Restoring a Nicolet NIC-80 back to life
Dwight K Elvey

In early December of 1999 I saw an interesting item for sale on eBay. It was a NIC-80 with core memory and floppy disk drive. I had resently seen a message from Sellam Ismail that he had gotten a Nicolet computer that was used as a data processor for a NMR Spectrometer. These use the same basic principle as the NMRI machines used in medicine. The difference is that it was used for chemical analysis. He said that he'd also gotten a number of paper tapes and some documentation. I figured I'd be able to put the two together and come up with a working machine.

I won the bid and started to deal with how to get two large rack mount units shipped out. The fellow I bought them from, Jay West, another collector, wasn't sure how to deal with shipping it to me but after several email messages, we worked it out. Since this all started around Christmas, I told him that there was no rush.

Finally I got the unit. By this time I'd met Sellam several times and asked about the items he had. I drove to his place and he dragged out several items. He was willing to loan out manuals and tapes if I was willing to read the tapes into an electronic form.

Like many of these older mini designs, the NIC-80 has a front panel with LED lights and switches. I read the manuals that Sellam loaned me and found the machine language was described in a form that was usable to create an assembler. I had written many assemblers in the past for various processors so it didn't take me long to crank out something to make code.

I toggled in some test code and hit the sequence to run the code. Nothing happened. Inspecting the first location, I found that the code was erased. This was strange because I knew that even to read the values for the display ment that the value needed to be written back. Core memory is a distructive read. This ment that the memory was working because I could display all day long with no problems. I also noticed the the instruction register, displayed on the front panel, was not changing. This gave me a starting point to track down the problem. None of the documents that I'd gotten from Sellam had any schematics for the processor. At first this seemed like an impossible task. Inspection of the boards in the machine showed that each board had the pins labled with things like MB12 and IR5. It didn't take long to figure that MB was memory bus and IR was instruction register. The instruction register was composed of several 74175's. I tracked the clock back to the failing part, a 7474.

I was now up and running code. It didn't take long before I had a simple echo program working with the serial interface. I found that one of the chunks of core was occationally losing one bit. This was in a bad location so I swapped it with one of the other blocks that I shouldn't need until later. I've since found the problem of a broken wire.

Nicolet had a program to read paper tapes the one could toggle in to get code loaded. This program was called "Nicoloaden" (I almost learned this by heart). Although, it is inteneded to stay resident, the thing that doesn't work right always seems to be the thing that needs that space. Working with core is great. You can put in an experiment one night and find it still there several nights later.

I didn't have a serial tape reader. Sellam loaned me a couple of readers that one was suppose to work with the NIC-80. I already had a reader and since I needed to recover data from tapes for Sellam, I desided to use the parallel reader I had. I threw together a program that used the parallel port on my lap top as an input port to read tapes. I was now able to read the tapes into a DOS file and later transfer them through the serial port with another program I wrote.

I loaded a few programs and they ran well. I still haven't tackled the floppy drive. The manuals that I had described a simple disk OS called DEMON II. This was designed to run a Diablo removable platter drive. I had a floppy drive so there were a number of potential differences. The manual said that to run the floppy, I needed DEMONF. I looked at every tape and could only find copies of DEMON II.

Using a quickly thrown together disassembler, I started looking into how the DEMON II worked. The manual also had a number of examples, both talking directly to disk and using their low level drivers. The fundimental data block size for the hard drive was a track of 3000 octal words. The word size on this mini is 20 bits. Doing some quick calculations, knowing that the drive, a SA900, was similar to a standard SA800, I knew that I could get a maximum of 4000 octal words on a floppy track. There were going to be diferences.

I needed to write the low end drivers to work with the floppy disk interface. This being an early mini, they didn't use some nice floppy interface chip, the disk interface was composed of 4 cards with TTL. The only way I was going to figure it out would be to trace the circuits and see how the various signal to the floppy were generated. I started tracing circuits into schematics. I expected that they would make registers that were similar to the hard drive. As it turned out, they were only slightly similar.

While debugging the floppy coding, I found a couple of other hardware problems. An old leaky tantilum capacitor that was being used to find the index was causing troubles. Later I found that I had a data pattern issue that turned out to me line termination on the home made cable I used to connect the processor to the floppy drive.

The writing of code had a few problems. While looking at the hardware, I determined that it expected 32 hard sectored disk and that it split the tack into two 2000 octal work blocks. This ment a certain incompatability with the hard drive software. I also found that although it was hard sectored, it expected some kind of tack address at the beginning of each track. This set me back about two weeks. I finally found that what it wanted was the track number written, bit reversed in the MSB's of the first word on that track.

I now looked at compatibility between the DEMON II and what I thought DEMONF should look like. I desided to keep the changes to a minimum and only write a low level driver that would work as much like the hard drive as possible. This wasn't an easy task because the Interface was just enough different that the code took a little more space. The code that they had written looked pretty good and I wasn't sure if I could cram more function into the same space and still keep entry vectors and public variables in the same place. Soon, I found places that their code could be improved. A word here and a word there. Soon, I had not only found space for my 4 extra words but 2 spares as well.

I wrote some code to patch the original DEMON II into my version of DEMONF. It required a complete overlay of the low level driver and some sixty patches to the code that called it. This was because the block select word was slightly different and the basic block size was different as well.

A couple of days of debugging on the system and I now had the basic DEMONF running. I now have restored a rare old computer back to a level that it can be used to execute code and use the original mass storage system. It has been a fun projected and doing this kind of thing is why I bought the machine in the first place.

I learned that the machine is much rarer than I'd realized. As far as I know, there is only one other still in existance and that is the one Sellam has. I have learned a lot about mini computers of this era and I believe I have made a contribution to keeping the history and hardware of this unique machine alive. There are a lot of other things that I left out of the story but it was long enough as it is.

FLOPPY DISK EXECUTIVE MONITOR
as Reconstructed by Dwight K Elvey
June 10, 2000

I. Description of Hardware

The floppy system is similar to the Nicolet 294 disk system. It uses 8 inch 32 hard sectored disk. The controller partitions each track into two sectors of 2000 octal (1024). This is the first difference between the floppy and hard drive systems. The hard drive system has 3000 octal (1536) per track as a single block. Like the hard drive system DEMONF allocates in the minimum block size of 2000 octal instead of 3000 octal. The floppy drive has a total of 115 octal (77) tracks. Since there are two blocks allocated per track, there are 232 octal (154) total blocks. In the DEMONF system, 5 words are used for each directory entry (the same as used in DEMON II). This means that the entire floppys directory can still fit within a single block. In the DEMON II, the directory is overlayed at 3000 octal when the DEMON is started from the disk head (loaded at 7600 octal). In DEMONF, the address is moved to 4000 octal. This has implications for accessing the directory directly but should be transparent to most code. By moving the directory location, the system useage of the first 20 octal floppy blocks can be kept the same as the hard drive. This helps to minimize the differences between the two systems.

Because of the differences between the controllers used in the hard and floppy drives, the selection of unit number and also the exact nature of reading and writing are slightly different. In the floppy drive system, drives are selected with 1000 octal for drive 1 and 2000 octal for drive 2. When selecting blocks to read or write, they are selected by combining the block number with the unit select, similar to the hard drive. The other difference is that besides setting the accumulator to 1 before doing a write, one must also mask in the bit pattern 400 octal to the unit/block word.

II. Introduction to Demonology

The floppy DEMONF is the same as DEMON II. The head resides at 7600 - 7777, and can be restarted by setting the switches to 7600, load PC/execute then continue/execute.

A. Loading the DEMONF

Refer to DEMON II manual.

B. Basic Commands

Refer to DEMON II manual.

C. Disk Error Messages

Refer to DEMON II manual.

D. System Start-Up and Shutdown

Refer to DEMON II manual.

E. DEMONF Bootstrap

This is similar to that that is described in the DEMON II manual. The code itself is different. If the system does not boot when started at 7600, it will be necessary to try to restart from the floppy. The following code will restart, using the copy of the disk head on the floppy. If this doesn't work it will be necessary to rebuild the system as described in section II.A.

This is the minimum code to restart DEMONF:

```
S.A. 7565   111000   1000 #VAL A =M
     7566     4631   LTRACK
     7567    44632   FPSTAT
     7570     2010   10 #VAL ?0 =A&M
     7571  2162000   ?0 =0
     7572     1570   7567 JMP
     7573    46634   FPRDF
```

```
7575      1574   7573 JMP
7576   2405772   7772 ADDR M =A
7577   2705576   7575 ADDR M =M-1
7600      1574   7573 JMP
```

F. Initializing Additional Disk Cartridges

Other than the use of 32 hard sectored 8 inch disk, the procedures for initializing the disk are similar to the DEMON II. The exception is that it is necessary to pre-format the floppies before using SYSGEN.

Format Process:

1. Use RUN FORMAT to start the format process. It will stop at PC=1 and wait for you to install the new floppy. Note: FORMAT can be loaded with Nico-Loadeon at 0-130;0
2. Place the new floppy in the drive A, with write tag attached.
3. Press the reset button on the front of the drive.
4. Do continue/execute.
5. When the drive activity stops, press the reset button on the front of the drive
6. Reinstall the original system disk.
7. Do continue/execute.

From this point, you can now run SYSGEN as stated in the DEMON II manual. If the new disk is to have SYSGEN loaded, it is a good idea to move FORMAT to the new disk as well.

G. Moving Files Between Disk

One can use DIR :F to determine address, size and start of files. Use the LOAD command to move the file to memory. One can then install the new floppy and mount the new disk by executing at 7600. Once this is done, use the STOre command to tranfer onto the new floppy.

III. Programming The Floppy Drive

The controller is somewhat similar to the hard drive. The major exception is that a write must be indicated by the write select bit being set in the LTRACK, even though there is a separate instruction to do the write and read.

Octal Code	Instruction	Meaning
4631	LTRACK	Load Track address from AC
44632	FPSTAT	Read Floppy controller status
46634	FPRDF	Read data word with skip
6634	FPWRF	Write data word with skip
		Note LTRACK difference with FPWRF

LTRACK Bit Map:

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Format	Recal								Drive2	Drive1	WriteBit	Block Address 0 - 231							

There are two blocks per track so bits 1 thru 7 select the track and bit 0 selects the block within that track. The controller compares the value from bits 1 thru 7 as the track number desired. If there is a mismatch, it automatically steps to the desired track. When ready to transfer data, it sets the Data Ready bit in the FPSTAT returned value.

When the Format bit is set, the controller ignores any value read from the disk and preforms a forward step equal to the value in bits 1 thru 7. Normal formating steps by 1. The first value written on the track should be the track number, written bit reversed in Bits 19 thru 13. This is the only part of the disk that needs to be initialize for write. Read do read quire that data be continued for both sectors worth of data.

Recal bit causes the controller to seek track zero. Success is indicated by the track 0 bit in the FPSTAT returned value.

FPSTAT Bit Map:

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																TransReady	Track0		ReadError

There are fewer status bit to worry about in the floppy controller. Track0 is only active while the head is loaded. After that the status is lost so one needs to read the status shortly after the Recal is executed.

IV. Programming with DEMONF Head

Programming is similar to DEMON II except that one of the drive bits must be set instead of one of the Unit bits. As an example, the hard drive 1000020 would address disk block 20 octal. In the floppy, it would be 1020. The minimum block size is 2000 octal instead of 3000 octal. Otherwise the calling conventions are the same as DEMON II. Programs Written for the original DEMONF will require some modification as well since the calling vector is 7613 instad of 7612 for new DEMONF. ErrorFlag is 7704 for new DEMONF.

V. Programming Using the Directory Function (DIRFUNC)

Refer to the DEMON II manual. The exception is that only 4000 to 7577 is swapped for the floppy instead of the hard drives 3000 to 7577. This effects some programs that were written for the hard drive.

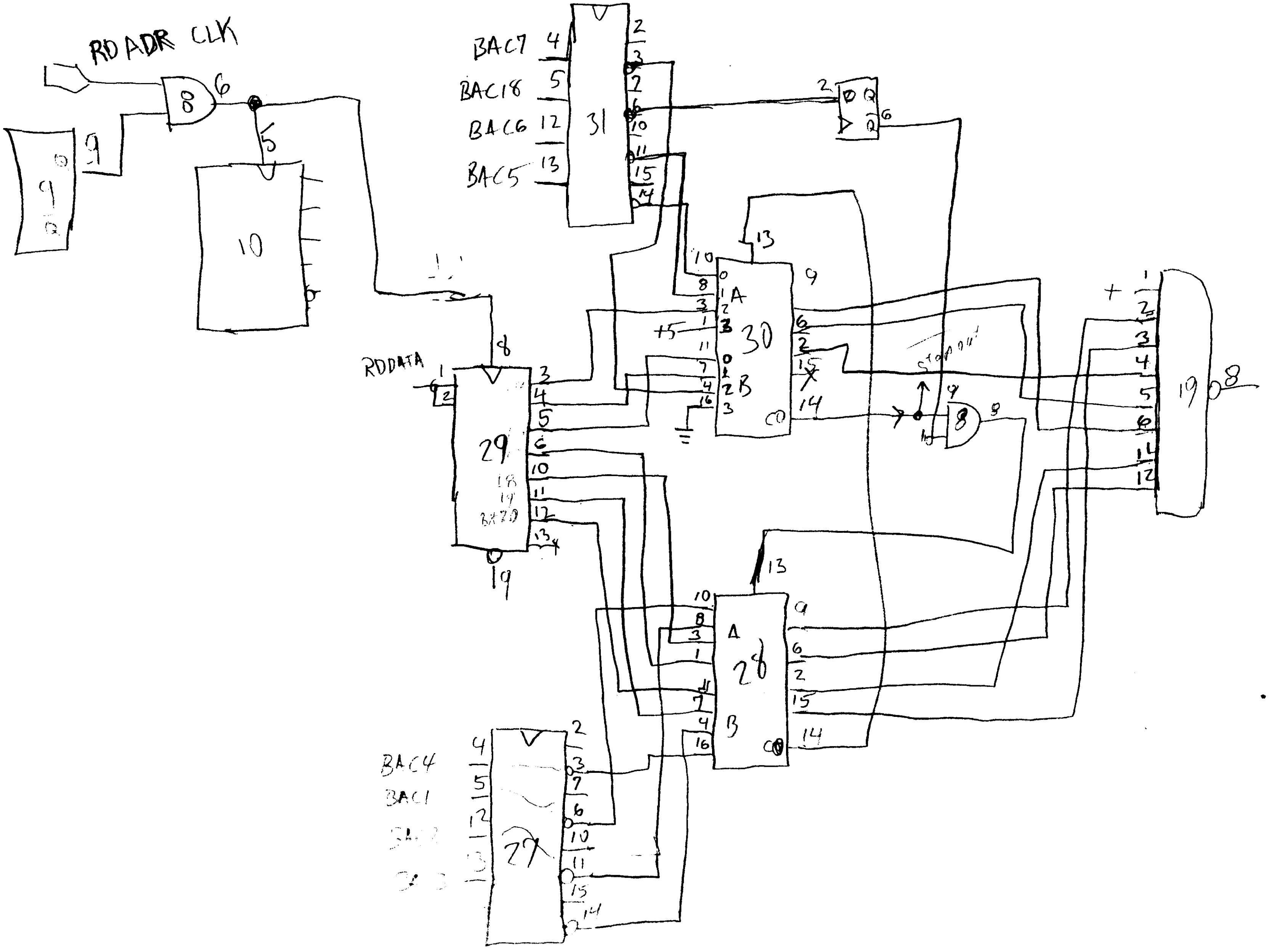
VI.

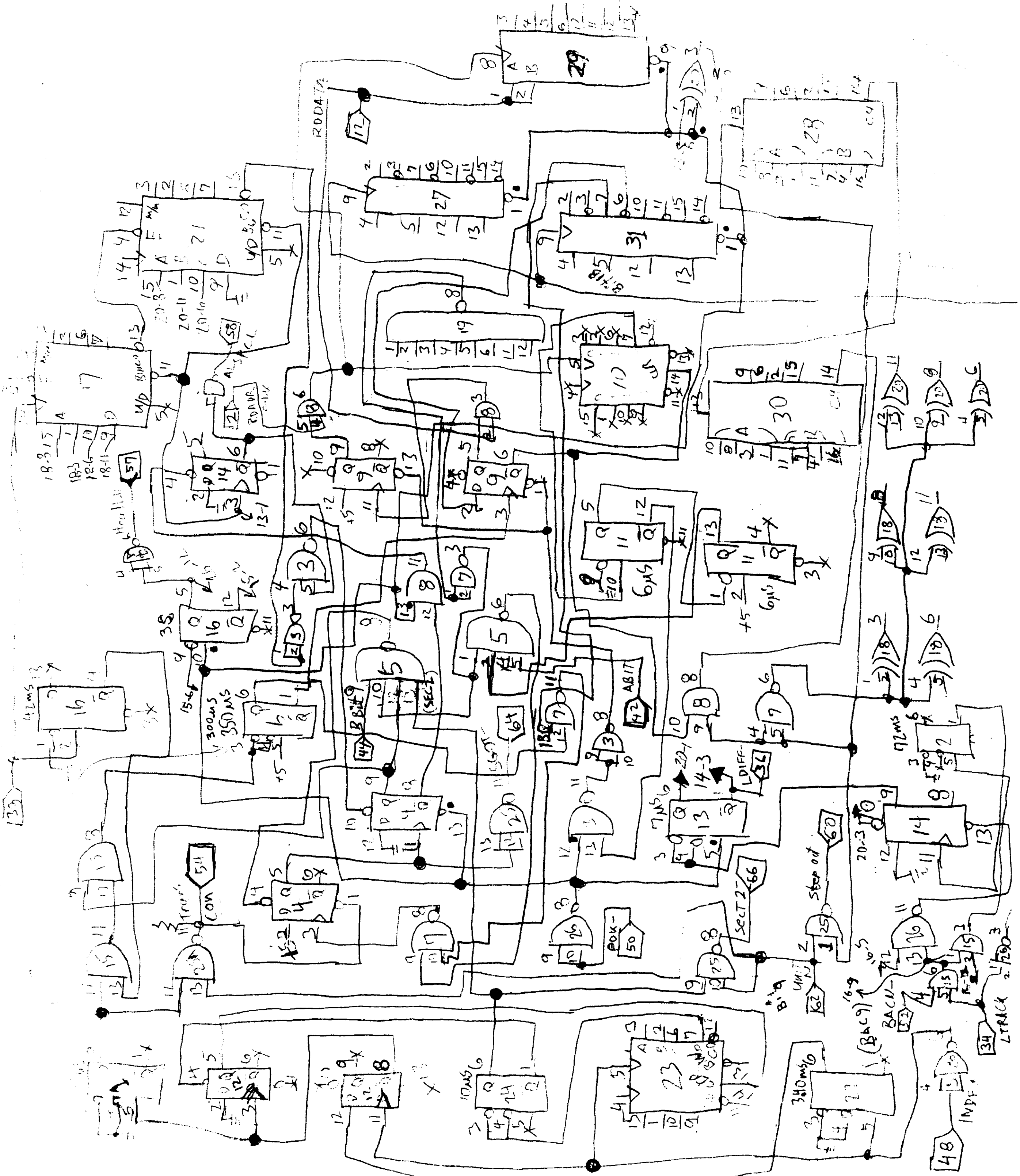
Refer to the DEMON II manual. Also refer to the listing FLOPPY.SEQ and FLOPPY.LST

```
FLOAD ASMN80
\ DEBUG PFLABEL
OCTAL
MAIN

    7565 ORG
LABEL BOOTFLOPPY
    1000 #VAL A =M
    LTRACK
LABEL STATWAIT
    FPSTAT
    10 #VAL ?0 =A&M
    ?0 =0
    STATWAIT JMP
LABEL RDLOOP
    FPRDF
    RDLOOP JMP
LABEL RDADDR
    7772 ADDR M =A
    RDADDR ADDR M =M-1
    RDLOOP JMP

comment:
  New boot strap for my Demon/f
  7565   11100Ø   100Ø #VAL A =M
  7566     4631   1 IOP 23 IO
  7567    44632   2 IOP ZAC 23 IO
  7570     2010     10 #VAL ?0 =A&M
  7571  2162000   ?0 =0
  7572     1567    7567 JMP
  7573    46634   4 IOP FLAG ZAC 23 IO
  7574     1573    7573 JMP
  7575  2405772    7772 ADDR M =A
  7576  2705575    7575 ADDR M =M-1
  7577     1573    7573 JMP


  Original boot strap for DemonII
  7566  2111566    7566 ADDR A =M
  7567     4546   6 IOP 14 IO
  7570    44541   1 IOP ZAC 14 IO
  7571     2036     36 #VAL ?0 =A&M
  7572     1570    7570 JMP
  7573    46526   6 IOP FLAG ZAC 12 IO
  7574     1573    7573 JMP
  7575  2405772    7772 ADDR M =A
  7576  2705575    7575 ADDR M =M-1
  7577     1573    7573 JMP
comment;

SAVECODE BOOTFLP.BIN
NCRNT .

 BOOTFLOPPY NCRNT BOOTFLOPPY -

 empty fload disn80
 rb BOOTFLP
 pfile BOOTFLP.lst
 print dis
```

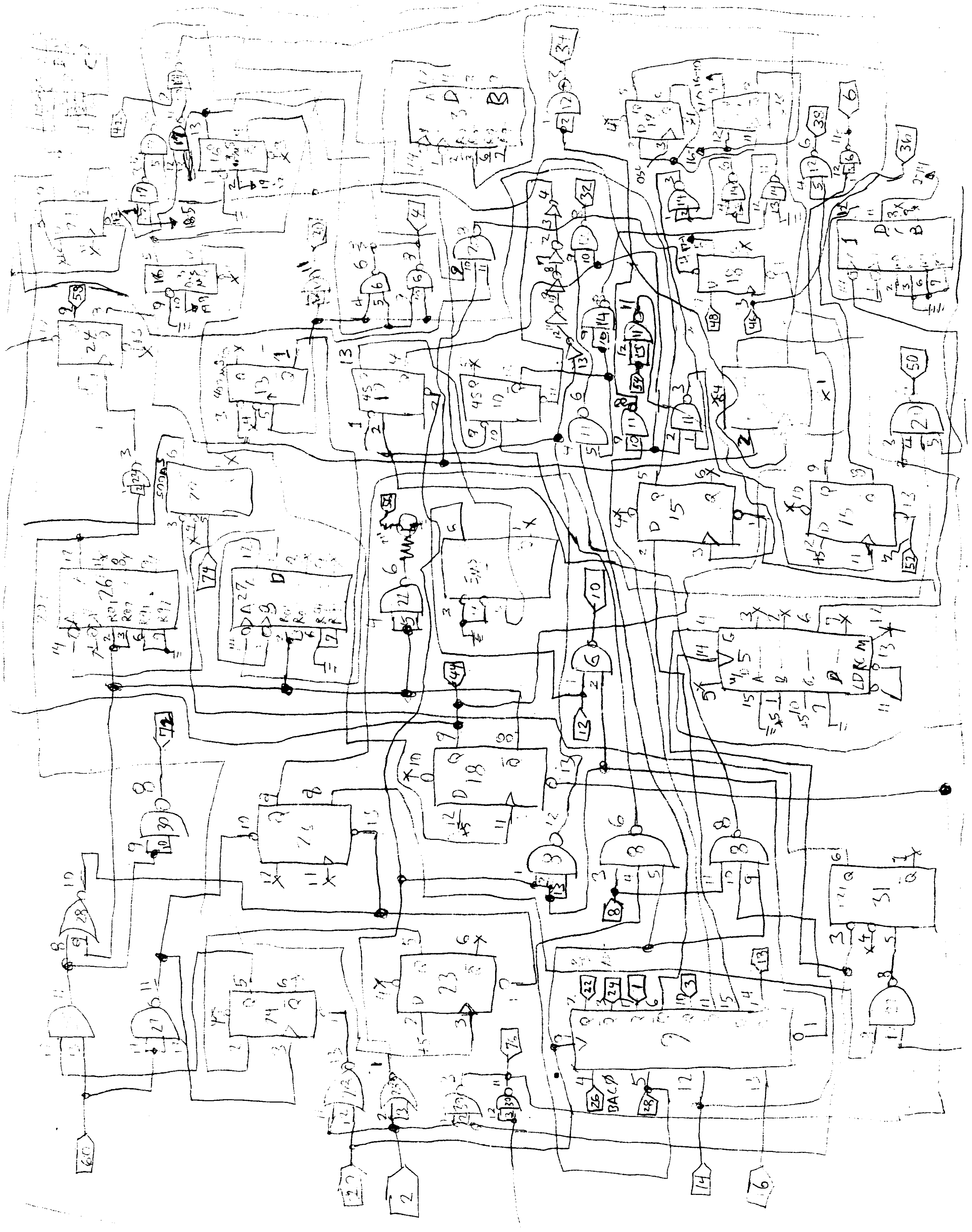
```
\ MKSYS.SEQ   Make entire floppy system
\    This file combines the parts of demonx.bin
\    with newer files and makes patches to original
\    binary

fload disn80

CREATE sBuf  BufZ ALLOT

: Buf->sBuf
   Buf sBuf BufZ  CMOVE ;

: sBuf->Buf
   sBuf Buf BufZ  CMOVE ;

: S! ( dVal Addr - )
  3 * sBuf +
  SWAP OVER C!
  1+ ! ;

: S@ ( Addr - dVal )
  3 * sBuf +
  DUP 1+ @
  SWAP C@ ;

: NMove  ( From To Count - )
    0 DO
      DUP>R
      OVER N@
      R> S!
      $00010001. D+
    LOOP 2DROP ;

: -NMove ( From To Count - )
    0 DO
      DUP>R
      OVER N@
      R> S!
      1- SWAP 1+ SWAP
    LOOP 2DROP ;

0 VALUE DemonOffset
0 VALUE LoadOffset

: sPatch ( dShouldBe dWas Addr - )
    LoadOffset -
    DemonOffset + DUP>R
    S@ D- OR
    ABORT" Patch not matched"
    R> S! ;

: xpatch .s cr
    LoadOffset -
    DemonOffset + dup . DUP>R
    S@ 2dup d. D- OR cr .s key drop
    ABORT" Patch not matched"
    R> S! ;

OCTAL
ReadBuf DEMONX
Buf->sBuf

\ sysgen
 0 =: DemonOffset
 0 =: LoadOffset
 \  5220. 0. 4211 sPatch  \ text operation

   1000. 100000. 3 sPatch
   1004. 100004. 23 sPatch
   1005. 100005. 30 sPatch
   1006. 100006. 35 sPatch
   1007. 100007. 42 sPatch
   1010. 100010. 47 sPatch
   1011. 100011. 54 sPatch
   1012. 100012. 61 sPatch
   1003. 100003. 66 sPatch
   1003. 100003. 74 sPatch
   2000. 3000.  67 sPatch
   2000. 3000.  75 sPatch
   1014. 100014.  101 sPatch
   1006. 100006.  106 sPatch
   102000. 103000. 70 sPatch
   102000. 103000. 76 sPatch
   4000. 6000. 102 sPatch

  ReadBuf SysGen
  125 125 137 NMove

 \ BootStrap ( MonHead Backwards )
  ReadBuf Floppy
  7600 2172 173 -NMove

 \ Key Board Monitor
  2200 =: DemonOffset
  6000 =: LoadOffset
   1007. 100007. 6323 sPatch
   1005. 100005. 7133 sPatch
   2000. 3000. 7551 sPatch
   4000. 3000. 7552 sPatch
   46634. 46526. 7115 sPatch
   2001630. 2001627. 7127 sPatch
   3000. 1700000. 6763 sPatch  \ mask drive only

   110002. 110005. 7277 sPatch  \ When 2 drives set to 110003.
 \ 5220. 221. 6661 sPatch \ debug stop
 \ 5220. 2103427. 6662 sPatch \ debug stop

\ Paper tape loader
ReadBuf FBINLDR
 7600 4000 16 NMove  \ Just the patched code

\ MonitorHead
ReadBuf Floppy
 7600 4200 160 NMove

 0 =: DemonOffset
 0 =: LoadOffset
\ 5220. 0. 4211 spatch   \ test code

\ DirFunc
 4400 =: DemonOffset
 7000 =: LoadOffset
 2000. 3000. 7020 sPatch
 4000. 3000. 7021 sPatch
 1010. 100010. 7175 sPatch
 2000. 3000. 7551 sPatch
 4000. 3000. 7552 sPatch
 4004. 3004. 7555 sPatch
 2000. 3000. 7247 sPatch
 2000. 3000. 7456 sPatch
```

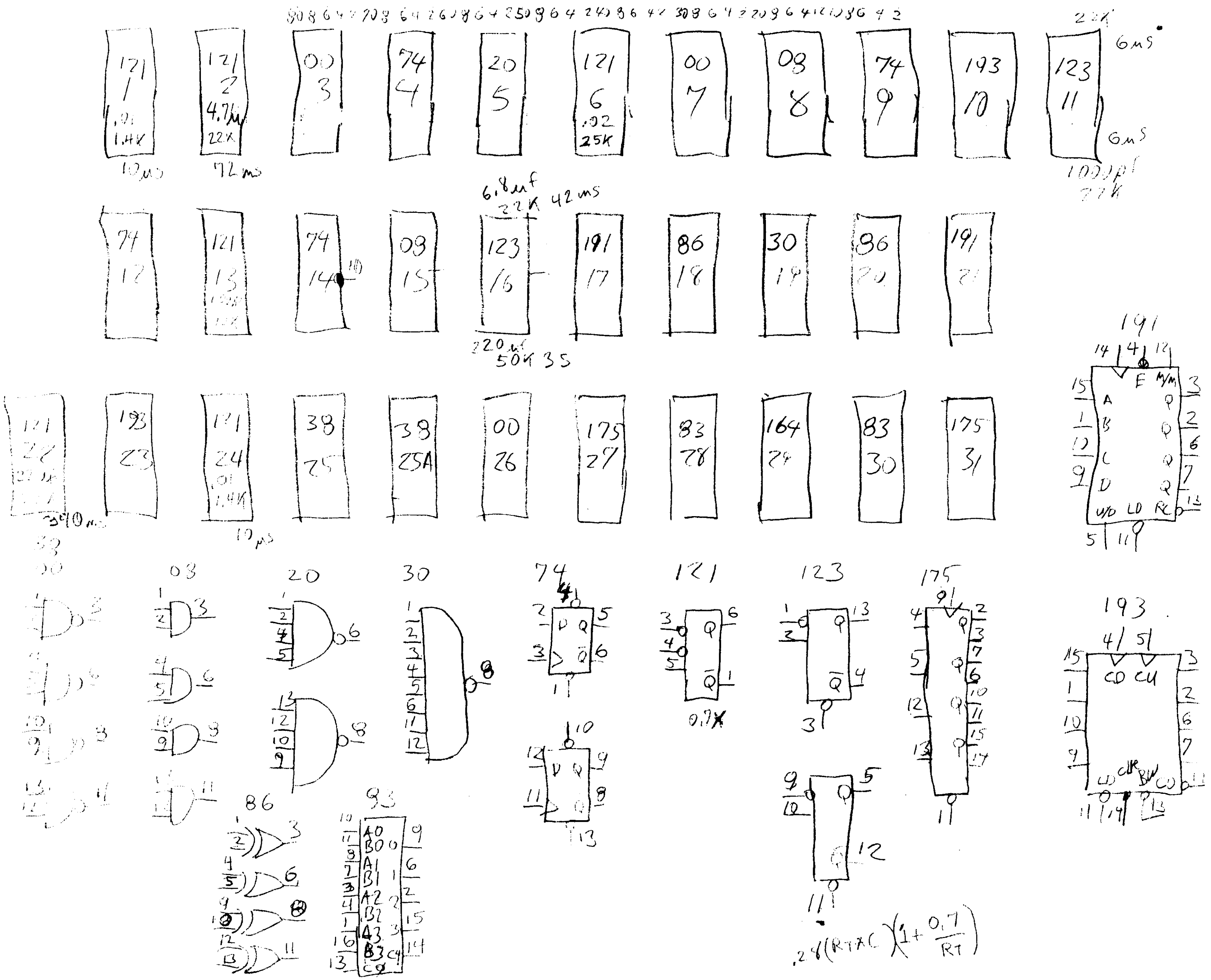
```
  1000. 100000. 7044 sPatch
  330232. 330625. 7165 sPatch
  2000. 200000. 7045 sPatch
  0. 400000. 7046 sPatch
  0. 1000000. 7047 sPatch
 \ test stops
 \ 5220. 2162000. 7005 sPatch
 \                            5220. 1034. 7553 sPatch

 \ DirLst
  5200 =: DemonOffset
  7200 =: LoadOffset
  2000. 3000. 7350 sPatch
  4000. 3000. 7351 sPatch
  110212. 110605. 7353 sPatch  \ used by kill

 \ GenIO
  5600 =: DemonOffset
  6000 =: LoadOffset
  1007. 100007. 6003 sPatch
  1001. 100001. 6263 sPatch
  2000. 3000. 6264 sPatch
  4000. 3000. 6265 sPatch
  2000. 3000. 6365 sPatch
  110002. 110005. 6221 sPatch  \ make 110003. for two drives

 \ IOHandlers
  6600 =: DemonOffset
  6000 =: LoadOffset
  1007. 100007. 6030 sPatch
  1000. 100000. 6062 sPatch
  2000. 200000. 6063 sPatch
  0. 400000. 6064 sPatch
  0. 1000000. 6065 sPatch
  1001. 100001. 6160 sPatch
  2000. 3000. 6161 sPatch
  4000. 3000. 6162 sPatch
  1002. 100002. 6171 sPatch
  4000. 3000. 6173 sPatch
  1002. 100002. 6462 sPatch
  4000. 3000. 6464 sPatch
  1001. 100001. 6470 sPatch
  2000. 3000. 6471 sPatch
  4000. 3000. 6472 sPatch
  1002. 100002. 6771 sPatch
  4000. 3000. 6773 sPatch


sBuf->Buf
WriteBuf DEMONF.bin

 empty fload mkpt
 octal
\ 0 173 mkpt DEMONF
 0 7600 mkpt DEMONF


empty fload disn80
rb demonf
pfile demonf.lst
print 0 7600 dis
pclose
```

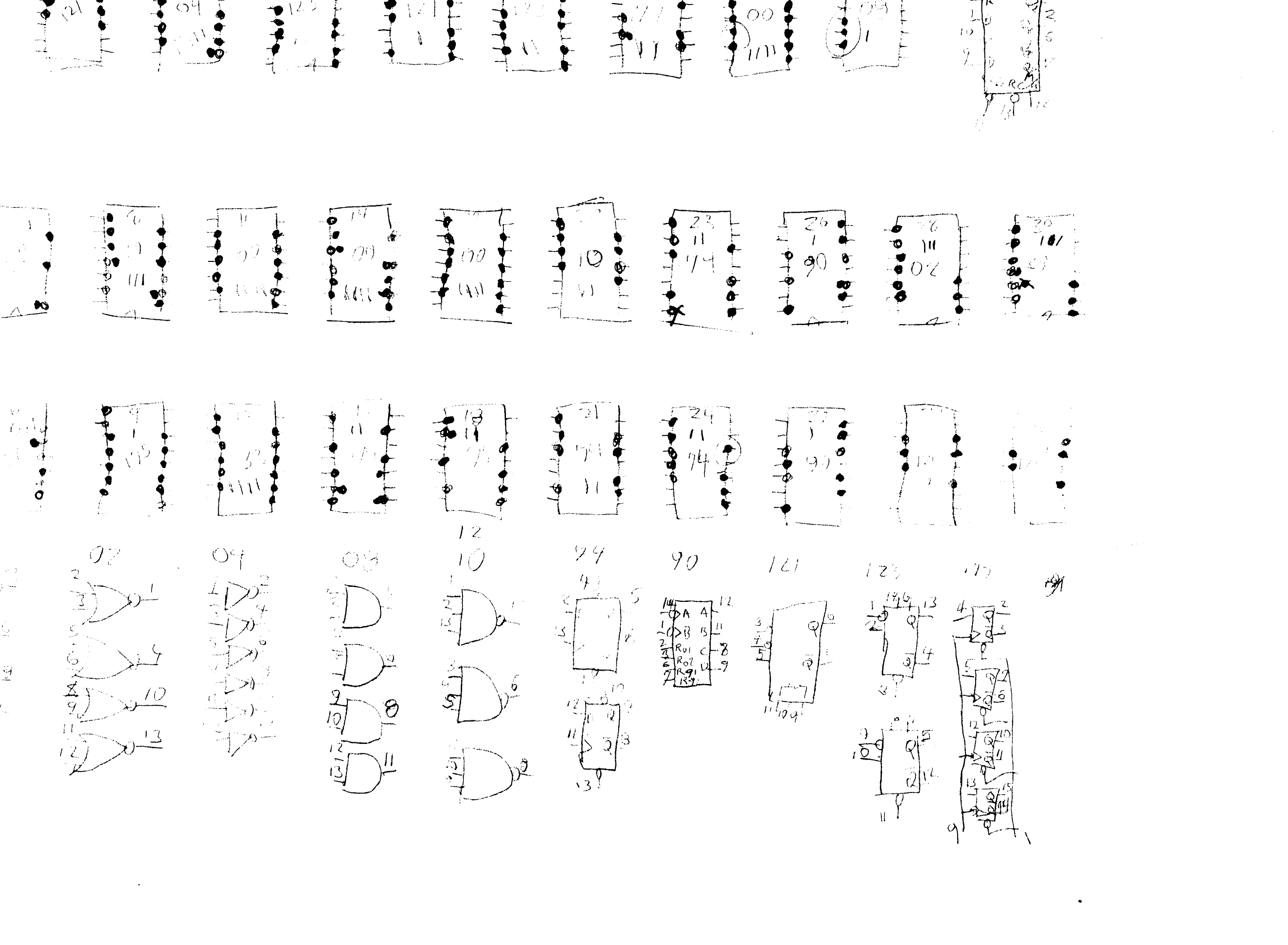

RD ADR CLK
9
8
6
5
10
RDDATA
29
8
19
BAC7
BAC18
BAC6
BAC5
31
30
+5
stop out
28
BAC4
BAC1
27
19
8
+



RDDATA
12
29
28
27
31
21
17
19
10
30
14
9
11
6μS
16
42MS
300MS
350MS
72MS
32
38
57
58
ALS
CON
54
41
BBIT
SECL
64
15B
ABIT
42
LDIFF
36
Step out
60
SECT 2
66
BOK
50
62
BAC9
BACN
LTRACK
34
INDF
48
23
10MS
380MS
+5


34
32
39
36
4
50
52
48
46
53
44
72
74
60
D
B
Q
D 15 Q
D 18 Q
23
31
LD
BACØ
26
28
22
24
12
2
14
16
10
8
13


121 1 .01 1.4K 10ms
121 2 4.7M 22K 72ms
00 3
74 4
20 5
121 6 .02 25K
00 7
08 8
74 9
193 10
123 11 22K 6ms 6ms 1000pf 22K
74 12
121 13
74 14 10
08 15
123 16 6.8uf 22K 42ms 220uf 50K 3S
191 17
86 18
30 19
86 20
191 21
121 22 390ms
193 23
121 24 .01 1.4K 10μs
38 25
38 25A
00 26
175 27
83 28
164 29
83 30
175 31
00
08
20
30
74
121
123
175
191
193
86
93
A0 B0 A1 B1 A2 B2 A3 B3 C0 C4
D Q Q̄
CD CU LD CLR BW CO
A B C D E M/m U/D LD RC
0.7K
.24(RT×C)(1+0.7/RT)



02
04
00
10
12
74
90
121
123
175
191

```
 Disk OS code and system utilities
  DEMONF.TAP  STORE SYSGEN 0-7577;0
     Builds disk OS
  GARBAGEF.TAP  STORE PACK 6000-6300;6000
     Recovers disk fragments
  FORMAT.TAP  STORE FORMAT 0-100;0
     Initalizes new disk before using SYSGEN
  BOOTFLP.SEQ  toggle in at 7566
     Restarts crashed system if disk OK
  NCLDN5B.TAP  Load by tape boot loader not saved to disk
     Nicoloaden Binary Tape Loader

 IMP package code
  DEDIT1.TAP  STORE DSKED 0-4300;0
     Part of IMP set
  DSKASSMF.TAP  STORE ASM 0-6500;0
     Part of IMP set
  MOVE.A    load w/ DSKED  funtion FMOVE
     Source for part of IMP, assemble to use
  MOVE.TAP  STORE MOVE 0-1500;0
     Partof IMP set
  DLUWDA.TAP  STORE LOADER 100000-101500;100000

BASIC files and programs
  BASICF.TAP  STORE BASIC 0-1777;0
     Main BASIC to RUN
  BASIC1F.TAP  STORE BASIC1 0-7677
     Part of Basic
  BASIC2F.TAP  STORE BASIC2 102000-107777
     Part of Basic
  BASDIRF.TAP  STORE BASDIR 100500-101677
     Part of Basic
  TREK.TAP  Use ASCIILOAD in BASIC to load, use NEW
     BASIC program to kill the Klingons

Misc Programs
  BJACKF.TAP  STORE BJACK 0-5200;0
     Black Jack Program
```

```
/ DISK MOVE PROGRAM
*0
IOSTRT, JMS SAVE /SAVE CORE
  ZERA
  JMS @ ZDISK /READ CD IN
    11  / MODIFIED FOR FLOPPY  1011
  1000  /WC
  6000  /BUFFER
  ZERM @ ZDEVDIR
  JMS @ A6000 /ENTER CD
ATABPNT, TABPNT  /ADDR OF I/O TABE
AOPTPNT, OPTPNT  /ADDR OF OPTION TABLE
  0 /NO ASSUMED EXTENSION
  MEMA ATABPNT
  ACCM ATEMP /LET'S FIND NUMBER OF FILES
  ZERM NINPUT
IINC, MPOZ @ ATEMP
  ZERZ
  JMP INC10 /DONE
  MPOM NINPUT /BUMP NUMBER OF FILES
  MEMA (3
  A+MM ATEMP
  JMP IINC
ZDISK, 7612
INC10, MPOM ATEMP /START OF OUTPUT FILES
  JMS FIRFLE /READ IN FIRST BUFFER
  JMS OUTSET /SET UP FOR OUTPUT
  MEMA ("C
  JMS OPTEST
  JMP CORE /CORE IMAGE FILE
  MEMA ("B
  JMS OPTEST
  JMP BIN /CONVERT CORE IMAGE TO BINARY
  JMS FETMC /JUST TRANSFER
  JMS PUTC
  JMP #-2

/CORE IMAGE FILE
CORE, MMOZ NINPUT
  JMP TOOCOR /MORE THAN 1
COR100, MEMA @ DBPNT /GET A WORD
  ACCM @ OUTPNT /STORE IT
  MPOM DBPNT
  MPOM OUTPNT
  MMOMZ IARG2
  JMP COR100 /GO AGAIN
  MEMZ DEVEND
  JMP COR200 /END OF FILE FLAG SSET
  JMS OUTTRN /SSTO FILE
  JMS IOTRN /GET NEW ONE
  JMS OTSPNT /SET UP OUTPUT POINTERS
  JMP COR100
COR200, MEMA IARG2A /REMAINDER
  ACCM OARG2
  M-AA C3000
  A+MA TOTCNT /MANIPULATE TOTAL COUNT
  JMS OUTTRN
  JMP CLS300  /CLOSE FILE

/SAVE 3000-7577
SAVE, 0
   JEA ONEA
  JMS DISTRN
  JMP @ SAVE
/RESTORE 3000-7577
```

```
  STOP /IMPOSSIBLE RETURN
  JMS RESTORE /RETURN CORE
  ZERAM @ ZERRFLG /CLEAR ERROR FLAG
  MEMA @ ZOARG2 /GET WORD COUNT
   XCT AC19 EXCT
   GA  /TAKE ABSOLUTE VALUE IF MINUS ANGA
  ACCM EMPCNT
  MEMA @ ZOARG1 /STARTING TRACK
  ACCM CLSTRK
  ACCM OARG3
  ZERM @ ZDEVDIR
  ZERM TOTCNT
  MEMA C3000
  ACCM OARG2
OUTS30, JMS OTSPNT /SET UP OUTPUT POINTERS
  JMP @ OUTSET
OUTS10, MONM POUTFG /SET PAPER TAPE FLAG
  MONM FIRFLG
  JMP OUTS30-2

/SET UP FOR INPUT TRANSFER
DEVSET, 0
  MEMA @ LSTADD /DEVICE
  EXCT MOAC
  JMP CLSFLE
  ACCM IARG1 /DEVICE
  A-MA (5
  SKIP AC19
  JMP DEVPT /SET UP FOR PAPER TAPE DEVICE
  MEMA ("H
  JMS OPTEST
  JMS TRKCAL /FILE > THAN 50 TRACKS
   POM LSTADD  MPOM
   EMA @ LSTADD MEMA
  ACCM IARG3 /STARTING TRACK
  MPOM LSTADD
  MEMA @ LSTADD /WORD COUNT
  ACCM IARG2A
  MPOM LSTADD /BUMP TO NEXT ENTRY
/SET UP RETURN ROUTIES FOR DISK
  MEMA (HARDER-IOSTRT
  ACCM ERRARG
  MEMA CBUMP
  ACCM ERRARG+1
  ZERM DEVEND /CLEAR END OF FILE FLAG
  JMP @ DEVSET
/SET UP FOR PAPER TAPE DEVICES
DEVPT, MEMA (3
  A+MM LSTADD
  MEMA (364
  ACCM @ Q6333 /MAKE SURE IT INITIALIZES
  ACCM @ Q6354
  ZERM DEVEND
  MEMA DPFST  /LARGE EMPTY SPACE
  ACCM IARG2A
  MEMA (IOTT10-IOSTRT /SET UP RETURN FOR PAPER TAPE
  ACCM ERRARG
  MEMA (IOTT20-IOSTRT
  ACCM ERRARG+1
  JMP @ DEVSET
/ERROR RETURN FOR IOTRN PAPER TAPE DEVICE (OUT OF TAPE)
I  T10, ZERM @ ZERRFLG  IOTT10,
  MONM DEVEND /SET END OF DEVICE FLAG
  JMP ERRARG+2
/NORMAL RETURN
```

```
  JMS @ ZDISK
  1012  / MODIFIED FOR FLOPPY
  1000
  6000
  MA NOP /LOOK IN CORE  MEMA
   CM @ PIN  ACCM
  JMP @ IOFTCH

/CALL IN DIRFUN
DIRIN, 0
  JMS SAVE
  ZERA
  JMS @ ZDISK
  1007  / MODIFIED FOR FLOPPY
  600
  7000
  JMP @ DIRIN

/CLOSE OUTPUT FILE
CLSFLE, MEMZ POUTFG /DON'T CLOSE PAPER TAPE
  JMP CLSPT /FINISH OUT WHATEVER
  JMS FINBUF /FILL BUFFER WITH ZEROS
CLS300, MEMA CLSTRK /CLOSE FILE
  ACCM @ ZOARG1
  MEMA TOTCNT /TOTAL NUMBER OF WORDS
  ACCM @ ZOARG2
  MEMA DPFST /BUFFER ADDRESS
  ACCM @ ZOARG3
  MEMA Y7600
  ACCM @ ZSYSTRT
  MEMA @ ATEMP /DEVICE
  ACCM CLS100
  OMA ATEMP /ADDRESS OF FILENAME  MPOMA
   CM CLS200    ACCM
  ZERM @ ZDEVDIR
  JMS DIRIN
  JMS @ ZDIRFUN /DO IT
CLS100, 0 /DEVICE
  1  /CLOSE
CLS200, 0  /POINTER TO FILENAME
  JMP NOROOM
  JMS RESTORE /RESTORE CORE
  MEMA (3
  A+MM ATEMP /FOR NEXT DEVICE
  JMP IOSTRT
CLSPT, ZERA /PUT A ZERO
  JMS PUTC
  MEMA OUTCNT
  M-AA C3000 /HOW MANY ARE THERE
  ACCMZ OARG2
  JMS OUTTRN /OUTPUT LAST BUFFER
  MEMA (4
  A+MM ATEMP /BUMP TO NEXT ENTRY
  JMP IOSTRT

/FINISH BUFFER
FINBUF, 0
  MNGA OUTCNT
  A+MA C3000 /# OF LOCATIONS LEFT
  EXCT ZAC
  MP@ FINBUF
   RA  ZERA
  MS PUTC
  JMP FINBUF+1
```

```
   MPOM TRK100
   TACMQ   /CONVERT TO WORDS
   MULT
   2000    / MODIFIED FOR FLOPPY WAS 3000
   SKIP ZAC
   JMP NOROOM
   TMQAC
   ACCM @ TRK100 /REALISTIC WORD COUNT
   JMS RESTORE
   JMP @ TRKCAL
/FETCH CHAR ROUTINE
FETMC, 0
   MEMA BCPNT  /CHAR ROUTINE POINTER
   A+MA CROUT
FET100, ACCM FETADD /CALCULATE ADDRESS OF ROUTINE
   MEMA @ FETADD /GET ADDRESS OF ROUTINE
   ACCM FETADD
   MEMA PUT300
   JMP @ FETADD
CROUT, CRLST
CRLST, CHAR0
 CHAR1
 CHAR2
 CHAR3
 CHAR4
CHAR0, MEMA @ DBPNT /GET WORD FRON DISK BUFFER
   LLSH 10
   JMP FCHEK /SEE IF FORM FEED
CHAR1, MEMA @ DBPNT
   RISH 4
   JMP FCHEK
CHAR2, MEMA @ DBPNT
   ANDA (17  /MASK FIRST PART
   LLSH 4
   ACCM FETADD /TEMP STORAGE
   MPOM DBPNT  /ACCESS NEXT BUFFER WORD
   MEMA @ DBPNT
   LLSH 4
   ANDA (17
   A+MA FETADD
   JMP FCHEK /CHECK FOR FORM FEED
CHAR3, MEMA @ DBPNT
   RISH 10
   JMP FCHEK
CHAR4, MEMA @ DBPNT
   MPOM DBPNT  /ACCESS NEXT WORD
   MONM BCPNT
FCHEK, ANDA (377
   ACCM FETADD
NOFORM, MPOMA BCPNT
   MEMA DBPNT  /DONE?
   A-MZ OARG4  /DONE WITH BUFFER?
   ZERZ
   JMS IOTRN  /GET NEW ONE
   MEMA FETADD /RETURN WITH CHAR IN AC
   JMP @ FETMC



/PUT CHARACTER INTO DISK BUFFER
PUTC, 0
   ACCM PUT300  /SCR
   MEMA BCPNTO
   A+MA CROUTO
   JMP FET100  /LET FETMC DO REST OF WORK
```

```
UNTYPE, 0
  ANDA (77
  A-MZ (77
  ZERZ
  JMP @ UNPCK /FOUND TERMINATOR
  MEMA (240
  JMS TYPE
  MEMA RESTORE
  JMP @ UNTYPE

/PRINT A CHAR
TYPE, 0
  TTYPF
  JMP #-1
  PRTTY
  JMP @ TYPE

/CRLF
CRLF, 0
  MEMA (215
  JMS TYPE
  MEMA (212
  JMS TYPE
  JMP@ CRLF

/ERROR MESSAGES
HARDER, JMS UNPCK  /HARDWARE ERROR
  MHARD
  JMP @ Y7600  /RETURN TO MONITOR
NOUT, JMS CRLF
  JMS UNPCK  /NO OUTPUT FILE
  MNOUT
  JMP IOSTRT
NOROOM, JMS UNPCK /NO ROOM ON DISK
  MNOROOM
  JMP @ Y7600
TOOCOR, JMS CRLF
  JMS UNPCK /MORE THAN 1 CORE IMAGE FILE
  MTOOCOR
  JMP IOSTRT
MTOOCOR, TEXT %MORE THAN ONE CORE IMAGE FILE!%
MHARD,   TEXT %HARDWARE ERROR! %
MNOUT,   TEXT %NO OUT PUT FILE?%
MNOROOM, TEXT %NO ROOM ON DISK!%
/OUTPUT IN BINARY FORMAT
BIN, MMOZ NINPUT
  JMP TOOCOR  /ONLY ONE CORE IMAGE FILE ALLOWED
  ZERM @ ZDEVDIR
  JMS DIRIN /WE HAVE TO LOOK UP BUFFER ADDRESS
  MEMA ATABPNT
  ACCM LEADER
  MEMA @ LEADER  /GET DEVICE
  ACCM FAK100
  A-MA (5  /CHECK FOR ILLRGAL INPUT
  SKIP AC19
  JMP ILLIN  /CAN'T READ CORE IMAGE IN FROM PAPER TAPE
  JMS @ ZDIRFUN
FAK100, 0 /DEVICE
  2  /DUMMY SEARCH
  ZPNT  /ZERO FILE NAME
  ACCA /PROBABLY RETURNS HERE
  ZERM @ ZERRFLG  /CLEAR ERROR FLAG
  MPOM LEADER /GET STARTING TRACK
  MEMA @ LEADER
  ACCM @ ZTRCK
```

```
JMP @ HBINP
```