OSItems

DECEMBER 1980

VOLUME II NO. 12

Base Conversions

Peter Schreiber
1609 Washington Avenue
Seaford, N.Y. 11783

Here is a short base conversion routine which will solve almost all of your paper and pencil problems of conversion. Execution of the program is as follows:

(1) Input the base of the number that you wish to convert. This must be between 2 and 36.

(2) Input the number in that base including any decimal part. When using a base larger then 10, you must use letters for digits.
Warning: (a) If you use a number that has a decimal form too large to store, the computer will do a lot of rounding off, or even give an OM error.
(b) If you try to use a digit that is not in that base, you will get the appropriate error.

(3) Input the output base that you wish the number to be converted to, again it must be between 2 and 36.

(4) If you had a number which had a decimal part, you must input to how meny places you want the conversion printed. This is necessary in the case of repeating decimals.

```
LIST
 100 FORX=1TO18:PRINT:NEXTX:PRINTSPC(14)"- BASE -":PRINT:PRINT
 110 PRINT"USE LETTERS FOR DIGITS LARGER THEN 9."
 120 FORX=1TO4:PRINT:NEXTX:CLEAR
 130 INPUT"INPUT BASE (2-36)  ";B:IFB<>INT(B)ORB<2ORB>36THEN130
 140 INPUT"INPUT BASE NUMBER  ";X$:L=LEN(X$):M=L
 150 INPUT"OUTPUT BASE (2-36) ";S:IFS<>INT(S)ORS<2ORS>36THEN150
 160 FORI=1TOL:V=ASC(MID$(X$,I,1))-48:IFV>16THENV=V-7
 170 IFV=-2ANDH=0THEND=0:H=1:M=I-1:GOTO160
 180 IFV=-2ANDH=1THENV=0:H=2:M=M+1
 190 IFV<0ORV>=BTHENPRINT" ? '"X$"' IS NOT IN BASE"B:GOTO120
 200 D=D+V*B^(M-I):NEXTI:E=D:M=LEN(STR$(INT(E)))
 210 C=INT(D/S):X=D-C*S:IFX>9THENX=X+7
 220 A$=CHR$(X+48)+A$:D=C:IFCTHEN210
 230 IFHTHENINPUT"ANSWER TO HOW MENY DECIMAL PLACES";J
 240 PRINT:PRINT"ANSWER:      "X$"("B") = "A$;:IFH=0THEN280
 250 Y$=RIGHT$(STR$(E),LEN(STR$(E))-M):Y=VAL(Y$):PRINT".";
 260 C=Y*S:X=INT(C):Y=(C*1E4)-(X*1E4):Y=Y/1E4:IFX>9THENX=X+7
 270 PRINTCHR$(X+48);:K=K+1:IFYTHENIFK<=JTHEN260
 280 PRINT"("S")":PRINT:INPUT"ANOTHER";A$:IFA$="YES"THEN120
 290 END
OK
```

MYSTERY PROGRAM

by M. Montalvo

```
10 FOR M=1 TO 23
110 READ CM
120 D$ = D$ + CHR$(CM): D = 53765
130 NEXT M
140 FOR X= 1 TO 32 : PRINT: NEXT
150 R=INT (1000 * RND (1000))
160 IF R=100 THEN 190
170 IF R < = 255 THEN PRINT CHR$ (R),
180 GOTO 150
190 FOR X=1 TO 32: PRINT: NEXT
200 GOSUB 1000
210 FOR T=0 TO 1000
220 NEXT T
230 GOTO 140
250 DATA 72,69,76,80,32,73
260 DATA 44,86,69,32,71,79
270 DATA 78,69,32,66,69,82
280 DATA 83,69,82,75,33
300 END
1000 FOR V = 1 TO LEN (D$): POKE D+V, ASC(MID$(D$,V,1)): NEXT:RETURN
```

The above falls into the category of short "cute" programs with an interesting twist or two thrown in. It might be worthwhile "decoding" the message before running the program and for those of you who have not come across it previously the simulated "print at" routine is a worthwhile addition to your subroutine library. My apologies to all the "ace" bit-hackers in the audience but this one was meant for the beginners in the crowd. Those beginners should tease it apart and make use of the various components .

Have Fun!

OS-65U UTILITY CORRECTIONS

by Dan Schwartz

Here is a list of errors and corrections I have discovered over the course of the last two months of working with the OS-65U system. This should , I hope, make up fr the spurious list of "memory locations for OS-65U" published two issues back (they were for ROM Basic only-and it was an honest mistake.)

1) In the program PACKER, line 390 is referenced but does not actually exist in the program, causing a US error when the program is run. Adding the line 390 REM will eliminate this problem. Note: as this program runs very close to the memory limit of a 32k system, do not put any explanatory comments on this line--just leave it as a simple 390 REM.

2) In the program FPRINT, there are two errors in the random file print section of the program. The first is that when the program asks for the last file index to print, it tells you that 0 OUTPUTS TO END OF FILE, but in fact entering zero causes a repeat of the input prompt. This can be corrected by changing line 550 from

```
550 IF FR<0 OR FR<SR GOTO 540
```

to

```
550 IF FR<SR AND FR<>0 GOTO 540
```

and adding

```
725 IF FR=0 THEN FR=1E9
```

The second, and more serious problem causes the file index to be set wrong when inputting the various fields of the random-file records, so that the wrong data altogether may be input and printed if your file contains data in between the valid fields in the record. To correct this, line 800 should be changed from

```
800 Y=INDEX(1)
```

TO

```
800 Y=RL*(X-1)+RO
```

3) In the program CHANGE, all output of disk data by the program is always given in hex, even if decimal mode was selected at the start of the run. This can be corrected by adding the line

```
695 if Q=10 THEN PRINT DA;:GOTO 710
```

Another problem with this program involves what happens if you attempt to change track zero on your disk. Track zero on the A drive can be changed, but if you try to change track zero on the B (or C or D) drive, the modified track will be written to the A drive instead.

This can be corrected by adding the line

505 DE=DS

4) In the program MULTI, quotation marks were somehow omitted from one of the string assignment statements. Line 370 should be changed from

370 IF S$<>"OK" THEN PRINT "EXPECTED OK":S$=OK

to

370 IF S$<>"OK" THEN PRINT "EXPECTED OK":S$="OK"

5) In the program EDITOR, if an attempt is made to run the program on a video system, the program (rightly) informs the user that it is only for use on serial systems, and erases itself. Before doing so, though, it will have "locked" the system, and the system will remain locked when the program is gone. Since the method used to lock the system is different than the one used by BEXEC*, running BEXEC* and typing UNLOCK will not unlock it, either. To avoid this problem, lines 2140 and 2150 should be changed from

2140 PRINT "THIS PROGRAM REQUIRES A SERIAL SYSTEM":NEW
2150 REM

to

2140 PRINT "THIS PROGRAM REQUIRES A SERIAL SYSTEM"
2150 POKE 2073,CC:POKE 2888,IM:NEW

6)In the program NECDRV, the program will not give the correct current values for lines per page and lines skipped between pages if an attempt is made to change these values on a video-based system (it will give both values as zero). To correct this, line 10070 should be changed from

10070 ON IO GOTO 10090,10170 : REM SERIAL/VIDEO

to

10070 ON IO GOTO 10090,10130 : REM SERIAL/VIDEO

To make this list comprehensive, I should point out that there is one more program with a serious error, namely FDUMP. When a track boundary is crossed while printing a file, this program gets all screwed up and prints garbage. Unfortunately, I haven't been able to discover the cause of this problem yet. If anyone knows the answer, please tell OSI-tems (and me!) about it. Another thing to watch out for when using this program is that some of the charcters the program tries to print will not actually be printed by the 540 video driver, and this can cause the following characters on the same output line to appear in the wrong column on the screen, so that what you think is at INDEX 1506 may actually be at INDEX 1507.

Extending OSI BASIC-IN-ROM: Version II

Peter Schreiber
1609 Washington Avenue
Seaford, N.Y. 11783

In the October issue of OSI-tems (V.II, No.8, p.7), I presented a machine language routine to extend your OSI Basic-In-Rom. Well, you learn new things everyday and so there came to be, version II. In this version, I have improved on the previous instructions and added one new routine. In improving the program, I have changed the syntax of the instructions because of some format problems, but I don't think that this should be too confusing. Also in making these changes, the machine code has been shifted which unfortunately means, if you want to use this version, you will have to retype the entire program.

Also in this version, I tried to account for any syntax errors you might make in typing in the extended instructions. I think I have covered most of them, but every once in a while an incorect instruction will do some nasty things, like wiping out your current basic program. Try to get the extended instructions typed in corectly, as I will show.

This version has three extended instructions. The first is, again, the machine language screen clear. The revised syntax of this instruction is as follows:

```
FNCLEAR          (in immediate mode); or

10 FNCLEAR       (in a program)
```

I like to read these instructions as "function clear", since it is somewhat of a function. This single instruction will produce a super fast screen clear, which can save you time and Basic code over other routines. The second extension is, again, the "PRINTON" instruction. There are now two types, and they are as follows:

```
(1)      FNPRINT"message"ONaddr                ; or

         10 FNPRINT "message" ON addr

(2)      FNPRINT ANY$ ON addr                  ; or

         10 FNPRINT ANY$ ON addr
```

In the string version of this instruction, there is no limit as to the name of the string variable you use. It can be any string form AA$ to ZZ$. But if you try to use a string that has not been defined, you will get no output to the screen and the machine routine will just pass the extended instruction up. You will also get the same result if you put the two quotes one after the other in the first example of this instruction. One other thing you might want to know is that you can use the "?" in replace of "PRINT" because Basic will still change it to its token. And last, remember, the address that you use must be a four(4) digit hex number or the routine will abort.

The third and newest extended instruction is the LET instruction, which is used as follows:

```
FNLET K = num      (in immediate mode); or

10 FNLET K = num   (in a program)
```

The purpose of this extended LET instruction is to change any four digit hex number to its equivalent decimal number. For example, if you type:

```
1Ø FNLET I = DØØØ
2Ø PRINT I
3Ø END
RUN
 53248
OK
```

The machine language routine will convert the hex number DØØØ to 53248 in decimal and place the result in the variable I, or which ever variable name you want to use.

Remember, before you try to use this code you must add that jump in the parser routine at hex ØØBC. The jump you must make in this version will be as follows:

Location	Manchine Language	Mnemonics
ØØBC	4C EØ 1E	JMP $1EEØ
ØØBF	EA EA EA	NOP

Again, I must thank the authors of the articals in OSI-tems, without which I could not do it. I hope we can have more authors in the future.

Location	Machine Language	Mnemonics
1EEØ	E6 C3	INC $C3
1EE2	DØ Ø2	BNE $1EE6
1EE4	E6 C4	INC $C4
1EE6	AØ ØØ	LDY #$ØØ
1EE8	B1 C3	LDA $C3,Y
1EEA	C9 9E	CMP #$9E
1EEC	FØ Ø3	BEQ $1EF1
1EEE	4C C5 ØØ	JMP $ØØC5
1EF1	E6 C3	INC $C3
1EF3	DØ Ø2	BNE $1EF7
1EF5	E6 C4	INC $C4
1EF7	B1 C3	LDA $C3,Y
1EF9	C9 9A	CMP #$9A
1EFB	FØ 11	BEQ $1FØE
1EFD	C9 87	CMP #$87
1EFF	FØ 26	BEQ $1F27
1FØ1	C9 97	CMP #$97
1FØ3	FØ 4D	BEQ $1F52

Location	Machine Language			Mnemonics
1FØ5	C6	C3		DEC $C3
1FØ7	1Ø	Ø2		BPL $1FØB
1FØ9	C6	C4		DEC $C4
1FØB	4C	C2	ØØ	JMP $ØØC2
1FØE	A2	ØØ		LDX #$ØØ
1F1Ø	86	FE		STX $FE
1F12	A9	DØ		LDA #$DØ
1F14	85	FF		STA $FF
1F16	A9	2Ø		LDA #$2Ø
1F18	91	FE		STA $FE,Y
1F1A	C8			INY
1F1B	DØ	FB		BNE $1F18
1F1D	E6	FF		INC $FF
1F1F	E8			INX
1F2Ø	EØ	Ø8		CPX #$Ø8
1F22	DØ	F4		BNE $1F18
1F24	4C	BC	ØØ	JMP $ØØBC
1F27	2Ø	BC	ØØ	JSR $ØØBC
1F2A	2Ø	ØB	AD	JSR $ADØB
1F2D	85	97		STA $97
1F2F	84	98		STY $98
1F31	2Ø	C2	ØØ	JSR $ØØC2
1F34	C9	AB		CMP #$AB
1F36	FØ	Ø3		BEQ $1F3B
1F38	4C	ØC	AC	JMP $ACØC
1F3B	2Ø	C4	1F	JSR $1FC4
1F3E	A5	EC		LDA $EC
1F4Ø	85	AD		STA $AD
1F42	A5	E8		LDA $EB
1F44	85	AE		STA $AE
1F46	A2	9Ø		LDX #$9Ø
1F48	38			SEC
1F49	2Ø	E8	B7	JSR $B7E8
1F4C	2Ø	74	B7	JSR $B774
1F4F	4C	BC	ØØ	JMP $ØØBC
1F52	2Ø	BC	ØØ	JSR $ØØBC
1F55	C9	22		CMP #$22
1F57	FØ	2Ø		BEQ $1F79
1F59	2Ø	ØB	AD	JSR $ADØB
1F5C	AØ	ØØ		LDY #$ØØ
1F5E	B1	95		LDA $95,Y
1F6Ø	85	EA		STA $EA
1F62	A2	ØØ		LDX #$ØØ
1F64	E6	95		INC $95
1F66	DØ	Ø2		BNE $1F6A
1F68	E6	96		INC $96
1F6A	B1	95		LDA $95,Y
1F6C	95	E8		STA $E8,X
1F6E	E8			INX
1F6F	EØ	Ø2		CPX #$Ø2
1F71	DØ	F1		BNE $1F64

Location	Machine Language			Mnemonics
1F73	2Ø	C2	ØØ	JSR $ØØC2
1F76	4C	9D	1F	JMP $1F9D
1F79	A9	ØØ		LDA #$ØØ
1F7B	85	EA		STA $EA
1F7D	85	CC		STA $CC
1F7F	2Ø	BC	ØØ	JSR $ØØBC
1F82	A6	C3		LDX $C3
1F84	86	E8		STX $E8
1F86	A6	C4		LDX $C4
1F88	86	E9		STX $E9
1F8A	4C	92	1F	JMP $1F92
1F8D	E6	EA		INC $EA
1F8F	2Ø	BC	ØØ	JSR $ØØBC
1F92	C9	22		CMP #$22
1F94	DØ	F7		BNE $1F8D
1F96	A9	EF		LDA #$EF
1F98	85	CC		STA $CC
1F9A	2Ø	BC	ØØ	JSR $ØØBC
1F9D	C9	9Ø		CMP #$9Ø
1F9F	FØ	Ø3		BEQ $1FA4
1FA1	4C	ØC	AC	JMP $ACØC
1FA4	2Ø	C4	1F	JSR $1FC4
1FA7	A2	ØØ		LDX #$ØØ
1FA9	E4	EA		CPX $EA
1FAB	DØ	Ø3		BNE $1FBØ
1FAD	4C	BC	ØØ	JMP $ØØBC
1FBØ	B1	E8		LDA $E8,Y
1FB2	91	EB		STA $EB,Y
1FB4	E8			INX
1FB5	E6	E8		INC $E8
1FB7	DØ	Ø2		BNE $1FBB
1FB9	E6	E9		INC $E9
1FBB	E6	EB		INC $EB
1FBD	DØ	Ø2		BNE $1FC1
1FBF	E6	EC		INC $EC
1FC1	4C	A9	1F	JMP $1FA9
1FC4	A2	ØØ		LDX $ØØ
1FC6	2Ø	BC	ØØ	JSR $ØØBC
1FC9	C9	3A		CMP #$3A
1FCB	1Ø	Ø5		BPL $1FD2
1FCD	29	ØF		AND #$ØF
1FCF	4C	D5	1F	JMP $1FD5
1FD2	38			SEC
1FD3	E9	37		SBC #$37
1FD5	95	EB		STA $EB,X
1FD7	85	E7		STA $E7
1FD9	A9	ØF		LDA #$ØF
1FDB	38			SEC
1FDC	E5	E7		SBC $E7
1FDE	1Ø	Ø3		BPL $1FE3
1FEØ	4C	ØC	AC	JMP $ACØC

Location	Machine Language	Mnemonics
1FE3	E8	INX
1FE4	EØ Ø4	CPX #$Ø4
1FE6	DØ DE	BNE $1FC6
1FE8	A5 EB	LDA $EB
1FEA	ØA	ASL A
1FEB	ØA	ASL A
1FEC	ØA	ASL A
1FED	ØA	ASL A
1FEE	18	CLC
1FEF	65 EC	ADC $EC
1FF1	85 EC	STA $EC
1FF3	A5 ED	LDA $ED
1FF5	ØA	ASL A
1FF6	ØA	ASL A
1FF7	ØA	ASL A
1FF8	ØA	ASL A
1FF9	18	CLC
1FFA	65 EE	ADC $EE
1FFC	85 EB	STA $EB
1FFE	6Ø	RTS

Note: Since I did not use page two, you must save the top of memory for the program. For my 8K system I must answer 79ØØ to "MEMORY SIZE? " when starting.

```
50000 K=0:FORJ=0TO240:IFPEEK(41092+J)>127THENPOKE547+K,J+1:K=K+1
50005 NEXT:POKE546,0:POKE517,1:A=769:B=10:I=10:L=B
50010 PRINTRIGHT$(STR$(L),LEN(STR$(L))-1);
50020 P=A+4
50025 IFPEEK(P)>127THEN51000
50027 IFPEEK(P)=0THEN50050
50030 PRINTCHR$(PEEK(P));
50040 P=P+1:GOTO50025
50050 L=L+I:A=PEEK(A)+256*PEEK(A+1):IFPEEK(A+3)>194THENPOKE517,0:END
50060 PRINT:GOTO50010
51000 C=PEEK(P):S=41092+PEEK(418+C)
51010 IFPEEK(S)>127THENPRINTCHR$(PEEK(S)-123);:GOTO52030
51020 PRINTCHR$(PEEK(S));:S=S+1:GOTO51010
52030 IFC<>136ANDC<>140ANDC<>160ANDC<>144THEN50040
52035 IFC=144THEN50040
52040 P=P+1:IFPEEK(P)>57THENP=P-1:GOTO50040
52050 M=0
52060 IFPEEK(P)<48ANDPEEK(P)<>32THENP=P-1:GOTO50040
52065 IFPEEK(P)=32THENP=P+1:GOTO52060
52070 M=M*10+PEEK(P)-48:IFPEEK(P+1)>57ORPEEK(P+1)<48THEN53000
52080 P=P+1:GOTO52065
53000 IFPEEK(P+1)=32THEN52080
53002 Z=0:N=769
53005 Q=PEEK(N+2)+256*PEEK(N+3):Z=Z+1
53007 IFPEEK(N)=0ORQ>MTHENPRINT"???";:GOTO53035
53010 IFQ<MTHENN=PEEK(N)+256*PEEK(N+1):GOTO53005
53030 T$=STR$(B+(Z-1)*I):PRINTMID$(T$,2,LEN(T$));
53035 IF(C=136ORC=140)ANDPEEK(P+1)=44THENPRINT",";:P=P+1
53050 GOTO52040
OK
```

Source Update

Mike Bassman
39-65 52 st.
Woodside, NY 11377

This is a continually published list of new products for OSI computers. For a complete list, use back issues of OSI-tems to compile a complete list.

Mile High Software
318 Linden Avenue
Boulder, CO 80302
These people are advertising an Adventure, a program editor, a checking account program, and a Basic tutor series.

Bob Retelle
2005 Whittaker
Ypsilanti, MI 48197
The advertisement mentions an interesting sounding line of games, with a descriptive sheet available on request.

Chuck Lewis
804 W.
North Moravia, Iowa 52571
Selling a light pen kit and a sound board kit for the C1P. Complete hardware and software list sent for a SASE.

CEL
P.O. Box 615
New Braunfels, Texas 78130
Disassembler for the C1P. Write for details.

Small Business Systems
Star Rt (40-A-10)
Gering, NE 69341
Line of business software for C2-C3 disk systems.

Send SASE for overview, $5 for manuals.

Optimum Integrated Systems
P.O. Box 11142
Indianapolis, Indiana 46201
Advertisement lists a Super Disassembler for OS65U, and a 2K Smart Terminal Program for C4P-MF.

Listing continued next page

Source Update:continued from last page.

EIS,inc.
P.O. Box 5893
Athens, GA 30604
EIS is selling a OS65U utility called BASXR. It will search for a variable or a Basic command. Locates specific lines on entry of decimal value of Basic commands.

Gaslight Software
3820 Byron
Houston, TX 77005
Their major product is an OS65D utility diskette. It includes a memory dump, improved Create, single disk copy, Verify, etc. Catalog send for SASE.

Prism Software
Box 928
College Park, MD 20740
Prism is producing a single disk copier for 24K OSI machines. Has color and sound(!)

Universal Systems
1647 East Old Shakopee Rd.
Minneapolis, MN 55420
Universal Systems has Magic Boot, an OS65D machine language extension to Basic. Adds renumber, full curosr editing, memory pack, screen and color clear to Basic. Write for price list.

Honders, Inc.
57 North St.
Middletown, NY 10940
Machine language sort for OS-DMS users. Will sort 20,000 byte file in under ten seconds. Needs OS-65U.

Computer Systems
3763 Airport Blvd.
Mobile, AL 36608
They have a number of programs to use CW and RTTY with any CIP. Also disassembler and backspace programs.

DUAL CASSETTE RECORDER MOTORCONTROL for OSI challenger 1 P
by Klaus Ernst

For some of you this is an old hat,but since I recently had a request for the plans for this unit,I thought I just as well publish them in OSI-tems and make them available to everybody.I wwote this about a year ago(for kilobaud).But never sent it in.Too busy with other projects!

Although I had a lot of fun building and programming the Volks-Volks-Computer COSMAC Elf,I found it very frustrating that it was ignored by most hobby computer magazines and felt it was time to look around for a BASIC speaking machine.Along came Volks-Computer OSI Challenger 1 P.Certainly a Superboard,but WHAT about some cheap I/O capability.The keyboard can take care of the input portion,but the output?

After intensive staring at the ckt schematics,I stumbled over 4 unused address outputs(y1 thru y4 of U23) which I ANDed with the R/W line.By using 'phoney' POKEs I could set and reset 2 flip-flops which in turn would turn on (or off) LEDs (the Elf Q !) or relays.As a special application I built a tele phone dialer.

At one of our OSI users group meetings somebody asked if anybody had worked on a cassette based sequential file system which requires a TRS 80-like motor control.This seemed a good applicatiman for my ckt.

In my machine I *PROTO area for the additional IC's,but for people who don't want to alter their 600 board I came up with a plug-in version.Also to make the ckt universally usable for machines with 610 boards,I did not use outputs y0,y1,y2(which address RAM up to 32K on the 610 board)but y4 which addresses 8000HEX to9FFFHEX (unused addressàecations tothe best of myknowledge) plus some additional decoding of addresslines A10 - A12 to get the necessary 4 outputs plus 4 spares(see table 1).I also decided to use a separate 5 to6 VDC powerpack (approx. 250mA) for the relays.To play it safe,the relay ckts are electrically separated form the 1 P by Opto Isolators.

Fig.1 shows the ckt.The 40 pin DIP header(24 pin will do) picks up the addresslines (10 thru 15) and ϕ2,R/W and GND.VCC is picked up with 16 pin DIP header from unused IC socket U6(pin 16)(unused,if no 610 board is hooked up). IC 1 and IC 2decode the addresslines.IC 3 'catches' the decoder pulses and turns on (or off) the LEDs of the Opto Isolators(IC 4,IC 5).All IC's plus R1 and R2 are mounted on a piggy back PC board mounted next to the fuseholder on the 600 board. Emitters and Collectors of the Opto Isolators are extended via 4-wire cable(and optional DIN plug and jack) to project box that contains the remainder of the ckt. LEDs CR3,CR4 are on-off indicators switched by a relay contact.The other relay contact remote controls the tape recorder motor.Toggleswitches S1,S2 are used to override the relay control.

OPeration: Cassette Based Sequential Filing is described on p.5ff of the OSI 8K BASIC-in-ROM Manual.On p.4 you will find POKEs to turn on and off LOAD and SAVE commands.Cassette 1 would load information into the computer,cassette 2 would save the upgrade data.

The motor control ckt does not have a reset button.If you want to turn the relays off (or on) you can POKE the appropiate POKEs in the direct mode.If you try to do this after a warm start,you'll get an OM error message.Try again and it will work(WHY ?).For a sample(test) program see listing 1.

Otehr uses: The Telephone Dialer by Allan S. Joffe (kilobaud nov.79) runs like a charm with the modifications per listing 2.

*USED

1) TEST PROGRAM FOR DUAL CASSETTE MOTOR CONTROL
FOR OSI CHALLENGER 1P BY KLAUS ERNST

```
10 INPUT"R or W or NC";N$
20 IF N$="R" THEN 110
30 IF N$="W" THEN 310
35 IF N$="NC" THEN C$=B$:GOTO 320
40 GOTO 10
110 POKE 34000,0
120 FOR I=1 TO 200:NEXT I
130 LOAD
140 INPUT A$
150 INPUT B$
160 POKE 515,0
170 POKE 33000,0
180 FOR J=1 TO 32:PRINT:NEXT J
190 PRINT B$
195 PRINT:PRINT:PRINT
200 GOTO 10
310 INPUT C$
320 POKE 36000,0
330 SAVE
340 PRINT"GARBAGE INVALID DISREGARD NOT USED"
350 PRINT C$
360 POKE 517,0
370 POKE 35000,0
380 GOTO 10
```

Note: "R" Read(play) off Recorder 1
"W" Write (record) on Recorder 2
"NC" No Change (record) on Recorder 2

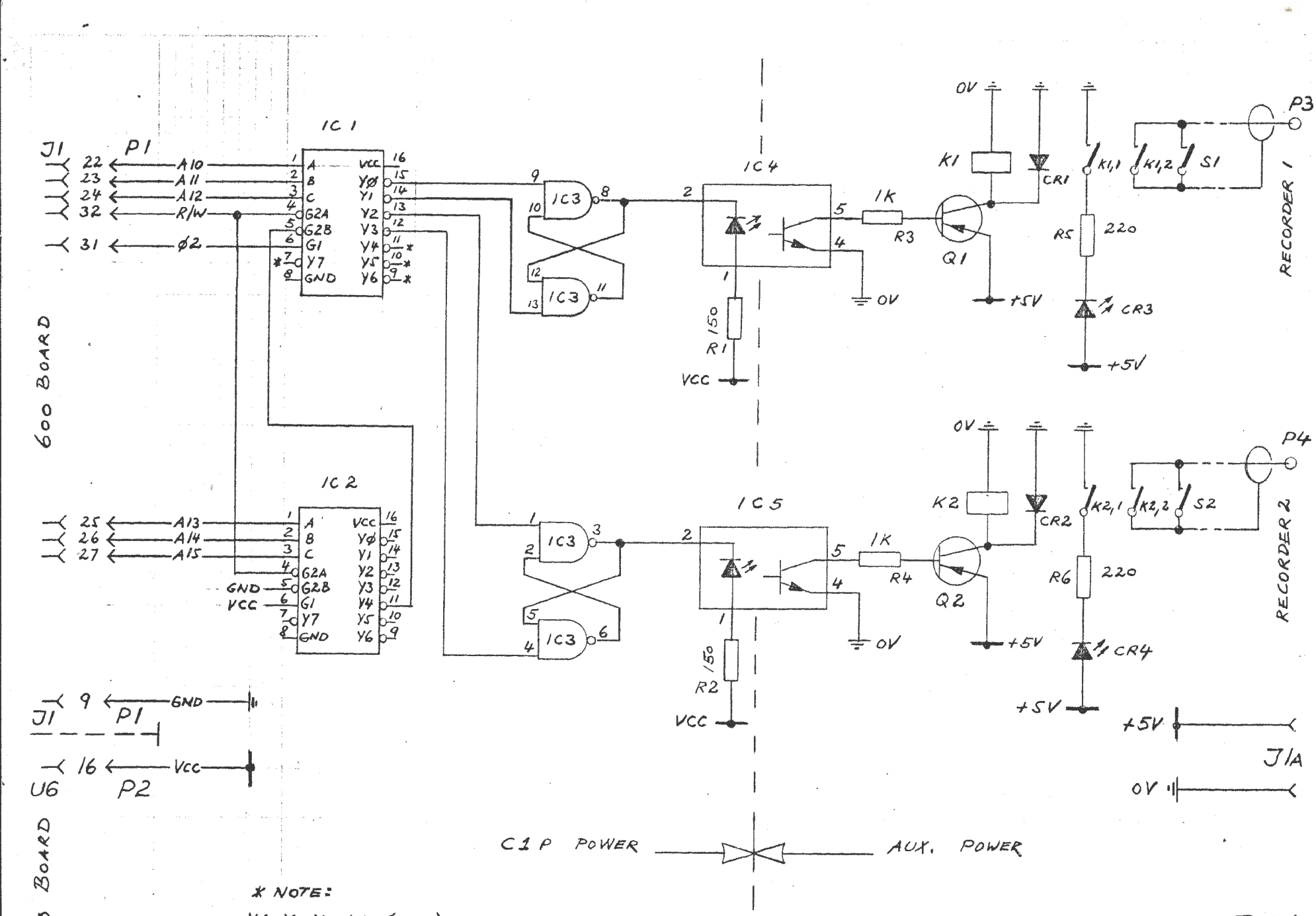
2) THE TRS-80 DIAL-A-PHONE BY ALLAN S. JOFFE, kilobaud Nov.79
MODIFIED FOR OSI CHALLENGER 1P BY KLAUS ERNST

CHANGE OR ADD THESE LINES :

```
35 POKE 34000,0
110 IF NUM(A)>9 THEN PRINT 0;:GOTO120
115 PRINT NUM(A);
220 POKE 33000,0
240 POKE 34000,0
295 POKE 33000,0
```

Address Decimal	Address Hex	Addresslines (decoded) A15 A14 A13 A12 A11 A10	LOW pulse (IC 1) at output:	Use
32768-33791	8000-83FF	1 0 0 0 0 0	Y0	Relay K1 OFF
33792-34815	8400-87FF	1 0 0 0 0 1	Y1	Relay K1 ON
34816-35839	8800-8BFF	1 0 0 0 1 0	Y2	Relay K2 OFF
35840-36863	8C00-8FFF	1 0 0 0 1 1	Y3	Relay K2 ON
36864-37887	9000-93FF	1 0 0 1 0 0	Y4	Spare
37888-38911	9400-97FF	1 0 0 1 0 1	Y5	Spare
38912-39935	9800-9BFF	1 0 0 1 1 0	Y6	Spare
39936-40959	9C00-9FFF	1 0 0 1 1 1	Y7	Spare

Table 1



* NOTE:
Y4, Y5, Y6, Y7 (IC 1)
CAN BE USED FOR
TWO MORE RELAY CKTS

FIG 1

DUAL CASSETTE RECORDER
MOTOR CONTROL
FOR OSI CHALLENGER 1P
11-18-79 BY K. ERNST

Software Reviews**********

by Salomon Lederman

There's a great new assembler available for the C1. It is from Bill's Micro Services 210 S. Kenilworth Oak Park, Illinois 60302. It costs about $15. and is the best deal of this sort that I've seen. The program performs and the documentation is very complete.

The package is an assembler/dissassembler/monitor, though the monitor features are not overwhelming. The package fits into 3K of RAM, it comes in two forms, for the lower 4k and for the upper 4k, of an 8k system. Both versions come on the tape so you can use each one whenever you need it. The assembler is screen oriented, very well done. The assembler assembles each mnemonic as you finish typing in the line, thus there is no need for a symbol table. Unfortunately you can't use labels because of this. There is a write modefor entering in your own code. You can view code one instruction at a time or you can advance a whole page with one keystroke. Supposedly there is an insert and delete that supposedly updates all your jumps; I wouldn't document that until I figure own how to get it to work.

There are quite a few frills that makes this package a real nice one to have around. You can view code as dissassembled mnemonics, as HEX code, or as ASCII. In HEX code you are looking at data, or machine code. You can view data as decimal or HEX. There is a built in screen clear. You can even dissassemble backwards if you like.

So you can see that this is indeed a nice package; go get one from Bill's Micro Services. You'll be glad you did!;

Programming the 6502- Machine Language for the Challenger Computer
Class 2, by Salomon Lederman December 4, 1980

This lesson is a followup of last month's class. Refer to last month's OSI-tems if you missed the class. Before I move on to new material I shall review the important information that we will need to use for this lesson.

We have learned several opcodes up to now. They are:

AD	LDA	Load accumulator from a 2 byte address
A9	LDA	Load accumulator with a constant (from next byte)
8D	STA	Store accumulator in 2 byte address
00	BRK	Break (stop the program)
EE	INC	Increment a 2 byte address
4C	JMP	Jump to the 2 byte address of the next 2 bytes
20	JSR	Jump to subroutine, indicated by next 2 bytes

These opcodes are enough for writing very simple programs but we must learn more codes to develop more complicated programs. In this lesson I will discuss two new concepts branching, and indexing. A simple branch is the JMP instruction, there are conditional branches that we should learn about to write programs that do things in loops. Indexing is a powerful tool that helps to count things, and to do things consecutively.

BRANCHING

The concept of conditional branching should not be new to anyone. Every language must have provisions for branching. An IF-THEN statement in BASIC is one example. The FORTRAN DO loop does essentially the same thing. In assembly language we have quite a few commands that help us transfer control of the program to a different section, depending on the result of some calculation. Consider the following opcode for CMP; its value is C9. The function of this code for CMP is to compare the contents of the accumulator to some numeric constant that is the next byte of your program. We shall soon see more precisely what CMP does. Study the following Example:

```
0500    20 00 FD  JSR $FD00
0503    C9 20     CMP #$20
0505    D0 F9     BNE $0500
0507    A9 00     LDA #$00
0509    8D C9 D1  STA $D1C9
050C    00        BRK
```

You will notice several new things in this program. Remember that the symbol '#' refers to a constant value and that the '$' stands for HEX. If you look at the assembly listing on the right then you should be able to figure out how this program works and what it does. The CMP instruction, to be a bit more precise than we were previously, subtracts the value of the constant from the value of the accumulator and sets some flags. I will not go into an explanation of flags in this lesson, for now it is sufficient only to understand that the BNE instruction depends on the flags that the CMP sets.

Remember from the previous lesson that JSR $FD00 calls a subroutine in ROM which waits for a keypress, then returns to your program. Thus at 0500 we call this routine and wait for a key. Once we reach 0503 we have the value of the keypress (in ASCII) stored in the accumulator- the ROM routine does all this.). At 0503 we compare this value to the constant $20. $20 is ASCII for the space bar. If this comparison does not result in equality, then we go on to the next instruction (we ignore the BNE). However, if the two values are equal then we transfer control to to whatever part of the program BNE points to. This is fairly complicated to explain, in class I will go into detail as to how to transfer control of a program.

INDEXING

For the sake of brevity I will provide only one example of indexing. The theory of indexing is quite complicated. The purpose of the class is to thoroughly explain the topics. This summary will provide you with a minimal knowledge of what indexing does. In further lessons I plan to go into much detail. Now consider the following example of indexing.

```
0500    A2 00     LDX #$00
0502    A9 20     LDA #$20
0504    9D 00 D0  STA $D000,X
0507    9D 00 D1  STA $D100,X
050A    9D 00 D2  STA $D200,X
050D    9D 00 D3  STA $D300,X
0510    E8        INX
0511    E0 00     CPX #$00
0513    D0 EF     BNE $0504
0515    00        BRK
```

This little program clears the screen for a C1. Can you figure out how to modify it for a C2? Note that the program is really not very long. CPX does basically the same thing as CMP except that it compares the X register to the constant value instead of the Accumulator. The really new instruction is 9D, which does a strange knid of store. What is done, in simple terms, is that the value of X is added to the address to obtain a new address. Remember that a byte is only 256 bytes big. Thus we need for special STAs to cover four pages of memory, one K. Thus X begins with the value 00 abd is incremented each time through the loop. After X gets to FF, or 255D, it goes back to 0. AT this point the BNE is not activated and the program falls out of the loop to stop the execution.

In the next lesson I shall discuss various methods commonly used to interface BASIC programs to Assembly Language subroutines. It is very impressive to have these short routines in your program to clear screen and color, to scan the keyboard, poke messages on the screen, etc. One or two of these routines can really speed your program up. Poking in BASIC is not incredibly fast. I know of several programs that were slowed down considerably because they had on screen scoring routines that had to updated often; in machine language you will never notice the delay. So if you don't think that you will be doing full machine language games in the future, at least you will be able to speed them up with various routines.