu program load other, longer programs. No variables need be passed between the programs, so a simple LOAD "nextprogram",8 suffices.

However, since the variable pointers are not reset, they will be pointing into the program. As soon as any variables are used, the program is disturbed, and a machine crash may result. Certainly, this will cause a non-recoverable error. This may be avoided by including the following line as the first of the program:

```
     POKE42,PEEK(201):POKE43,PEEK(202):CLR
```

This resets the bottom of text pointer and then the CLR resets all the other pointers. The program will now run.

If this is the first line in a program, and you modify the program, DO NOT use the RUN command with no parameters. Start runing the program from below this line, or the pointers will be reset to the previous end of the program. In general, you would lose the same number of bytes as were added.

But what about doing a link of two programs and passing data between them?

This is relatively simple. A scratch file can be created and filled with the variables to be passed. These variables are then read by the second program.

However, because of the disk's handling of sequential files, it is advisable to generate your own carriage returns between data items. Using the format

```
PRINT#3,A;CHR$(13);B;CHR$(13);D$;CHR$(13);
```

will avoid any complications due to unwanted line feed characters.

It would be a good idea to have the first program check for a pre-existing file and either delete it, or warn the user to change disks. Obviously, a different filename cannot be used, unless user intervention is provided, which in general would slow the system down.

The existence of a previous file can be checked most easily by opening the filename, either for read or write, and checking the error channel. If a read produces "FILE NOT FOUND" or a write succeeds then the filename has not been used. Otherwise, some action should be taken. Scratch files should also be deleted when they are of no further use.

This also allows long programs of the number-crunching variety to be interrupted and restarted. One could write programs with a very long run time so that they can "go to sleep" either by keyboard command or after a set time.

The machine will hang up if the next program to be loaded is not found.

Programming

PET DOS SUPPORT PROGRAM

By R. J. Fairbairn

Now that the COMMODORE 2040 Floppy Disk System is reaching PET owners more support programs are needed. The PET DOS SUPPORT Program is an aid to the 2040 User which humanizes the PET to 2040 interface better than direct mode BASIC statements.

This program consists of two routines; a BASIC driver routine and a machine language routine. The BASIC program calls the machine language which moves the working portion of itself up into high memory. The subroutine then links itself into the CHRGET subroutine in page zero and before returning moves the top of memory pointer down so BASIC will not destroy the working portion. The BASIC program then clears the PET screen and displays an abbreviated set of instructions before executing a NEW command.

Figure A and Figure B are the BASIC and ASSEMBLY Listings of the DOS SUPPORT Program. The programs are entered into the PET as follows. First reset the PET so the memory is initialized, this makes entry of machine code simpler. After the PET has been reset type in the BASIC program exactly as listed in figure A. Then using the machine language monitor enter the object code for the machine language subroutine at $0700 hex. After entry save both routines from the monitor (SA = $0400,EA = $08B8). Finally, using the instructions included in this article test the program to insure correct operation. Good luck and happy computing.

WARNING: It is advisable to use diskettes that are new or that contain no valuable data during the test phase. This will avoid loss of important data and your time.

The purpose of this program is to aid the CBM or PET 2001 User in operating the 2040 Dual Floppy Disk System. This instruction sheet has been written with the assumption that the reader has a working knowledge of the 2001 series and the 2040.

NOTE: This program has been placed in the public domain but if you would like us to produce a copy for you, send us a blank disk and we'll duplicate the DOS SUPPORT Program on it at no charge. Though, we do ask that you include a self-addressed ,stamped envelope. If you have any comments or suggestions on the following, please refer them to the editor.

The normal method with which the PET communicates with an IEEE Buss device is by the BASIC commands OPEN, PRINT, GET, INPUT and CLOSE. These statements are somewhat verbose in nature and therefore more prone to operator error. There is also the limitation that INPUT and GET cannot be used in direct mode due to shared buffer areas. These conditions are easily handled with the DOS SUPPORT PROGRAM.

DOS SUPPORT PROGRAM may be loaded (saved) as if it were a normal BASIC program. Note should be made of the fact that the 2040 has a special load file name '*' which if used immediatly after power up (reset) executes the following:

1. Initalizes Drive 0
2. Loads the first file on that drive

Thus if the command LOAD"*",8 is executed and the DOS SUPPORT Program is the first directory entry it will be loaded. When the DOS SUPPORT Program is executed it relocates itself up into the highest available RAM memory locations, links into the CHRGET routine and adjusts BASIC's top of memory pointer down. This technique uses about 350 bytes of the Users memory but normal machine operations may proceed without having to reload the DOS SUPPORT Program until such time that a system reset is performed.

The DOS SUPPORT Program functions by capturing the data that the PET operating system passes to BASIC, before the interpreter has a chance to parse it. Thus we can look for Key (escape) characters and process the disk command which follows without the use or knowledge of the BASIC interpreter.

There are four key characters that are recognized by the DOS SUPPORT Program. They will be processed only when they are found in column one of an input line, otherwise a SYNTAX ERROR will occur.

DOS SUPPORT KEY CHARACTERS

@ or > - Passes commands to the Disk.
/ - LOAD's a program.
↑ - LOAD's and RUN's a program.

The greater than symbol when used preceeding a 2040 Disk command passes that command directly to the Floppy Disk System. See the following examples.

```
Thus:
>IØ
is the same as:
PRINT#15,"IØ"
and:
>SØ:FILE1
is equal to:
PRINT#15,"SØ:FILE1"
```

As you can see the > symbol is a substitute for the PRINT#15 statement. Remember that an OPEN statement is required before a PRINT may be executed but no OPEN is required for the DOS SUPPORT Program.

The second function of the > command is the directory list command. As you know the directory of a minidisk can be loaded with a LOAD"$Ø",8. This LOAD will destroy any program you might have in memory. To avoid the destruction of the current program the DOS SUPPORT program prints the directory on the screen.

To avoid possible directory scrolling, you may depress the SPACE key to stop the listing of a directory. Depress any key to continue the listing - or you may depress the RUN/STOP key to stop the directory listing and return to BASIC.

```
>$Ø
```

Means - Display the entire directory of Drive Ø

```
>$1:Q*
```

Means - Display the directory entries of all files on Drive 1 that have names starting with the letter Q.

The third function of the > command is the error channel interrogation feature. The error channel is read by typing a > followed immediately by a RETURN. This is equivilent to the following program segment.

```
10 OPEN 15,8,15
20 INPUT#15,ER,MSG$,DRV,SEC
30?ER",MSG$","DRV","SEC
```

For Users that have the CBM Model Business Keyboard the "@" key may be used in place of the > for key entry convience. This eliminates shifting for this command.

The LOAD / and LOAD-RUN ↑ command characters operate the same as their BASIC counterparts only with a simplified syntax as follows: /WUMPUS

- This command will load the program file WUMPUS. Both drives will be searched if required.

↑1:COPY DISK FILES

-This command will load the program COPY DISK FILES from Drive 1 (if it is there) and execute it.

The following requirements and limitations are placed on the DOS SUPPORT Program User.

1. The DOS SUPPORT commands may only be used in the direct mode.

2. The commands must start in Column 1.

The user may print the directory by using the following commands:

OPEN 4,4: CMD4	: Opens device 4 and changes the primary output device to 4
>$Ø	: Print the directory
PRINT#4 : CLOSE 4	: Return the default output device to the screen and close the file

```
5 SYS2222
10 PRINT"■"TAB(11)"______________________"
20 PRINTTAB(11)"■ PET DOS SUPPORT "
30 PRINTTAB(14)"NOW  LOADED
40 PRINTTAB(9)"  COMMANDS FOLLOWING"
50 PRINTTAB(7)"A > OR @  IN COLUMN 1 WILL"
60 PRINTTAB(9)"BE PASSED TO THE DISK.■"
90 PRINTTAB(7)"CMD         DESCRIPTION"
140 PRINTTAB(7)"$      DIRECTORY BOTH DRIVES
150 PRINTTAB(7)"$0     DIRECTORY DRIVE 0
160 PRINTTAB(7)"$1     DIRECTORY DRIVE 1■"
180 PRINTTAB(7)"  ALL 2040 COMMANDS MAY BE
190 PRINTTAB(7)"ENTERED AS IF THEY WERE IN
200 PRINTTAB(7)"A PRINT# STATEMENT.
220 PRINTTAB(11)"■■SPECIAL COMMANDS
230 PRINTTAB(7)"■/      LOAD A PROGRAM
240 PRINTTAB(7)"↑      RUN  A PROGRAM
250 PRINT"  SPECIAL COMMANDS START IN COL 1 AND
260 PRINT"ARE FOLLOWED BY A 2040 FILENAME.
270 NEW
```

```
PETDOS4.0.

LINE# LOC    CODE          LINE


0001  0000                 ;******************************
0002  0000                 ;*
0003  0000                 ;*  PET DOS SUPPORT
0004  0000                 ;*
0005  0000                 ;*    04-27-79
0006  0000                 ;*
0007  0000                 ;*  BOB FAIRBAIRN
0008  0000                 ;*
0009  0000                 ;******************************
0010  0000                 ;*
0011  0000                 ;* VERSION 3.1 6/14/79
0012  0000                 ;*    ADD @ PROMPT FOR BUSINESS
0013  0000                 ;*    KEYBOARD. ADD STOP KEY CHECK
0014  0000                 ;*    IN DIRECTORY PRINT. ADD
0015  0000                 ;*    HALT IN DIRECTORY PRINT
0016  0000                 ;*
0017  0000                 ;* VERSION 3.2 7/2/79
0018  0000                 ;*    FOR (-04) ROM
0019  0000                 ;*    WITH LOAD ADDRESS ONE OFF
0020  0000                 ;*    BYTE LOW.
0021  0000                 ;*
0022  0000                 ;* VERSION 3.3 7/2/79
0023  0000                 ;*    ADD STACK LOOKUP FOR
0024  0000                 ;*    ACTIVATION.
0025  0000                 ;*
0026  0000                 ;* VERSION 4.0 7/5/79
0027  0000                 ;*    ADD CONTROL FOR CMD DURING
0028  0000                 ;*    A DIRECTORY LISTING.
0029  0000                 ;*


0031  0000                 ;
0032  0000                 ;BASIC VARIABLES USED
0033  0000                 ;
0034  0000                 VERCK   =$9D           ;VERIFY FLAG
0035  0000                 SAL     =$C7           ;INDIRECT POINTER LO
0036  0000                 SAH     =$C8           ;HI
0037  0000                 WSW     =$B3           ;UNUSED FLAG (BASIC)
0038  0000                 CNTDN   =$BA           ;SAVE AREA
0039  0000                 GRBTOP  =$5C           ;INDIRECT POINTER
0040  0000                 MEMSIZ  =$34           ;POINTER TO TOP MEM
0041  0000                 TXTPTR  =$77           ;POINTER TO BUF
0042  0000                 SPERR   =$10           ;EOI ERROR BIT
0043  0000                 BUF     =$0200         ;BASIC INPUT BUFFER
0044  0000                 SATUS   =$96           ;STATUS BYTE
0045  0000                 SA      =$D3           ;SECONDARY ADDRESS
0046  0000                 FA      =$D4           ;PRIMARY ADDRESS
0047  0000                 LA      =$D2           ;LOGICAL DEVICE #
0048  0000                 FNLEN   =$D1           ;FILE NAME LENGTH
0049  0000                 FNADR   =$DA           ;FILE NAME ADDRESS
0050  0000                 EAL     =$C9           ;END ADDR LO
0051  0000                 EAH     =$CA           ;HI
```

```
0052  0000                DFLTO   =$B0        ;DEFAULT OUTPUT DEV.
0053  0000                VARTAB  =$2A        ;END OF BASIC PGM.
0054  0000                TMP2    =$FD        ;TEMP VARIABLE
0055  0000                ;
0056  0000                ;PROGRAM VARIABLES
0057  0000                ;
0058  0000                CR      =$0D        ;SYMBOLIC CARRIAGE RETURN
0059  0000                FLAG    =WSW        ;BYTE USED AS A FLAG
0060  0000                PIAK    =$E812      ;KEYBOARD I/O PORT
0061  0000                CMDLN   =CMDEND-CMD ;LENGTH OF RELCOATE

0063  0000                ;
0064  0000                ;PET ROUTINES USED
0065  0000                ;
0066  0000                LINPRT  =$DCD9      ;PRINT LINE #
0067  0000                SPMSG   =$F315      ;SEND A MESSAGE
0068  0000                LD15    =$F322      ;LOAD ROUTINE
0069  0000                TWAIT   =$F8E6      ;WAIT FOR STOP KEY
0070  0000                CHRGET  =$70        ;INPUTS CHARACTERS
0071  0000                CHRGOT  =$76        ;GET LAST CHAR
0072  0000                NEWSTT  =$C6C4      ;NEW STATEMENT EXEC
0073  0000                PRT     =$E3D8      ;PRINT A CHARACTER
0074  0000                LISTN   =$F0BA      ;SEND LISTEN
0075  0000                SECND   =$F128      ;SEND SA
0076  0000                CIOUT   =$F16F      ;SEND CHARACTER
0077  0000                UNLSN   =$F183      ;UN LISTEN
0078  0000                ACPTR   =$F18C      ;GET A CHARCATER
0079  0000                TALK    =$F0B6      ;SEND TALK
0080  0000                OPENI   =$F466      ;OPEN FILE
0081  0000                FCLOSE  =$F2AE      ;CLOSE FILE
0082  0000                READY   =$C389      ;REENTER BASIC
0083  0000                RUNC    =$C572      ;CLEAR VARIABLES
0084  0000                LNKPRG  =$C442      ;LINK BASIC LINES
0085  0000                UNTLK   =$F17F      ;UN TALK
0086  0000                STXTPT  =$C5A7      ;SET START TEXT POINTER
0087  0000                CHKIN   =$F770      ;CHECK IN
0088  0000                CHKOUT  =$F7BC      ;CHECK OUT
0089  0000                CLRCHN  =$FFCC      ;CLEAR CHANNEL
0090  0000                BASIN   =$FFCF      ;BASIC IN
0091  0000                STOP1   =$F301      ;CHECK FOR STOP KEY
0092  0000                BSOUT   =$FFD2      ;BASIC OUT
0093  0000                FOPEN   =$F524      ;FILE OPEN
0094  0000                LD209   =$F3E6      ;LOAD ERROR

0096  0000                ;
0097  0000                ;WEDGE IN ROUTINE WITH THE
0098  0000                ;COMMAND PARSER AND EXECUTITION
0099  0000                ;
0100  0000                        *=$0700
0101  0700                ;
0102  0700  EA            CMD     NOP         ;THROWN AWAY
0103  0701  E6 77                 INC TXTPTR  ;BUMP POINTER
0104  0703  D0 02                 BNE WG100
0105  0705  E6 78                 INC TXTPTR+1
0106  0707  86 B3         WG100   STX WSW     ;SAVE X IN WSW
```

```
0709  BA                 TSX             ;GET STACK POINTER
070A  BD 01 01           LDA $0101,X
070D  C9 9B              CMP #$9B        ;WERE WE CALLED BY MAIN
070F  D0 3A              BNE NOMAIN      ;NO...
0711  BD 02 01           LDA $0102,X     ;MAYBE?
0714  C9 C3              CMP #$C3
0716  D0 33              BNE NOMAIN      ;NOT THERE...
0718  A5 77              LDA TXTPTR      ;FIRST COLUMN
071A  D0 2C              BNE WG997       ;GET OUT NOT FIRST CHR
071C  A5 78              LDA TXTPTR+1
071E  C9 02              CMP #>BUF       ;IN BUFFER?
0720  D0 26              BNE WG997
0722              ;
0722  A0 00       WG110  LDY #0          ;.Y IS BUF INDEX
0724  84 B3              STY FLAG        ;FLAG SET FOR DIR
0726  B1 77              LDA (TXTPTR),Y
0728  C9 3E              CMP #'>         ;COMMAND PROMPT?
072A  F0 11              BEQ WG115       ;YES...
072C  C9 40              CMP #'@         ;BUSINESS KEYBOARD PROMPT
072E  F0 0D              BEQ WG115       ;YES...
0730  C8                 INY
0731  85 B3              STA FLAG        ;SET FLAG FOR LOAD
0733  C9 2F              CMP #'/         ;LOAD PROMPT
0735  F0 63              BEQ DODIR
0737  C9 5E              CMP #94         ;CHECK FOR ARROW
0739  F0 5F              BEQ DODIR
073B  D0 0B              BNE WG997
073D  C8          WG115  INY
073E  B1 77              LDA (TXTPTR),Y
0740  F0 32              BEQ RDERR       ;READ ERROR CHANNEL
0742  C9 24              CMP #'$         ;DIRECTORY?
0744  F0 54              BEQ DODIR       ;YES
0746  D0 08              BNE NOTDIR
0748  4C 76 00    WG997  JMP CHRGOT
074B  A6 B3       NOMAIN LDX WSW         ;RESTORE .X AND
074D  4C 76 00           JMP CHRGOT      ;RETURN TO CHRGOT

0750              ;
0750              ; SEND COMMAND TO DISK
0750              ;
0750  A9 08       NOTDIR LDA #8          ;GET DEVICE ADDRESS
0752  85 D4              STA FA
0754  A9 6F              LDA #$6F        ;SECONDARY ADDRESS 15
0756  85 D3              STA SA
0758  20 BA F0           JSR LISTN
075B  A5 D3              LDA SA
075D  20 28 F1           JSR SECND       ;SEND SECONDARY ADDR
0760  E6 77       BUMP   INC TXTPTR
0762  A0 00              LDY #0          ;INDEX=0
0764  B1 77              LDA (TXTPTR),Y  ;GET THE FIRST CHARACTER
0766  F0 06              BEQ WG120       ;ZERO IS LAST CHAR
0768  20 6F F1           JSR CIOUT       ;SEND THE CHAR
076B  B8                 CLV
076C  50 F2              BVC BUMP        ;MORE
076E              ;
076E  20 83 F1    WG120  JSR UNLSN       ;UN LISTEN
```

```
0163  0771  B8                 CLV
0164  0772  50 23              BVC WG998
0165  0774             ;
0166  0774             ; READ THE ERROR CHANNEL
0167  0774             ;
0168  0774  84 77      RDERR   STY TXTPTR       ;FIX POINTER
0169  0776  A9 08              LDA #8           ;SET FA
0170  0778  85 D4              STA FA
0171  077A  20 B6 F0           JSR TALK
0172  077D  A9 6F              LDA #$6F         ;COMMAND CHANNEL SA
0173  077F  85 D3              STA SA
0174  0781  20 28 F1           JSR SECND        ;SEND SA
0175  0784  20 8C F1   WG140   JSR ACPTR        ;GET BYTE FROM DISK
0176  0787  C9 0D              CMP #CR
0177  0789  F0 06              BEQ WG130
0178  078B  20 D8 E3           JSR PRT          ;PRINT BYTE TO SCREEN
0179  078E  B8                 CLV
0180  078F  50 F3              BVC WG140        ;LOOP FOR MORE
0181  0791  20 D8 E3   WG130   JSR PRT          ;PRINT CR
0182  0794  20 7F F1           JSR UNTLK        ;UN TALK
0183  0797  4C 76 00   WG998   JMP CHRGOT       ;DONE WITH CMD

0185  079A             ;
0186  079A             ;PRINT THE DIRECTORY
0187  079A             ;
0188  079A  C8         DODIR   INY              ;GET LENGTH OF CMD
0189  079B  B1 77              LDA (TXTPTR),Y
0190  079D  D0 FB              BNE DODIR
0191  079F  88                 DEY
0192  07A0  84 D1              STY FNLEN        ;SET LENGTH (-1)
0193  07A2  A9 01              LDA #<BUF+1      ;FILE NAME ADDRESS
0194  07A4  85 DA              STA FNADR
0195  07A6  A9 02              LDA #>BUF
0196  07A8  85 DB              STA FNADR+1
0197  07AA  A9 08              LDA #8           ;DEVICE ADDRESS
0198  07AC  85 D4              STA FA
0199  07AE  A5 B3              LDA FLAG         ; 0 MEANS DIR
0200  07B0  D0 53              BNE LOADB        ;DO A LOAD
0201  07B2  A5 D2              LDA LA           ;SAVE LA
0202  07B4  85 B3              STA WSW
0203  07B6  A5 B0              LDA DFLTO        ;SAVE DFLTO
0204  07B8  85 BA              STA CNTDN
0205  07BA  A9 60              LDA #$60         ;SECONDARY ADDR
0206  07BC  85 D3              STA SA
0207  07BE  A9 0E              LDA #14          ;OPEN THE FILE
0208  07C0  85 D2              STA LA
0209  07C2  20 83 F1           JSR UNLSN        ;DON'T LISTEN TO FLOPPY
0210  07C5  20 24 F5           JSR FOPEN
0211  07C8  A9 00              LDA #0
0212  07CA  85 96              STA SATUS        ;SET STATUS TO 0
0213  07CC  A0 03              LDY #$03         ;LOOP THREE TIMES
```

```
0215  07CE  84 D1     WG220   STY FNLEN      ;SAVE NEW COUNT
0216  07D0  A2 0E             LDX #14        ;DISK CHANNEL
0217  07D2  20 70 F7          JSR CHKIN
0218  07D5  20 CF FF          JSR BASIN
0219  07D8  85 FD             STA TMP2
0220  07DA  A4 96             LDY SATUS      ;CHECK STATUS
0221  07DC  D0 29             BNE WG235B     ;BAD STATUS
0222  07DE  20 CF FF          JSR BASIN
0223  07E1  85 FE             STA TMP2+1
0224  07E3  A4 96             LDY SATUS      ;CHECK STATUS
0225  07E5  D0 20             BNE WG235B
0226  07E7  A4 D1             LDY FNLEN      ;MORE TO DO?
0227  07E9  88                DEY
0228  07EA  D0 E2             BNE WG220      ;NOT DONE YET
0229  07EC  20 CC FF          JSR CLRCHN     ;CLEAR CHANNEL
0230  07EF  A6 BA             LDX CNTDN      ;CHECK DFLTO FOR SCREEN
0231  07F1  E0 03             CPX #3
0232  07F3  F0 05             BEQ *+7
0233  07F5  A6 B3             LDX WSW        ;OPEN THE PRINT CHANNEL
0234  07F7  20 BC F7          JSR CHKOUT
0235  07FA  A6 FD             LDX TMP2
0236  07FC  A5 FE             LDA TMP2+1
0237  07FE  20 D9 DC          JSR LINPRT     ;PRINT LINE NUMBER
0238  0801  A9 20             LDA #'         ;PRINT A SPACE
0239  0803  D0 06             BNE SKIPB      ;SKIP OVER BRANCHES
0240  0805  D0 6C     LOADB   BNE LOAD       ;(JMP)
0241  0807  D0 5D     WG235B  BNE WG230      ;(JMP)
0242  0809  D0 C3     WG220B  BNE WG220      ;(JMP)
0243  080B  20 D2 FF  SKIPB   JSR BSOUT
0244  080E  20 CC FF          JSR CLRCHN
0245  0811  A2 0E     WG250   LDX #14        ;DISK CHANNEL
0246  0813  20 70 F7          JSR CHKIN
0247  0816  20 CF FF          JSR BASIN
0248  0819  48                PHA
0249  081A  20 CC FF          JSR CLRCHN
0250  081D  68                PLA
0251  081E  A6 96             LDX SATUS
0252  0820  D0 44             BNE WG230      ;BAD
0253  0822  C9 00             CMP #0         ;EOL
0254  0824  F0 26             BEQ WG240
0255  0826  A6 BA             LDX CNTDN      ;CHECK DFLTO FOR SCREEN
0256  0828  E0 03             CPX #3
0257  082A  F0 05             BEQ *+7
0258  082C  A6 B3             LDX WSW
0259  082E  20 BC F7          JSR CHKOUT
0260  0831  20 D2 FF          JSR BSOUT
0261  0834  20 CC FF          JSR CLRCHN
0262  0837                ;
0263  0837                ;CHECK FOR STOP KEY AND PAUSE
3264  0837                ;
0265  0837  20 01 F3          JSR STOP1      ;STOP KEY
0266  083A  F0 2A             BEQ WG230      ;YES...
0267  083C  20 E4 FF          JSR $FFE4      ;GET A CHAR FROM KEYBOARD
0268  083F  F0 D0             BEQ WG250      ;NOTHING...
0269  0841  C9 20             CMP #$20       ;SPACE BAR?
```

```
0843  D0 CC              BNE WG250      ;NO...
0845  20 E4 FF   WG255   JSR $FFE4      ;ANY KEY STARTS
0848  F0 FB              BEQ WG255
084A  D0 C5              BNE WG250      ;(JMP)
084C             ;
084C  A9 0D      WG240   LDA #CR
084E  A6 BA              LDX CNTDN      ;CHECK DFLTO FOR SCREEN
0850  E0 03              CPX #3
0852  F0 05              BEQ *+7
0854  A6 B3              LDX WSW
0856  20 BC F7           JSR CHKOUT
0859  20 D2 FF           JSR BSOUT
085C  20 CC FF           JSR CLRCHN
085F  20 83 F1           JSR UNLSN
0862  A0 02              LDY #$02       ; DO TWICE
0864  D0 A3              BNE WG220B
0866             ;
0866             ;CLOSE FLOPPY AND RETURN
0866             ;
0866  20 CC FF   WG230   JSR CLRCHN
0869  A9 0E              LDA #14        ;CLOSE FLOPPY
086B  20 AE F2           JSR FCLOSE
086E  68                 PLA            ;CLEAN UP THE STACK
086F  68                 PLA
0870  4C 89 C3           JMP READY      ;RETURN "READY"

0873             ;
0873             ; LOAD A FILE
0873             ;
0873  A9 00      LOAD    LDA #0
0875  85 96              STA SATUS      ;CLEAR STATUS
0877  85 9D              STA VERCK      ;LOAD NOT VERIFY
0879  20 22 F3           JSR LD15       ;LOAD A PROGRAM
087C  A5 96              LDA SATUS
087E  29 10              AND #SPERR     ;CHECK STATUS (EOI OK)
0880  D0 28              BNE LDERR
0882  AD 84 F3           LDA $F384      ;CHECK FOR (-04) ROM
0885  30 06              BMI LOAD1      ;NOT (-04)....
0887  E6 C9              INC EAL        ;FIX THE LOAD (-04) ROM
0889  D0 02              BNE LOAD1
088B  E6 CA              INC EAH
088D  A5 CA      LOAD1   LDA EAH        ;SET BASIC'S POINTERS
088F  85 2B              STA VARTAB+1
0891  A5 C9              LDA EAL
0893  85 2A              STA VARTAB
0895  20 72 C5           JSR RUNC       ;FIX POINTERS
0898  20 42 C4           JSR LNKPRG     ;FIX LINKS
089B  A5 B3              LDA FLAG       ;CHECK FOR LOAD OR RUN
089D  C9 2F              CMP #'/        ;LOAD ?
089F  D0 03              BNE WG300      ;NO...
08A1  4C 89 C3           JMP READY      ;LOAD RETURN TO BASIC
08A4  20 A7 C5   WG300   JSR STXTPT     ;SET TXTPTR FOR RUN
08A7  4C C4 C6           JMP NEWSTT     ;RUN PROGRAM
08AA  4C E6 F3   LDERR   JMP LD209      ;PRINT "LOAD ERROR"
08AD             CMDEND
```

```
0326  08AD              ;
0327  08AD              ;THIS ROUTINE POKES TOP OF MEMORY
0328  08AD              ;DOWN RELOCATES THE PARSER AND
0329  08AD              ;SETS THE WEDGE
0330  08AD              ;
0331  08AD   A5 34      POKE    LDA MEMSIZ        ;POKE TOP DOWN
0332  08AF   18                 CLC               ;MINUS ONE
0333  08B0   E9 AD              SBC #<CMDLN
0334  08B2   85 34              STA MEMSIZ
0335  08B4   A5 35              LDA MEMSIZ+1
0336  08B6   E9 01              SBC #>CMDLN
0337  08B8   85 35              STA MEMSIZ+1
0338  08BA              ;
0339  08BA              ;MOVE THE CODE
0340  08BA              ;
0341  08BA   A0 01      MOVE    LDY #$01          ;SET UP FROM ADDR
0342  08BC   A9 00              LDA #<CMD
0343  08BE   85 C7              STA SAL
0344  08C0   A9 07              LDA #>CMD
0345  08C2   85 C8              STA SAH
0346  08C4   A5 34              LDA MEMSIZ        ;SET UP TO ADDR
0347  08C6   85 5C              STA GRBTOP
0348  08C8   A5 35              LDA MEMSIZ+1
0349  08CA   85 5D              STA GRBTOP+1
0350  08CC   B1 C7      MOV1    LDA (SAL),Y       ;RELOCATE
0351  08CE   91 5C              STA (GRBTOP),Y
0352  08D0   C8                 INY
0353  08D1   D0 F9              BNE MOV1
0354  08D3   E6 5D              INC GRBTOP+1
0355  08D5   E6 C8              INC SAH
0356  08D7   A5 C8              LDA SAH
0357  08D9   C9 08              CMP #>CMDEND
0358  08DB   F0 02              BEQ MOV2
0359  08DD   B0 04              BCS WEDGE
0360  08DF   A0 00      MOV2    LDY #0
0361  08E1   F0 E9              BEQ MOV1
0362  08E3              ;
0363  08E3              ;WEDGE INTO BASIC
0364  08E3              ;
0365  08E3   A9 4C      WEDGE   LDA #$4C          ;JUMP INSTRUCTION
0366  08E5   85 70              STA CHRGET
0367  08E7   A4 34              LDY MEMSIZ
0368  08E9   A6 35              LDX MEMSIZ+1
0369  08EB   C8                 INY
0370  08EC   D0 01              BNE WEDGE1
0371  08EE   E8                 INX
0372  08EF   84 71      WEDGE1  STY CHRGET+1
0373  08F1   86 72              STX CHRGET+2
0374  08F3   60                 RTS
0375  08F4                      .END

ERRORS = 0000
```

```
SYMBOL TABLE

SYMBOL  VALUE
 ACPTR    F18C    BASIN    FFCF    BSOUT    FFD2    BUF      0200
 BUMP     0760    CHKIN    F770    CHKOUT   F7BC    CHRGET   0070
 CHRGOT   0076    CIOUT    F16F    CLRCHN   FFCC    CMD      0700
 CMDEND   08AD    CMDLN    01AD    CNTDN    00BA    CR       000D
 DFLTO    00B0    DODIR    079A    EAH      00CA    EAL      00C9
 FA       00D4    FCLOSE   F2AE    FLAG     00B3    FNADR    00DA
 FNLEN    00D1    FOPEN    F524    GRBTOP   005C    LA       00D2
 LD15     F322    LD209    F3E6    LDERR    08AA    LINPRT   DCD9
 LISTN    F0BA    LNKPRG   C442    LOAD     0873    LOAD1    088D
 LOADB    0805    MEMSIZ   0034    MOV1     08CC    MOV2     08DF
 MOVE     08BA    NEWSTT   C6C4    NOMAIN   074B    NOTDIR   0750
 OPENI    F466    PIAK     E812    POKE     08AD    PRT      E3D8
 RDERR    0774    READY    C389    RUNC     C572    SA       00D3
 SAH      00C8    SAL      00C7    SATUS    0096    SECND    F128
 SKIPB    080B    SPERR    0010    SPMSG    F315    STOP1    F301
 STXTPT   C5A7    TALK     F0B6    TMP2     00FD    TWAIT    F8E6
 TXTPTR   0077    UNLSN    F183    UNTLK    F17F    VARTAB   002A
 VERCK    009D    WEDGE    08E3    WEDGE1   08EF    WG100    0707
 WG110    0722    WG115    073D    WG120    076E    WG130    0791
 WG140    0784    WG220    07CE    WG220B   0809    WG230    0866
 WG235B   0807    WG240    084C    WG250    0811    WG255    0845
 WG300    08A4    WG997    0748    WG998    0797    WSW      00B3

END OF ASSEMBLY
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

THE WALL STREET JOURNAL

"No! I don't want any middlemen, put me right through to your computer."
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Random Access File Indexing

For those writing programs that have random access record handling, a routine has been developed by Jim Hindson of Burlington, Ontario. The routine is basically an algorithm that will convert a record number into the location of the record within the file.

2040 Disk   Jim Hindson

Index and Main Record locations for:

a) Index file of records at 10 records per sector
b) Main file of records at 3 records per sector

Task A - Divide available sectors into sectors to be used as the index file and sectors to be used for the main file and to obtain an equal number of each record type (index and main) on a diskette.

For 10 index records/sector and 3 main records/sector, one plan would be as follows:

Index Records

Record No.	Track No.	Sector No.	
1 - 200	1	1 - 20	(1)
201 - 400	2	1 - 20	
401 - 600	3	1 - 20	
601 - 800	4	1 - 20	
801 - 1000	5	1 - 20	
1001 - 1200	6	1 - 20	
1201 - 1400	7	1 - 20	
1401 - 1500	8	1 - 10	

Main Records

Record No.	Track No.	Sector No.	
1 - 567	9 - 17	0 - 20	
Track 18 reserved for directory			
568 - 927	19 - 24	0 - 19	
928 - 1251	25 - 30	0 - 17	
1252 - 1500	31 - 35	0 - 16	(2)

Each of the four Main Record areas will be known as track zones.

Note (1) Although sector 0 is available on tracks 1 - 8, it is not used in this example.
(2) Sector 15 & 16 of track 35 not used

Task B - Write a subroutine to convert any record number (say NR) to the track, sector and record number within the sector.

Variable Identification

NR : Number of the Record, the location of which is required
TR(1) : Index file track number for NR
TR(2) : Main file track number for NR
SN(1) : Index file sector number for NR
SN(2) : Main file sector number for NR
SR(1) : Index file record number for NR (1-10)
SR(2) : Main file record number for NR (1-3)

Z(1) - Z(4) : delimiters for the track zones which have a different number of available sectors
B1 : number of records per track (within a track zone)
A : B1-1
C : 1 less than the lowest track number in a track zone

By using this subroutine it is not necessay to carry any information on the index file about where the record is located on the main file.

Subroutine Convert

Fed NR, this subroutine will return TR(1), SN(1), SR(1) and TR(2), SN(2), SR(2) for a 1500 record file of 1500 index records at 10 records/sector and 1500 main records at 3 records/sector.

```
40500 REM *** SUBROUTINE CONVERT ***
40501 REM +++ FIND INDEX FILE LOCATION +++
40502 Z = (NR + 199)/200
40505 TR(1) = INT(Z)
40510 Z1 = NR - ((TR(1) - 1)*200)
40515 Z2 = (Z1 + 9)/10
40520 SN(1) = INT(Z2)
40525 Z3 = Z1 - ((SN(1) - 1)*10)
40530 SR(1) = INT(Z3)

40550 REM +++ FIND MAIN FILE LOCATION +++
40549 Z(1) = 567 : Z(2) = 927
40552 Z(3) = 1251 : Z(4) = 1506
40560 FOR J = 1 TO 4                       :find track
40565 IF NR - Z(J) <= 0 THEN 40576          zone
40575 NEXT J
40576 NZ = NR
40578 IF J > 1 THEN NZ = NR - Z(J-1)       :convert to number
                                            within track zone
40580 ON J GOTO 40591,40592,40593,40594
40591 A=62 : B1=63 : C=8  : GOTO 40600     :define
40592 A=59 : B1=60 : C=18 : GOTO 40600      zone
40593 A=53 : B1=54 : C=24 : GOTO 40600      parameters
40594 A=50 : B1=51 : C=30
```

```
40600 Z =(NZ + A)/B1                      find
40605 TR(2) = INT(Z)                      track,
40610 Z1 = NZ - ((TR(2) - 1)*B1)
40615 Z2 = (Z1 + 2)/3
40620 SN(2) = INT(Z2)                     sector,
40625 Z3 = Z1 - ((SN(2) - 1)*3)
40630 SR(2) = INT(Z3)                     record
40640 TR(2) = TR(2) + C                   compensate for # of
40650 SN(2) = SN(2) - 1                   tracks in lower and
                                          availabilty of
                                          sector 0.

40660 RETURN
```

Editor's Note

You may be asking, "Why an index file routine and a main file routine when the whole purpose is to do away with the index ?". The index file really doesn't do any indexing and might have been called a 'sub-main' file. Jim developed the program for his own use and found it more efficient to split each entry into 2 files: an "index" file for name and Social Insurance Number and a main file for any remaining info (address, phone #, etc.). It was anticipated that 110 characters would be required for each entry. With 255 byte sectors, this would impose a restriction of 2 entries per sector, wasting 35 bytes. The maximum would also be restricted to 2*670 (blank disk has 670 sectors) or 1340. By splitting up the entries into 25 and 85, each sector or block can filled to capacity allowing 1500 entries. This figure could also be increased as some blocks are unused.

This method of indexing has only one drawback: NR. That is, each item in the file must have a number (1, 2, 3...etc.) that may be irrelevant to the data being recorded. Therefore, access to a record requires entry of the corresponding 'NR' and in the above example NR has a range of 1 to 1500.

This would be ideal for applications such as a mailing list where each subscriber has a number, but for a inventory it becomes somewhat impractical since 'NR' will probably not be your part number. However, Jim's method is still simpler than recording disk co-ordinates. Consider this; have PET assign "NR's" to the record element that will be primarily used for record recall. For example

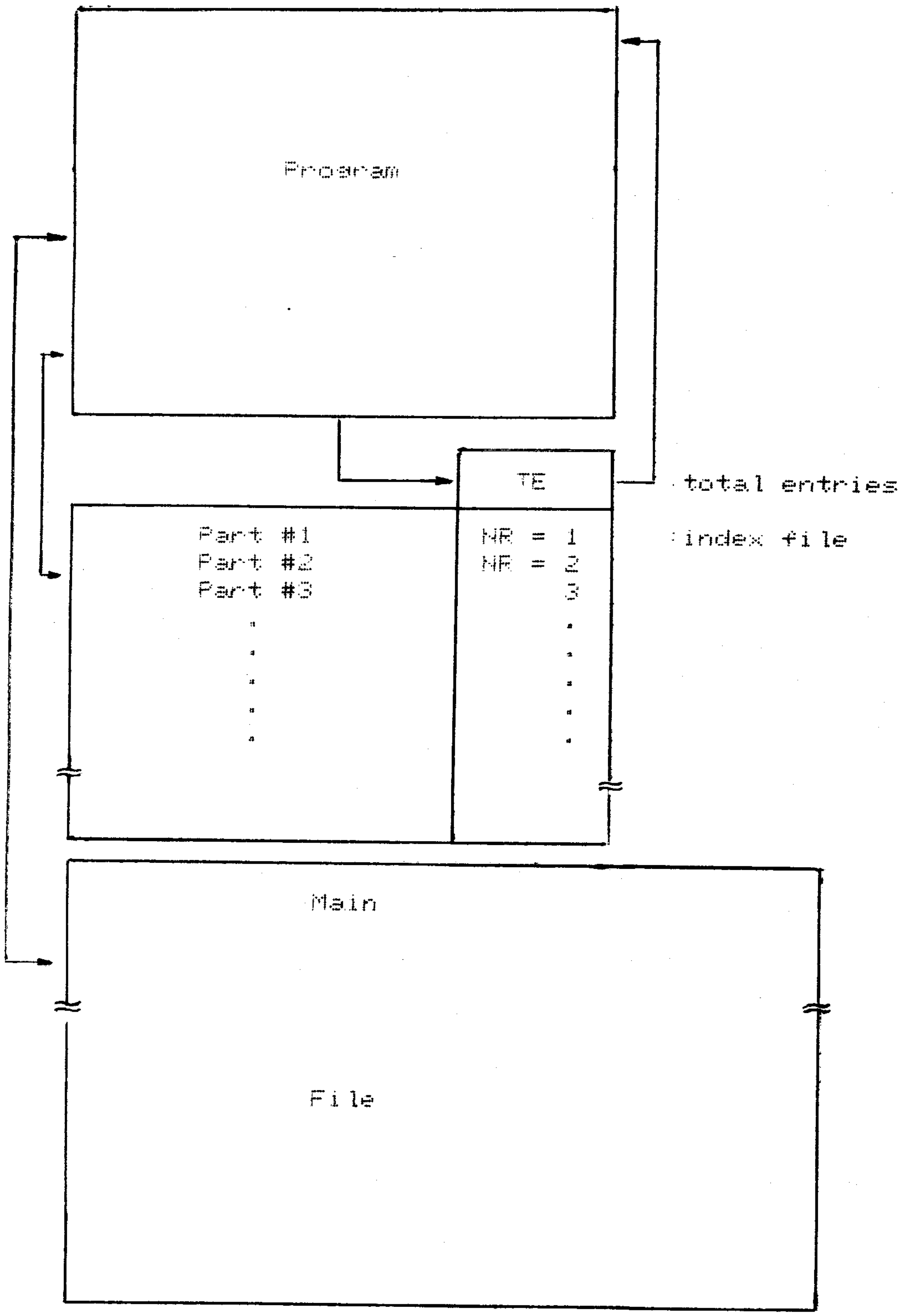
```
(Part #1) , X
(Part #2) , X+1
(Part #3) , X+2
```

...and so on. This information could be stored in an random index file along with the total number of entries (TE) so that PET would know where to start assigning new NR's to new entries.

With the desired Part # entered, the index file could be searched, NR extracted and passed into Jim's main file subroutine.

Once the track and sector co-ordinates are determined (TR(2) and SN(2)), they can now be inserted in the Block-Read command and SN(2) in the Buffer-Pointer command for rapid record access. You might also consider using Bill Maclean's Block Get routine for transfering data from disk to PET.

System layout for above:



INDEXING PROGRAMMES ON CASSETTE

by Michael Casey

At the April meeting of the Pet Users Club, I discussed a revised tape index programme which I had developed. Since then, I have had several requests to document my theory so that everyone can take advantage of it. I started thinking about this subject after reading David Wilcox's index programme in the Pet User Notes. I tried his index programme and it worked well. However, it had a few disadvantages which I was determined to eliminate:

1) it was very tedious to create an indexed cassette (the index programme had to be run each time you added a programme)
2) a lot of tape was wasted
3) his formula was too restrictive.

Items (2) and (3) were partially resolved by including an array of the programme lengths in the index programme itself, calculating the summations of these lengths for each programme and applying these summations to his formula to calculate the FFWD times. Item (1) could not be resolved without changing his entire concept which involved the splitting of the tape into FFWD time segments.

My objective, then, was to find an indexing formula which would divide the tape in terms of length rather than time. It took me quite a while to find the solution. My biggest hangup was trying to forget the notion of a "fast-forward ratio" which, by the way, is a totally insignificant and utterly meaningless term.

The description which I am about to describe is based, to some extent, on the specifications of my particular PET. Because of the different characteristics of PETs, tapes and tape drives, it may be necessary for you to adjust some of the parameters that I used. So that you will be able to do this, I have included all of the factors and all of the formulae.

Basic Factors

a) the fast-forward reel revolves at a constant rate although it may take a few jiffies to get up to normal speed and it doesn't stop on a dime.

b) I was using a C60 Realistic Supertape which contained 281' of tape or 3372". The diameter of the empty spool was .875" and the thickness of the tape was .0068". The diameter of a full spool was 2.0".

c) the total time to FFWD the entire tape including front & back leaders was 5906 jiffies (obtained through experimentation)

d) normal play speed = 1.875"/second

e) both the programme and its label are written twice on a SAVE. The length of 1 label = 192 bytes.

f) the front leader on a SAVE is 10 seconds and there is a 2 second leader between the first and second set of recordings. In addition, there is a small gap between the programmes and the labels.

g) the formula for the circumference of a circle = π X diameter

h) the formula for determining the roots of a quadratic equation is

$$\frac{-b \text{ +or-square root of } b^2 - 4ac}{2a}$$

i) the formula for the sum of an arithmetic progression is

$$\frac{m \text{ X } (a_1 + a_n)}{2}$$

where a_1 is the first term
a_n is the last term
n is the number of terms and
m is the difference between terms

Step 1 Determine the number of revolutions to wind full tape

Length of tape = $C_1 + C_2 + \ldots C_n$ where C_1 is the circumference of the empty spool, C_2 is the circumference of the spool after 1 revolution, C_2 is the circumference of the spool afer 2 revolutions... and C_n is the circumference of the spool after n revolutions.

Therefore, L = $\pi d_1 + \pi d_2 + \ldots \pi d_n$ where d_1 is the diameter of the empty spool, d_2 is the diameter of the spool after 1 revolution... and d_n is the diameter of the full spool.

After plugging in the parameters, I used the formula for an arithmetic progression and solved the resultant equation for n, using the formula for the roots of a quadratic equation.

Based on my parameters, n = 796 meaning that winding 3372" of tape requires 796 revolutions of the spool. From this I was able to calculate the FFWD time per revolution 5906 / 796 or 7.419 jiffies per revolution.

Step 2 Determine the SAVE rate in terms of bytes per jiffy

The initial formula that I started with was

$$\text{Time to SAVE} = \text{Constant (for leaders)} + \frac{\text{2 X (prog. + label lengths)in bytes}}{\text{bytes per jiffy}}$$

This equation has two unknown factors - the constant (which I knew was 10+2 seconds + the time to play the gaps between labels and programmes) and the rate.

I saved a couple of programmes of different lengths, noted the SAVE times and programme lengths in bytes and used this data in the above formula. This gave me two equations which I solved giving:

Constant = 730.15 jiffies
Rate = 1.8496890 bytes per jiffy

Step 3 Calculate length of a programme in inches of tape

Distance = Rate X Time

Using the formula in Step 2, I can calculate the length of time to SAVE any programme. Given that the rate of SAVE is 1.875" per second, the formula for determining the length of any programme in inches is:

1.875 X (730.15 + (2 X (192 + L) / 1.84968901)) / 60   where L = prog. length in bytes

Step 4 Calculate # of revolutions corresponding to Y inches of tape

Using the same formula in Step 1, I can plug in Length and solve for n.

Number of revolutions for tape length L =

-734.77912 + SQR((F X F) + 4 X (534.974599 X L) / 2) where F = 1469.58824

Step 5 Calculate FFWD time for Z revolutions

Number of revolutions in Step 4 X 7.419 (calculated in Step 1).

The formulae in Steps 3 and 4 are in the Index Programme and by using the summations of programme lengths in bytes, the FFWD times can be calculated. To use this programme, SAVE it as the first programme on the cassette with dummy entries in the DATA statements. Then simply continue to load your programmes until you run out of tape. Keep a note of the programme names and (7167 - FRE(0)) in sequence as you do this so that you can plug them into the DATA statements when you have finished saving. (include INDEX) When you have loaded all the programmes, rewind tape, load INDEX, alter the DATA statements, rewind tape and SAVE"INDEX". Then you can RUN and see if it works.
It takes several seconds to do the calculations. To eliminate this delay, you could take the calculations out of INDEX, put them in a separate programme and plug the FFWD times into an array in INDEX.
Initially, I found that I wasn't able to hit programmes near the back end of the tape and I attributed this to the fact that there is some "run on" after the programme cuts the motor switch. So I reduced the FFWD times by a factor (maximum 120 jiffies) which varies directly with the relative position of each programme. This eliminated the problem.

So there it is. I hope that you will be able to use this technique to free up a pile of cassettes and start using your over 30's more efficiently.

By the way, I am using this theory, together with some others, to develop a system which I call 'CRAMPET' - Cassette Random Access Method for the PET. The concept has several limitations but I know it can work. Its main advantage is the ability to access any "record" on a tape file without reading the "file" sequentially and, the ability to go from "record" to "record". I am having a few problems with the run-on due to the fact that the "record" lengths are relatively short. However, I think this can be ironed out. Any of you who use data files and are not planning to purchase tha disk may be interested in pursuing this idea with me. Please give me a call or write to me. Together, we may be able to achieve the ultimate: 'CRUMPET' - Cassette Random Update Method for the PET.

Michael L. Casey
BCS
6105 Yonge Street
Willowdale, Ontario
M2M 3W2
416-223-8901

*** INDEX ***

```
 1 DIML(20,3)
 2 DATA1,5.9,2,3.1,3,6.8,4,6.4,5,2.6,6,3.4,7,4.2,8,1.8,9,5.8,10,1.5
 3 DATA11,4.1,12,1.1,13,3.1,14,7.1,15,1.1,16,7.1,17,7.1,18,7.1,19,7.1,20,7.1
 10 FORI=1TO20
 11 FORJ=1TO2
 12 READL(I,J)
 13 L(I,3)=L((I-1),2)/7.1*10+L((I-1),3)
 14 L(I,3)=INT(L(I,3))
 16 NEXTJ:NEXTI
 29 PRINT"]"
 40 N$=""
 70 PRINT"THE PROGRAMMES ON THIS TAPE ARE
 80 PRINT" #  NAME        LENGTH F.FWD-SEC
 85 PRINT"___ ____        ______ _________"
 100 PRINT" 1  OSERO      ";"  ";L(1,2);"    ";L(1,3)
 105 PRINT" 2  NUMGAME    ";"  ";L(2,2);"    ";L(2,3)
 110 PRINT" 3  BLACKJACK  ";"  ";L(3,2);"    ";L(3,3)
 115 PRINT" 4  STAR TREK  ";"  ";L(4,2);"    ";L(4,3)
 120 PRINT" 5  TREND LINE";"  ";L(5,2);"    ";L(5,3)
 125 PRINT" 6  MOONLANDER";"  ";L(6,2);"    ";L(6,3)
 130 PRINT" 7  CHECKERS   ";"  ";L(7,2);"    ";L(7,3)
 135 PRINT" 8  CLOCK      ";"  ";L(8,2);"    ";L(8,3)
 140 PRINT" 9  COMDEMO    ";"  ";L(9,2);"    ";L(9,3)
 145 PRINT" A  BNSLOGO    ";"  ";L(10,2);"    ";L(10,3)
 147 PRINT" B  EXCDEMO    ";"  ";L(11,2);"    ";L(11,3)
 149 PRINT" C  TAPEDUMP   ";"  ";L(12,2);"    ";L(12,3)
 151 PRINT" D  HANGMAN    ";"  ";L(13,2);"    ";L(13,3)
 153 PRINT" E  BNSDEMO    ";"  ";L(14,2);"    ";L(14,3)
 155 PRINT" F  CREATEFILE";"  ";L(15,2);"    ";L(15,3)
 157 PRINT" G  DUMMY      ";"  ";L(16,2);"    ";L(16,3)
 159 PRINT" H  DUMMY      ";"  ";L(17,2);"    ";L(17,3)
 160 PRINT" I  DUMMY      ";"  ";L(18,2);"    ";L(18,3)
 161 PRINT" J  DUMMY      ";"  ";L(19,2);"    ";L(19,3)
 162 PRINT" K  DUMMY      ";"  ";L(20,2);"    ";L(20,3)
 164 PRINT"PRESS F.FWD KEY ON CASSETTE THEN"
 166 PRINT"ENTER # OF DESIRED PROGRAMME";"]"
 200 GETN$:IFN$<>""THEN230
 210 IFPEEK(519)=0THENPOKE519,52:POKE59411,61
 220 GOTO200
 230 N=ASC(LEFT$(N$,1))-49:PRINT"SEARCHING FOR ";N$;"                "
 231 IFN>9THENN=N-7
 240 POKE59411,53:TS=TI+L(N+1,3)*60
 250 IFTI<TSTHEN250
 260 POKE59411,61
 270 PRINT"]NPRESS 'STOP' ON CASSETTE AND LOAD"
 280 PRINT"SELECTED PROGRAMME NORMALLY....N"
 285 PRINT"                 ORN"
 290 PRINT"SAVE NEW PROGRAMME ON THIS TAPE HERE"
READY.
```

16 spaces

(THIS IS A MODIFIED VERSION OF DAVID WILCOX'S INDEX PROGRAMME)

TAPE INDEX

```
1 DIML(36,5)
2 DIMP$(36)
5 PRINT"■■■■■■■■■■■■■■■■■■■■■■*CREATING INDEX TABLE*"
9 L(0,1)=0.0:L(0,2)=0:L(0,3)=0:L(0,4)=0
10 FORI=1TO36
11 READX:READX
12 L(I,1)=X
13 L(I,2)=1.875*(730.15+(2*(192+X)/1.84968901))/60
15 NEXTI
20 FORI=1TO36
25 L(I,3)=L((I-1),3)+L(I,2)
27 B=1469.58824
30 L(I,4)=(-734.77912+SQR(B↑2+4*534.974599*L(I,3))/2)
31 L(I,5)=INT(L(I,4)*7.419-L(I,3)*120/3372)
40 NEXTI
60 PRINT"■"
65 N$=""
70 PRINT"THE PROGRAMMES ON THIS TAPE ARE:";"■"
80 PRINT" #  NAME           BYTES INCH  REV  JIFFY
85 PRINT"——  ————           ————— ————  ———  —————"
86 GOTO1000
164 PRINT"PRESS F.FWD KEY ON CASSETTE THEN"
166 PRINT"ENTER # OF DESIRED PROGRAMME";"■■"
200 GETN$:IFN$<>""THEN230
210 IFPEEK(519)=0THENPOKE519,52:POKE59411,61
220 GOTO200
230 N=ASC(LEFT$(N$,1))-49:PRINT"SEARCHING FOR PROGRAMME #";N$;"          "
231 IFN>9THENN=N-7
235 PRINT"FFWD TIME =";"     ■■■■■";INT((L(N,5)-L(N,1))/60);"SECDS          ";"■
"
243 POKE59411,53:TS=TI+L(N,5)-L(1,5)
250 IFTI<TSTHEN250
260 POKE59411,61
270 PRINT"■■PRESS 'STOP' ON CASSETTE AND LOAD"
280 PRINT"SELECTED PROGRAMME NORMALLY....■"
300 END
500 DATA01,2618,02,0502,03,1558,04,1537
502 DATA05,1359,06,1403,07,1545,08,2370
504 DATA09,2401,10,1525,11,1503,12,1425
506 DATA13,6700,14,5308,15,1724,16,5730
508 DATA17,2244,18,5491,19,6783,20,1772
510 DATA21,4823,22,5056,23,2858,24,5845
512 DATA25,2565,26,0669,27,0669,28,0000
514 DATA29,0000,30,0000,31,0000,32,0000
516 DATA33,0000,34,0000,35,0000,36,0000
551 DATA1,"TAPE-INDEX  "
552 DATA2,"CREATEFILE  "
553 DATA3,"CREATE14/2  "
554 DATA4,"READFILE15/2"
555 DATA5,"READFILE10/2"
556 DATA6,"READ11/2/1  "
557 DATA7,"READ11/2/2  "
558 DATA8,"YAHTZEE#1   "
559 DATA9,"YAHTZEE#2   "
560 DATA10,"KCDRAW#1    "
561 DATA11,"KCDRAW#2    "
562 DATA12,"FLYING-S    "
563 DATA13,"BNSDEMO     "
564 DATA14,"BLACKJACK   "
565 DATA15,"TIME        "
```

```
 566 DATA16,"COMDEMO     "
 567 DATA17,"TDCOMBO     "
 568 DATA18,"PETDEMO     "
 569 DATA19,"TDDEMO      "
 570 DATA20,"TD-FAE      "
 571 DATA21,"HORSERACE   "
 572 DATA22,"OSERO       "
 573 DATA23,"NUMGAME     "
 574 DATA24,"STARTREK    "
 575 DATA25,"TRENDLINE   "
 576 DATA26,"LOBLOGO#1   "
 577 DATA27,"LOBLOGO#2   "
 578 DATA28,"NO          "
 579 DATA29," MORE       "
 580 DATA30,"  SPACE     "
 581 DATA31,"   ON       "
 582 DATA32,"    SIDE    "
 583 DATA33,"     #1     "
 584 DATA34,"      OF    "
 585 DATA35,"       CASS."
 586 DATA36,"        BCSX"
 1000 FORI=1TO36:READP,X$:P$(I)=X$:NEXT
 1002 FORI=1TO36
 1005 J=I+48:IFI>9THENJ=J+7
 1010 J$=CHR$(J)
 1011 IFI>1GOTO1015
 1012 PRINT"                                      ┐"
 1013 PRINTJ$;"   ";P$(I);INT(L(I,1));INT(L(I,2));" ";INT(L(I,4));" ";L(I,5)
 1014 GOTO1020
 1015 PRINT"                                      ┐"
 1016 PRINTJ$;"   ";P$(I);INT(L(I,1));INT(L(I,2));INT(L(I,4));L(I,5)
 1017 GOTO1020
 1020 IFI<>18GOTO1040
 1022 REM***Q$<>"="WILL SCROLL PROGS. ON SCREEN***
 1025 GETQ$:IFQ$=""GOTO1025
 1026 IFQ$<>"="GOTO1055
 1027 PRINT"SQQQQQ";
 1037 Q$=""
 1040 NEXT
 1050 GETQ$:IFQ$="="THENQ$="":GOTO1060
 1051 IFQ$=""THENGOTO1050
 1055 GOTO164
 1060 PRINT"SQQQQQ";:RESTORE:GOTO1002
READY.
```

PRELIMINARY REPORT ON THE ON-SITE USE OF A MICROCOMPUTER FOR ARCHAEOLOGICAL FIELDWORK

EAST KARNAK EGYPT 1979

© by: G.D. Hathaway , P.Eng.
85 Alcorn Ave.
Toronto, Ontario
M4V 1E5
(416) 923 8586

ABSTRACT

The summer season of 1979 at East Karnak witnessed the first on-site use of a microcomputer for all aspects of archaeologocal fieldwork and report preparation. The purpose of the present paper is to highlight some of the more important operations that are able to be carried out by computer, as well as give a summary of the summer season's work.

****

The concept of using computers in archaeology is not new. Significant advances in the use of large scale computers has been taking place for the last dozen years in North America, Great Britain, Europe, and Israel. It was noted early that a computer's ability to handle large amounts of data with unmatched speed and efficiency would be a boon to the social and historical sciences and archaeology in particular. Thus there have been many recent articles in the literature devoted to this idea.

However, when one examines in more depth the systems referred to, one sees that very few of them have even arrived at the stage of feeding the site data into the machine. The constraints and restrictions placed on archaeologists preventing them from gaining access to computer facilities prompted the present author (G.D.H.) to attempt to rectify the situation.

Late in 1978, the author approached Professor Donald B. Redford of the Dept. of Near Eastern Studies, University of Toronto, for assistance and a testing ground for a proposed computer scheme. It was decided to put together a preliminary program of activities for field trials at E. Karnak the following summer. If such a scheme would work in the heat of the Egyptian day, it would likely work anywhere.

It was determined that a totally portable computer was required for at least the following six purposes:

- site artifact description and recording for long term storage
- physical site structures (features and loci/strata) recording
- ability to review, alter, or delete any site data already recorded
- ability to sort or partition all or part of this data by means of specific requests
- ability to perform basic statistical analyses on all or part of the data
- ability to store schematic representations of the site plans for future retrieval and use

A comprehensive data base management approach was therefore needed. A microcomputer was the only machine that offered the necessary characteristics and was truly portable.

The overall design objective was that the system would be used as the exclusive tool of the archaeologist. It could be taken on site and offered a complete , comprehensive system of data management. All other attempts to date had been only piecemeal.

The proposed program of activities carried out were as follows:

Task 1 - specification and procurement of microcomputer and associated peripheral devices

Task 2 - writing and testing of germinal computer programs for the aforementioned purposes

Task 3 - transporting the computer intact to E. Karnak from Toronto and back again

Task 4 - setting up and field testing the computer itself including performing test runs on actual site data

TASK 1. SPECIFICATION AND PROCUREMENT OF MICROCOMPUTER

A low power, low cost, compact, portable computer was required for the job. In addition, the machine needed to have a relatively large memory storage capacity, operate with various peripheral devices (e.g. printer and mass storage devices), as well as posess the ability to handle graphic characters. The P.E.T. 2001 Microcomputer by Commodore offered the optimal combination of these factors as well as having a very powerfull and fast programming language.

The mass storage device chosen was a dual flexible disk system, and the printer was designed to print a wide range of graphic characters and symbols in addition to the standard alphabetic and numeric characters (diag. 1).

TASK 2. WRITING AND TESTING OF GERMINAL PROGRAMS

This section describes the computer programs that were written to encompass the six purposes outlined previously.

The stipulations of archaeological fieldwork required that the computer be taken directly onto the site or at least close enough so that data from the site books or the physical artifacts themselves could be fed into the machine directly. This mode of operation had the additional advantage that the director could oversee and verify the data immediately upon its entry into the computer.

Another requirement was that no voluminous, complicated coding forms would be allowed. Data had to be typed intothe computer's memory directly from the keyboard. This was accomplished by developing in advance the typography for all types and descriptions of artifacts. In this paper, artifact will be used to denote both site artifacts, e.g. pottery, coins bones, small finds etc. as well as physical features, e.g. walls, pits, strata/loci etc. The typography is in the form of a table or 'Typography Chart' (diag. 2) which is resident inside the computer. It represents the most likely set of artifacts and their attributes to be expected on the site, based on previous excavation or the director's judgement. Since the table can be altered to suit any site, this set of programs is truly universal.

It is this table that serves to drive the data input section of the program. This input section allows even unskilled operators to handle efficient data entry. The operator simply specifies the artifact which he or she is about to describe and the computer responds by presenting the operator with a set of multi-choice questions. These questions correspond to each of the attributes found on the Typography Chart for the selected artifact. The operator simply answers the questions one by one by pressing single keys on the computer keyboard. This is done sequentially until the attributes are complete for that artifact. At this point the computer automatically stores all the descriptions of the artifact on its mass storage disks and waits for the next artifact to be specified. This allows any artifact to be entered at any time and obviates the need for grouping, for example, all pots together and entering them at once. The method corresponds most closely to the way data is recorded by site supervisors in their site books.

After the day's data has been entered into the mass memory disks, any number of requests may be made of it, or any data stored previously. The director can ask to see, for example, all green glazed rim sherds from feature 25A ; stratum 15. The computer responds by printing out complete descriptions of all pots corresponding to this request. In this way, the director can perform his searching and sorting on all the data for that season before the dig ends and thus save weeks of work. This is one of the more important and unique features of this set of programs.

In order to revise and update the data already in memory, a comprehensive scheme of editing is included in the programs. If it is found that an artifact has been improperly entered, the operator can retrieve that datum by means of its description or number, and alter it in any manner desired.

Once the data has been entered to the satisfaction of the director, the sorted and partitioned data set created by his requests can be subjected to numerous statistical analyses. These include the simple stats such as frequency histograms, averages, standard deviations, correlation coefficients etc., as well as more sophisticated techniques such as analysis of variance, seriation, multi-variate regression, and eventually discriminant analysis and cluster analysis. These last are included at the discretion of the director and must be of a size small enough to fit onto the microcomputer used.

Another unique feature of the P.E.T. computer is its ability to draw schematized representations of the site plans on its screen and print them on paper (diag. 1) . These are particularly useful when new features are discovered and areas of the site need to be highlighted. The director can hand these to his respective site supervisors with pertinent information pinpointed.

A future version of the programs will include the ability to show automatically on the proper site schematic the responses to the archaeologist's requests. For example, at the archaeologist's request, all green glazed rim sherds could be plotted automatically on the correct site plan without human intervention.

The data input, basic stats and site plan drawing sections of the program were completed before travelling to Egypt, as was the format for the Typography Chart. This chart was then altered while in Egypt to suit the E. Karnak site artifact distributions. Work is now almost complete on the remaining sections of the programs - data requests and retrieval, data alteration, and advanced statistics. By Autumn 1979 final testing will take place in Toronto on data brought back from E. Karnak.

TASK 3. TRANSPORTING THE COMPUTER INTACT

This was the part of the program of activities that held the greatest potential for trouble, but it all went quite smoothly. The computer was dismantled into 4 small sectional components: Printer, TV screen (CRT), Main Computer and Keyboard, and Disk Drives (mass storage). These components were packed in sturdy boxes, the whole package taking up less than 4 cu.ft. The disk drives and CRT were put on the airplane as cargo and the computer and printer were taken as hand luggage. Disks with programs recorded on them were packed inside the main computer for protection from magnetic fields and X-rays at airport security.

All customs checkpoints were passed with no interruptions or delays. The computer system was carried by two people (G.D.H. and wife) from Canada to New York, to London, to Cairo, to Luxor, to Karnak and still worked perfectly. The printer was adjusted for operation at 50 Hz. and the whole system was powered by a variable rheostat to enable constant 120 V.AC operation from varying (and intermittent) Egyptian 220 V.AC mains current.

TASK 4. FIELD TESTING

The system was installed on a table in the office of the Inspector of Karnak Antiquities, Sayed Abdul Hamid, to whom we owe a debt of gratitude. It was discovered that air conditioning was required to keep the system cool. Fortunately, the inspector's office offered the necessary air conditioning and a relatively dust-free atmosphere. This was especially important for the delicate disk drives which are notoriously susceptible to dust and dirt. Thus each night, special plastic bags were put around the disks and disk drives.

In actual operation, every afternoon the voltage was adjusted and the computer system turned on. The morning's site books or artifacts were brought to the operator for entry. At the end of the computer session, with all the data recorded and requests made, the machine was turned off and enclosed in the bags. All systems and modes operated successfully as designed.

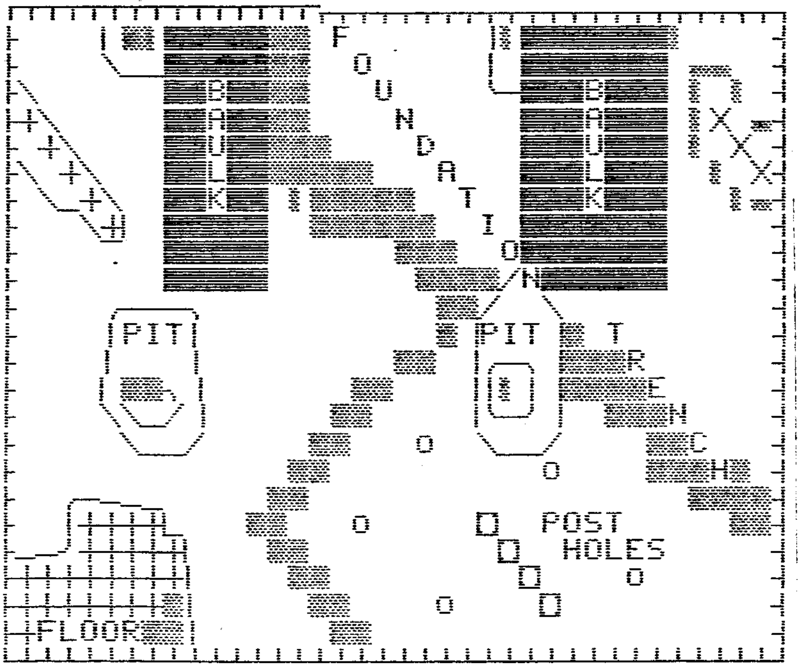
Actual test data from the site were used to demonstrate the usage of the programs, and suggest alterations to both modes of data entry and expected typography of the site. These tests as well were carried out to the satisfaction of the director and the operators.

SUMMARY

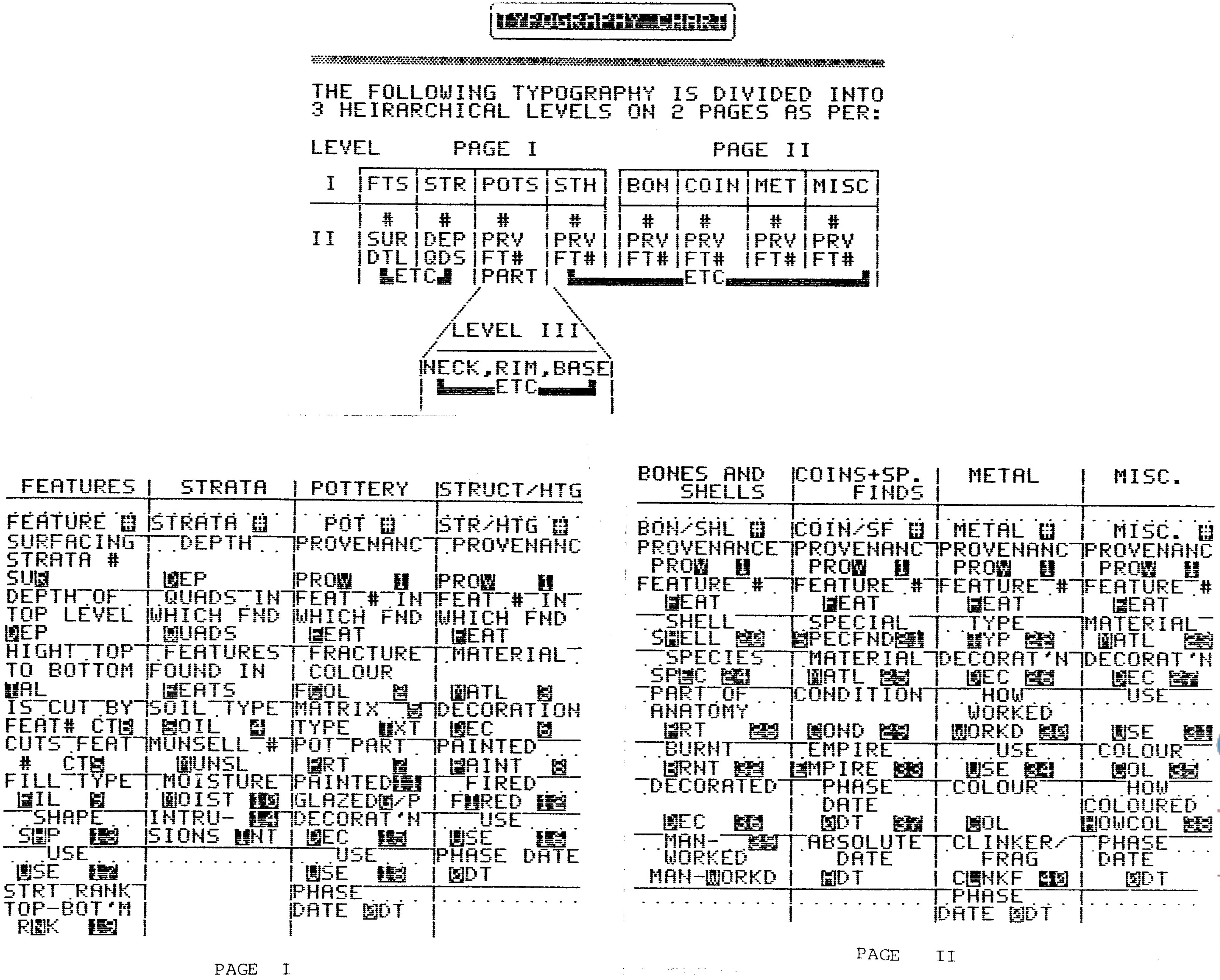
For under $5000 (Can.) archaeologists are now able to purchase a computer system capable of operating a comprehensive set of programs dedicated to fieldwork and analysis. The programs offer a complete data base management system for handling both physical site features and artifacts, as well as portraying graphical representations of the site. An additional feature is the consultation service offered by the author to prospective users, enabling a typography to be created which is particular to each individual site.

The summer season of 1979 at E. Karnak offered an ideal, if harsh, testing ground and proved that such a system was viable. It is hoped that the savings to archaeologists in terms of both fieldwork and report preparation will prompt further interest in and elaboration of the programs, and promote their use.

July 1979



DIAG. 1 Hypothetical site printed by P.E.T. printer



```
TYPOGRAPHY CHART

THE FOLLOWING TYPOGRAPHY IS DIVIDED INTO
3 HEIRARCHICAL LEVELS ON 2 PAGES AS PER:

LEVEL       PAGE I              PAGE II

 I   |FTS |STR |POTS |STH | |BON |COIN |MET |MISC|
     |  # |  # |  #  |  # | |  # |  #  |  # |  # |
 II  |SUR |DEP |PRV  |PRV | |PRV |PRV  |PRV |PRV |
     |DTL |QDS |FT#  |FT# | |FT# |FT#  |FT# |FT# |
     | ▬ETC▬   |PART |    | ▬▬▬▬▬▬▬▬ETC▬▬▬▬▬▬▬▬▬ |
                 LEVEL III
             |NECK,RIM,BASE|
             | ▬▬▬ETC▬▬▬   |


 FEATURES  |  STRATA    | POTTERY    |STRUCT/HTG
FEATURE ▨  |STRATA ▨    |  POT ▨     |STR/HTG ▨
SURFACING  |  DEPTH     |PROVENANC   |PROVENANC
STRATA #   |            |            |
SUR        | DEP        |PROV    ▨   |PROV    ▨
DEPTH OF   | QUADS IN   |FEAT # IN   |FEAT # IN
TOP LEVEL  |WHICH FND   |WHICH FND   |WHICH FND
DEP        | QUADS      | FEAT       | FEAT
HIGHT TOP  | FEATURES   | FRACTURE   | MATERIAL
TO BOTTOM  |FOUND IN    | COLOUR     |
▨AL        | FEATS      |FCOL    ▨   | MATL  ▨
IS CUT BY  |SOIL TYPE   |MATRIX   ▨  |DECORATION
FEAT# CT▨  | SOIL  ▨    |TYPE  ▨XT   | DEC   ▨
CUTS FEAT  |MUNSELL #   |POT PART    |PAINTED
 #  CT▨    |  MUNSL     | PRT    ▨   | PAINT  ▨
FILL TYPE  | MOISTURE   |PAINTED▨    |  FIRED
 FIL  ▨    | MOIST ▨    |GLAZED▨/P   | FIRED ▨
  SHAPE    |INTRU-  ▨   |DECORAT'N   |    USE
 SHP  ▨    |SIONS ▨NT   | DEC   ▨    | USE   ▨
    USE    |. . . . . . |     USE    |PHASE DATE
 USE  ▨    |            | USE   ▨    | PDT
STRT RANK  |            |PHASE       |. . . . . .
TOP-BOT'M  |            |DATE PDT    |
 RNK  ▨    |            |            |

                 PAGE  I


BONES AND   |COINS+SP.  |   METAL     |  MISC.
  SHELLS    |    FINDS  |             |
BON/SHL ▨   |COIN/SF ▨  | METAL ▨     |  MISC. ▨
PROVENANCE  |PROVENANC  |PROVENANC    |PROVENANC
 PROV  ▨    | PROV  ▨   | PROV   ▨    | PROV   ▨
FEATURE #   |FEATURE #  |FEATURE #    |FEATURE #
  FEAT      |  FEAT     |   FEAT      |  FEAT
  SHELL     | SPECIAL   |   TYPE      |MATERIAL
 SHELL ▨    |SPECFND▨   |  TYP ▨      | MATL  ▨
  SPECIES   | MATERIAL  |DECORAT'N    |DECORAT'N
 SPEC ▨     | MATL ▨    |  DEC ▨      |  DEC ▨
 PART OF    |CONDITION  |   HOW       |    USE
 ANATOMY    |           |  WORKED     |
  PRT    ▨  | COND ▨    | WORKD ▨     |  USE   ▨
  BURNT     | EMPIRE    |     USE     | COLOUR
  BRNT ▨    |EMPIRE ▨   |  USE ▨      |  COL ▨
 DECORATED  |  PHASE    | COLOUR      |   HOW
            |  DATE     |             |COLOURED
  DEC   ▨   |  PDT  ▨   |  COL        |HOWCOL ▨
  MAN-   ▨  | ABSOLUTE  | CLINKER/    | PHASE
  WORKED    |   DATE    |    FRAG     | DATE
 MAN-WORKD  |  ADT      | CLNKF ▨     |  PDT
. . . . . . |. . . . . .| PHASE       |. . . . . .
            |           |DATE PDT     |

                 PAGE  II
```

DIAG. 2   Hypothetical Typography Chart (2 pages). Note: 3rd heirarchical level omitted