Display Control Panel /

Graphics Controllerr

96 SHEETS • 5 x 5 QUAD
10⅛ x 7⅞ • 53-110

1984

NATIONAL BLANK BOOK COMPANY, INC.
Holyoke, Massachusetts 01040 • Made in USA

Display Control Panel / Graphics Controller

Memory Panel

4 planes of 512 x 512 pixels -

- each plane may be
cleared
written to
disabled
read

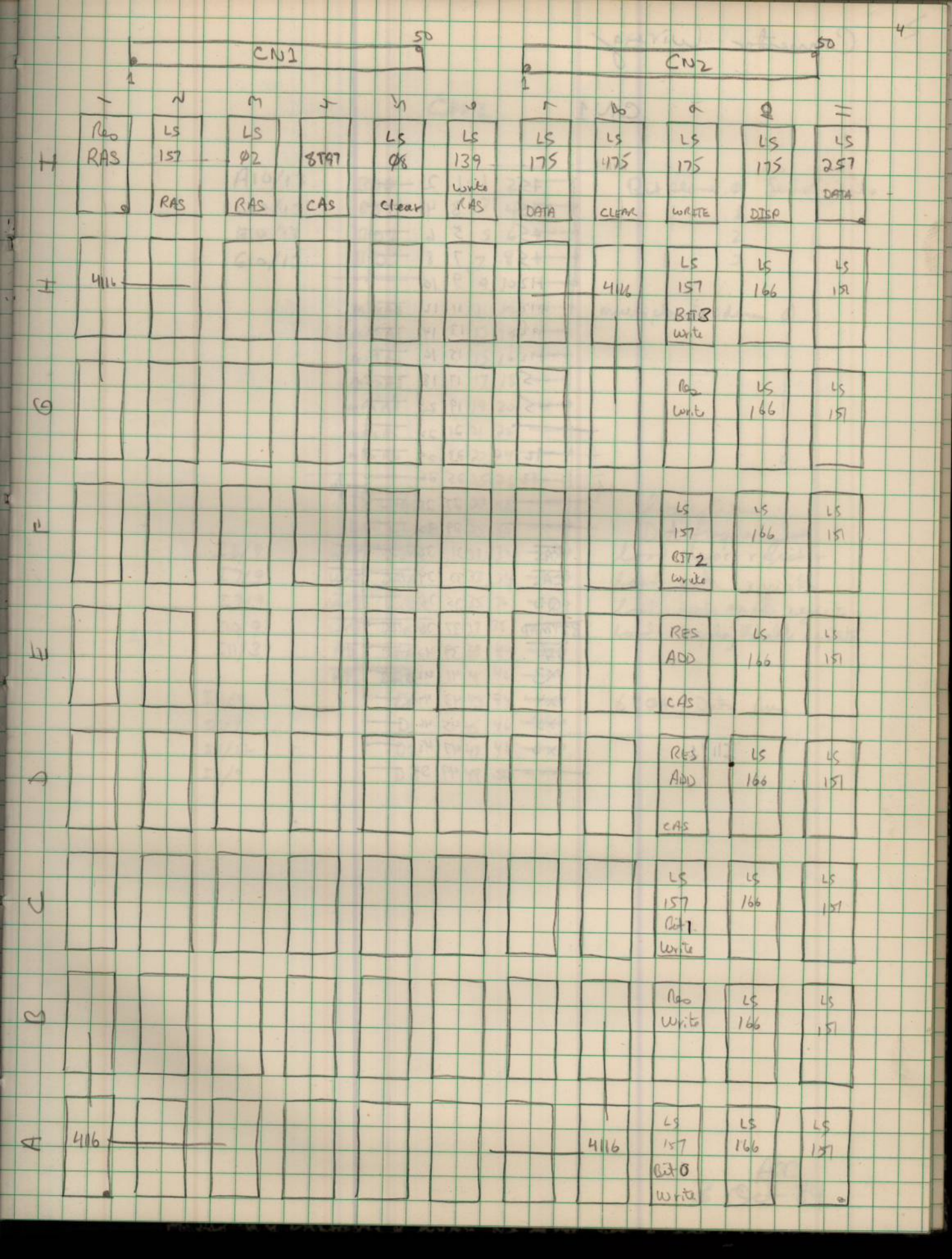
independently -

this organization allows 4 independent monochrome planes -

or combinations of color planes

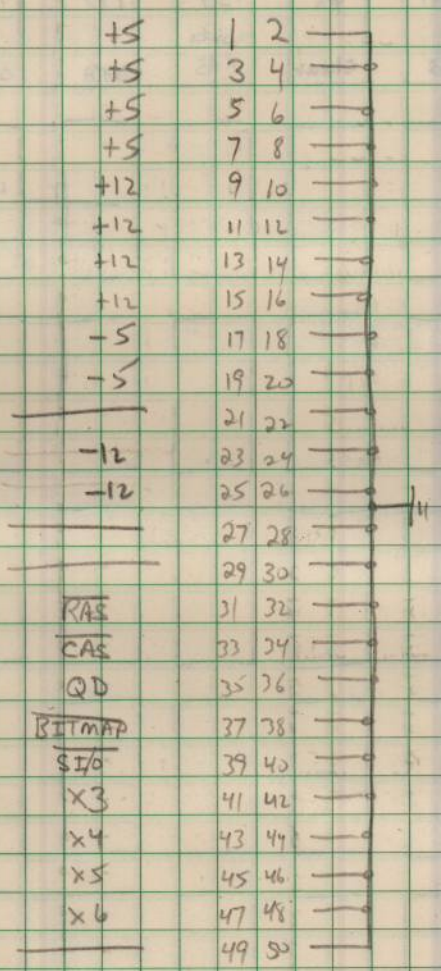
ABD
30 Oct 84

Board Layout –


CN1
50
1
CN2
50
1
M
N
P
R
S
T
U
V
Neo RAS
LS 157 RAS
LS Φ2 RAS
8T97 CAS
LS Φ8 Clear
LS 139 Write RAS
LS 175 DATA
LS 175 CLEAR
LS 175 WRITE
LS 175 DISP
LS 257 DATA
4116
4116
LS 157 BIT3 Write
LS 166
LS 151
Neo Write
LS 157 BIT2 Write
RES ADD CAS
LS 157 Bit1 Write
LS 157 Bit0 Write
H
G
F
E
D
C
B
A

Connector Wiring

CN1

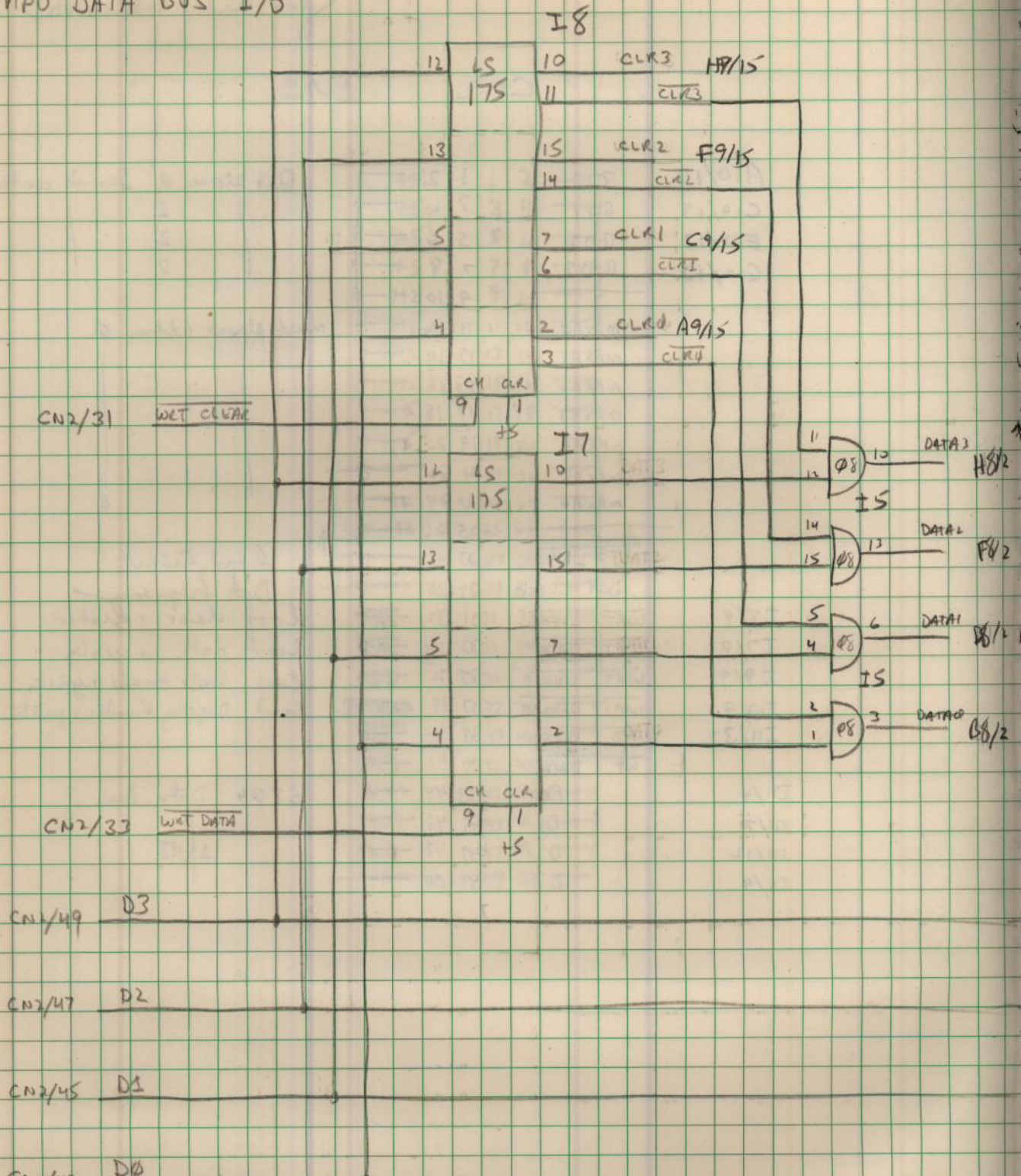


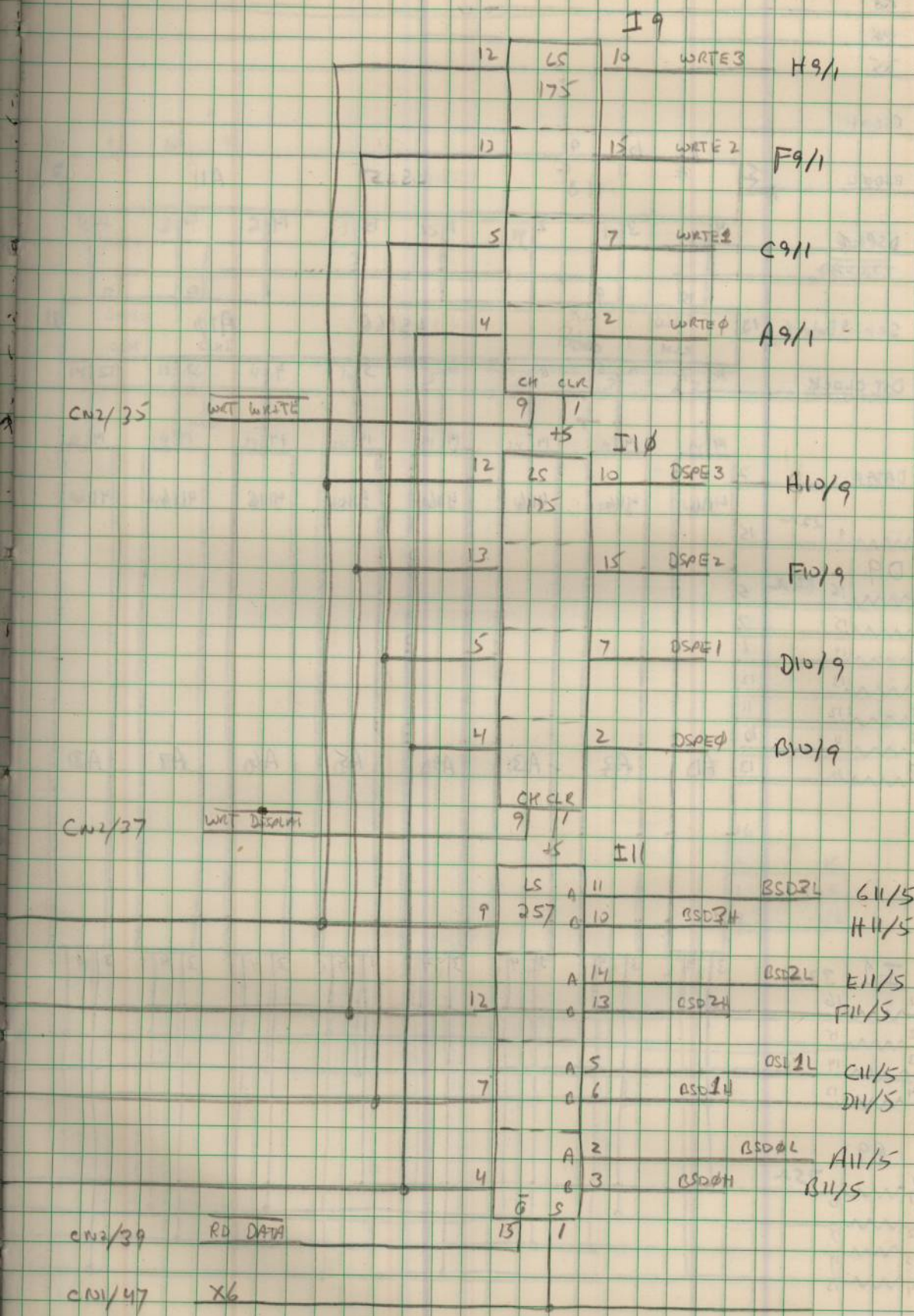
III/1

CN2

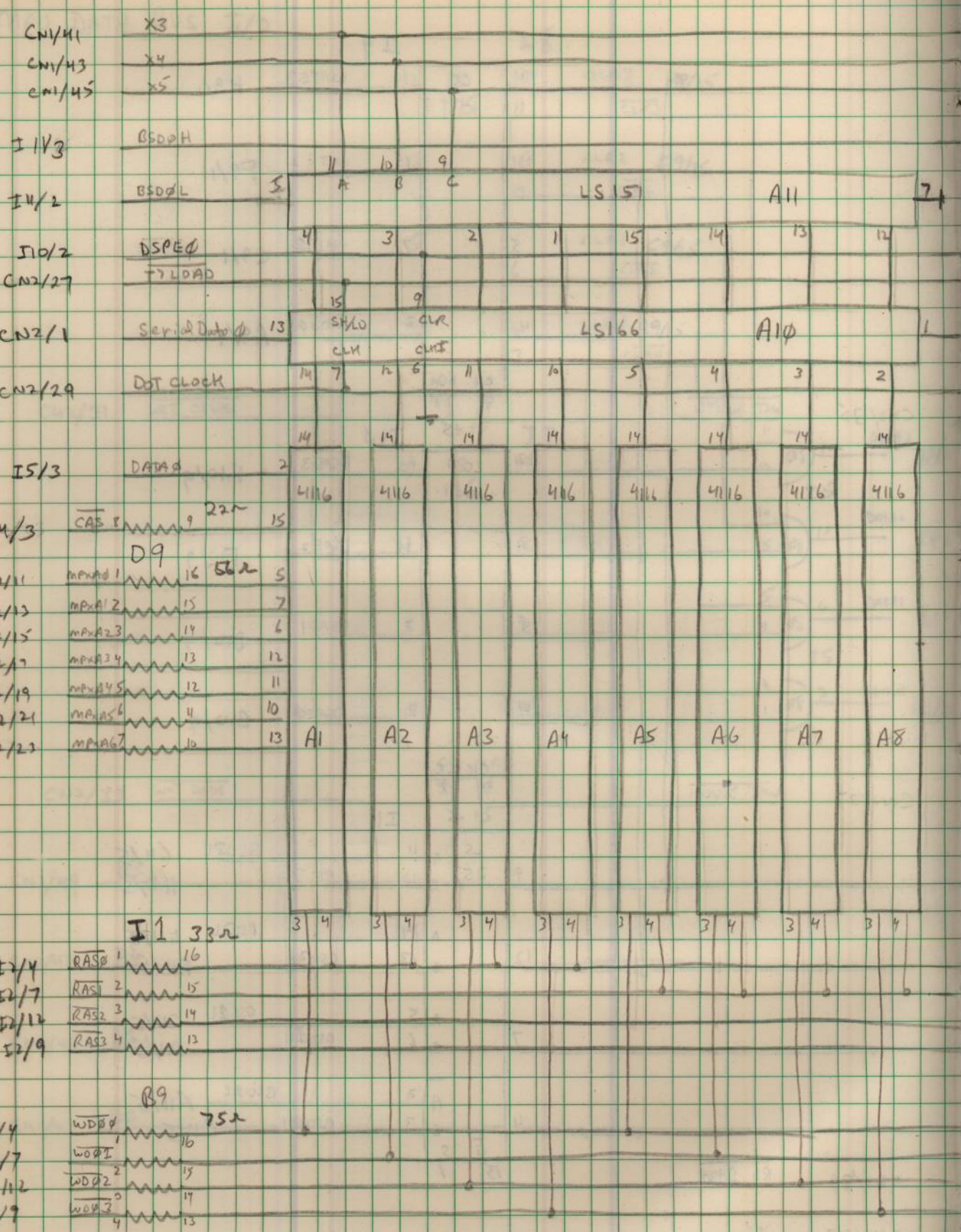
A10/13	BD0	1	2	Bit plane 0 Serial data
C10/13	BD1	3	4	1
E10/13	BD2	5	6	2
G10/13	BD3	7	8	3
		9	10	
	MAXA0	11	12	multiplexed Address 0
	MAXA1	13	14	
	MAXA2	15	16	
	MAXA3	17	18	
	MAXA4	19	20	
	MAXA5	21	22	
	MAXA6	23	24	6
		25	26	
	T7 LOAD	27	28	load time
	DOT CLOCK	29	30	Dot frequency
I8/9	WRT CLEAR	31	32	load clear register
I7/9	WRT DATA	33	34	load data register
I9/9	WRT WRITE	35	36	load write enable register
I10/9	WRT DISPLAY	37	38	Load Display Enable register
I11/39	RD DATA	39	40	
	WT DATA	41	42	
I11/4	D0	43	44	6809 Data Bus
I11/7	D1	45	46	
I11/12	D2	47	48	
I11/9	D3	49	50	

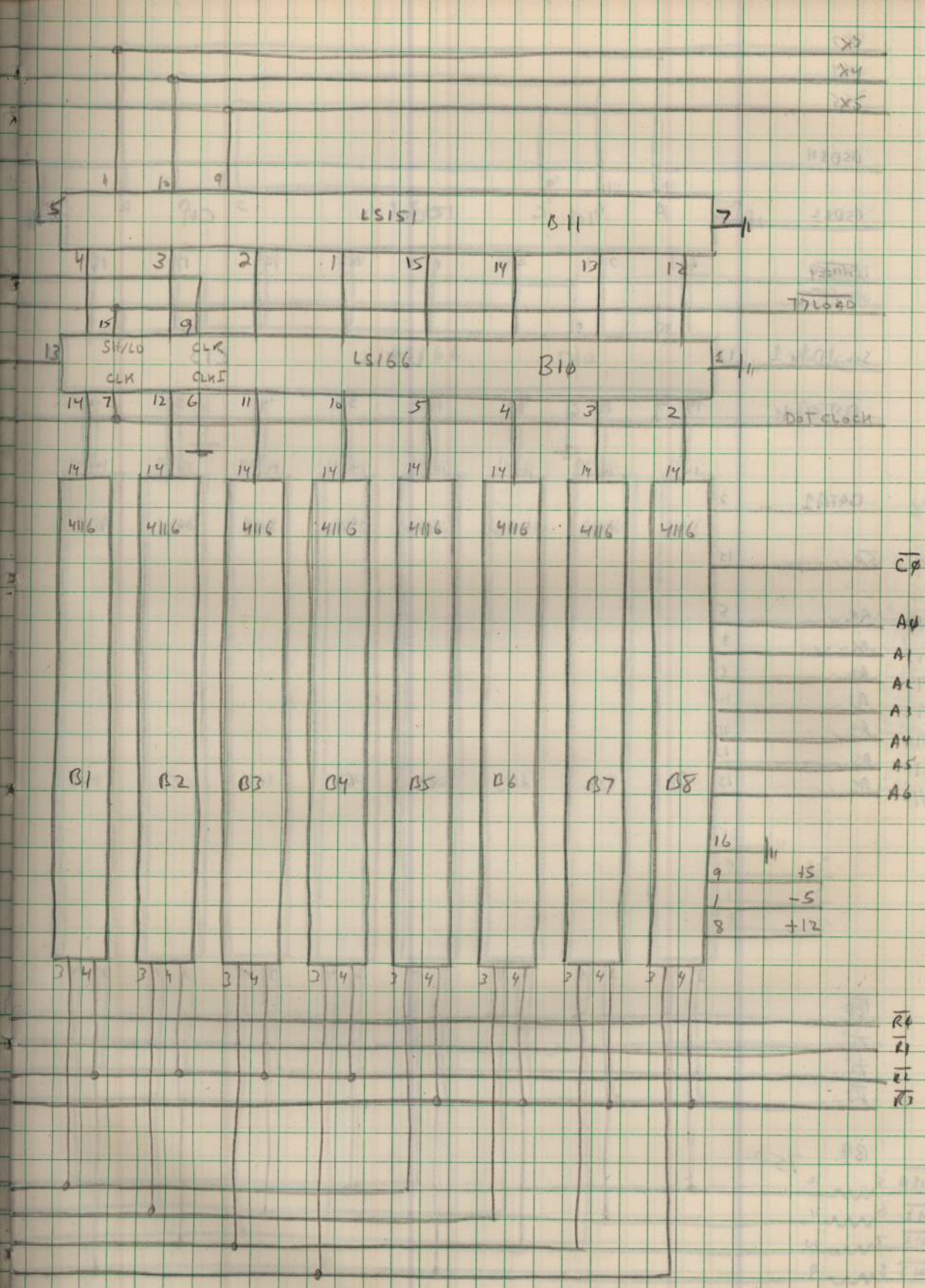
ADD
3 Oct 84


MPU DATA BUS I/O
I8
LS 175
CLR3
CLR2
CLR1
CLR0
CK
CLR
CN2/31
WRT CLEAR
I7
CN2/33
WRT DATA
I5
DATA3
DATA2
DATA1
DATA0
CN2/49
D3
CN2/47
D2
CN2/45
D1
CN2/43
D0

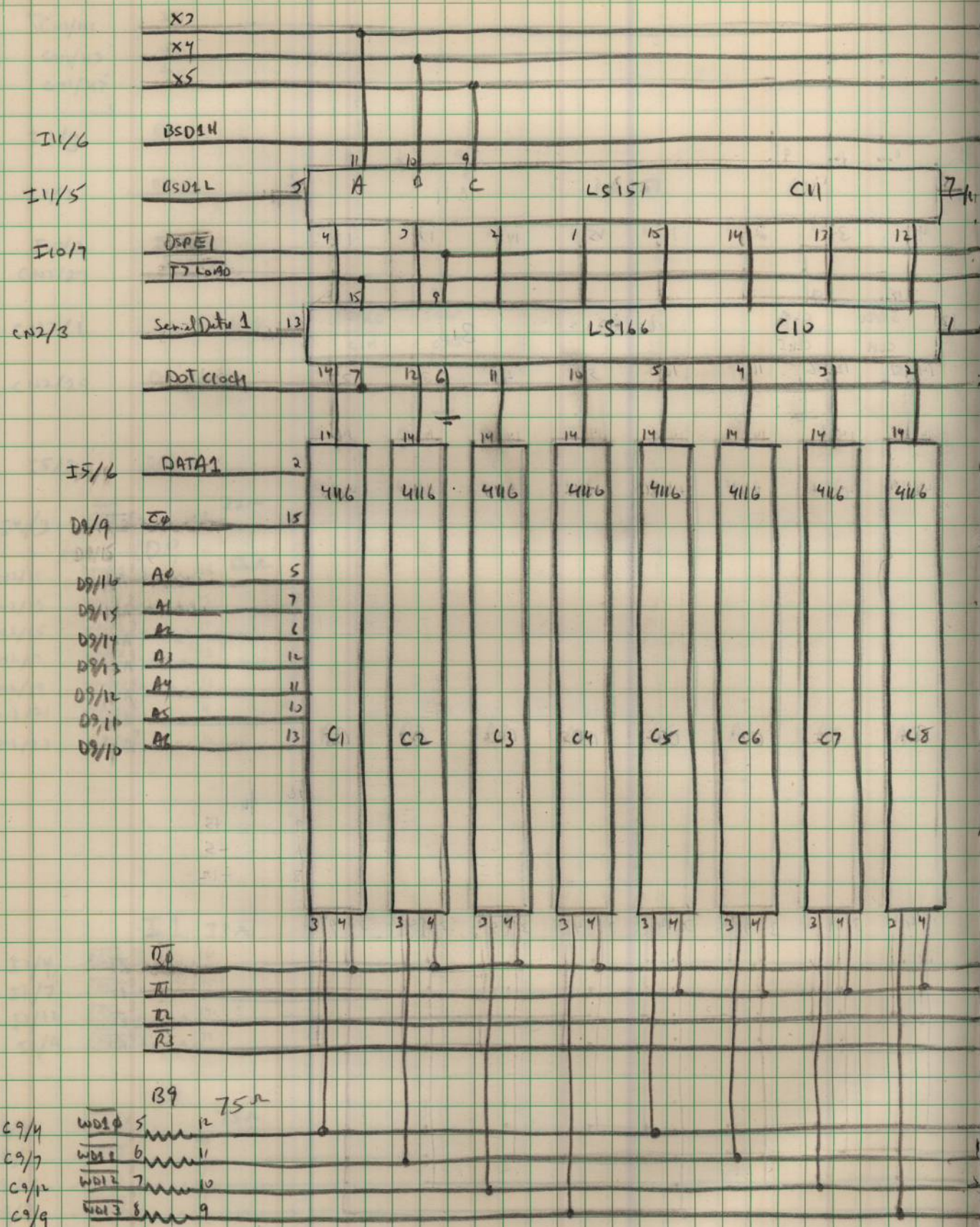

I9
LS 175
12
10
WRTE3
H9/1
13
15
WRTE2
F9/1
5
7
WRTE1
C9/1
4
2
WRTEØ
A9/1
CK CLR
9
1
+5
CN2/35
WRT WRITE
I1Ø
LS 125
12
10
DSPE3
H10/9
13
15
DSPE2
F10/9
5
7
DSPE1
D10/9
4
2
DSPEØ
B10/9
CK CLR
9
1
+5
CN2/37
WRT DISPLAY
I11
LS 257
A
B
11
10
BSD2L
BSD2H
G11/5
H11/5
9
12
A
B
14
13
BSD2L
BSD2H
E11/5
F11/5
A
B
5
6
BSD1L
BSD1H
C11/5
D11/5
7
A
B
2
3
BSDØL
BSDØH
A11/5
B11/5
4
Ḡ
S
15
1
CN2/39
RD DATA
CN11/47
X6

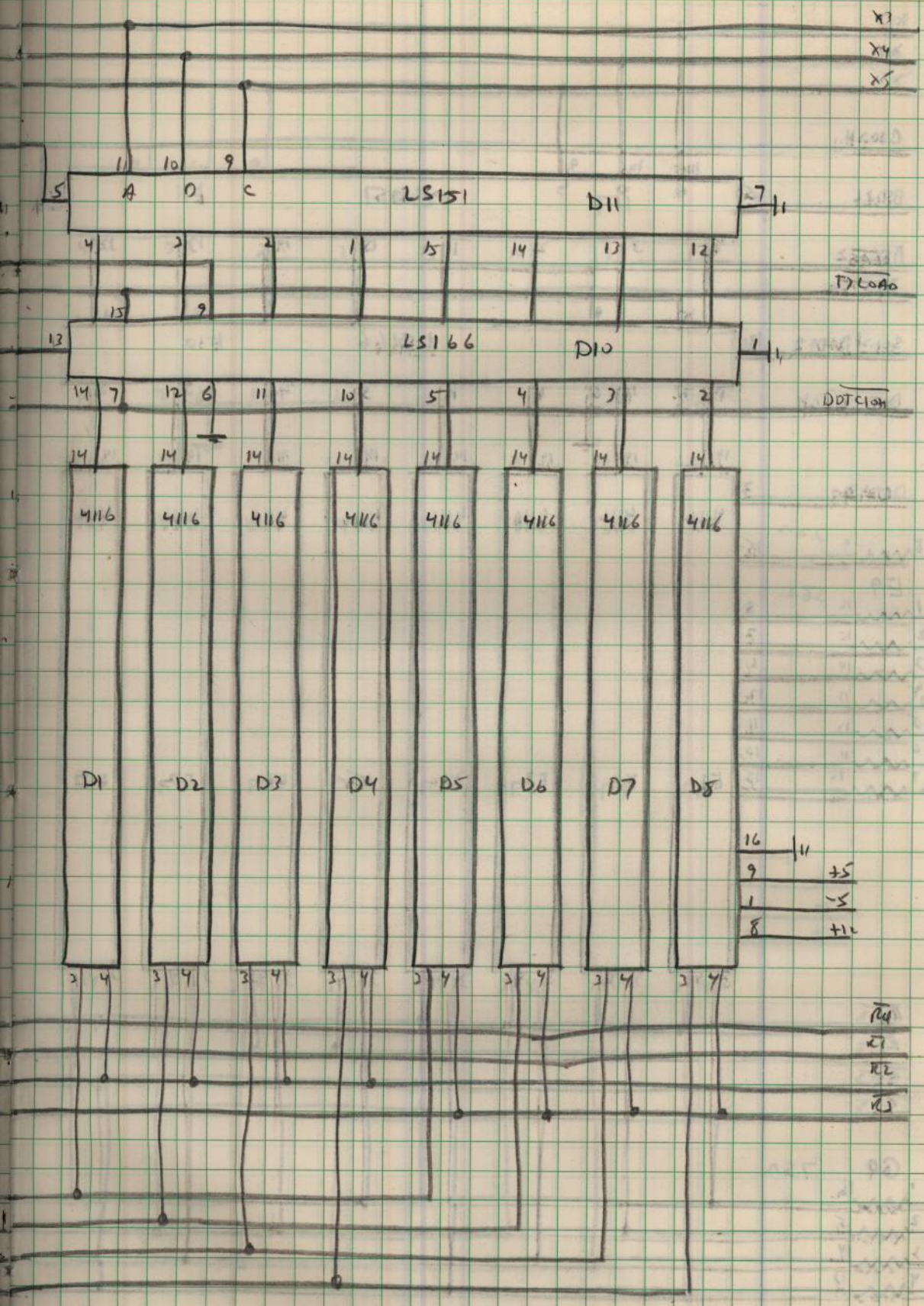
30 Oct 84
BRD


CN1/41
X3
CN1/43
X4
CN1/45
X5
I1/3
BSDØH
I4/2
BSDØL
I10/2
DSPEØ
CN2/27
FJ LOAD
CN2/1
Serial Data Ø
CN2/29
DOT CLOCK
I5/3
DATAØ
I4/3
CAS
D9
22Ω
CN2/11
MPXAØ 1
56Ω
CN2/13
MPXA1 2
CN2/15
MPXA2 3
CN2/17
MPXA3 4
CN2/19
MPXA4 5
CN2/21
MPXA5 6
CN2/23
MPXA6 7
LS157
A11
LS166
A10
A1
A2
A3
A4
A5
A6
A7
A8
4116
I1
33Ω
I2/4
RASØ
I2/7
RAS1
I2/12
RAS2
I2/9
RAS3
R9
75Ω
A9/4
WDØØ
A9/7
WDØ1
A9/12
WDØ2
A9/9
WDØ3

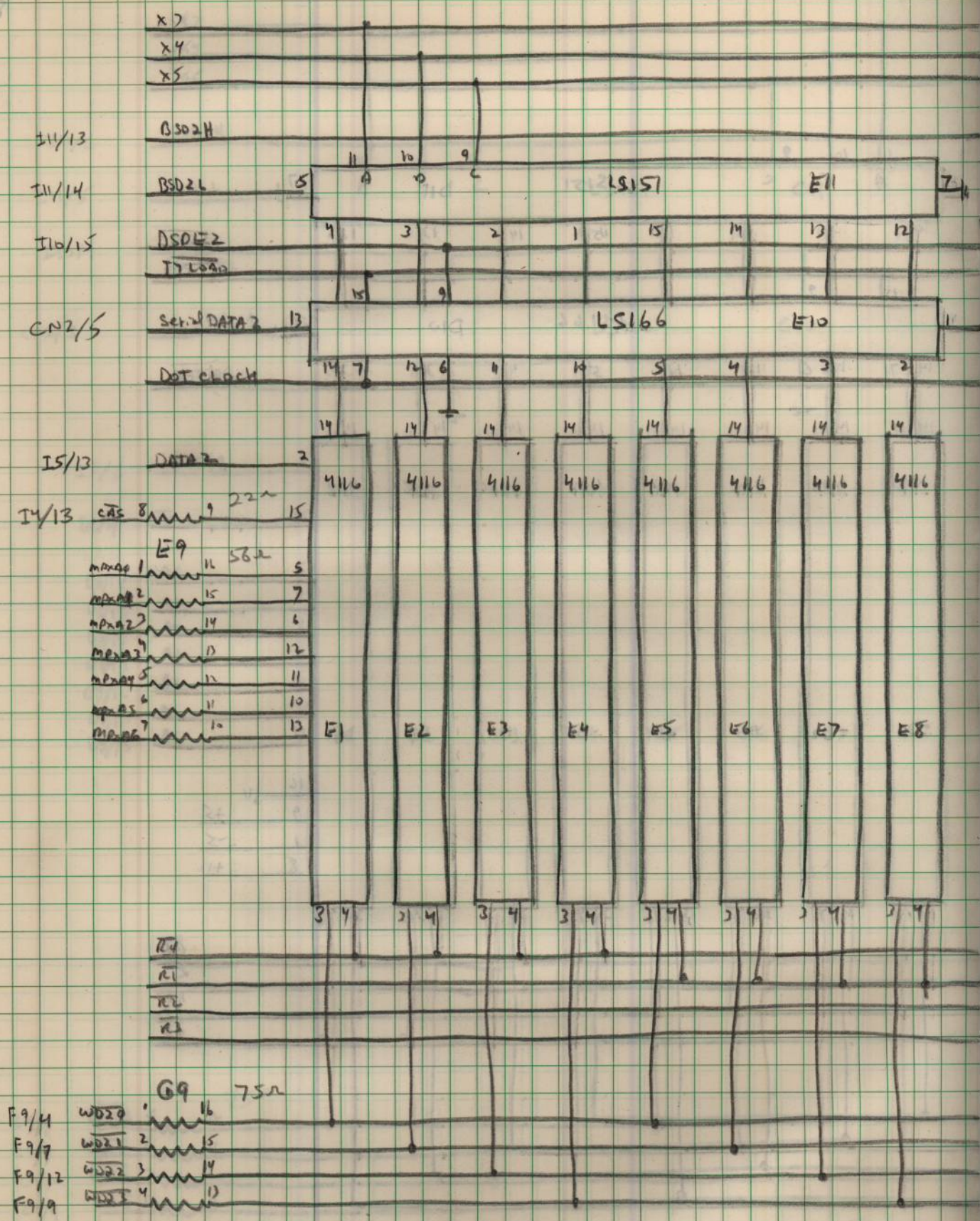

X3
X4
X5
LS151
B11
LOAD
LS166
B10
SH/LD
CLK
CLR
DOT CLOCK
4116
B1
B2
B3
B4
B5
B6
B7
B8
+5
-5
+12
A0
A1
A2
A3
A4
A5
A6

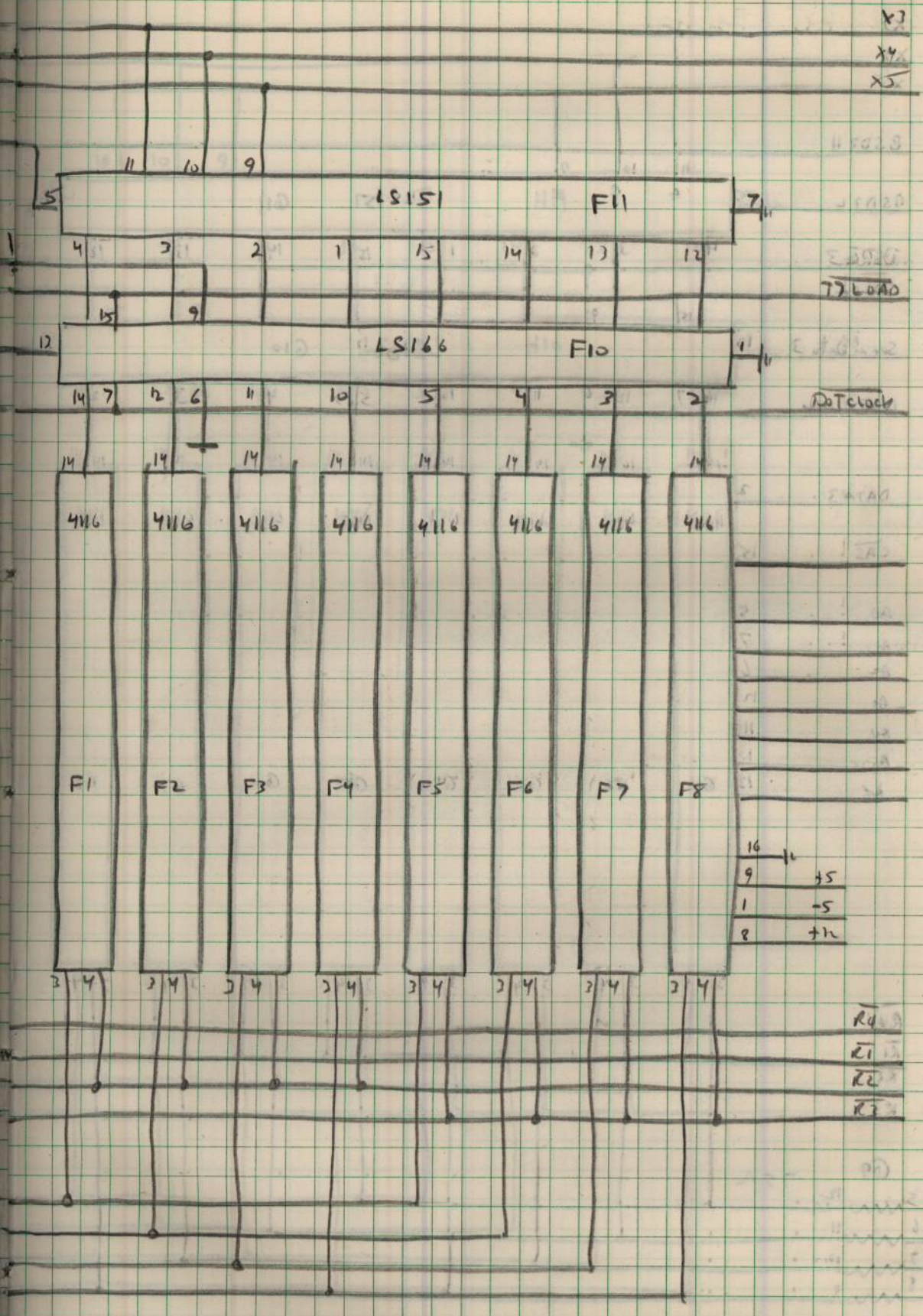
7 Oct 84
ARB


X3
X4
X5
I11/6
BSD1H
I11/5
BSD2L
A
B
C
LS151
C11
I10/7
DSPEI
T7 LOAD
CN2/3
Serial Data 1
LS166
C10
DOT CLOCK
I5/6
DATA1
4116
D9/9
D9/16
D9/15
D9/14
D9/13
D9/12
D9/11
D9/10
C1
C2
C3
C4
C5
C6
C7
C8
B9
75Ω
C9/4
C9/7
C9/12
C9/9

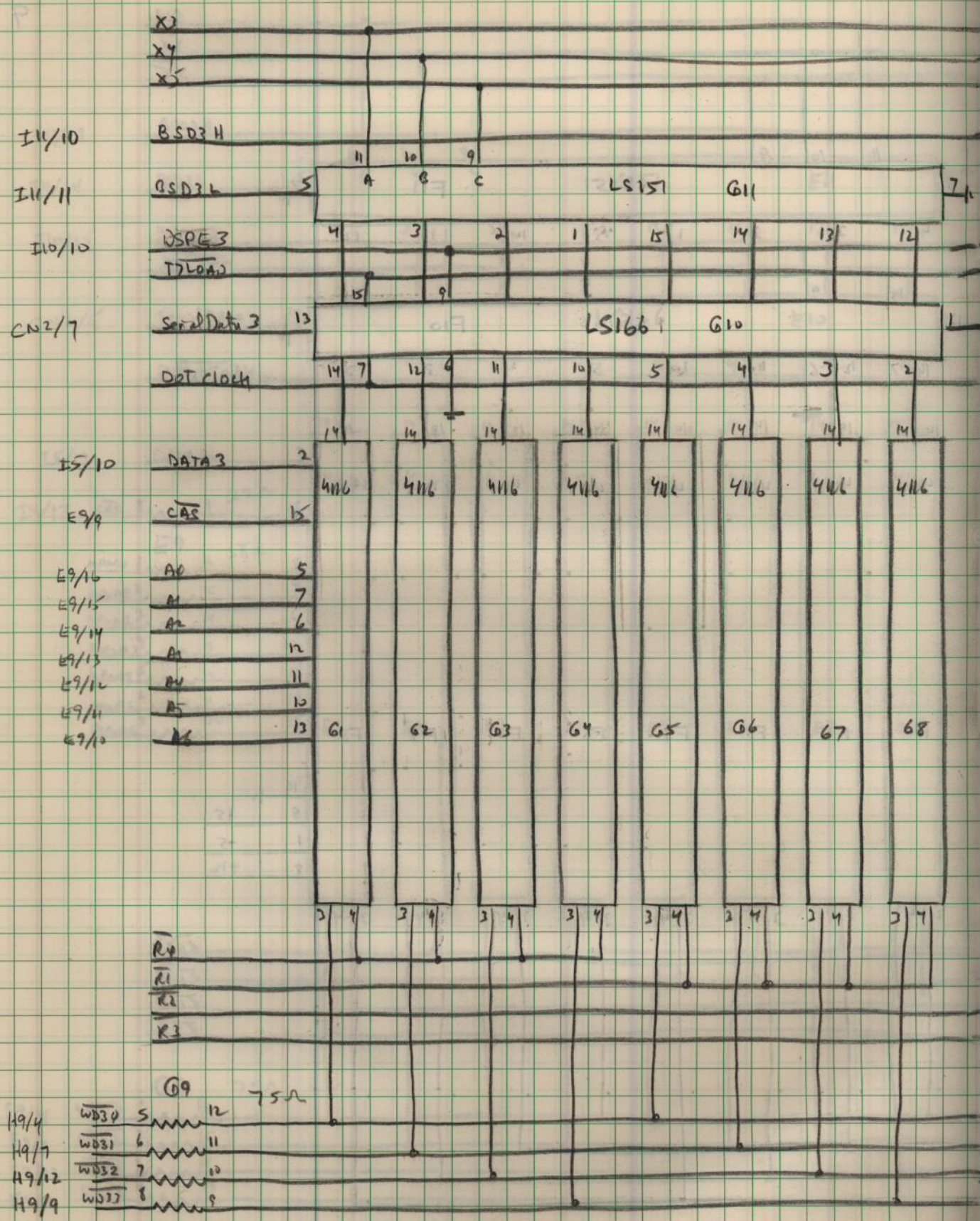

X3
X4
X5
A
B
C
LS151
D11
LOAD
LS166
D10
DOTCLOCK
4116
D1
D2
D3
D4
D5
D6
D7
D8
+5
-5
+12
RA4
RA1
RA2
RA3

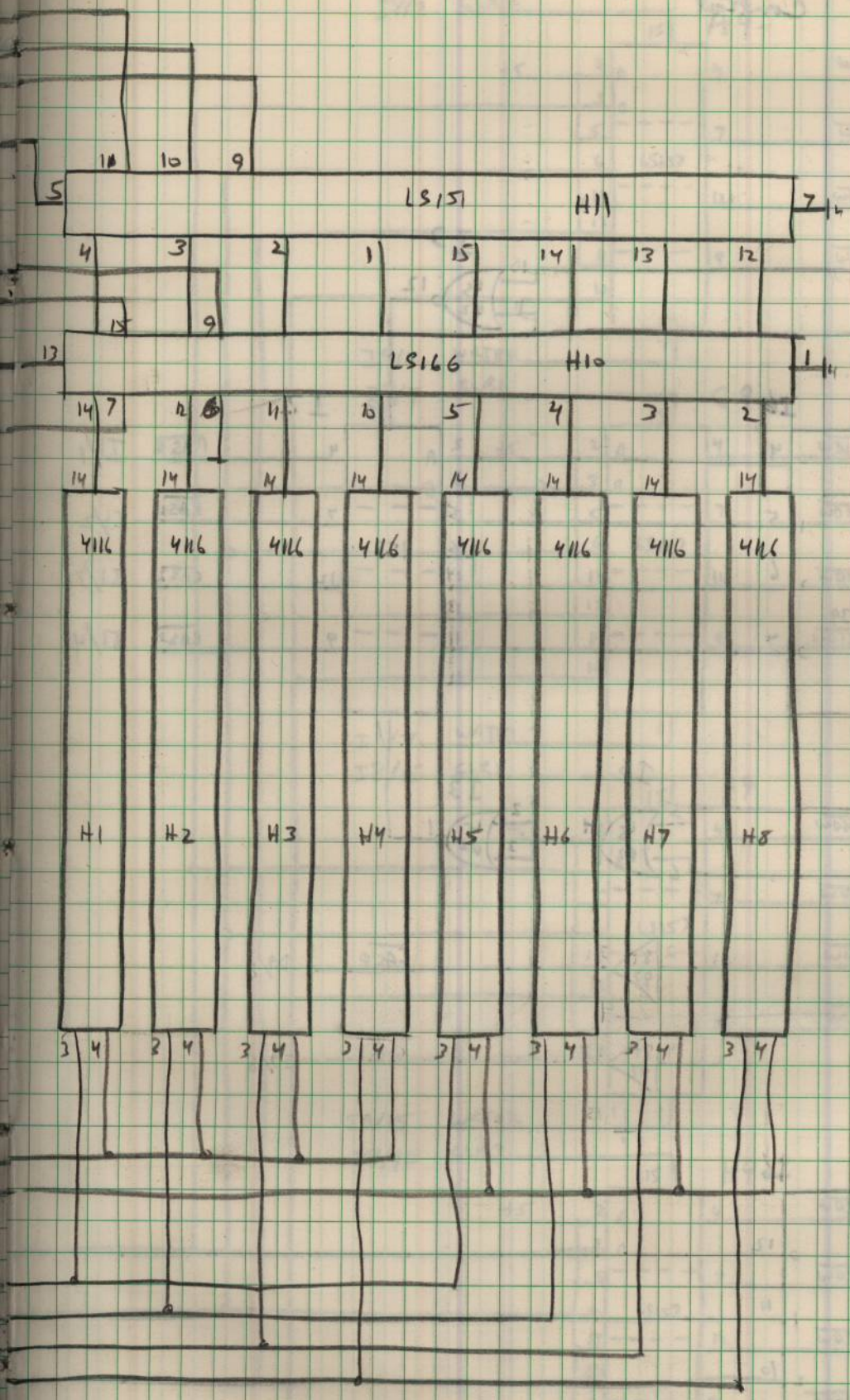
7 Oct 84
ARD


X3
X4
X5
I11/13
BS02H
I11/14
BS02L
LS151
E11
I10/15
DSOE2
TP LOAD
CN2/5
SERIAL DATA 2
LS166
E10
DOT CLOCK
I5/13
DATA 2
4116
I4/13
CAS
22Ω
E9
MAD0
MAD1
MAD2
MAD3
MAD4
MAD5
MAD6
56Ω
E1
E2
E3
E4
E5
E6
E7
E8
RA0
RA1
RA2
RA3
G9
75Ω
F9/4
F9/7
F9/12
F9/9
WD20
WD21
WD22
WD23


X3
X4
X5
LS151
F11
LS166
F10
LOAD
DOTclock
4116
F1
F2
F3
F4
F5
F6
F7
F8
+5
-5
+12
R0
R1
R2
R3

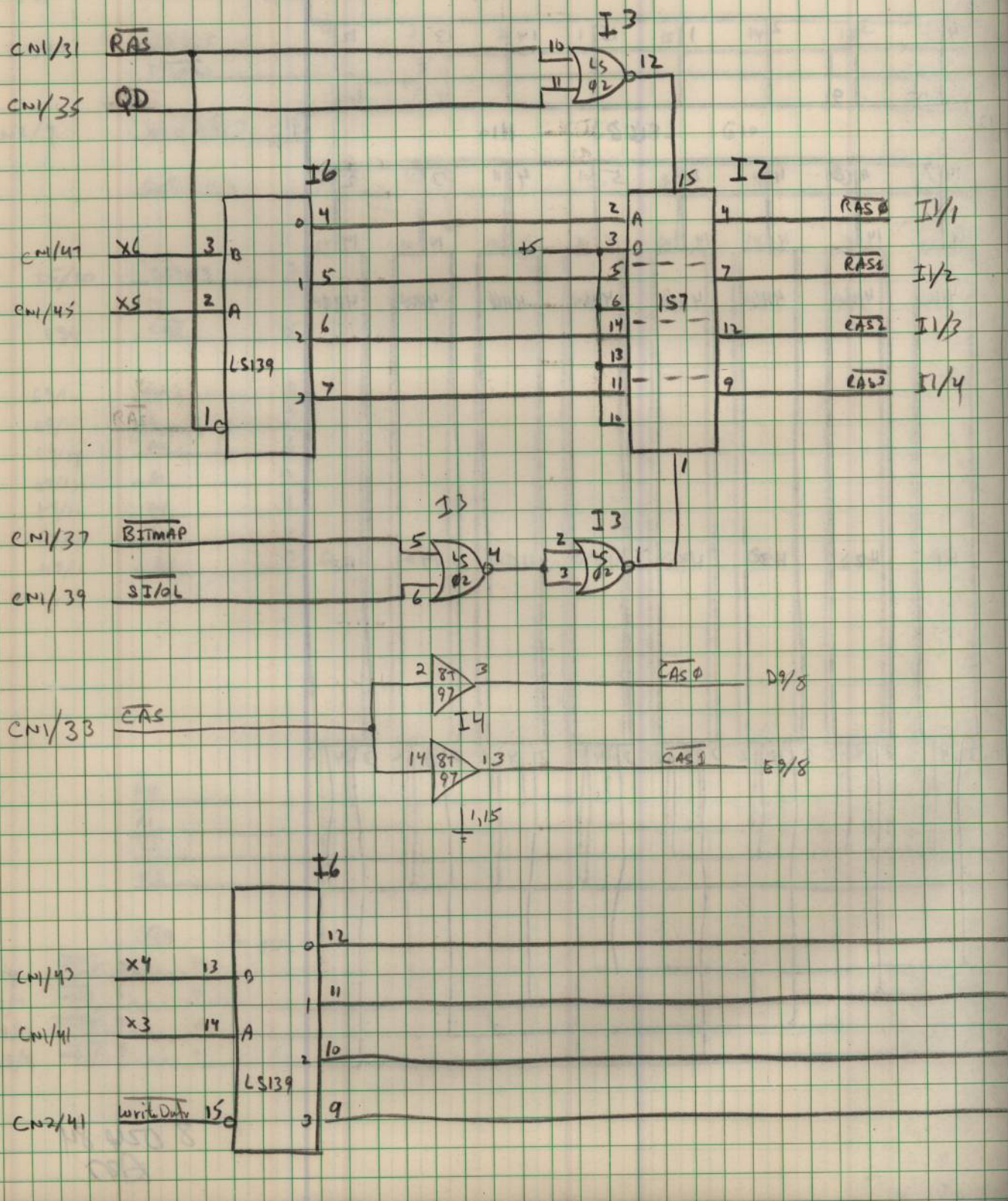
8024 24
ABS

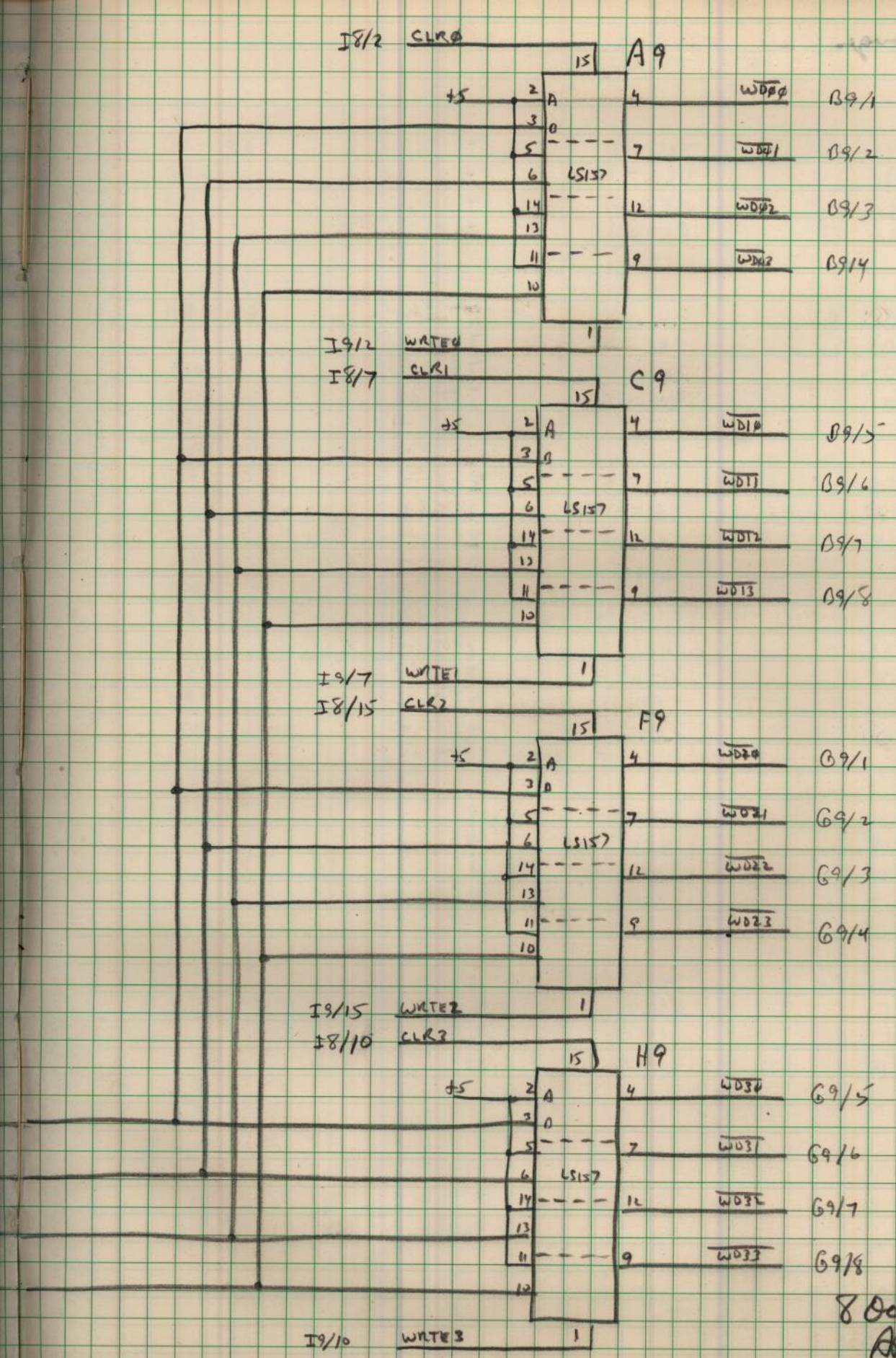

X3
X4
X5
I11/10
BSD3H
I11/11
BSD3L
I10/10
WSPE3
TDLOAD
CN2/7
Serial Data 3
DOT CLOCK
LS157
G11
LS166
G10
E5/10
DATA 3
E9/9
CAS
E9/16
E9/15
E9/14
E9/13
E9/12
E9/11
E9/10
A0
A1
A2
A3
A4
A5
A6
4116
G1
G2
G3
G4
G5
G6
G7
G8
R4
R1
R2
R3
G9
75Ω
H9/4
H9/7
H9/12
H9/9
WD30
WD31
WD32
WD33



8 02 84
AD

RAS and WT control




I8/2 CLR0
A9
+5
WD00
B9/1
WD01
B9/2
WD02
B9/3
WD03
B9/4
LS157
I9/2 WRITE0
I8/7 CLR1
C9
WD10
B9/5
WD11
B9/6
WD12
B9/7
WD13
B9/8
I9/7 WRITE1
I8/15 CLR2
F9
WD20
G9/1
WD21
G9/2
WD22
G9/3
WD23
G9/4
I9/15 WRITE2
I8/10 CLR3
H9
WD30
G9/5
WD31
G9/6
WD32
G9/7
WD33
G9/8
I9/10 WRITE3

8 Oct 84
ABD

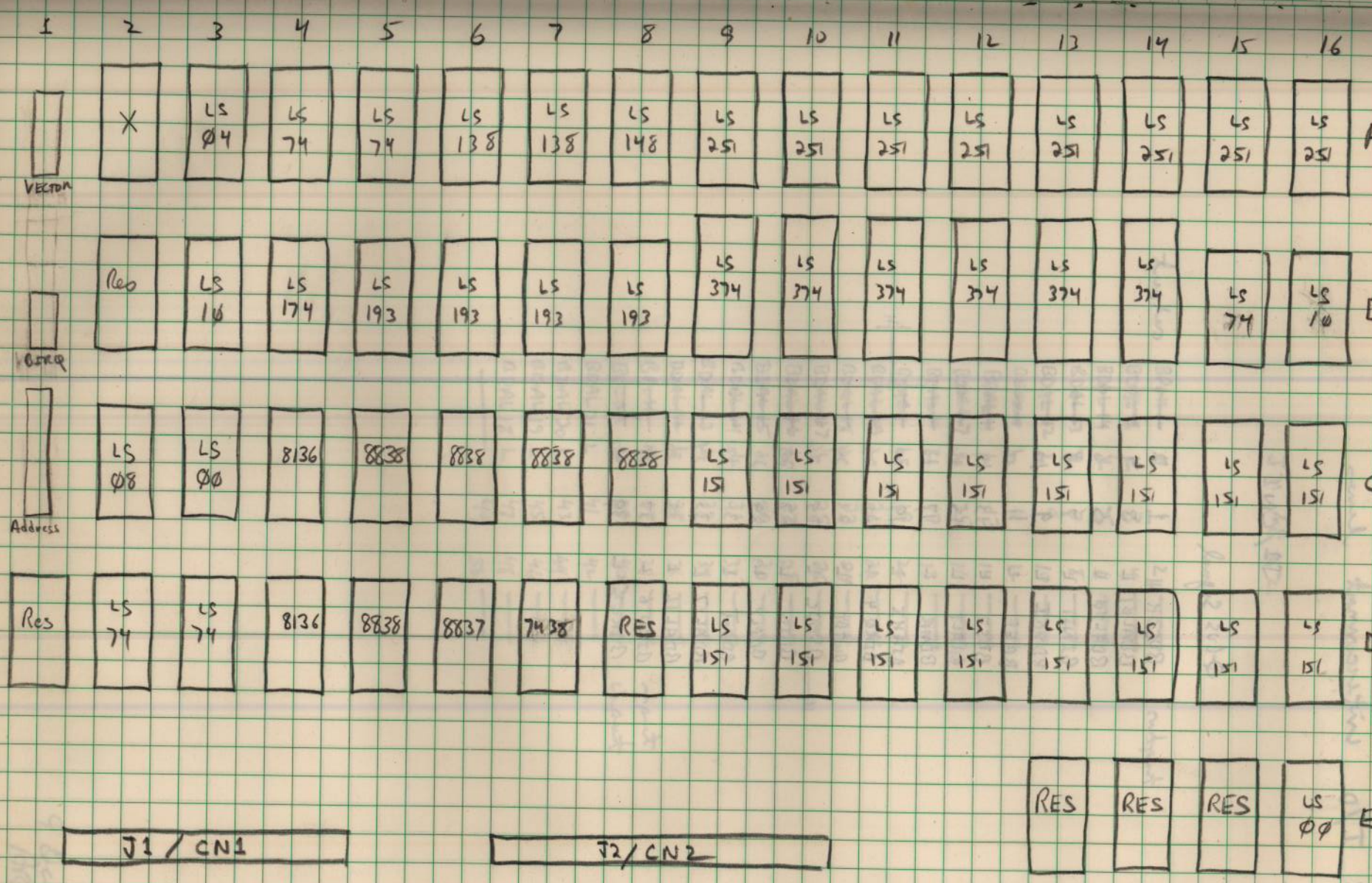
IC Delays

A1 - A8	4116	7
A9	LS157	11
A10	LS166	7
A11	LS151	7
B1 - B8	4116	7
B9	RES	7, 8
B10	LS166	7
B11	LS151	7
C1 - C8	4116	8
C9	LS157	11
C10	LS166	8
C11	LS151	8
D1 - D8	4116	8
D9	RES	7
D10	LS166	8
D11	LS151	8
E1 - E8	4116	9
E9	RES	9
E10	LS166	9
E11	LS151	9
F1 - F9	4116	9
F9	LS157	11
F10	LS166	9
F11	LS151	9

G1 – G8	4116	10
G9	RES	9, 10
G10	LS166	10
G11	LS151	10
H1 – H8	4116	10
H9	LS157	11
H10	LS166	10
H11	LS151	10
I1	RES	7
I2	LS157	11
I3	LS02	11
I4	—	
I5	—	
I6	LS139	11
I7	LS175	6
I8	LS175	6
I9	LS175	6
I10	LS175	6
I11	LS257	6

8 Oct 84
ARB

Q-BUS Interface


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
A
VECTOR
X
LS 04
LS 74
LS 74
LS 138
LS 138
LS 148
LS 251
LS 251
LS 251
LS 251
LS 251
LS 251
LS 251
LS 251
B
BERQ
Res
LS 10
LS 174
LS 193
LS 193
LS 193
LS 193
LS 374
LS 374
LS 374
LS 374
LS 374
LS 374
LS 74
LS 10
C
Address
LS 08
LS 00
8136
8838
8838
8838
8838
LS 151
LS 151
LS 151
LS 151
LS 151
LS 151
LS 151
LS 151
D
Res
LS 74
LS 74
8136
8838
8837
7438
RES
LS 151
LS 151
LS 151
LS 151
LS 151
LS 151
LS 151
LS 151
E
RES
RES
RES
LS 00
J1 / CN1
J2/CN2
Out
RDO

1953 I/O interconnect lines

J1/CN1

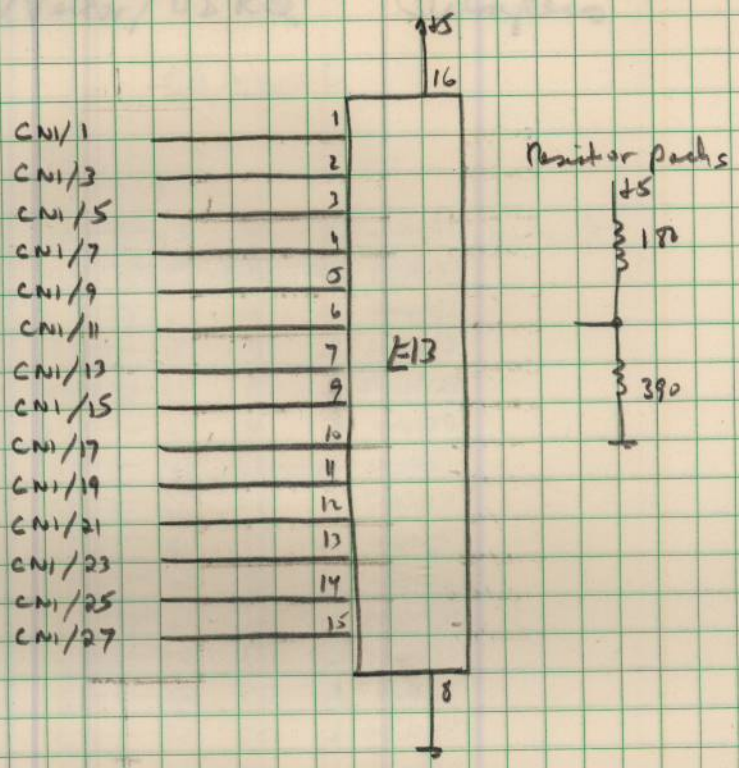
Q-BUS Signal				
BDCOK	H	1	2	output
BDAL01	L	3	4	
BDAL00	L	5	6	
BINIT	L	7	8	
BDMGI	L	9	10	
BREF	L	11	12	
BGS7	L	13	14	
BHALT	L	15	16	
BDMR	L	17	18	
BIAKI	L	19	20	H
BIRQ4	L	21	22	
BWTBT	L	23	24	
BSYNC	L	25	26	
BDIN	L	27	28	
BRPLY	L	29	30	
BDOUT	L	31	32	
BDAL17	L	33	34	
BDAL16	L	35	36	
BIRQ6	L	37	38	
BIRQ5	L	39	40	

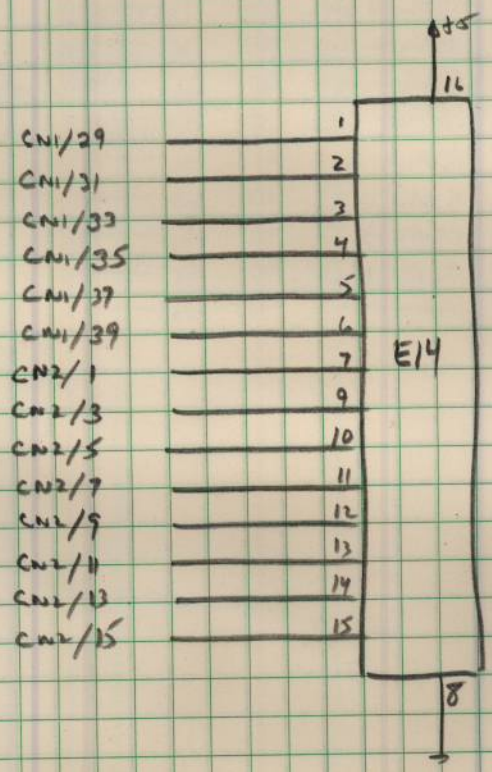
J2 / CN2

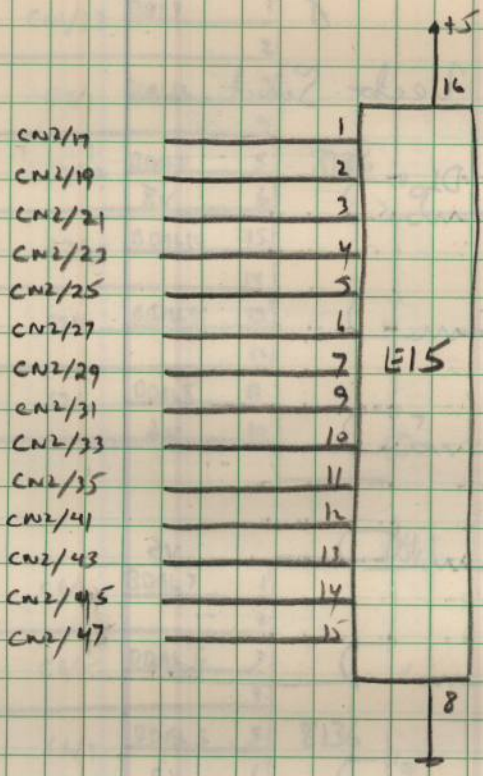
BPOK	H	1	2	output
BDAL 15	L	3	4	
BDAL 14	L	5	6	
BDAL 13	L	7	8	
BDAL 12	L	9	10	
BEVENT	L	11	12	
BDAL 11	L	13	14	
BIRQ7	L	15	16	
BDAL 10	L	17	18	
BSACK	L	19	20	
BDAL 09	L	21	22	
BDAL 08	L	23	24	
BDAL 07	L	25	26	
BDAL 06	L	27	28	
BDAL 05	L	29	30	
BDAL 04	L	31	32	
BDAL 03	L	33	34	
BDAL 02	L	35	36	
BPOK	H	37	38	input
BDCOK	H	39	40	input
BDAL 21	L	41	42	
BDAL 20	L	43	44	
BDAL 19	L	45	46	
BDAL 18	L	47	48	
———		49	50	

9 Oct 84
ARD

J1/J2 Resistive Terminations



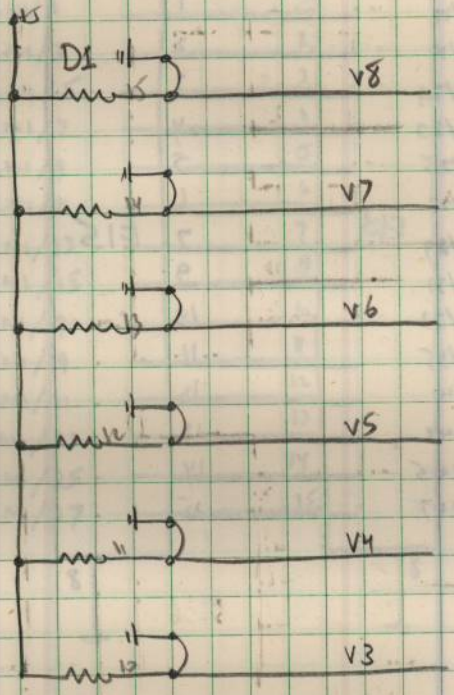



+5
16
CN2/17
CN2/19
CN2/21
CN2/23
CN2/25
CN2/27
CN2/29
CN2/31
CN2/33
CN2/35
CN2/41
CN2/43
CN2/45
CN2/47
1
2
3
4
5
6
7
9
10
11
12
13
14
15
LE15
8

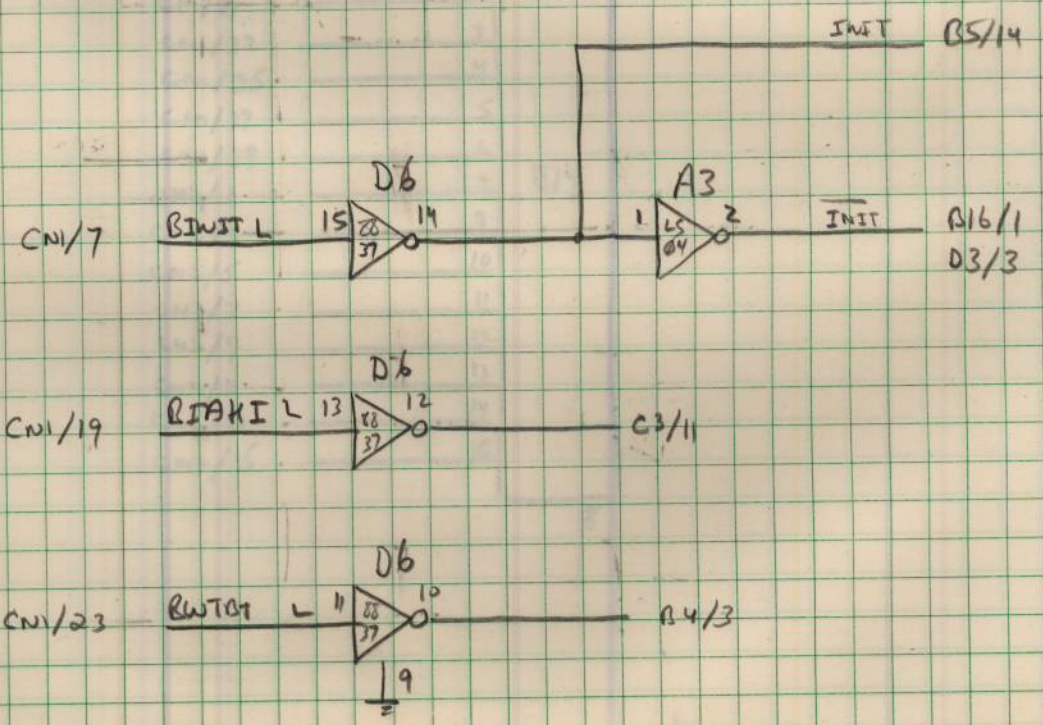
9 Oct 89

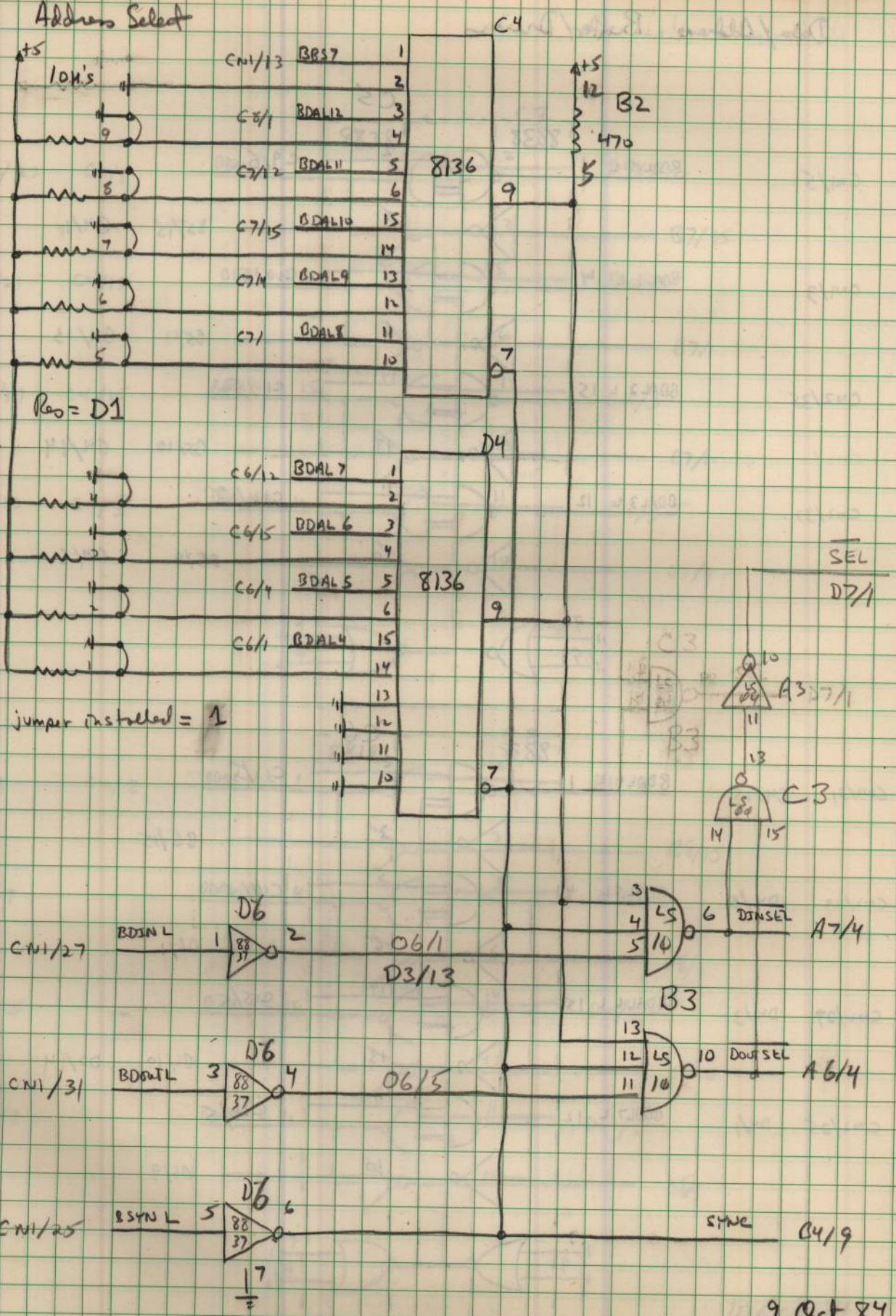
Q-BUS Addres Decoding

Vector Select

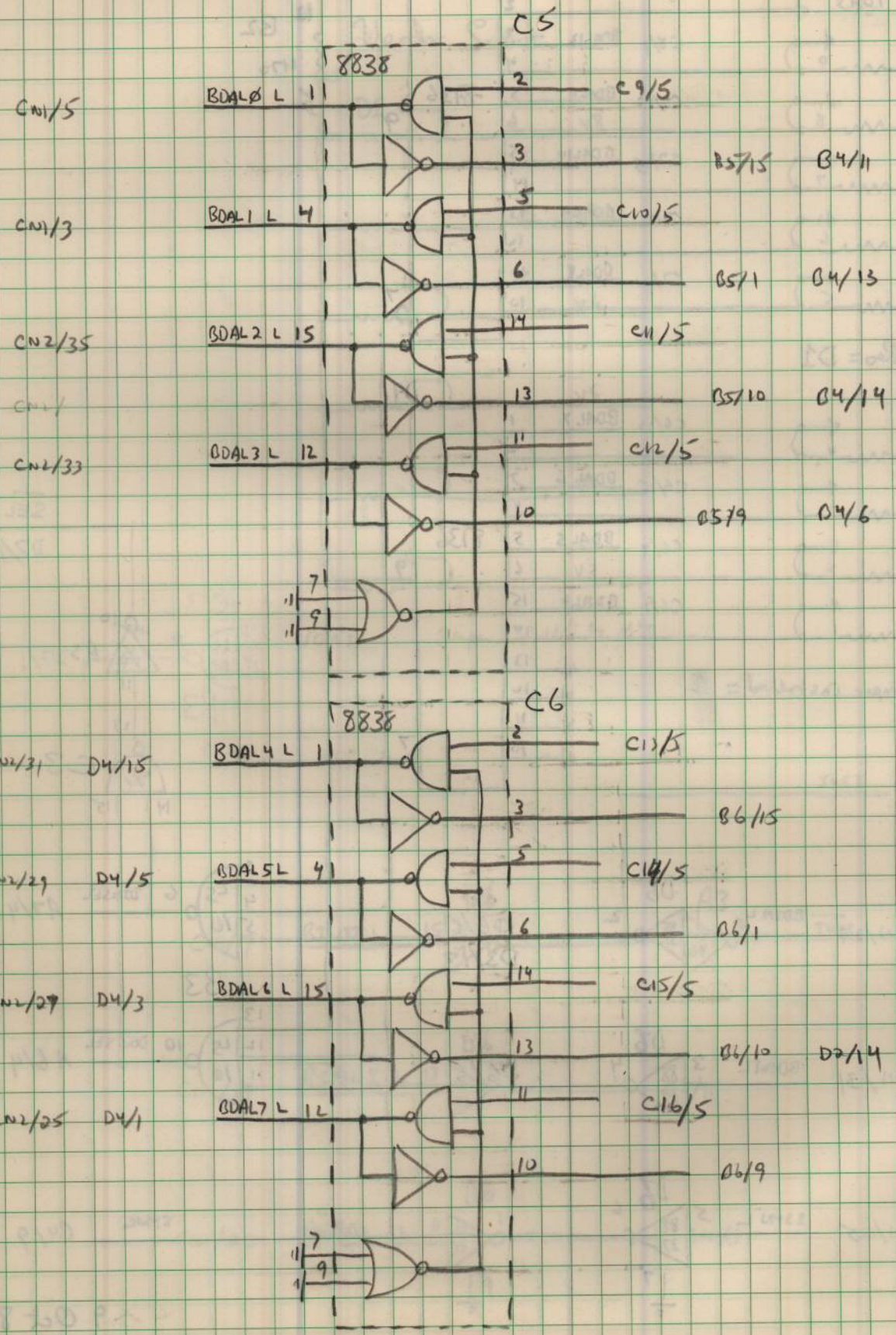


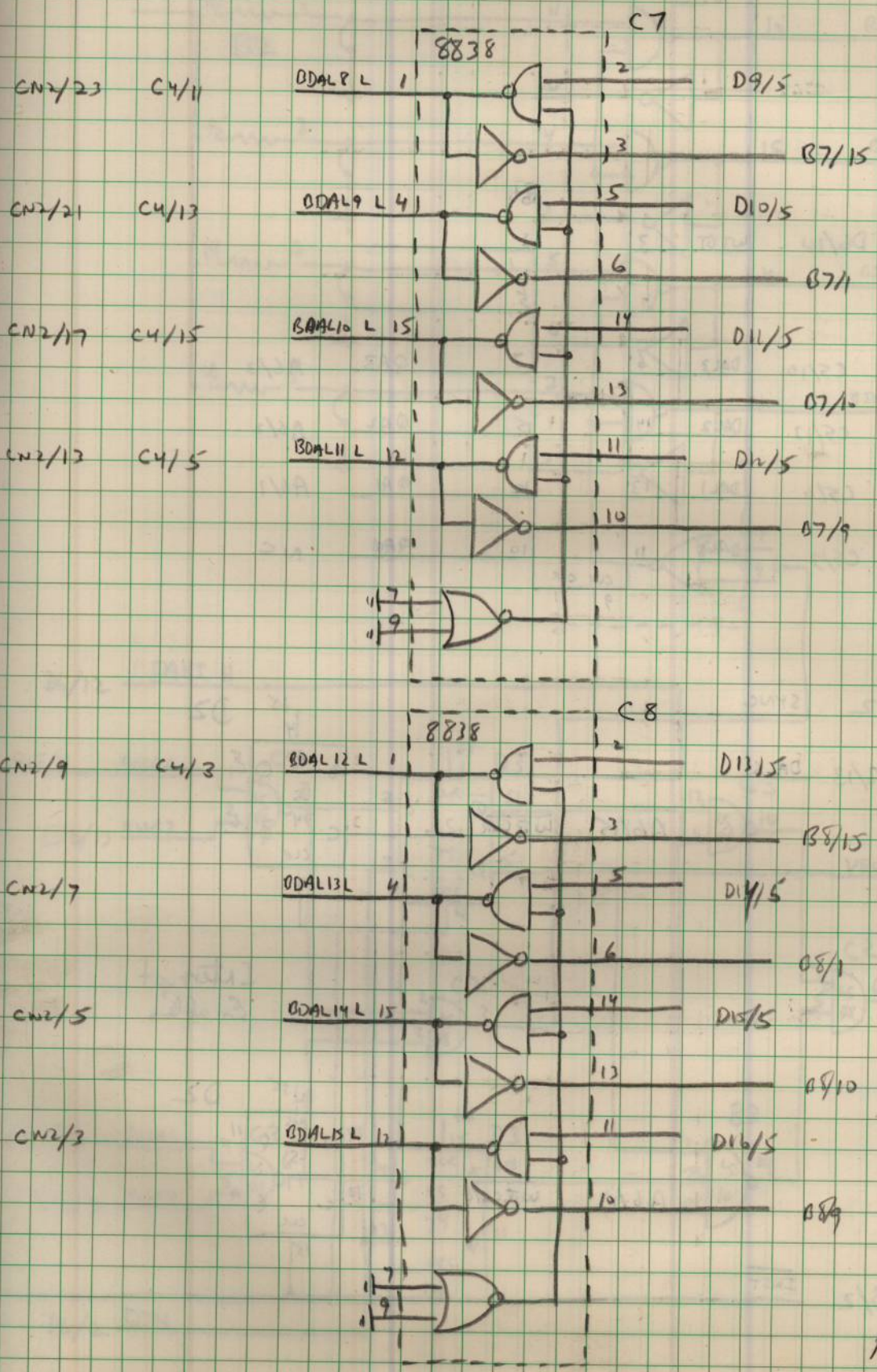
10H's Jumper installed = ø




Address Select
C4
+5
10K's
CN1/13
BRST
BDAL12
BDAL11
BDAL10
BDAL9
BDAL8
8136
B2
470
Pas = D1
D4
BDAL7
BDAL6
BDAL5
BDAL4
jumper installed = 1
SEL
D7/1
A3
C3
B3
CN1/27
BDINL
D6
D6/1
D3/13
DINSEL
A7/4
CN1/31
BDOUTL
D6/5
DOUTSEL
A6/4
CN1/25
BSYNL
SYNC
B4/9
9 Oct 84
ARB

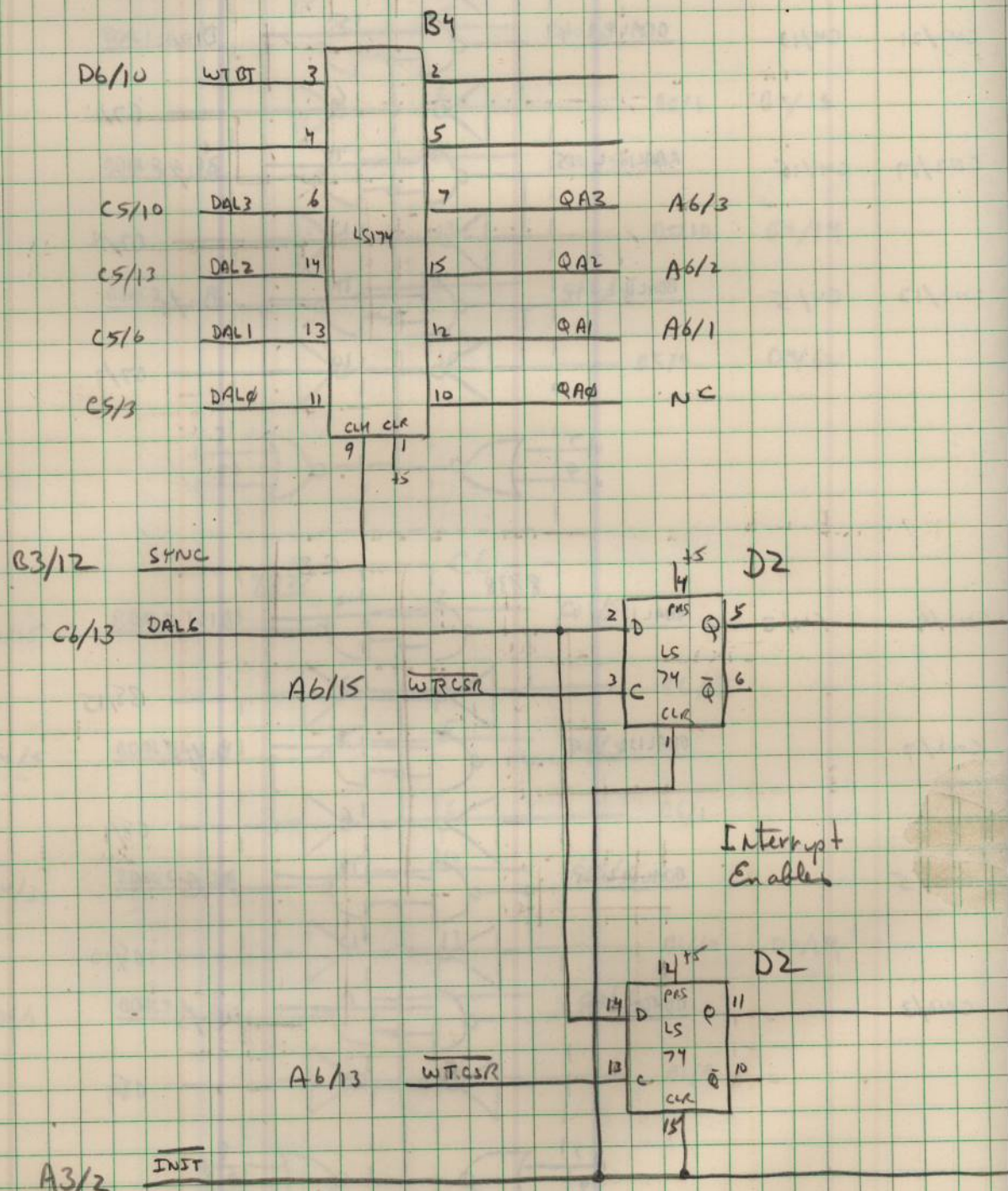
Data / Address Buffer / Drivers



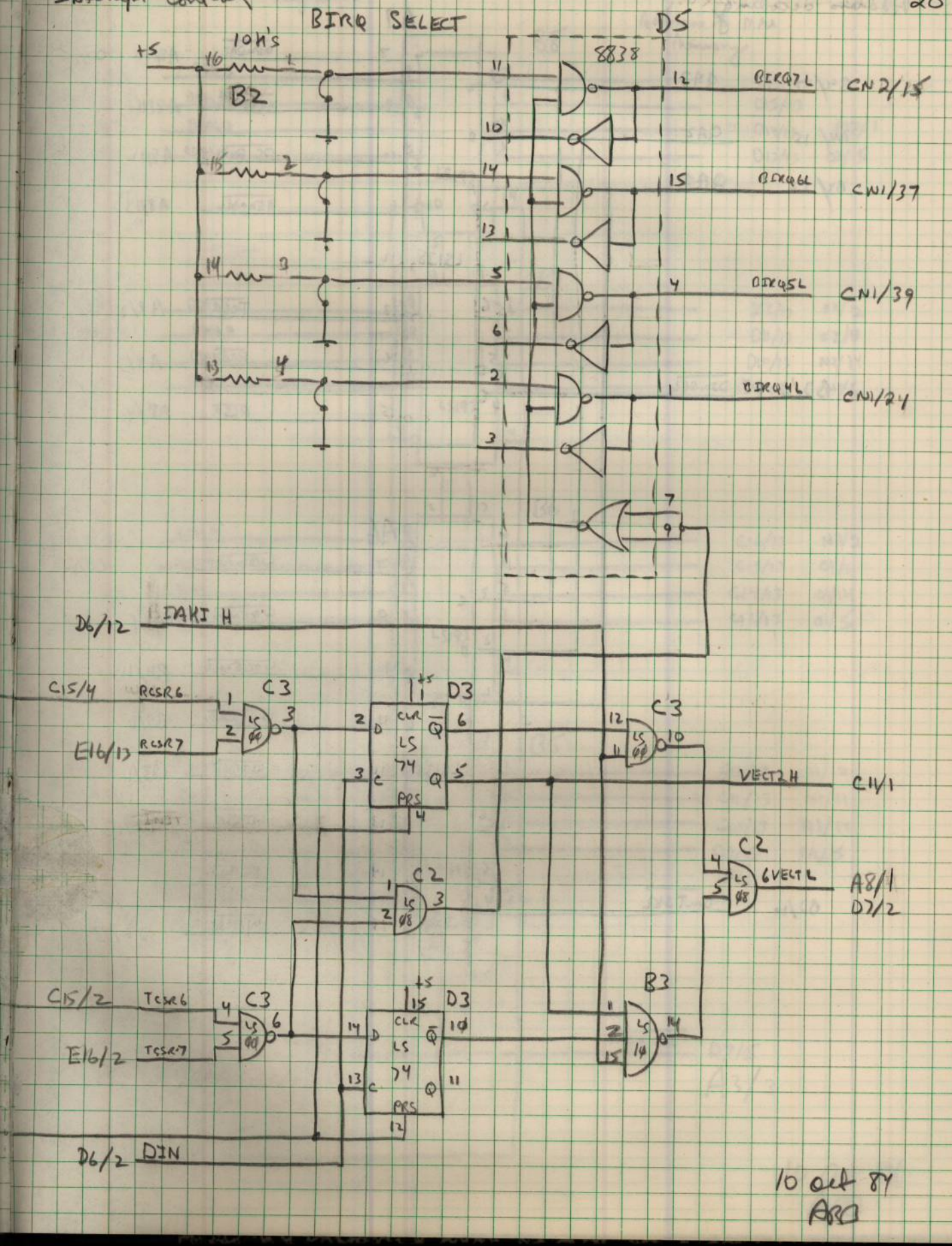

C7
8838
CN2/23
C4/11
BDAL8 L 1
2
D9/5
3
B7/15
CN2/21
C4/13
BDAL9 L 4
5
D10/5
6
B7/1
CN2/17
C4/15
BDAL10 L 15
14
D11/5
13
B7/1
CN2/13
C4/5
BDAL11 L 12
11
D12/5
10
B7/9
7
9
C8
8838
CN2/9
C4/3
BDAL12 L 1
2
D13/5
3
B8/15
CN2/7
BDAL13L 4
5
D14/5
6
B8/1
CN2/5
BDAL14 L 15
14
D15/5
13
B8/10
CN2/3
BDAL15 L 12
11
D16/5
10
B8/9
7
9

10 Oct 84
ABB

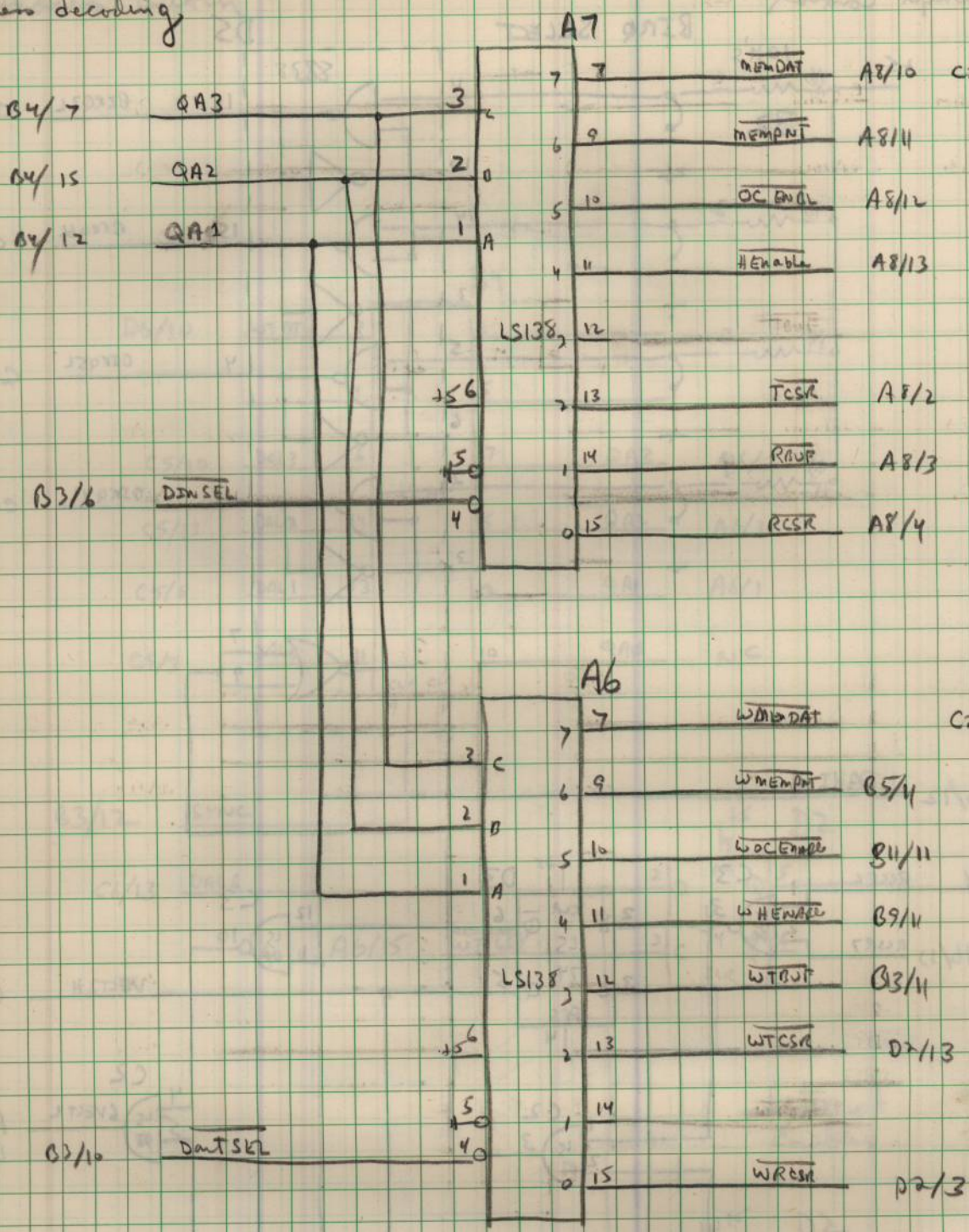
Address Latch



Interrupt Control

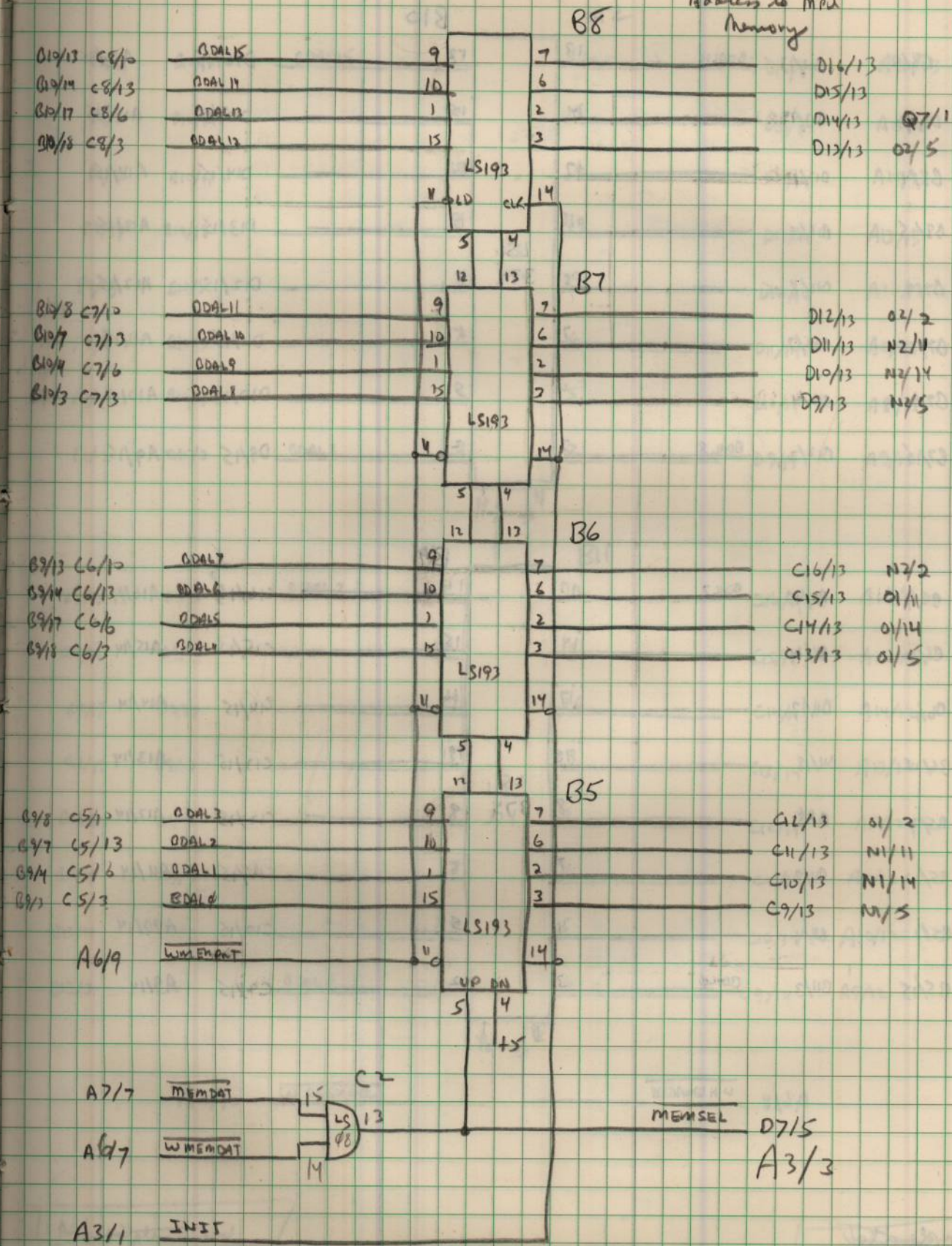


Address decoding



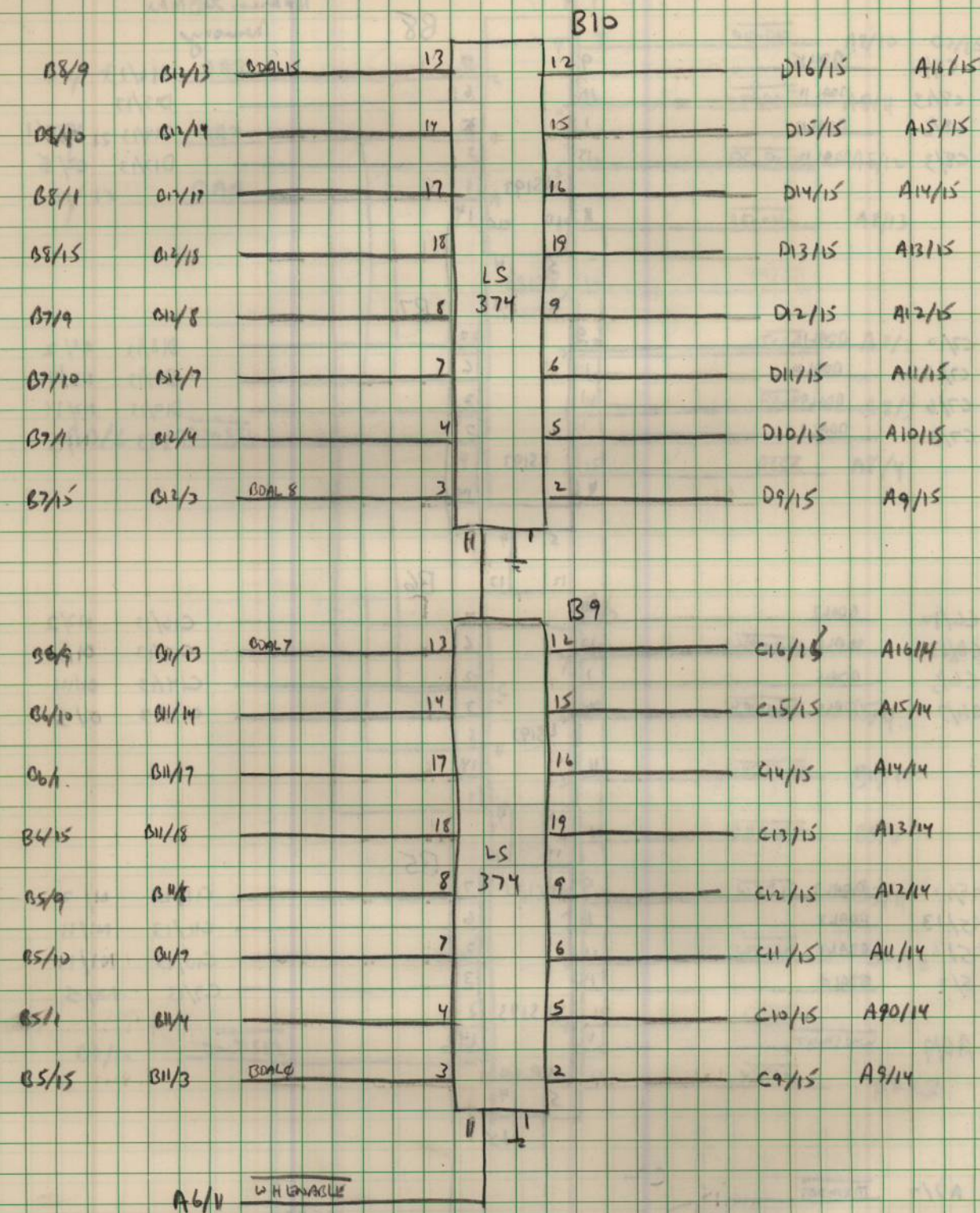
MEMPNTR Register

Address to MPU Memory



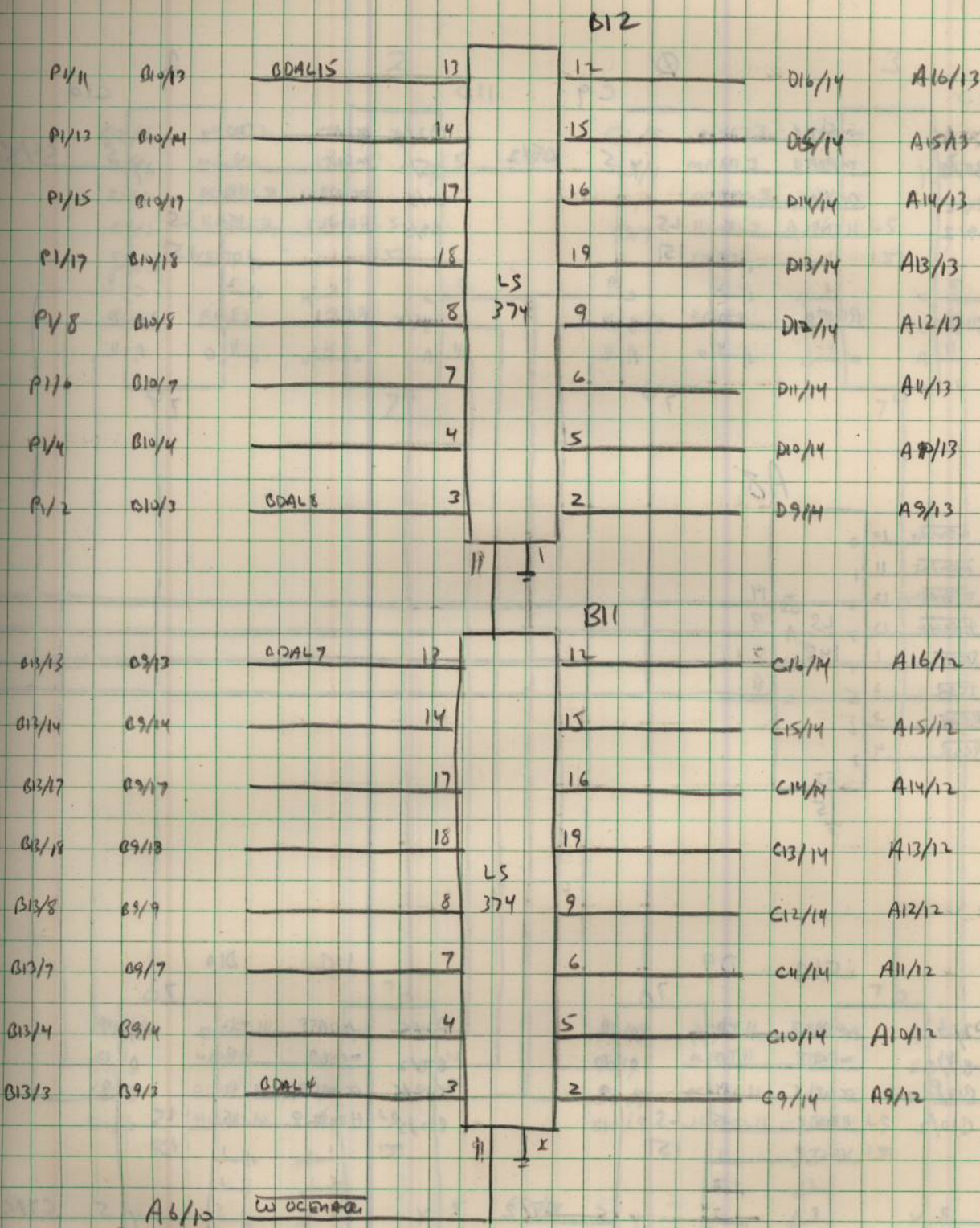
10 Oct 84
ARD

Histo Enable Register



Not implemented

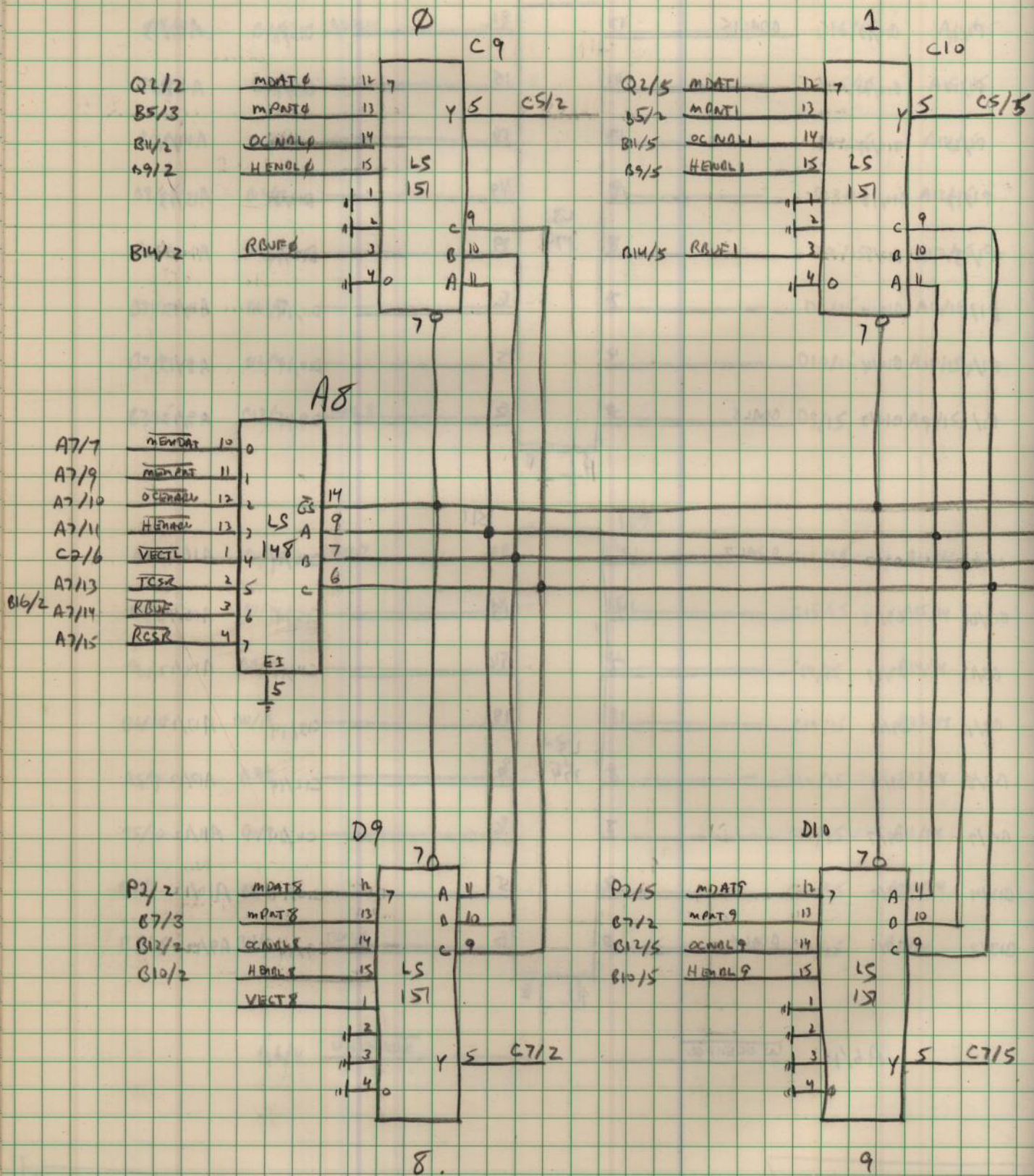
OC/Coron Enable Register

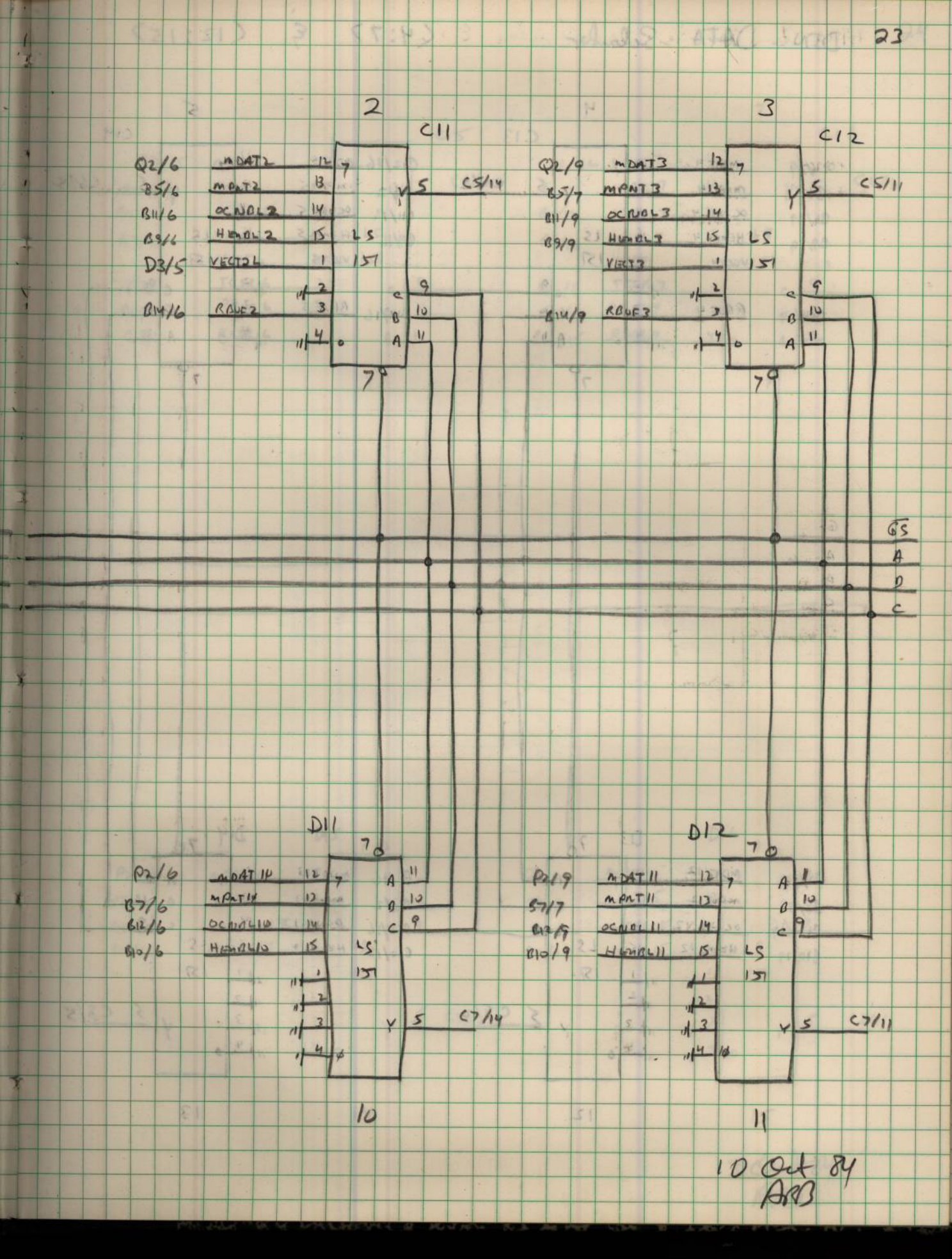


Not implemented

10 Oct 84
BBO

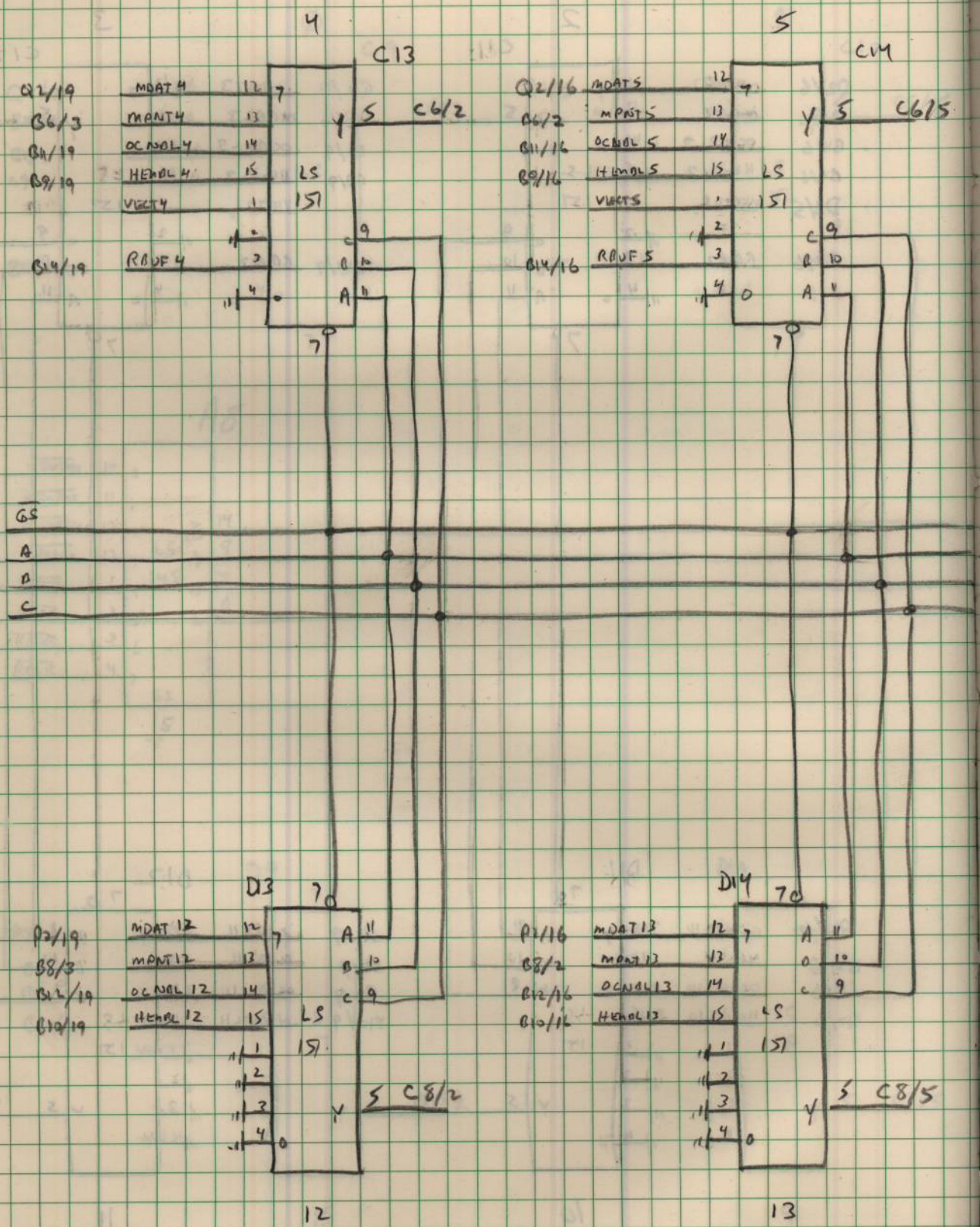
DIN DATA Selector <0:3> & <8:11>

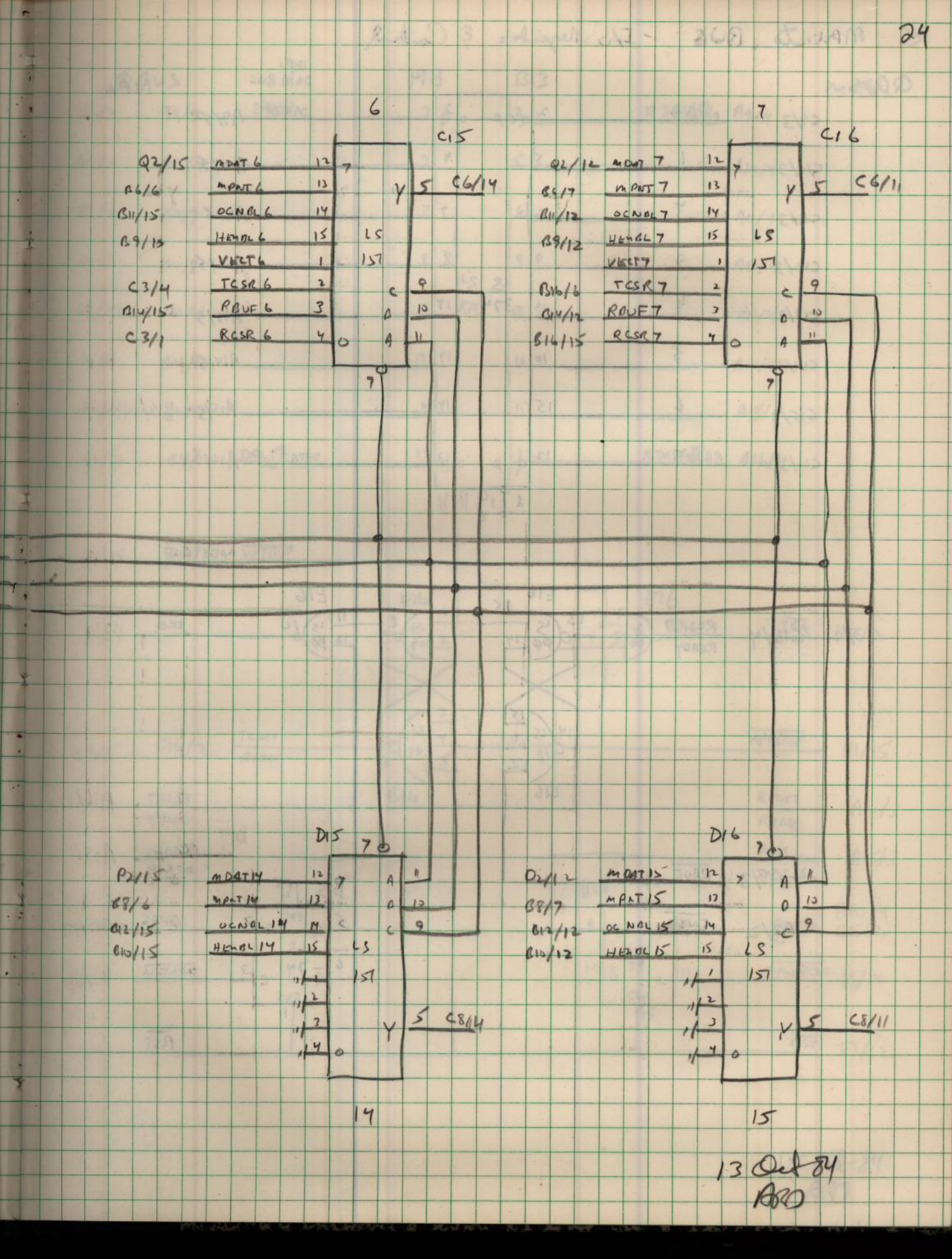



2
C11
Q2/6
B5/6
B11/6
B9/6
D3/5
B14/6
MDAT2
MPAT2
OCNOL2
HENOL2
VECT2L
RBUF2
LS
157
CS/14
3
C12
Q2/9
B5/7
B11/9
B9/9
B14/9
MDAT3
MPAT3
OCNOL3
HENOL3
VECT3
RBUF3
CS/11
GS
A
D
C
D11
P2/6
B7/6
B12/6
B10/6
MDAT10
MPAT10
OCNOL10
HENOL10
C7/14
10
D12
P2/9
B7/7
B12/9
B10/9
MDAT11
MPAT11
OCNOL11
HENOL11
C7/11
11

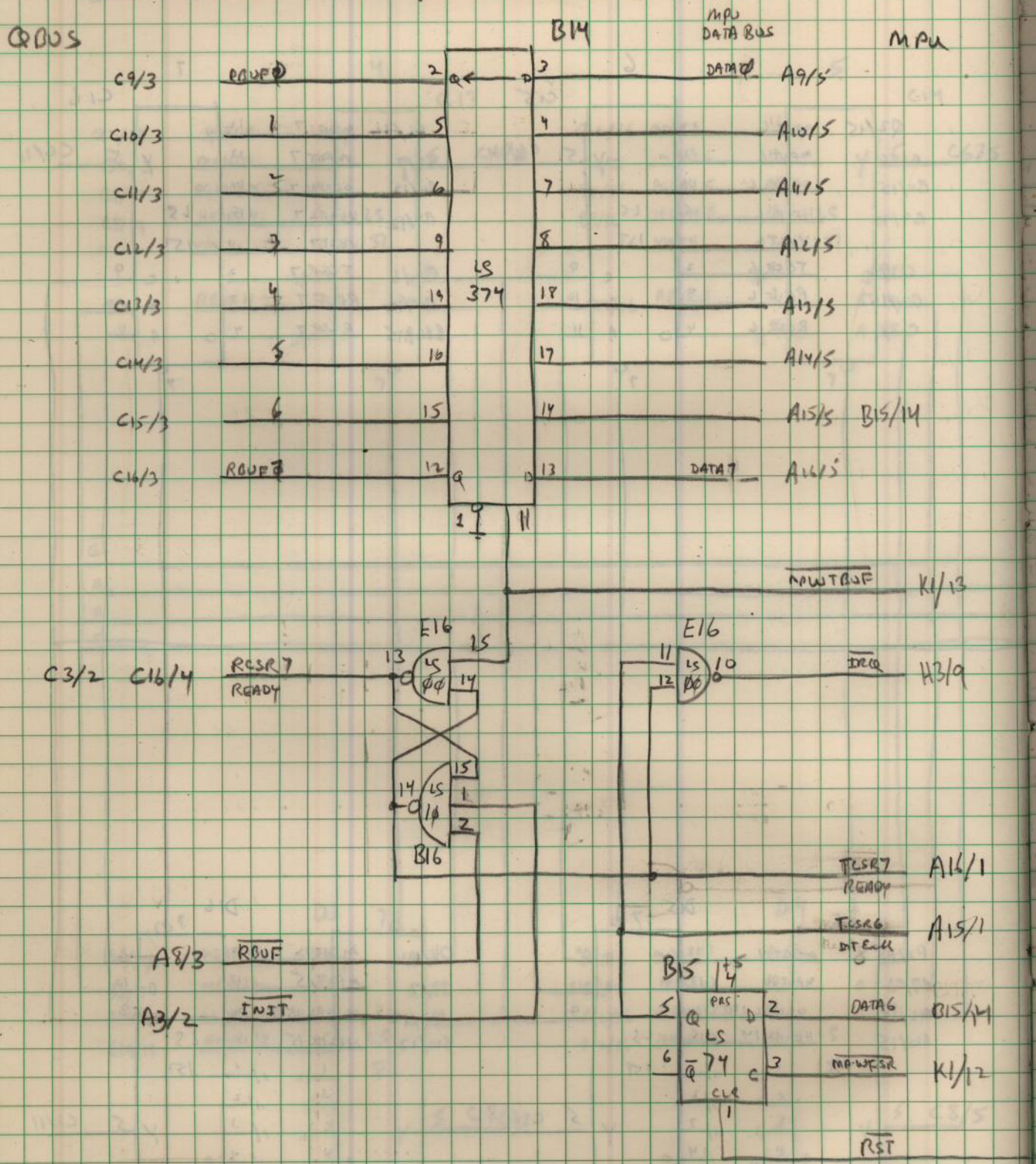
10 Oct 84
ARB

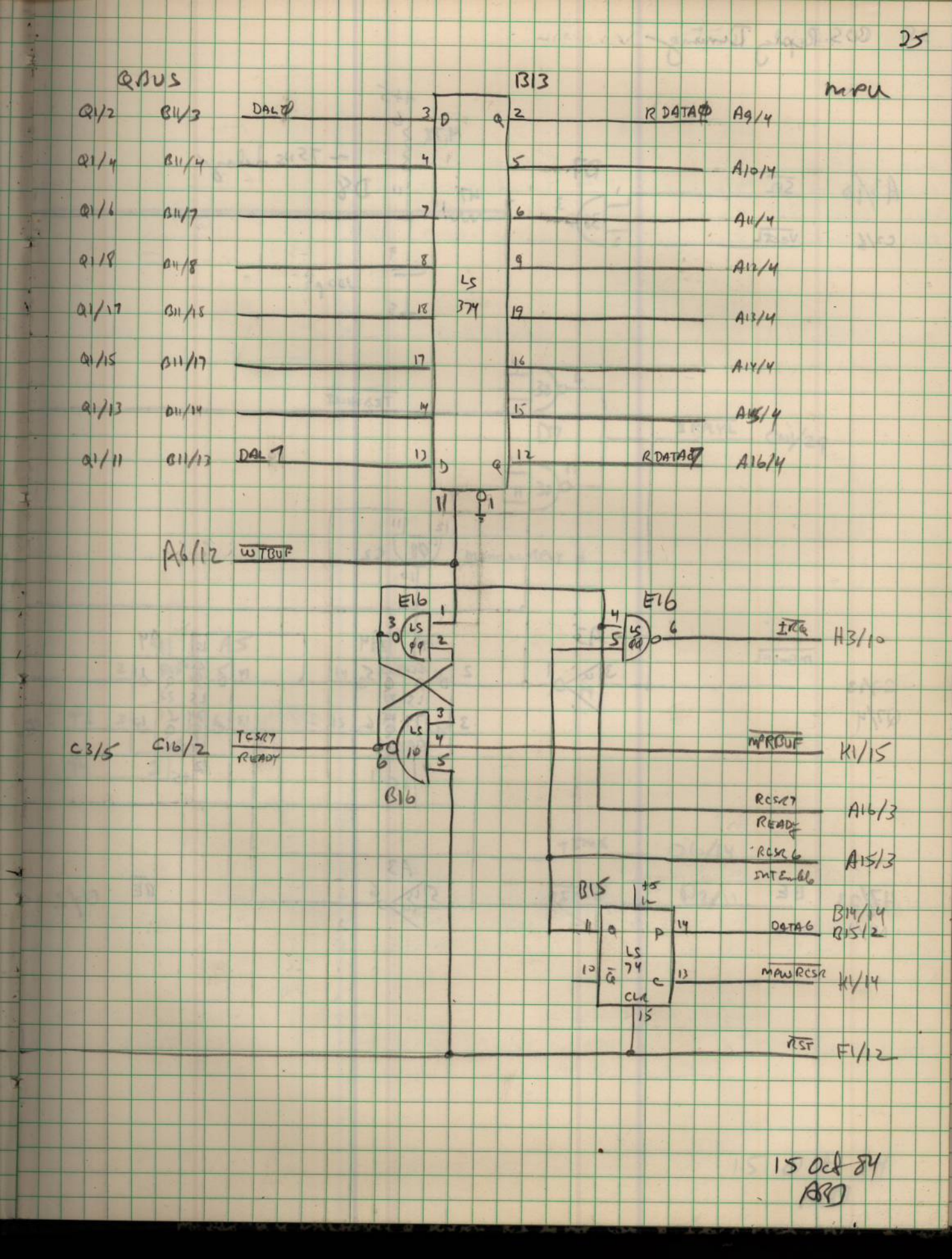
DIN DATA Selector <4:7> & <12:15>



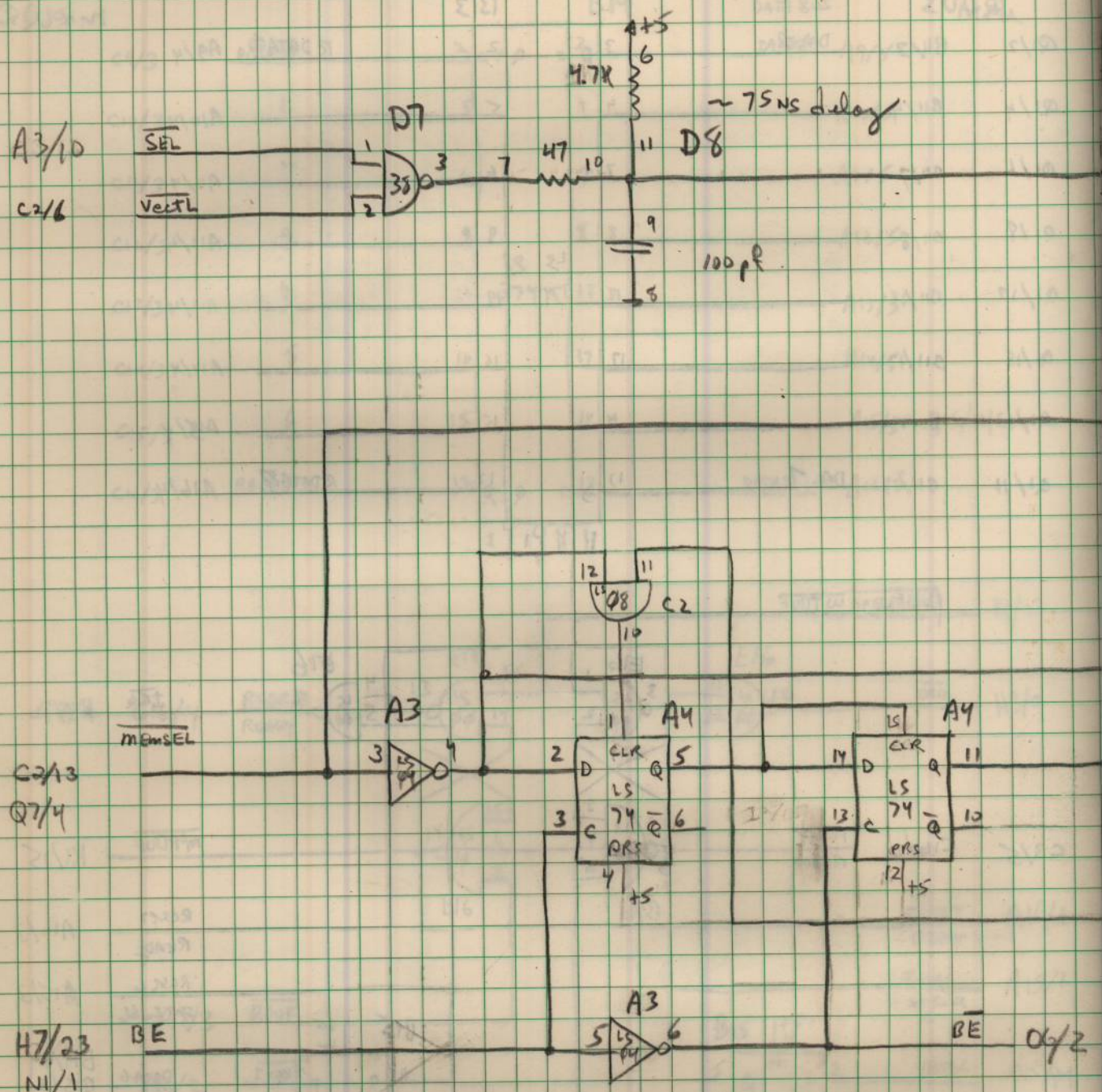

6
C15
Q2/15
A6/6
B11/15
B9/13
C3/4
A14/15
C3/1
MDAT 6
MPAT 6
OCNOL 6
HENOL 6
VECT 6
TCSR 6
RBUF 6
RCSR 6
LS
157
Y
C6/14
7
C16
Q2/12
B9/7
B11/12
B9/12
B16/6
A14/12
B14/15
MDAT 7
MPAT 7
OCNOL 7
HENOL 7
VECT 7
TCSR 7
ROUF 7
RCSR 7
C6/11
D15
P2/15
B8/6
A12/15
B10/15
MDAT 14
MPAT 14
OCNOL 14
HENOL 14
C8/14
14
D16
D2/12
B8/7
B12/12
B10/12
MDAT 15
MPAT 15
OCNOL 15
HENOL 15
C8/11
15
13 Oct 84
ABD

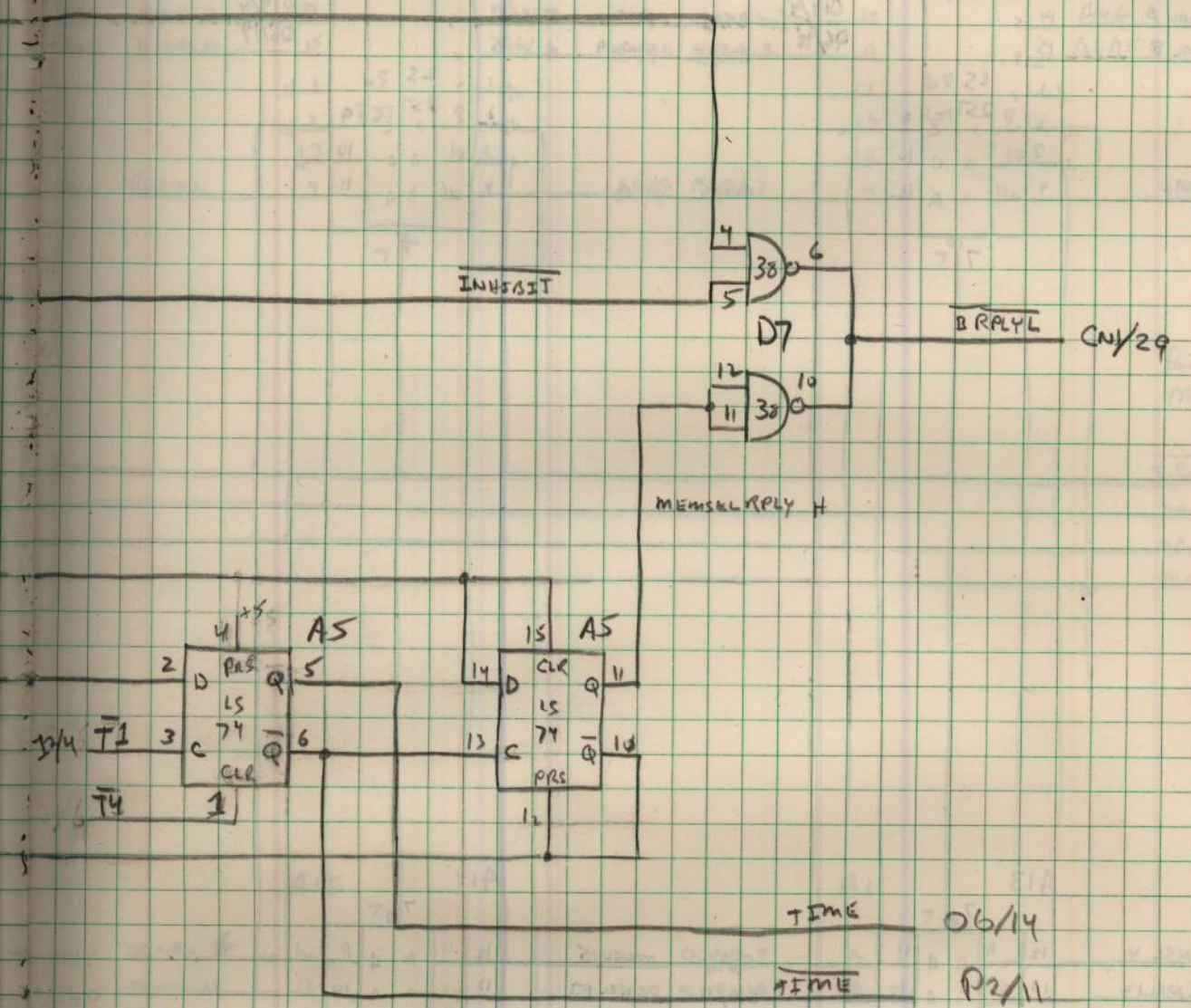
MPU to BUS I/O Registers & Control




QBUS
B13
MPU
Q1/2 B11/3 DAL0 3 D Q 2 RDATA0 A9/4
Q1/4 B11/4 4 5 A10/4
Q1/6 B11/7 7 6 A11/4
Q1/8 B11/8 8 9 A12/4
LS 374
Q1/17 B11/18 18 19 A13/4
Q1/15 B11/17 17 16 A14/4
Q1/13 B11/14 14 15 A15/4
Q1/11 B11/13 DAL7 13 D Q 12 RDATA7 A16/4
11 1
A6/12 WTBUF
E16
LS 00
E16
LS 00
IRQ H3/10
C3/5 C16/2 TCSR7 READY
LS 10
MPRBUF K1/15
B16
RCSR7 READY A16/3
RCSR6 INTEnbl6 A15/3
B15
LS 74
DATA6 B14/14 B15/2
MAINT RCSR K1/14
CLR
RST F1/12
15 Oct 84
ABD

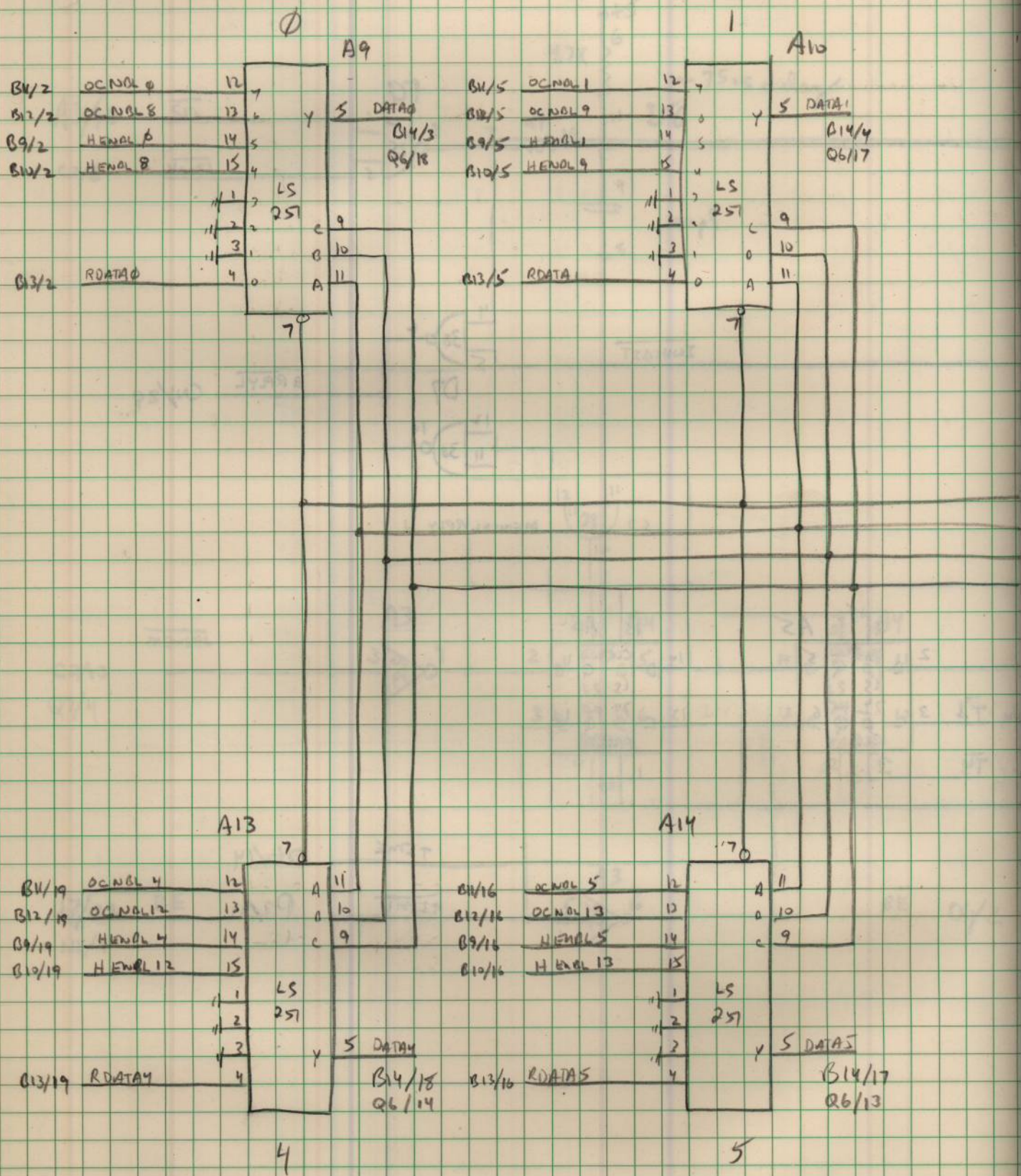
BUS Reply Timing

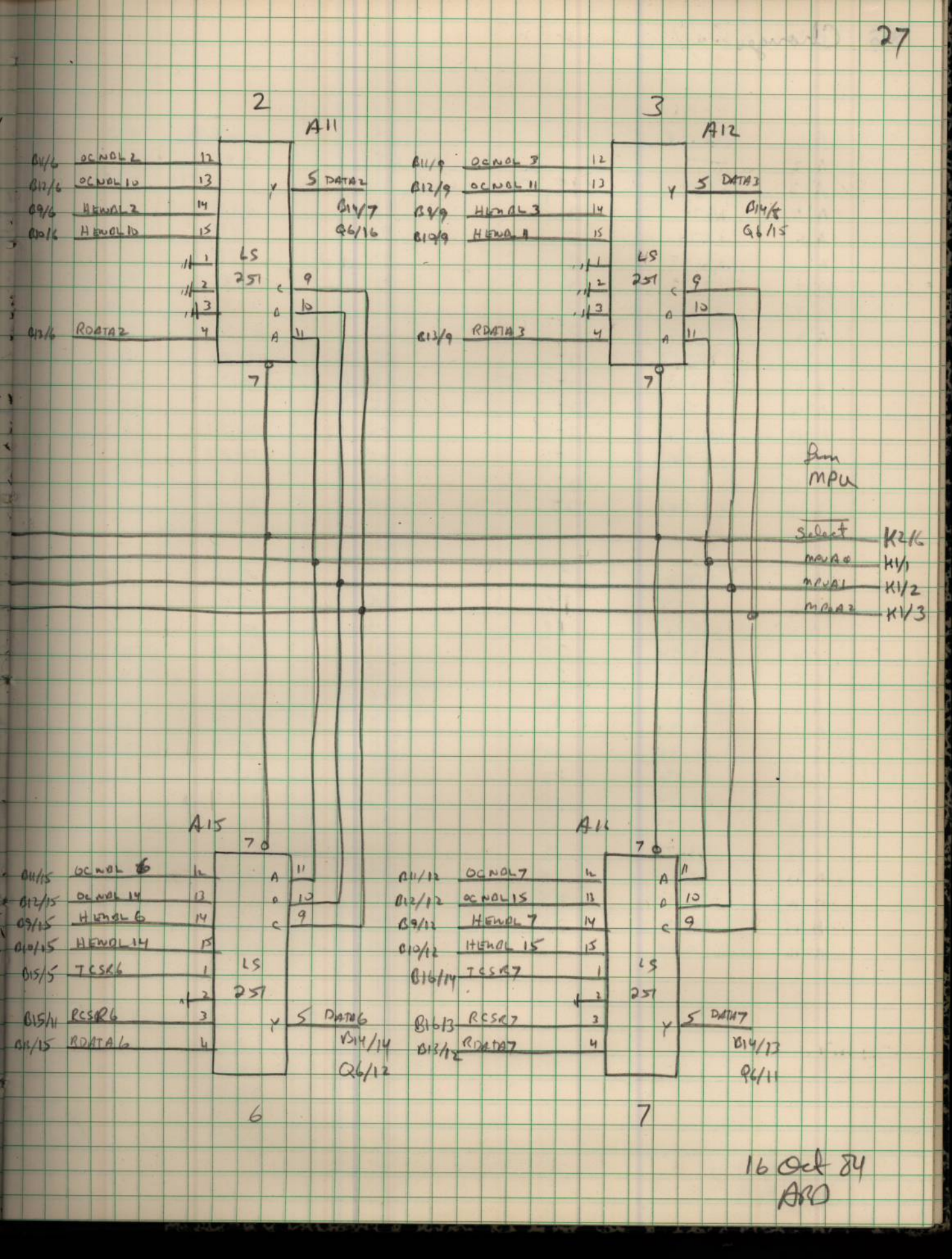



INHIBIT
4
5
38
6
D7
BRPLYL
CNV/29
12
11
38
10
MEMSELRPLY H
A5
A5
LS 74
PRS
CLR
D
C
Q
T1
T4
TIME
06/14
TIME
P2/11

15 Oct 84
ARD

MPU Data Selector




2
A11
B4/6 OCNOL 2
B12/6 OCNOL 10
C9/6 HENOL 2
C10/6 HENOL 10
B13/6 RDATA2
LS 251
5 DATA2
B14/7
Q6/16
3
A12
B11/9 OCNOL 3
B12/9 OCNOL 11
C9/9 HENOL 3
B10/9 HENOL 11
B13/9 RDATA3
LS 251
5 DATA3
B14/8
Q6/15
Zum MPU
Select K2/6
MEMAD K1/1
MPUAI K1/2
MPUA2 K1/3
A15
B11/15 OCNOL 6
B12/15 OCNOL 14
C9/15 HENOL 6
C10/15 HENOL 14
B15/5 TCSR6
B15/11 RCSR6
B11/15 RDATA6
LS 251
5 DATA6
B14/14
Q6/12
6
A11
B11/12 OCNOL 7
B12/12 OCNOL 15
C9/12 HENOL 7
C10/12 HENOL 15
B16/14 TCSR7
B16/13 RCSR7
B13/12 RDATA7
LS 251
5 DATA7
B14/13
Q6/11
7
16 Oct 84
ARD

IC Index

A 1	VECTOR	18
A 2	—	
A 3	LS04	18, 26
A 4	LS74	26
A 5	LS74	26
A 6	LS138	21
A 7	LS138	21
A 8	LS148	23
A 9	LS251	27
A 10	LS251	27
A 11	LS 251	27
A 12	LS 251	27
A 13	LS 251	27
A 14	LS 251	27
A 15	LS 251	27
A 16	LS 251	27

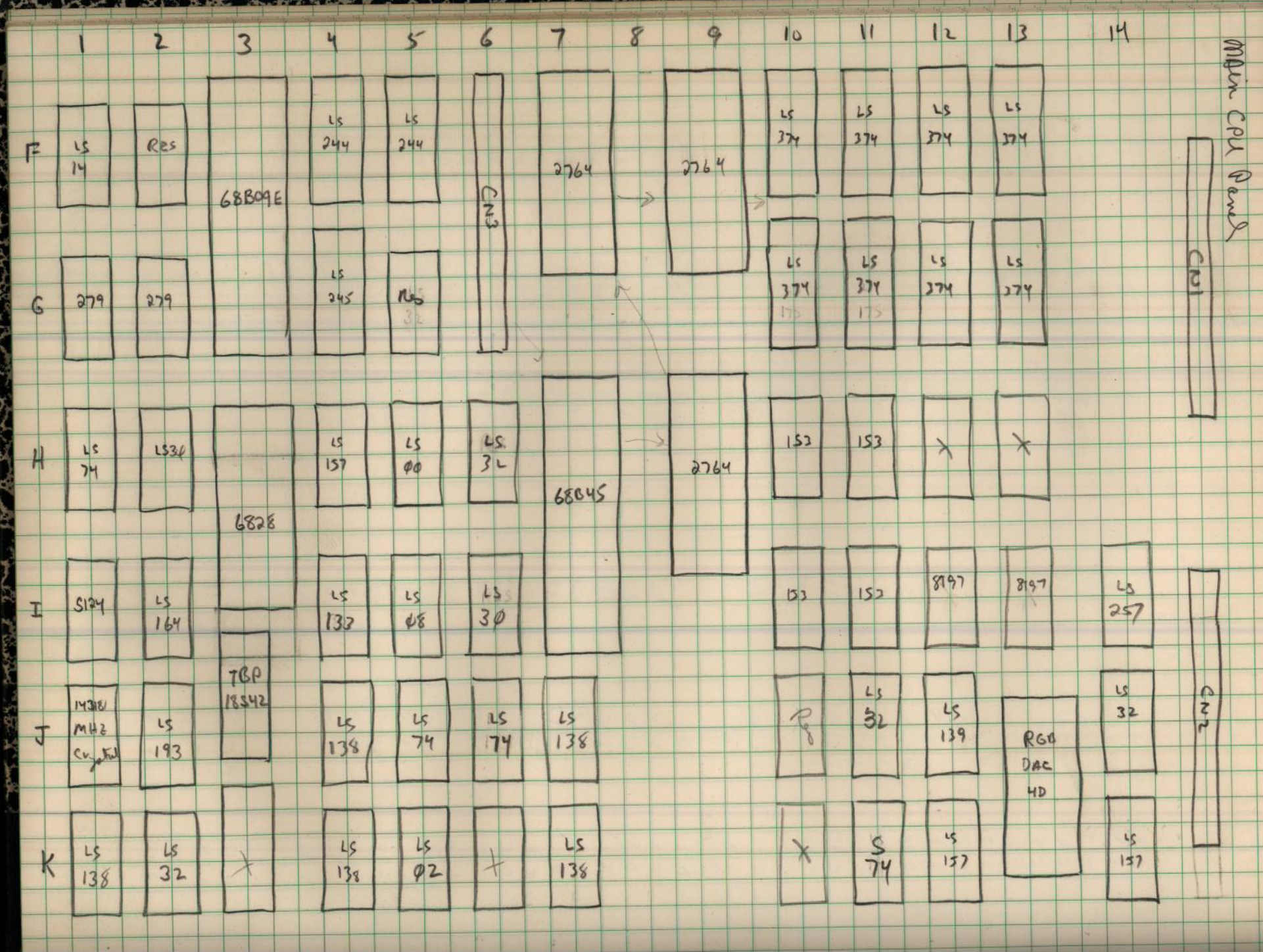
B 1	BINQ	20
B 2	RES	18
B 3	LS10	18, 20
B 4	LS174	20
B 5	LS193	21
B 6	LS193	21
B 7	LS193	21
B 8	LS193	21
B 9	LS374	22
B 10	LS374	22
B 11	LS374	22
B 12	LS374	22
B 13	LS 374	25
B 14	LS 374	25
B 15	LS74	25
B 16	LS10	25

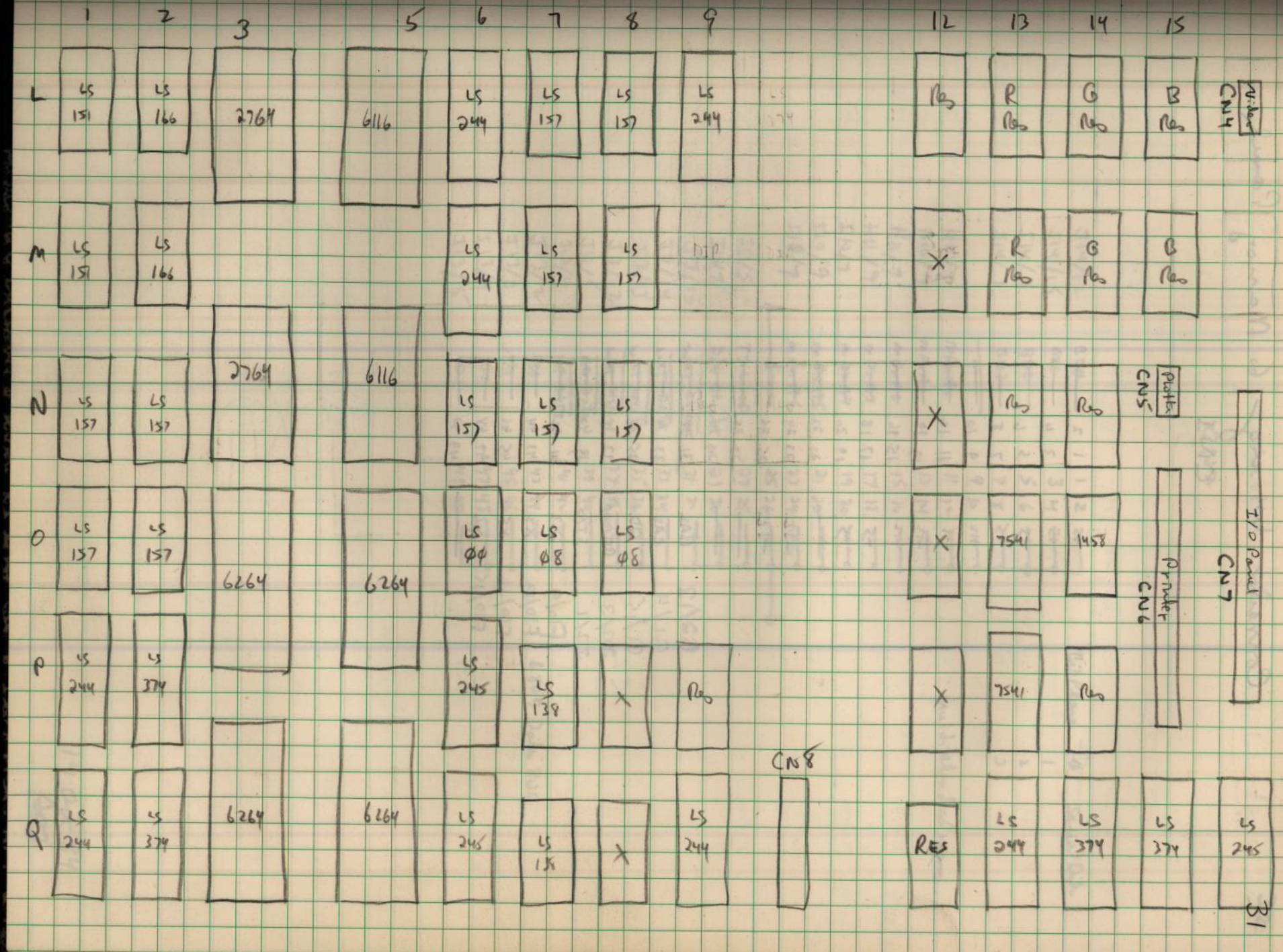
C 1	Address	18
C 2	LS08	20, 21
C 3	LS00	18, 20
C 4	8136	18
C 5	8838	19
C 6	8838	19
C 7	8838	19
C 8	8838	19
C 9	LS151	23
C 10	LS151	23
C 11	LS151	23
C 12	LS151	23
C 13	LS151	24
C 14	LS151	24
C 15	LS151	24
C 16	LS151	24

D 1	RES	
D 2	LS74	20
D 3	LS74	20
D 4	8136	18
D 5	8838	20
D 6	8837	18
D 7	7438	26
D 8	RES	26
D 9	LS151	23
D 10	LS151	23
D 11	LS151	23
D 12	LS151	23
D 13	LS151	24
D 14	LS 151	24
D 15	LS 151	24
D 16	LS151	24

E 13	RES	17
E 14	RES	17
E 15	RES	17
E 16	LS00	25

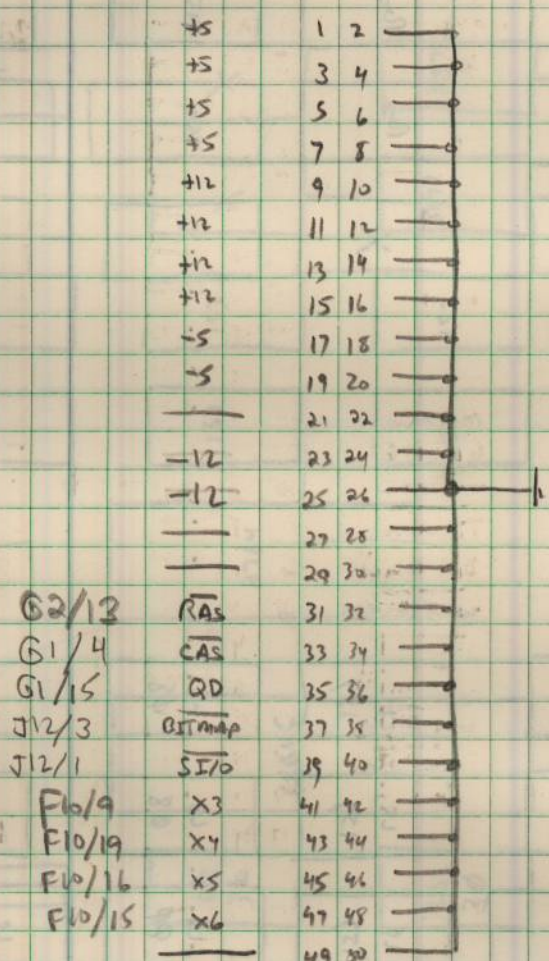
16 Oct 84
BRD


1 2 3 4 5 6 7 8 9 10 11 12 13 14
F
G
H
I
J
K
LS 14
RES
68B09E
LS 244
LS 244
CN3
2764
2764
LS 374
LS 374
LS 374
LS 374
LS 245
LS 374
LS 374
LS 374
279
279
6828
LS 157
LS 00
LS 32
68B45
2764
153
153
LS 74
LS 30
S124
LS 164
LS 138
LS 08
LS 30
153
153
8097
8097
LS 257
TBP 18S42
14318 MHz Crystal
LS 193
LS 138
LS 74
LS 74
LS 138
LS 32
LS 139
RGB DAC 4D
LS 32
LS 138
LS 32
LS 138
LS 02
LS 138
S 74
LS 157
LS 157
Main CPU Panel
CN1
CN2


1
2
3
5
6
7
8
9
12
13
14
15
L
M
N
O
P
Q
LS 151
LS 166
2764
6116
LS 244
LS 157
LS 245
LS 138
LS 244
LS 374
6264
DIP
R
G
B
RES
CN4
CN5
CN6
CN7
CN8
Video
Plotter
Printer
I/O Panel
7541
1458
B1

Connector Wiring to Memory Panel

CN1



CN2

J14/12	BD0	1	2	
J14/15	BD1	3	4	
J14/4	BD2	5	6	
J14/1	BD3	7	8	
		9	10	
H10/7	MAxA0	11	12	
H10/9	MAxA1	13	14	
H11/7	MAxA2	15	16	
H11/9	MAxA3	17	18	
I10/7	MAxA4	19	20	
I10/9	MAxA5	21	22	
I11/7	MAxA6	23	24	
		25	26	+5
I13/5	T7 Load	27	28	
I13/7	Dot Clock	29	30	
J7/15	WRT CLOCK	31	32	
J7/14	WRT DATA	33	34	
J7/13	WRT WRITE	35	36	
J7/12	WRT Display	37	38	
J12/5	RD Data	39	40	
J12/4	WR DATA	41	42	
J13/2	D0	43	44	
J13/3	D1	45	46	
J13/4	D2	47	48	
J13/5	D3	49	50	

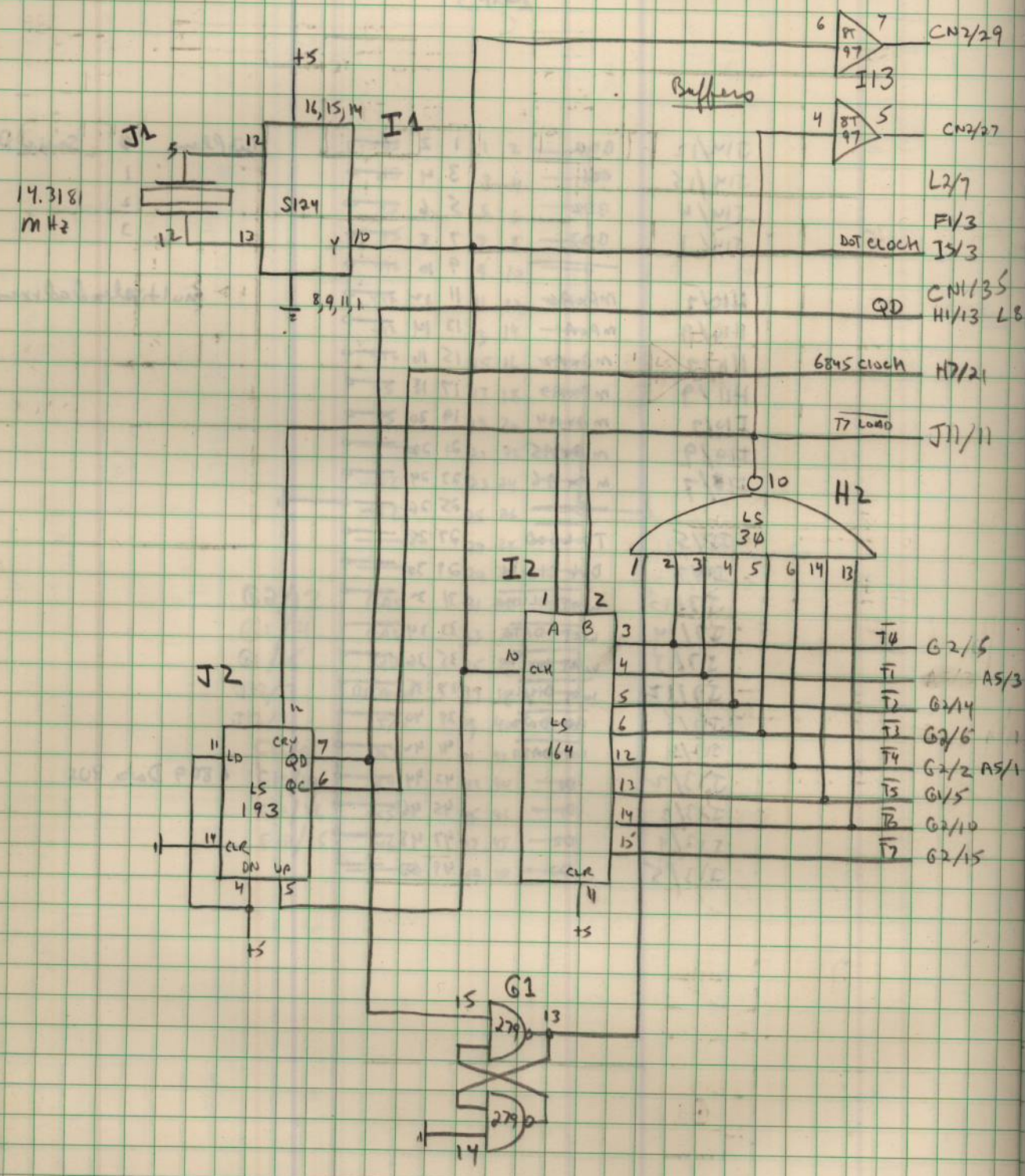
Bit Plane 0 Serial Data
1
2
3

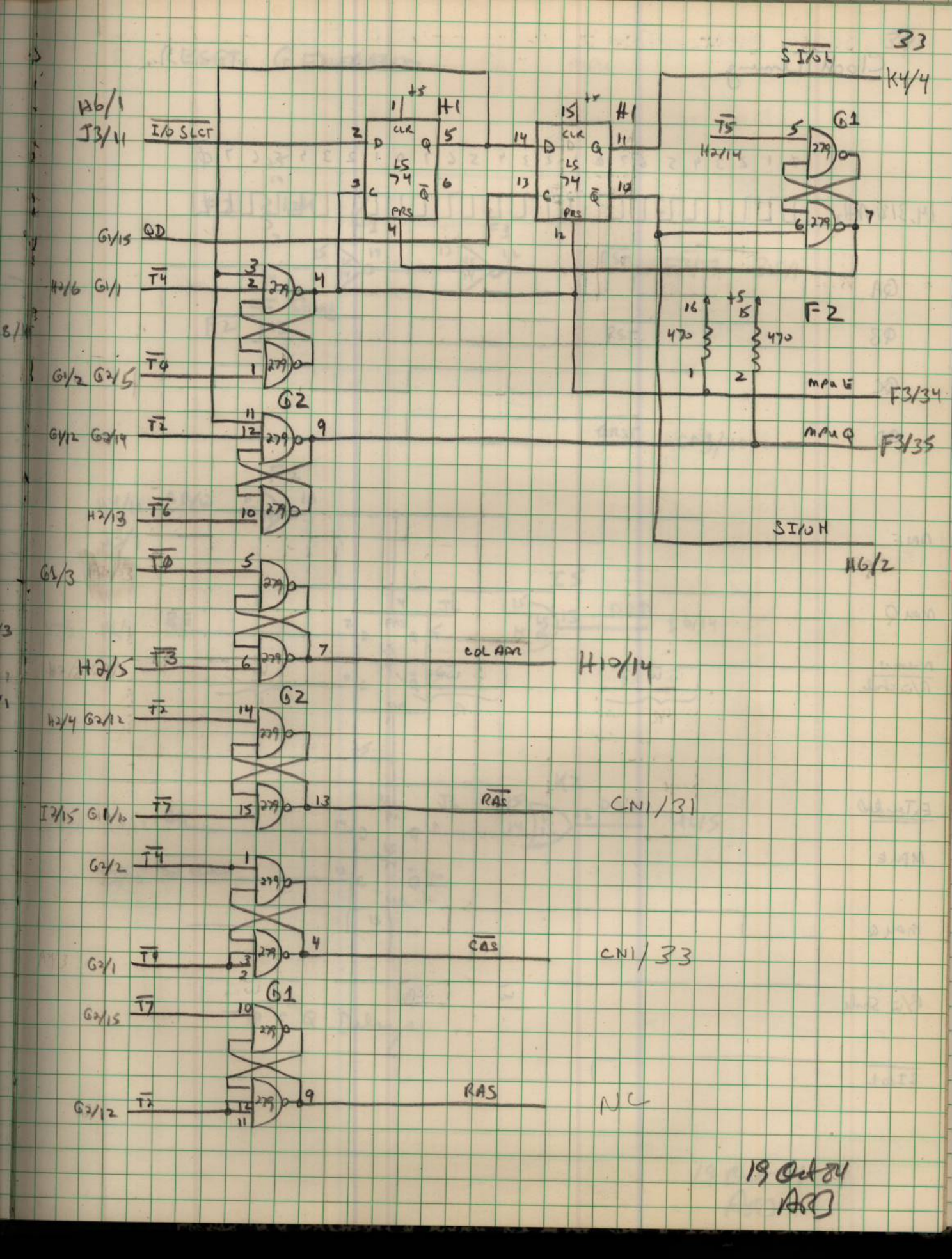
multiplexed address

6809 Data BUS

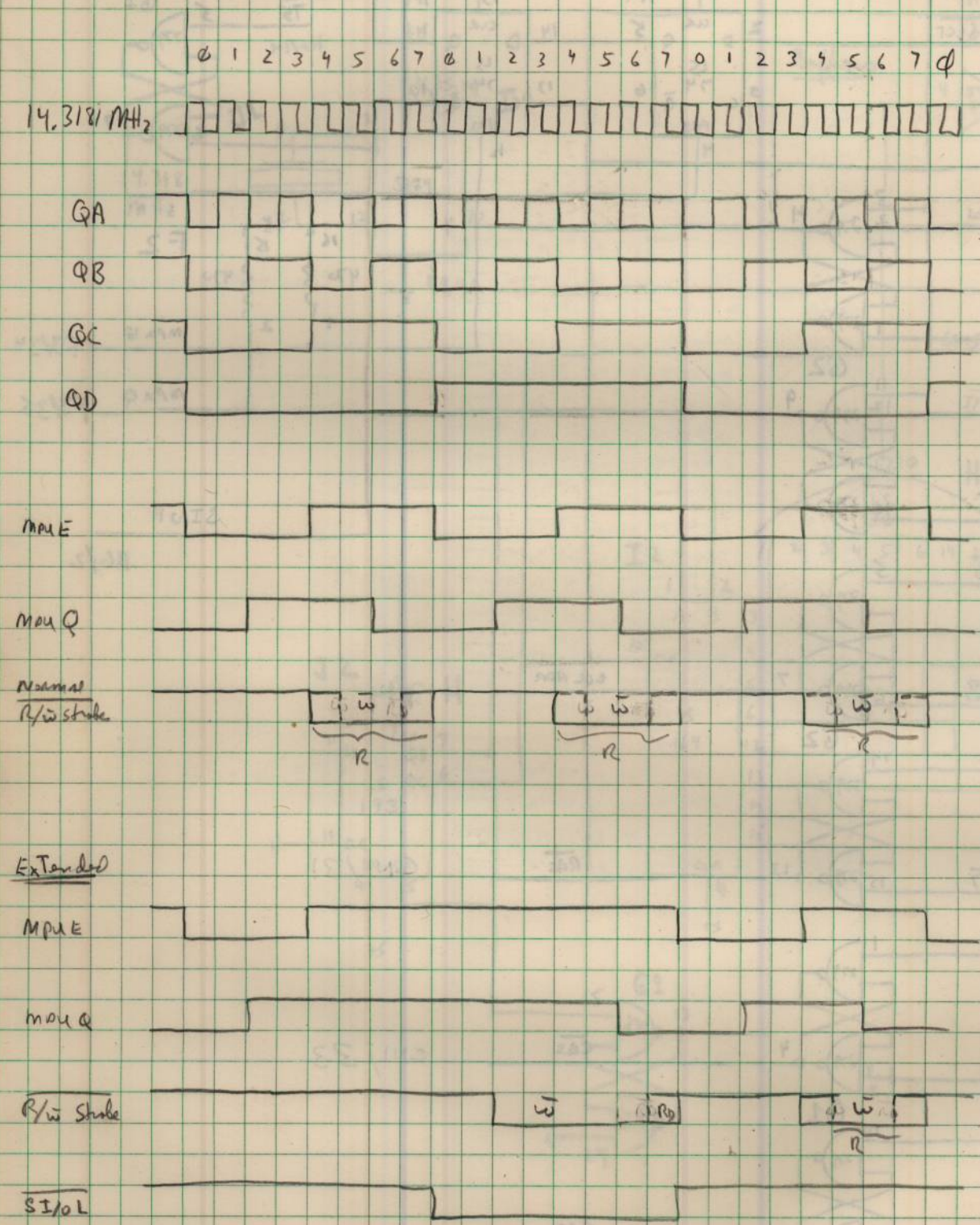
19 Oct 84
ARD

System Clock with Cycle Stretcher for Synchronizing with the graphics Rom

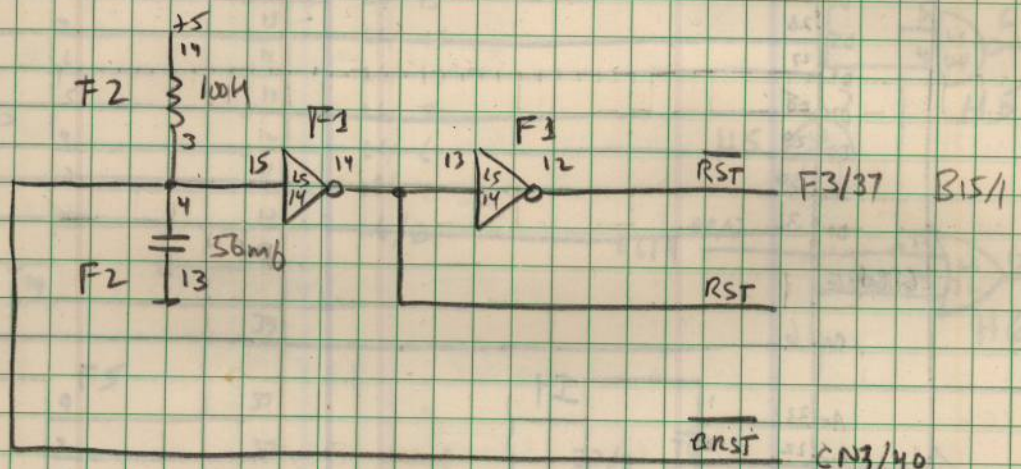


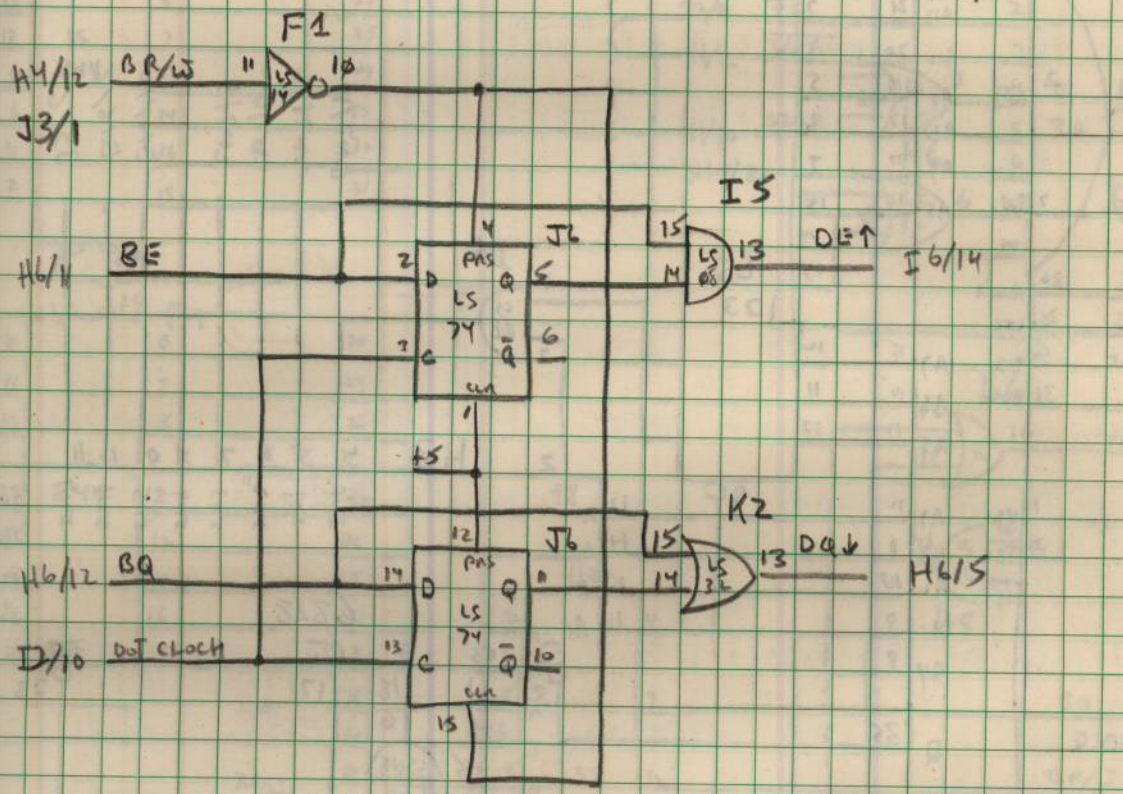

SI/OL
K4/4
H6/1
J3/11
I/O SLCT
H1
LS 74
CLR
D
C
PRS
Q
$\overline{Q}$
+5
G1
$\overline{T5}$
H2/14
279
G1/15
QD
$\overline{T4}$
H2/6
G1/1
G1/2
$\overline{T0}$
G2/6
G2
$\overline{T2}$
G1/12
G2/14
$\overline{T6}$
H2/13
$\overline{T1}$
G1/3
$\overline{T3}$
H2/5
COLADR
H10/14
H2/4
G2/12
$\overline{T7}$
I2/15
G1/10
RAS
CN1/31
G2/2
$\overline{T4}$
G2/1
$\overline{T0}$
CAS
CN1/33
G2/15
$\overline{T7}$
G1
G2/12
$\overline{T2}$
RAS
NC
470
F2
MAUG
MAUQ
F3/34
F3/35
SI/O H
H6/2
19 Oct 84
AB

Clock Timing



RESET GENERATION

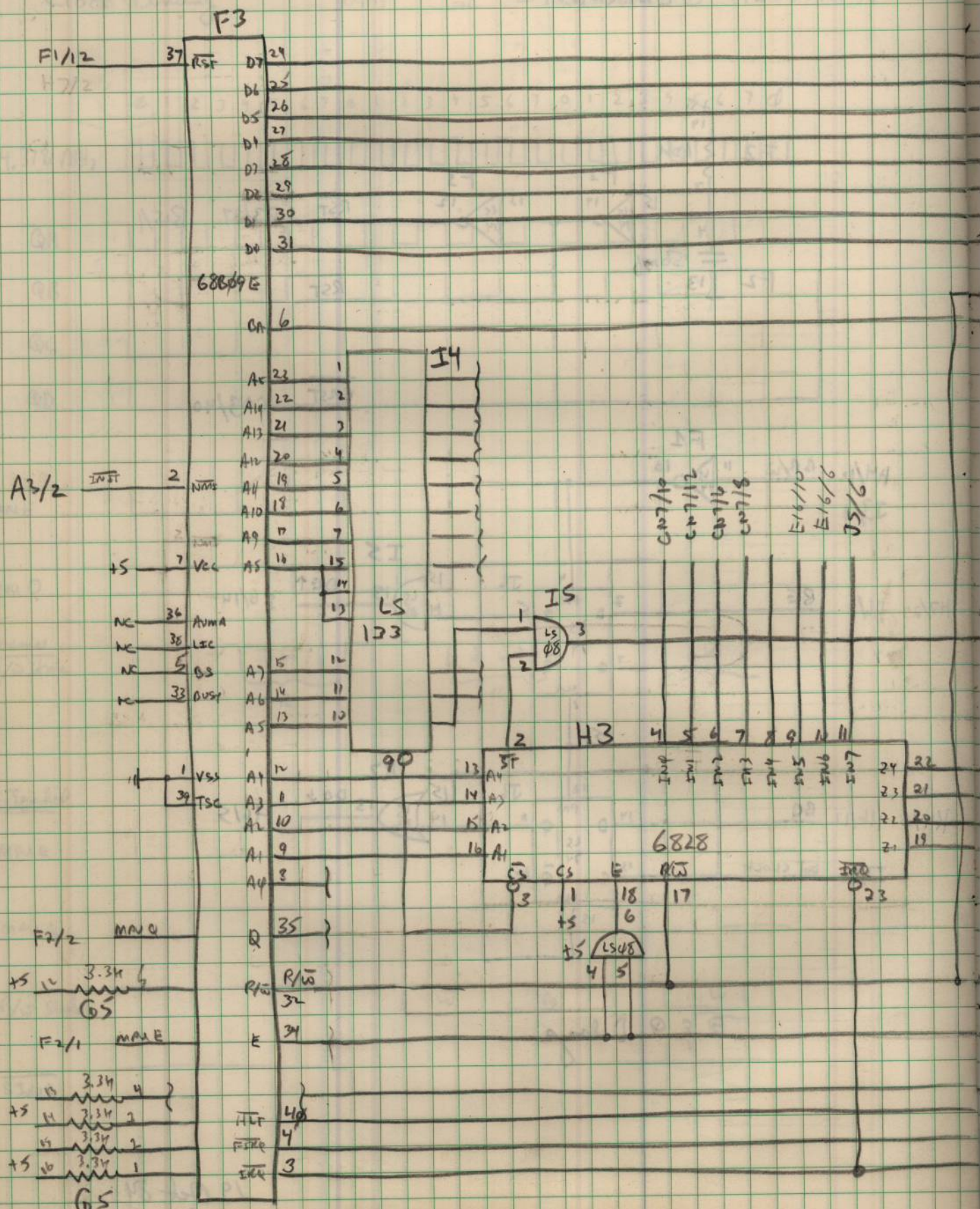




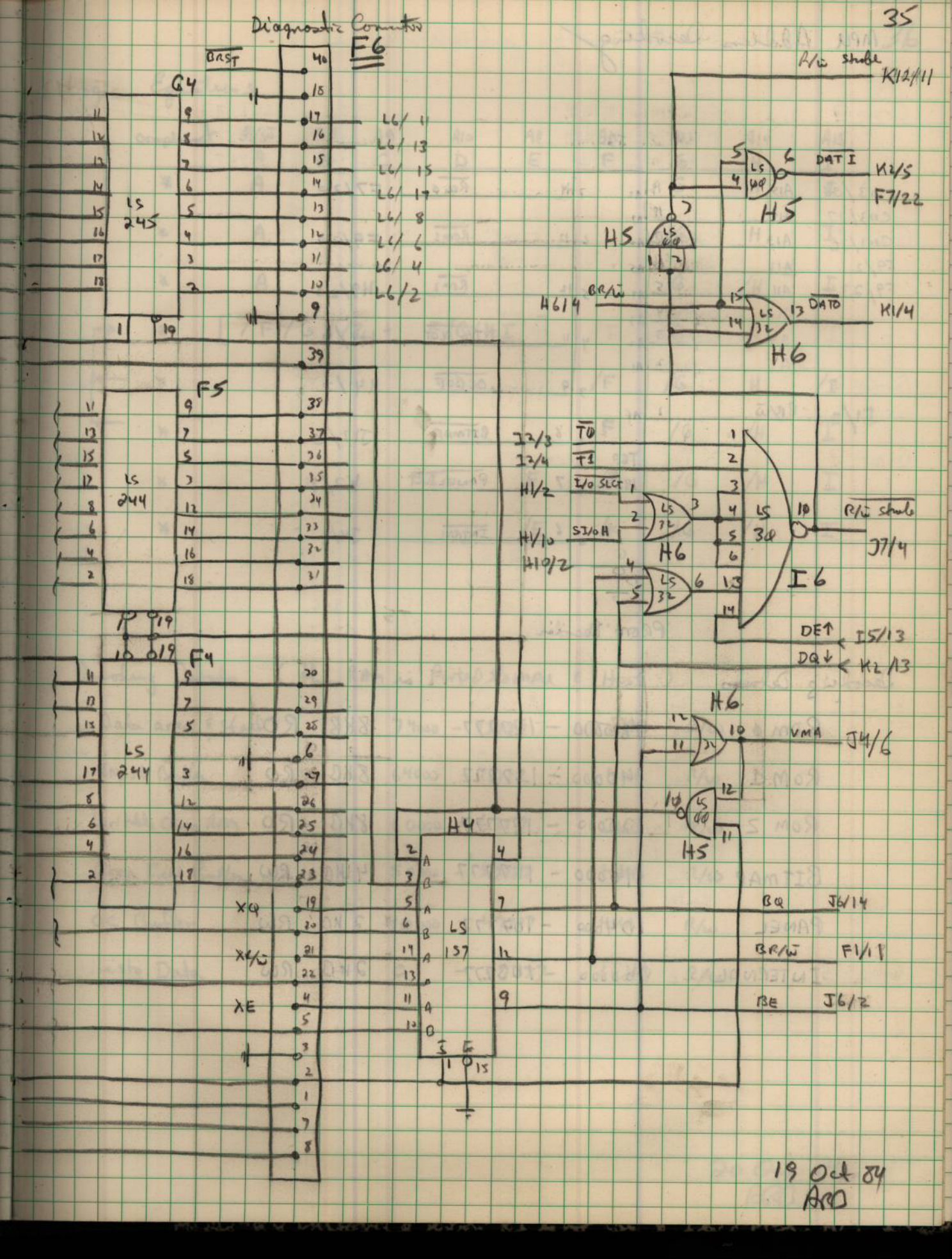
E & Q Delays

19 Oct 84

68B09E MPU Logic

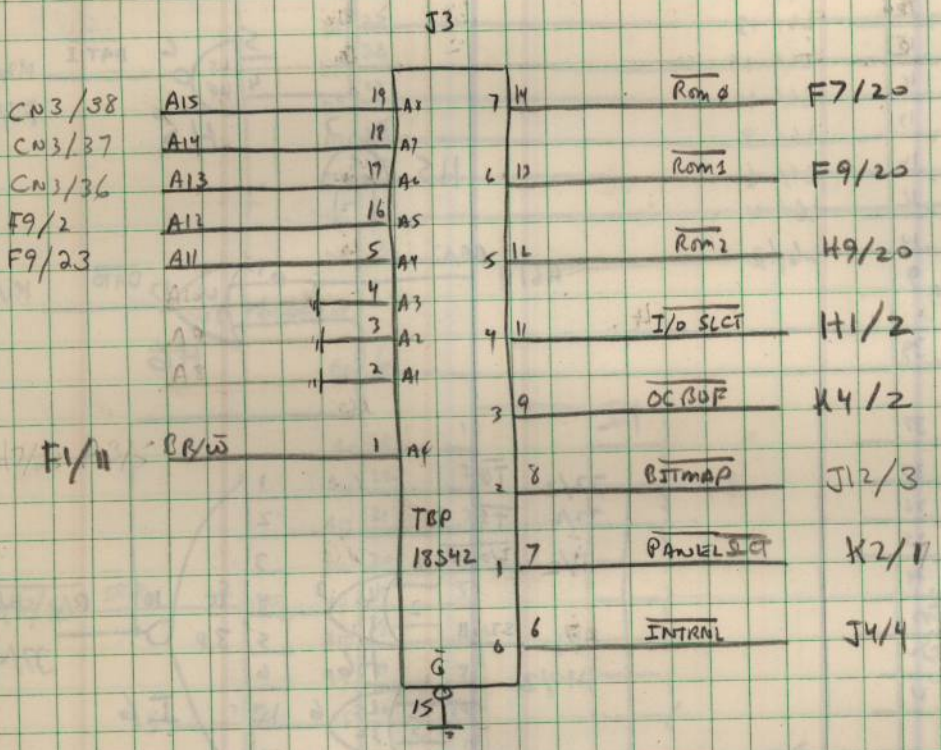


Diagnostic Counter F6



19 Oct 84
ARD

MPU Address Decoding



PRom Decoder

Decoding Details

Rom 0	160000 - 177777	8KB RO
ROM 1	140000 - 157777	8KB RO
Rom 2	120000 - 137777	8KB RO
BITMAP	110000 - 117777	4 KB RW
PANEL	104000 - 107777	2 KB RW
INTERNAL	100000 - 103777	2KB RW

Logic Equations

complement	R/$\bar{W}$	A8	A9	A10	A11	A12	A13	A14	A15
	A	B	C	D	E	F	G	H	I
*	A						G	H	I
*	A						/G	H	I
*	A						G	/H	I
*	[/F/GH/I + F/G/HI]					/F			
*						/F	/G	H	/I
*						F	/G	/H	I
*					E	/F	/G	/H	I
*					/E	/F	/G	/H	I

Decoding Details (This RAM is Ported to map & Host)

Data area & stack	74000 - 77777	2KB	R/W
INput Buffer	64000 - 73777	4KB	R/W
Writable Char RAM	60000 - 63777	2KB	R/W
Area Fill Buffer	50000 - 57777	4KB	R/W
OC Buffer	40000 - 47777	4KB	R/W
Histo Data	0 - 37777	16KB	R/W

20 Oct 84
ARD

I/O Register Addressing (Internal)

Group 7	RGB DAC RMAP		100700 – 100717	WO
	RGB DAC GMAP		100720 – 100737	WO
	RGB DAC BMAP		100740 – 100757	WO
Group 1	Bit plane	CLEAR	100100	WO
	Bit plane	DATA	100101	WO
	Bit plane	WRITE	100102	WO
	Bit plane	Display	100103	WO
	Status	Power	100107	
Group 2	Clock	clear interrupt	100200	WO
Group 3	QB Reg	RDATA	100300	RO
	QB Reg	RCSR	100301	R/W
	QB Reg	TDATA	100302	WO
	QB Reg	TCSR	100303	R/W
	QB Reg	HENBL MSB	100304	RO
	QB Reg	HENBL LSB	100305	RO
	QB Reg	OCENBL MSB	100306	RO
	QB Reg	OCENBL LSB	100307	RO
Group 4	XY Reg	XAXIS MSB	100400	WO
	XY Reg	XAXIS LSB	100401	WO
	XY Reg	YAXIS MSB	100402	WO
	XY Reg	YAXIS LSB	100403	WO
	Plotter Reg	Load	100404	X
	Plotter/Printer	Status	100405	RO
	Plotter/Printer	Control	100406	WO
	Printer	Data	100407	WO
Group 5	MC68045	CRTC	100500	R/W
			100501	WO

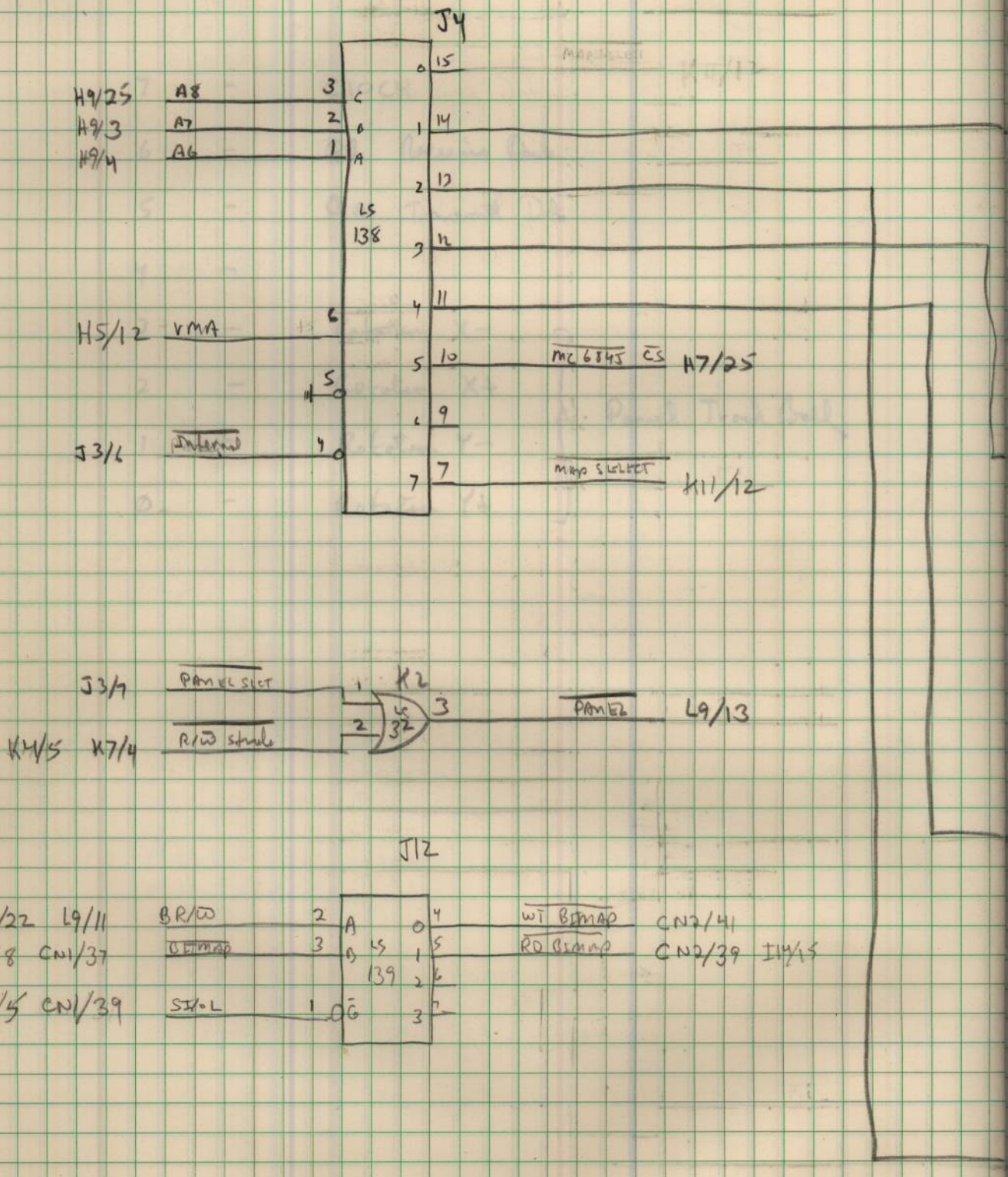
21 Oct 84
ABO

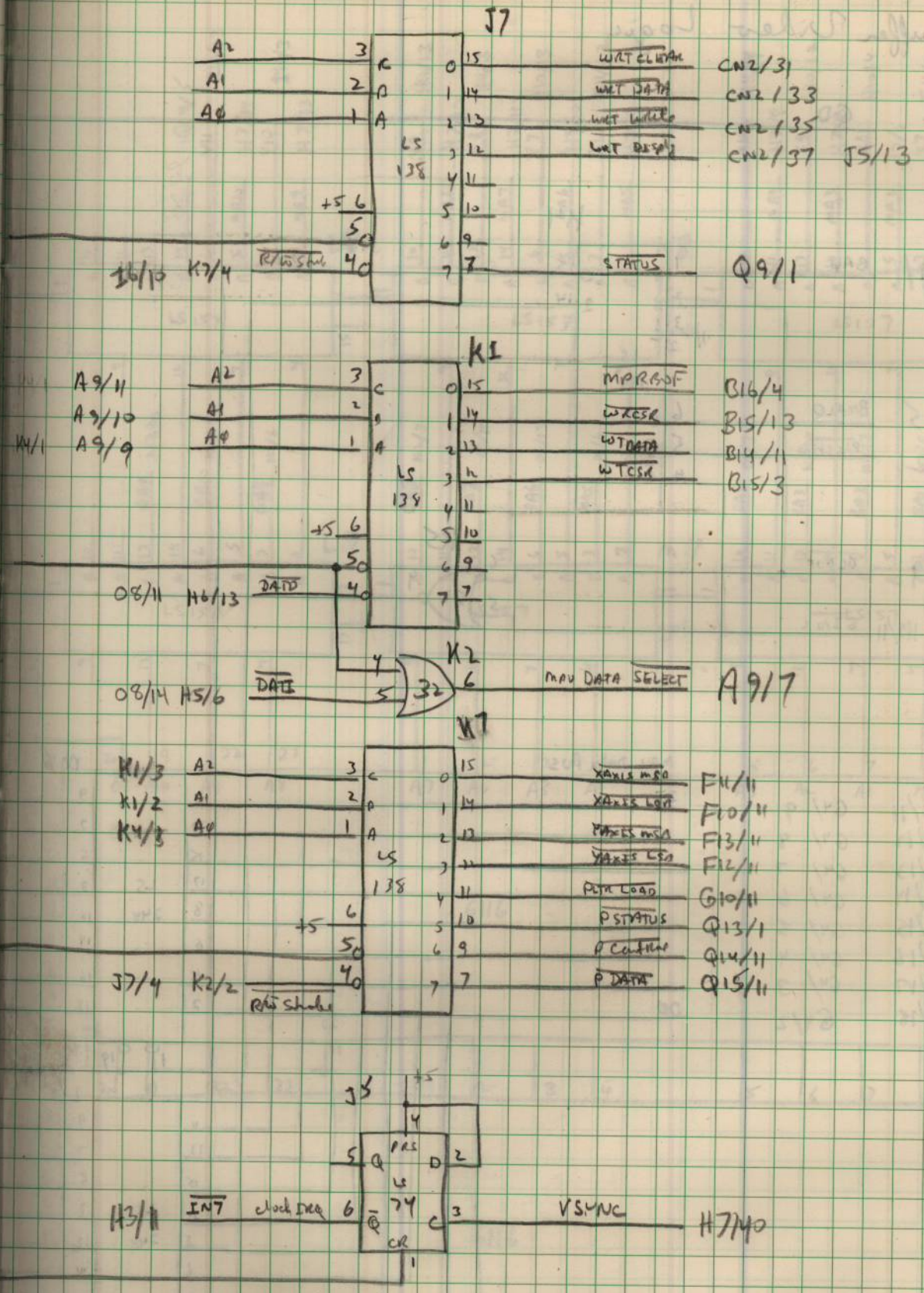
Interrupt levels

7	-	CLOCK Receive Data
6	-	QB Receive Data
5	-	QB Transmit Data
4	-	
3	-	Rotation X-
2	-	Rotation X+
1	-	Rotation Y-
0	-	Rotation Y+

} Panel Track Ball

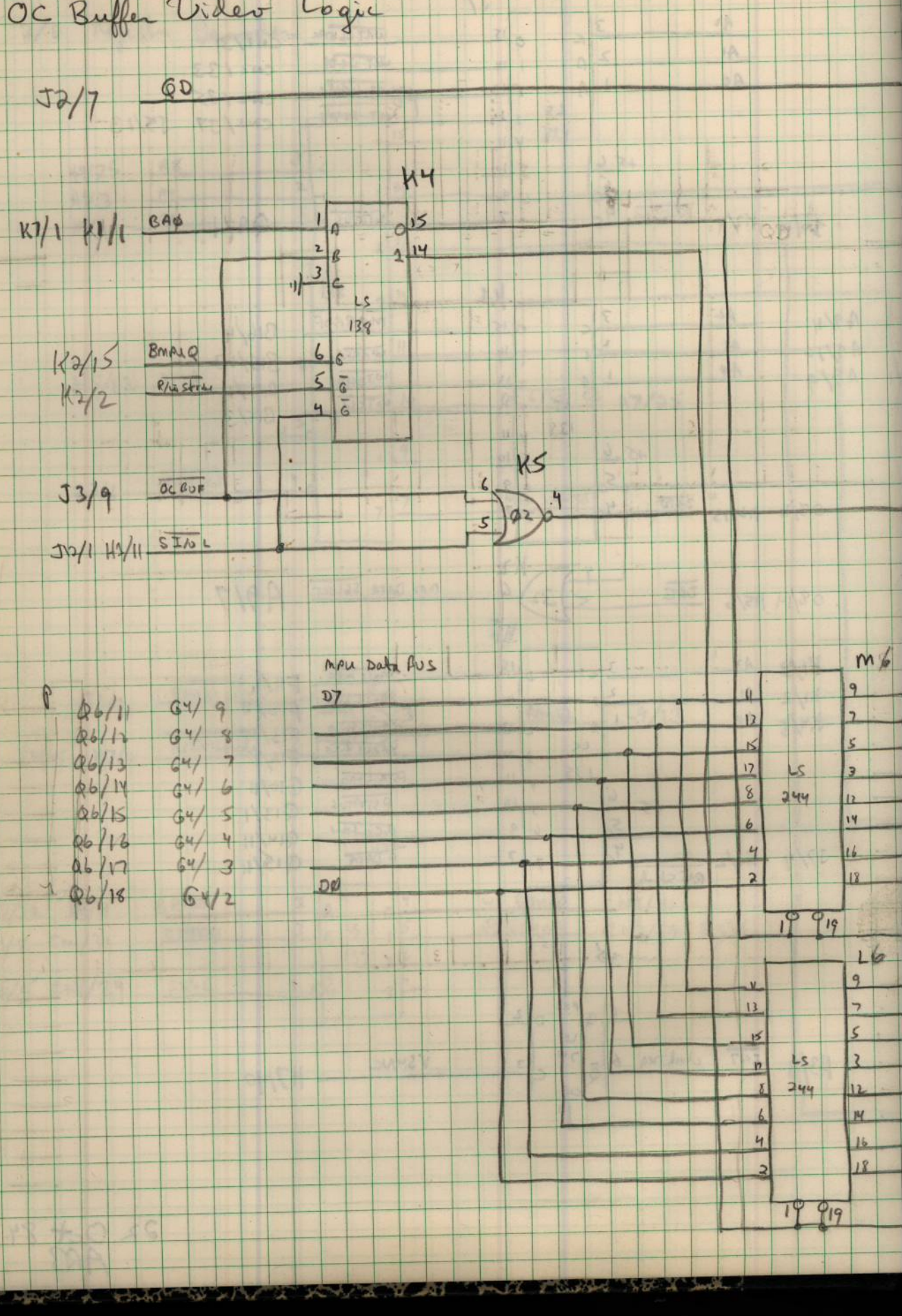
I/O Register Decoder

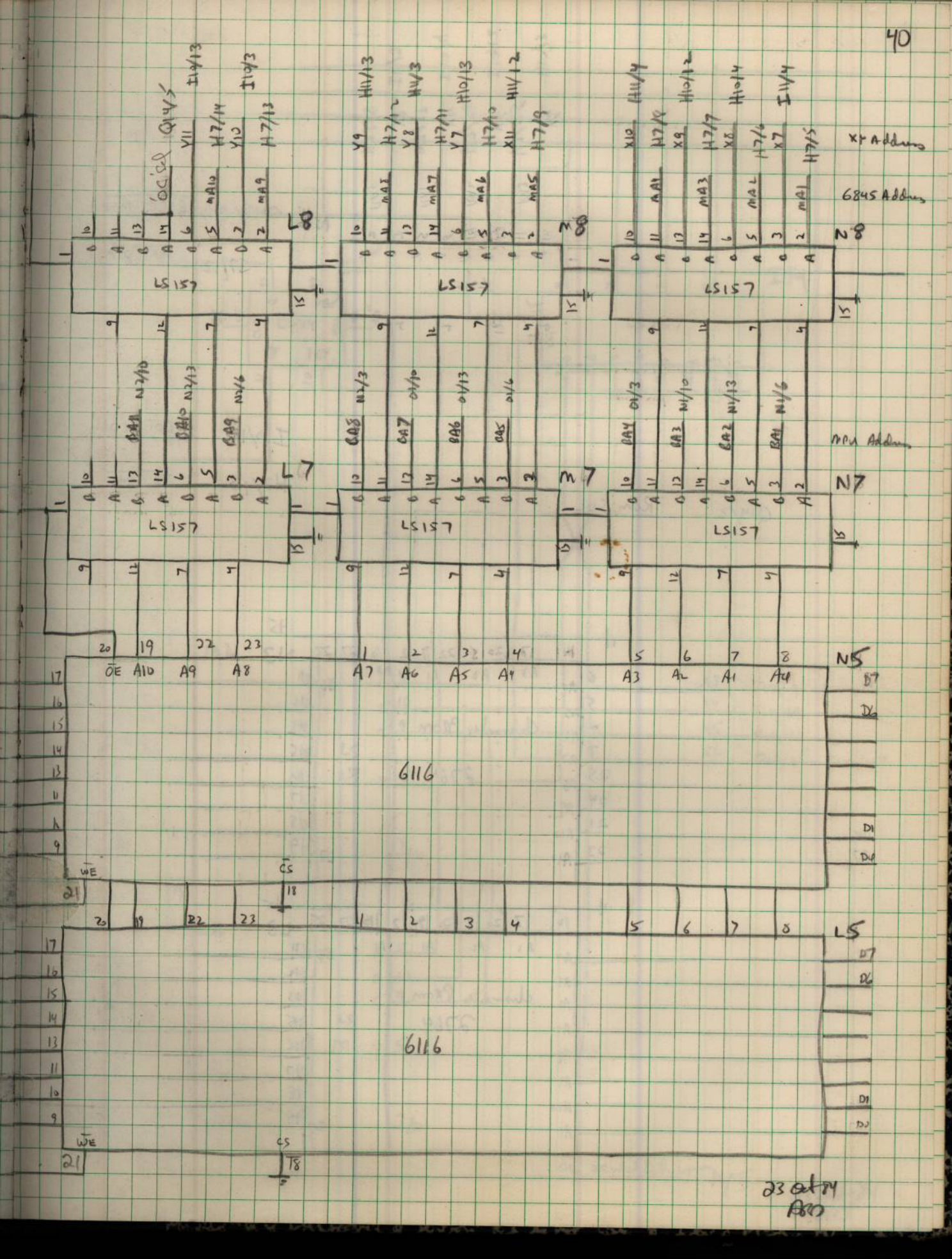


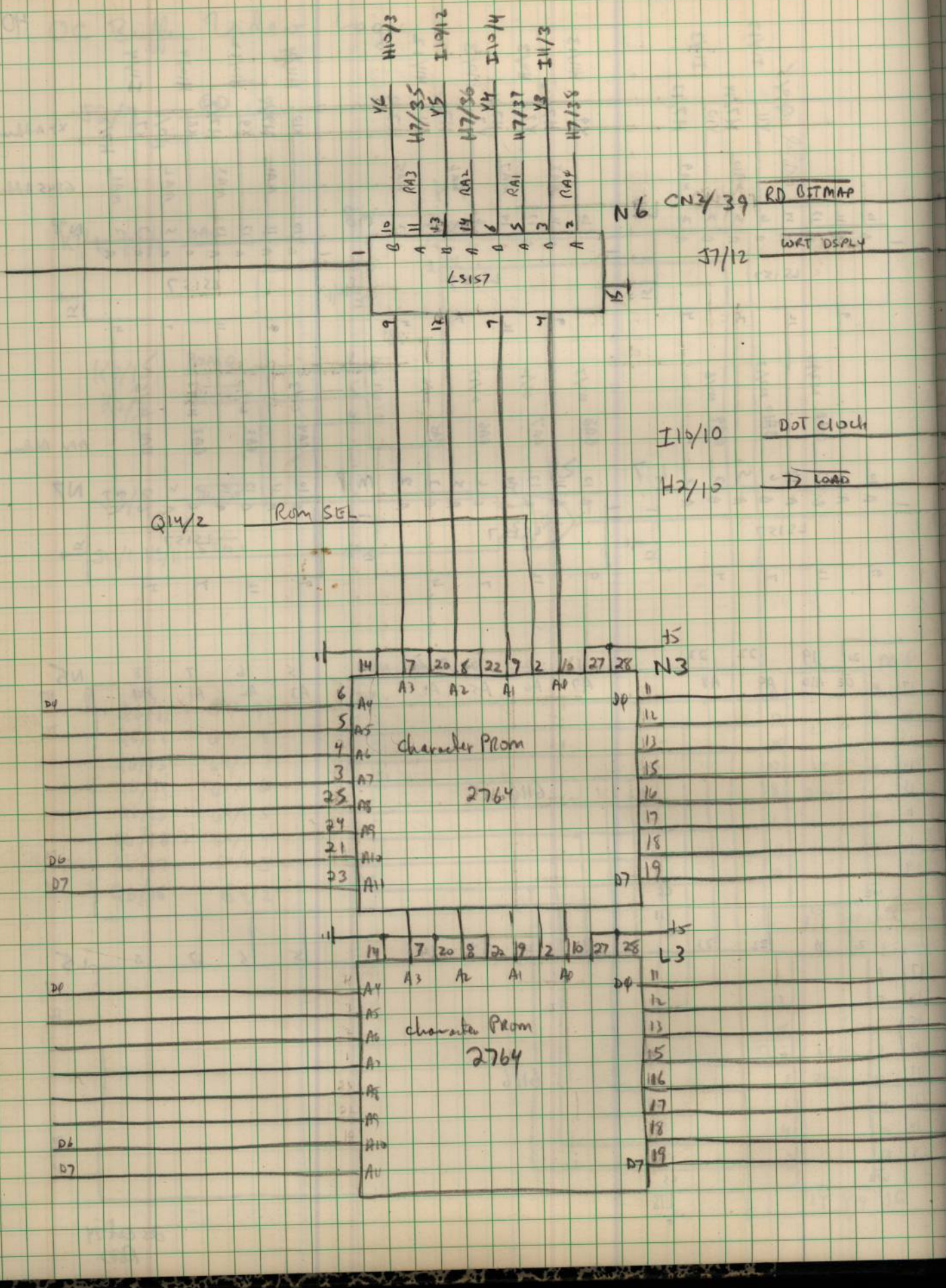

J7
A2
A1
A0
LS 138
+5
RD/WR Strb
16/10 K7/4
WRT CLEAR
WRT DATA
WRT WRITE
WRT DISP
CN2/31
CN2/33
CN2/35
CN2/37 J5/13
STATUS
Q9/1
K1
A9/11
A9/10
A9/9 K4/1
MPRASF
WXESR
WTDATA
WTCSR
B16/4
B15/13
B14/11
B15/3
DATD
08/11 H6/13
K2
32
08/14 H5/6
DATI
MAP DATA SELECT
A9/7
K7
K1/3
K1/2
K4/3
J7/4 K2/2
RAS Strb
XAXIS MSB
XAXIS LSB
YAXIS MSB
YAXIS LSB
PLTR LOAD
P STATUS
P CONTROL
P DATA
F11/11
F10/11
F13/11
F12/11
G10/11
Q13/1
Q14/11
Q15/11
J5
74
PRE
CLR
INT
Clock Req
13/11
VSYNC
H7/10

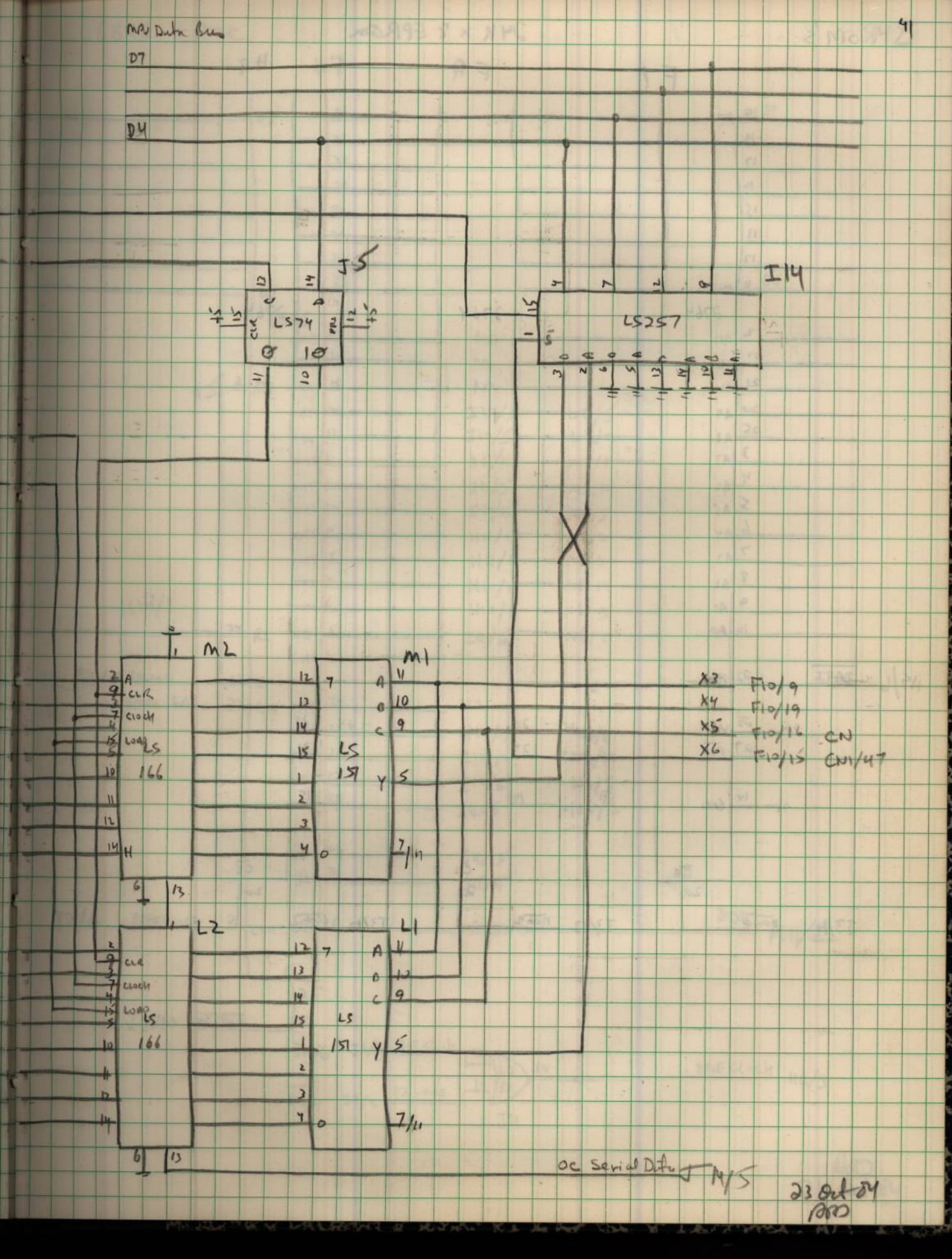
22 Oct 84
ARB

OC Buffer Video Logic



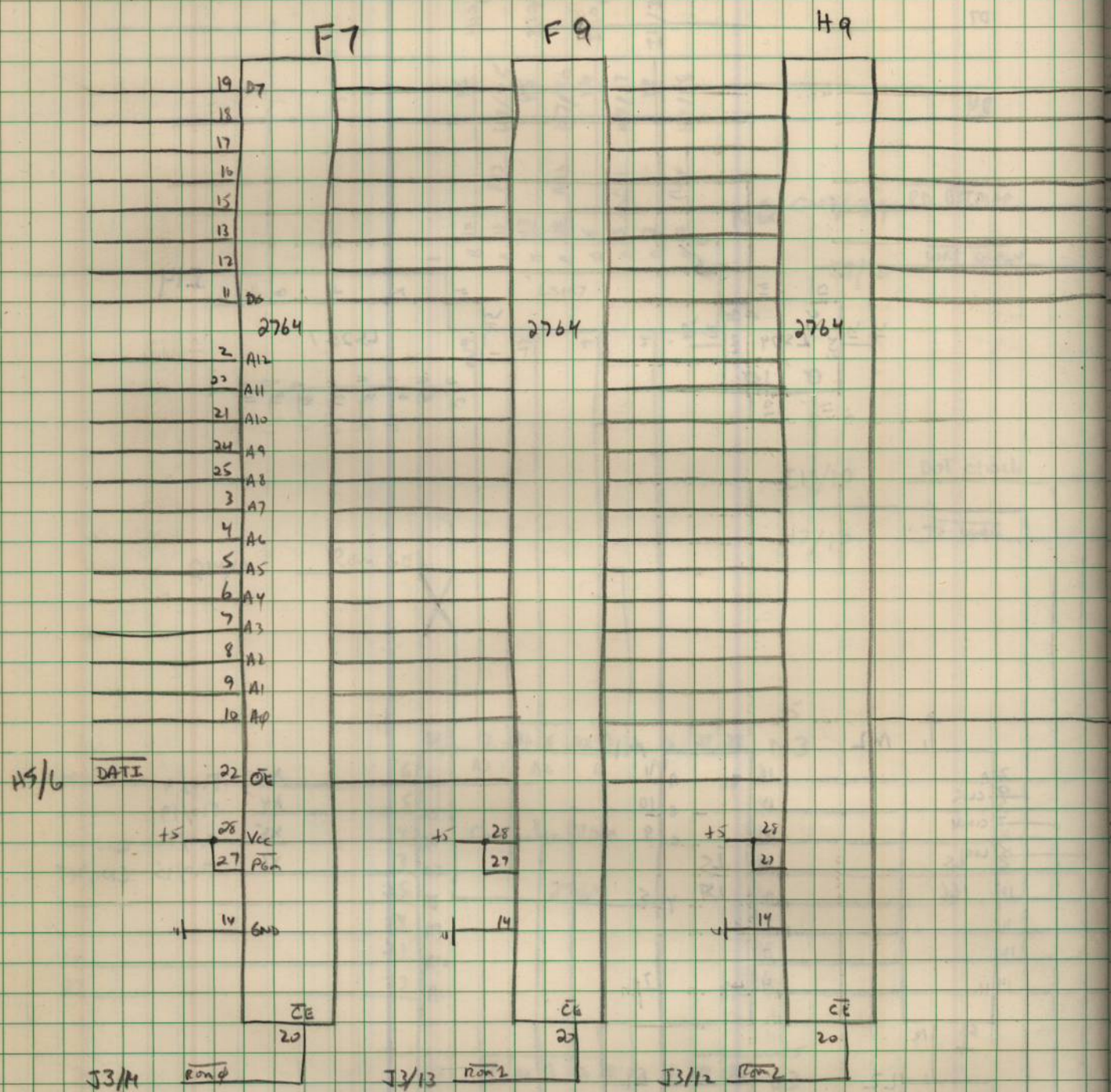

LS157
6116
6116
OE
WE
CS
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
D7
D6
D1
D0
N5
N7
L5
L8
I14/13
I14/3
I14/4
I14/5
H11/13
H11/12
H11/4
H10/13
H10/12
H10/4
H7/14
H7/13
H7/12
H7/11
H7/10
H7/9
H7/8
H7/7
H7/6
H7/5
N2/13
N2/10
N2/6
O1/13
O1/10
O1/6
N1/13
N1/10
N1/6
O1/3
N2/3
BA10
BA9
BA8
BA7
BA6
BA5
BA4
BA3
BA2
BA1
BA0
6845 Addr
MA11
MA10
MA9
MA8
MA7
MA6
MA5
MA4
MA3
MA2
MA1
MA0
23 of 24


H10/3
I10/12
I10/4
I11/3
Y6
Y5
Y4
Y3
H7/35
H7/36
H7/37
H7/38
RA3
RA2
RA1
RA0
N6
CN3/39
RD BITMAP
J7/12
WRT DSPLY
LS157
I10/10
DOT clock
H2/10
LOAD
Q14/2
ROM SEL
N3
L3
A3
A2
A1
A0
Character PRom
2764
character PRom
2764
D0
D6
D7


MPU Data Bus
D7
D4
J5
LS74
I14
LS257
M2
M1
L2
L1
LS 166
LS 151
CLR
CLOCK
LOAD
X3 F10/9
X4 F10/10
X5 F10/16 CN
X6 F10/15 CN1/47
OC Serial Data
23 Oct 84

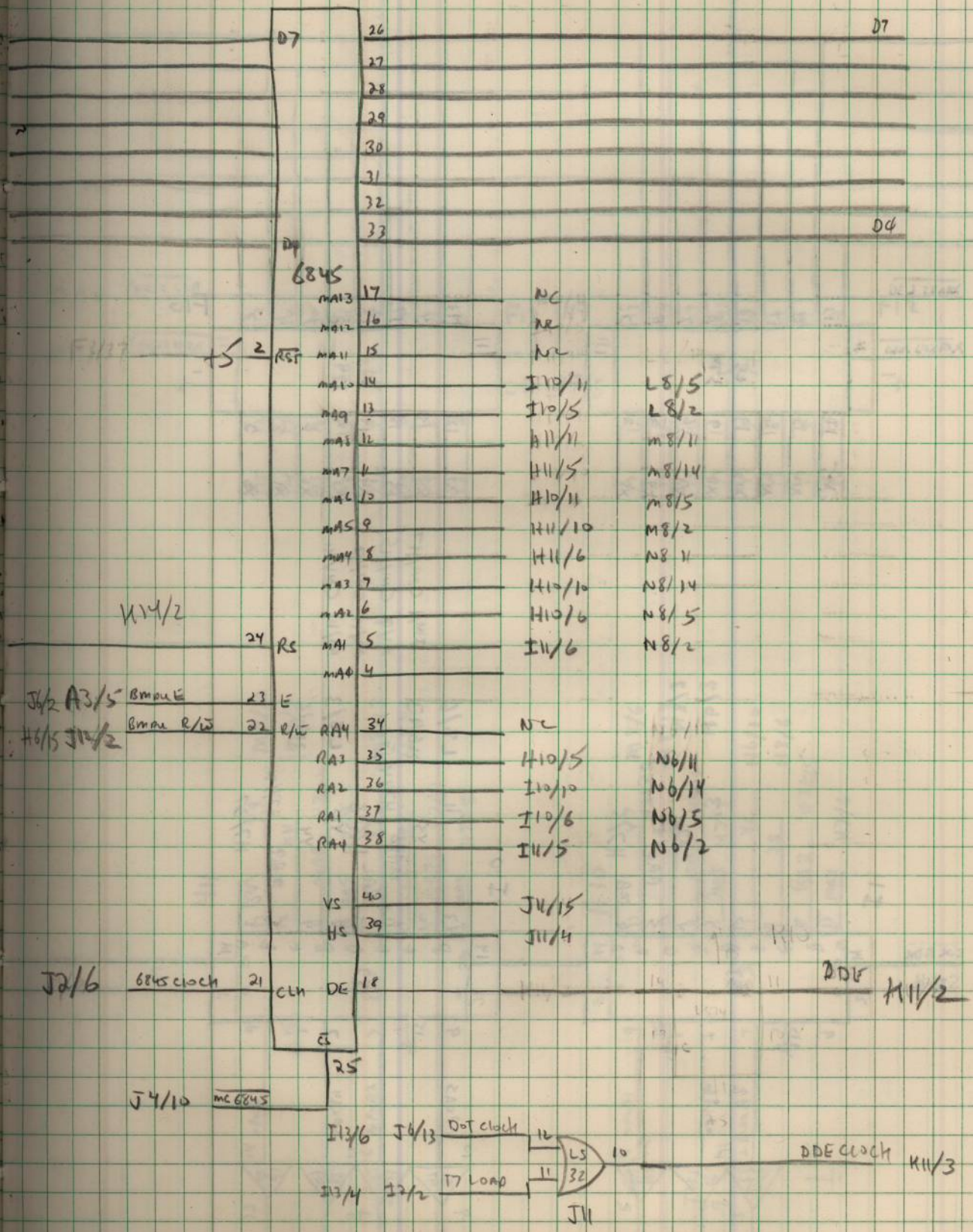
Rom's

24K x 8 EPROM

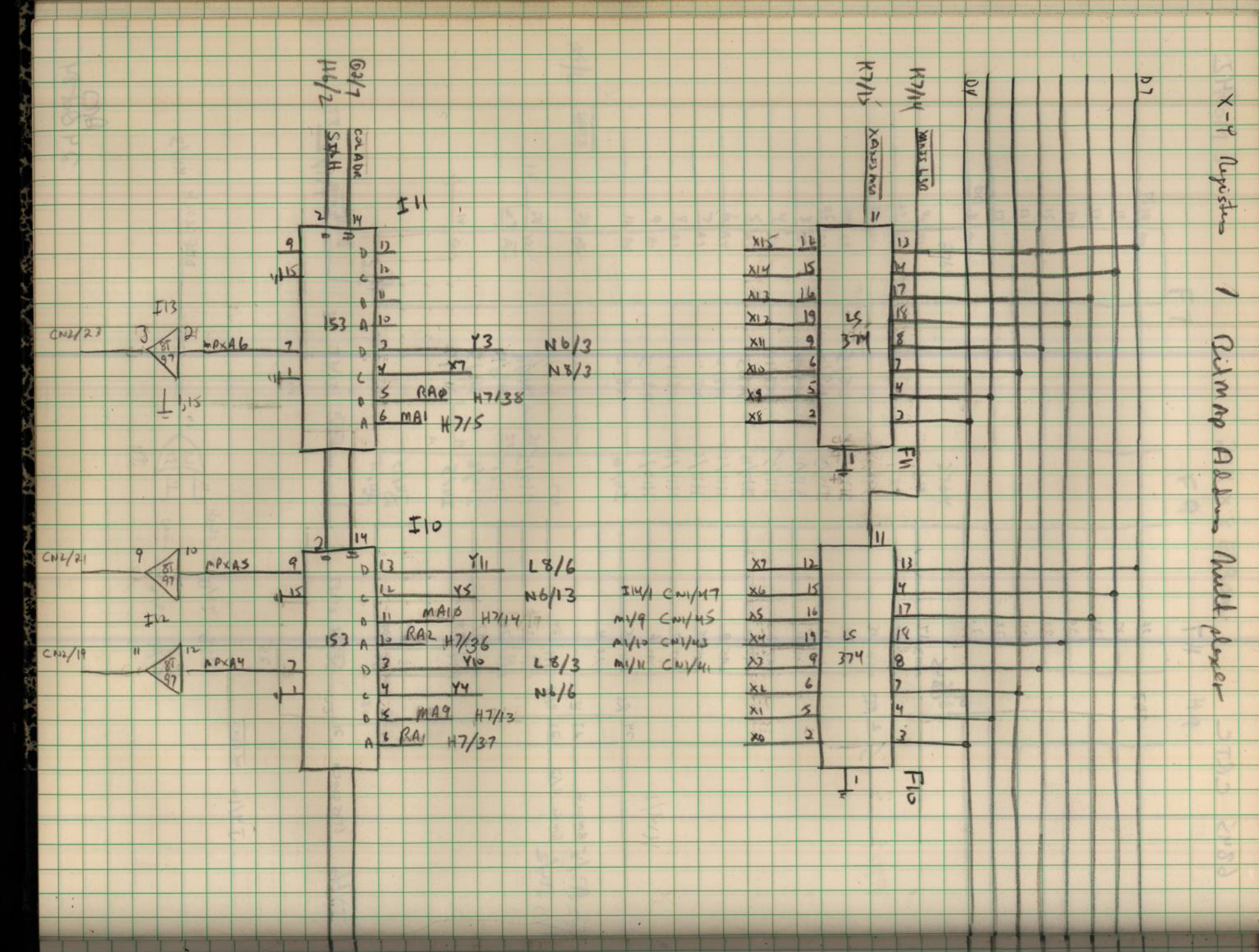


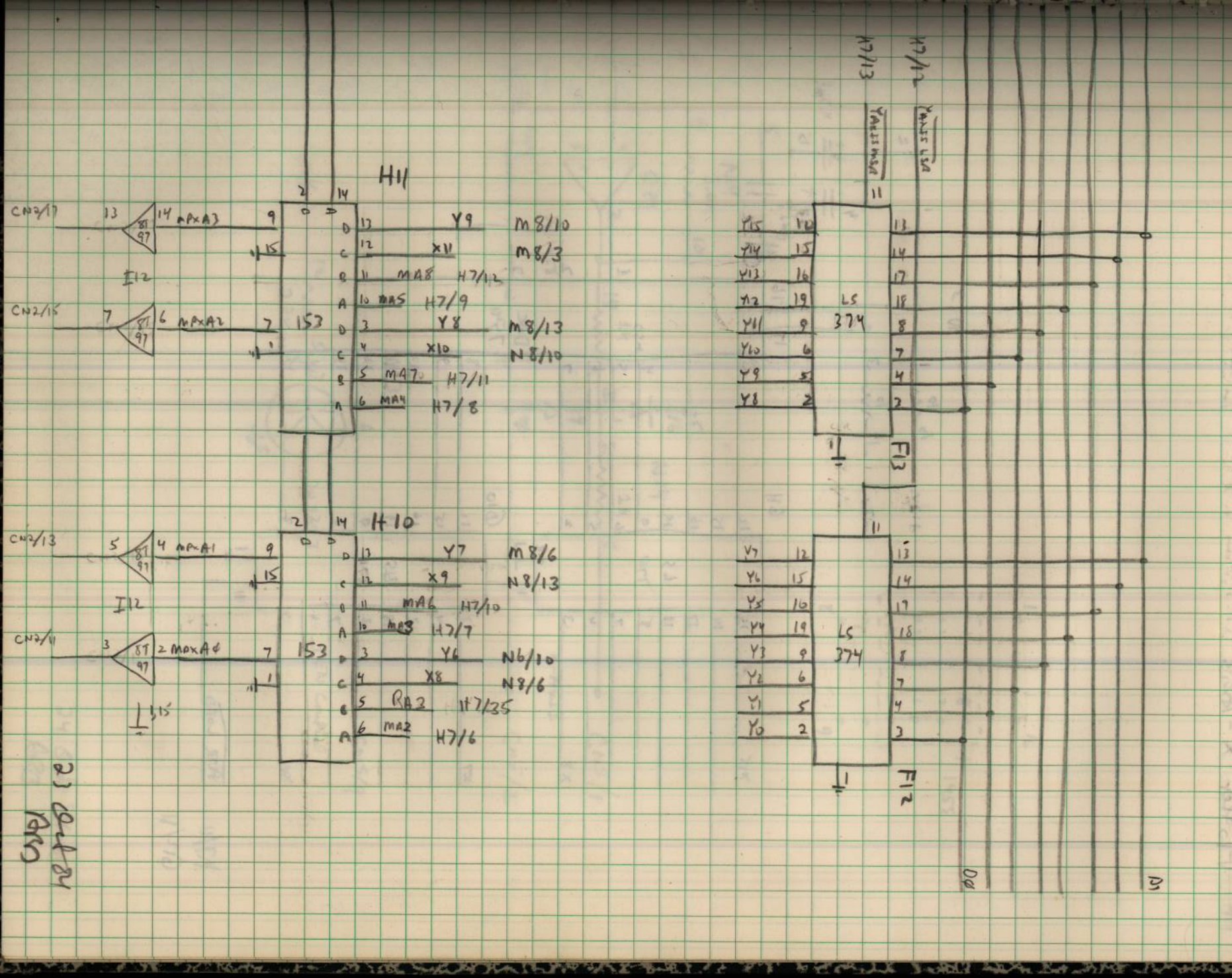
6845 CRTC

#7

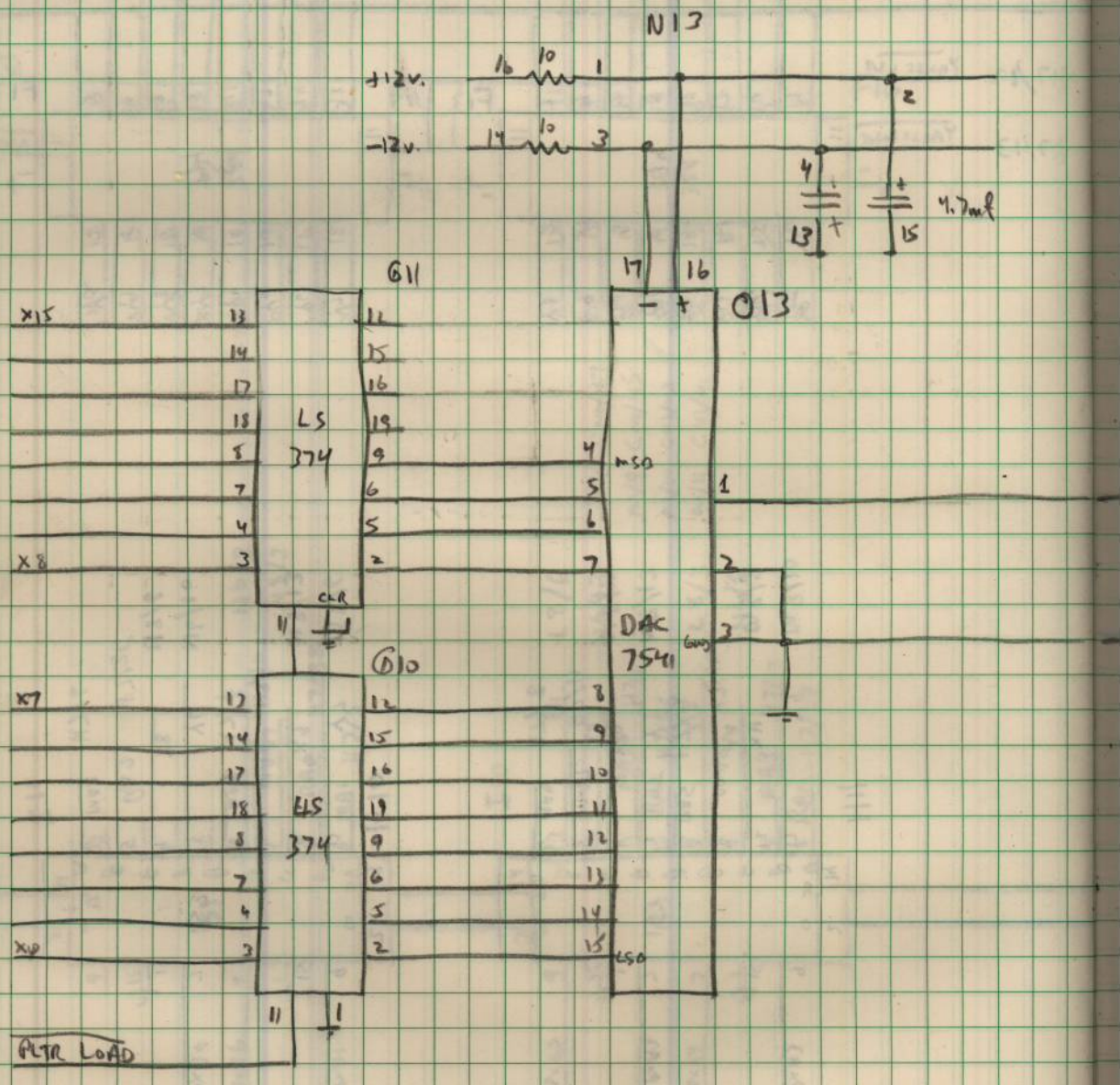


AO
24 Oct 84


X-Y Register / Bitmap Address Multiplexer
D7
D0
H7/15
XADDRMUX
H7/14
YADDRMUX
F11
F10
I11
I10
I12
I13
153
CN2/23
CN2/21
CN2/19
MPXA6
MPXA5
MPXA4
Y3
X7
RA0
MA1
Y11
Y5
MA10
RA2
Y10
Y4
MA9
RA1
N6/3
N8/3
H7/38
H7/5
L8/6
N6/13
H7/14
H7/36
L8/3
N6/6
H7/13
H7/37
I14/1 CN1/17
M1/9 CN1/45
M1/10 CN1/43
M1/11 CN1/41
LS 374

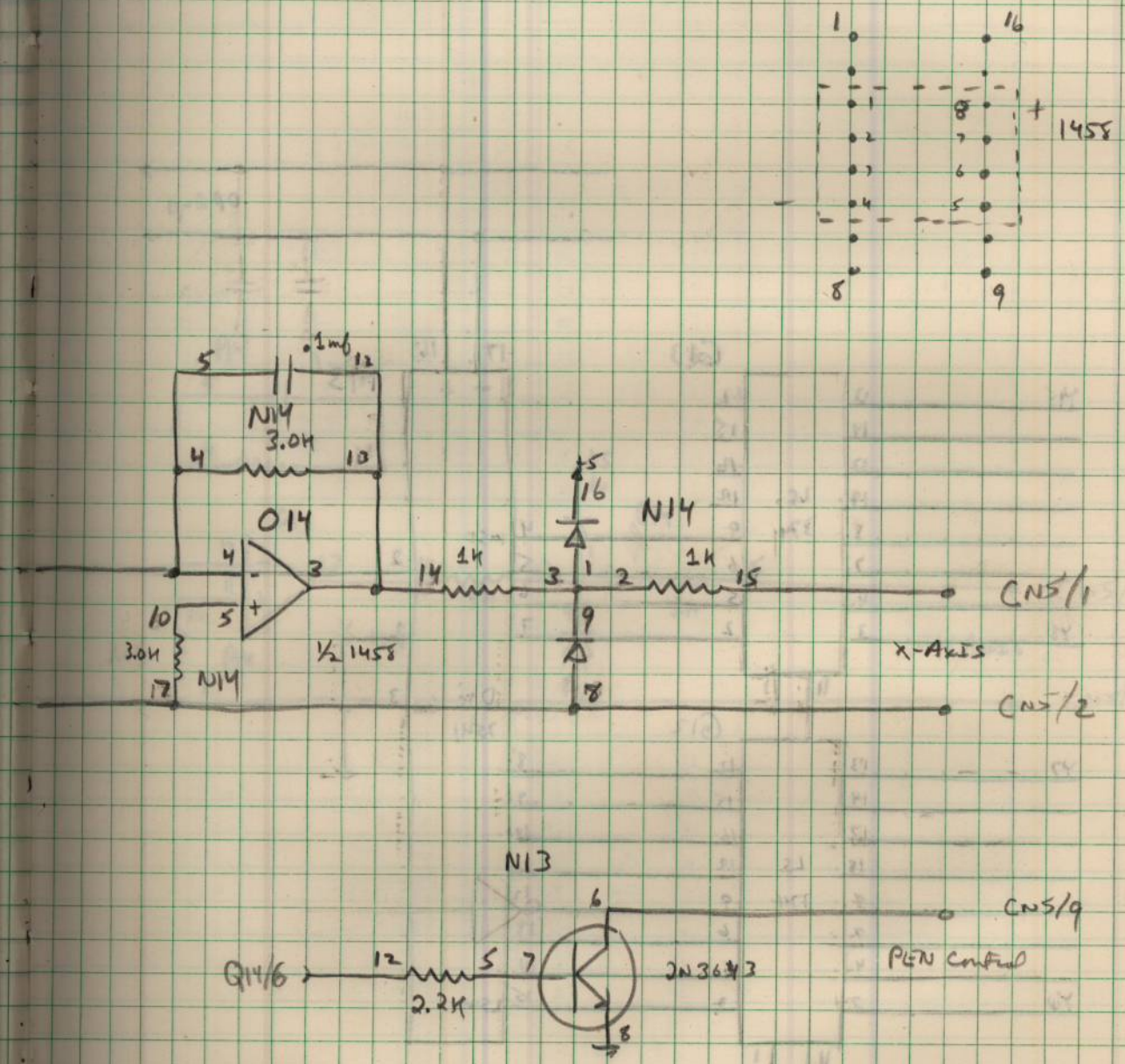

H11
H10
153
153
CN2/17
CN2/15
CN2/13
CN2/11
MAXA3
MAXA2
MAXA1
MAXA0
I12
I12
I15
Y9
X11
MA8
H7/12
MA5
H7/9
Y8
X10
MA7
H7/11
MA4
H7/8
Y7
X9
MA6
H7/10
MA3
H7/7
Y6
X8
RA3
H7/35
MA2
H7/6
M8/10
M8/3
M8/13
N8/10
M8/6
N8/13
N6/10
N8/6
H7/13
H7/12
YADDRMSB
YADDRLSB
LS
374
F13
F12
DB
13
43

Plotter X-Axis



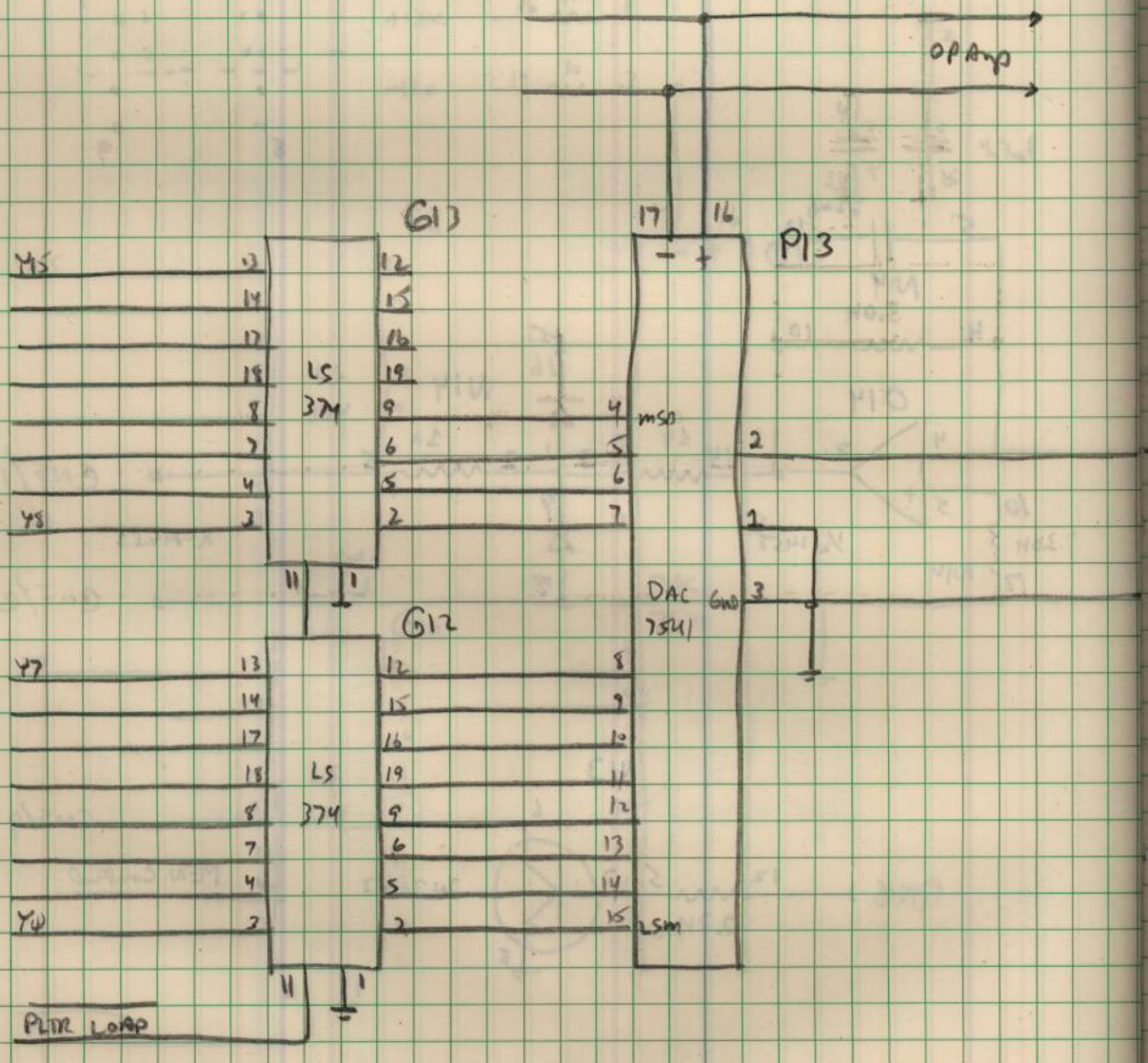
A7/11

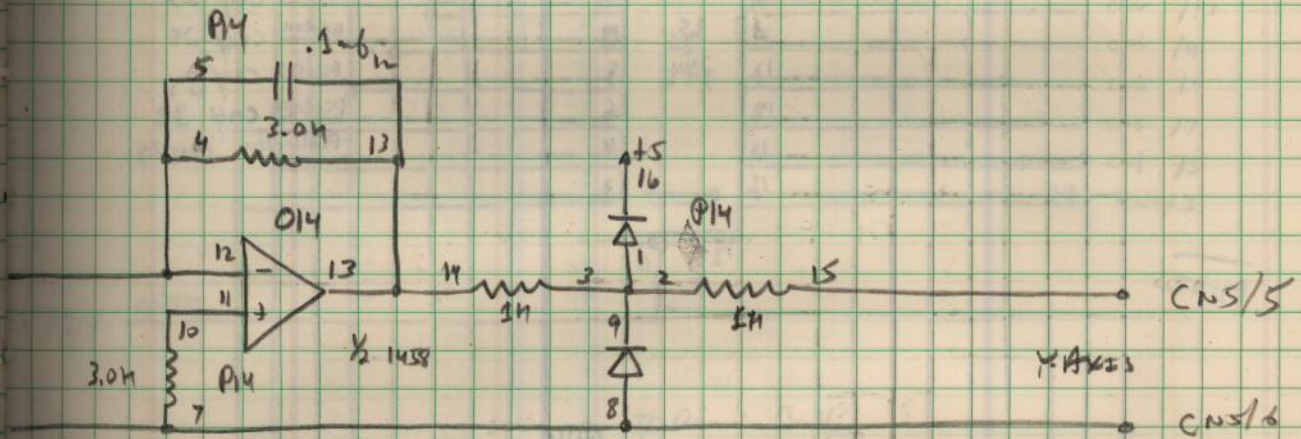
G12/11


1
16
8
9
1458
.1mf
5
12
N14
3.0H
4
10
O14
+5
16
N14
1K
1K
14
3
1
2
15
CN5/1
X-AXIS
9
10
5
3.0H
17
N14
½ 1458
8
CN5/2
N13
6
CN5/9
PEN CONTROL
Q14/6
12
5
7
2N3643
2.2K
8

24 Oct 84
ABB

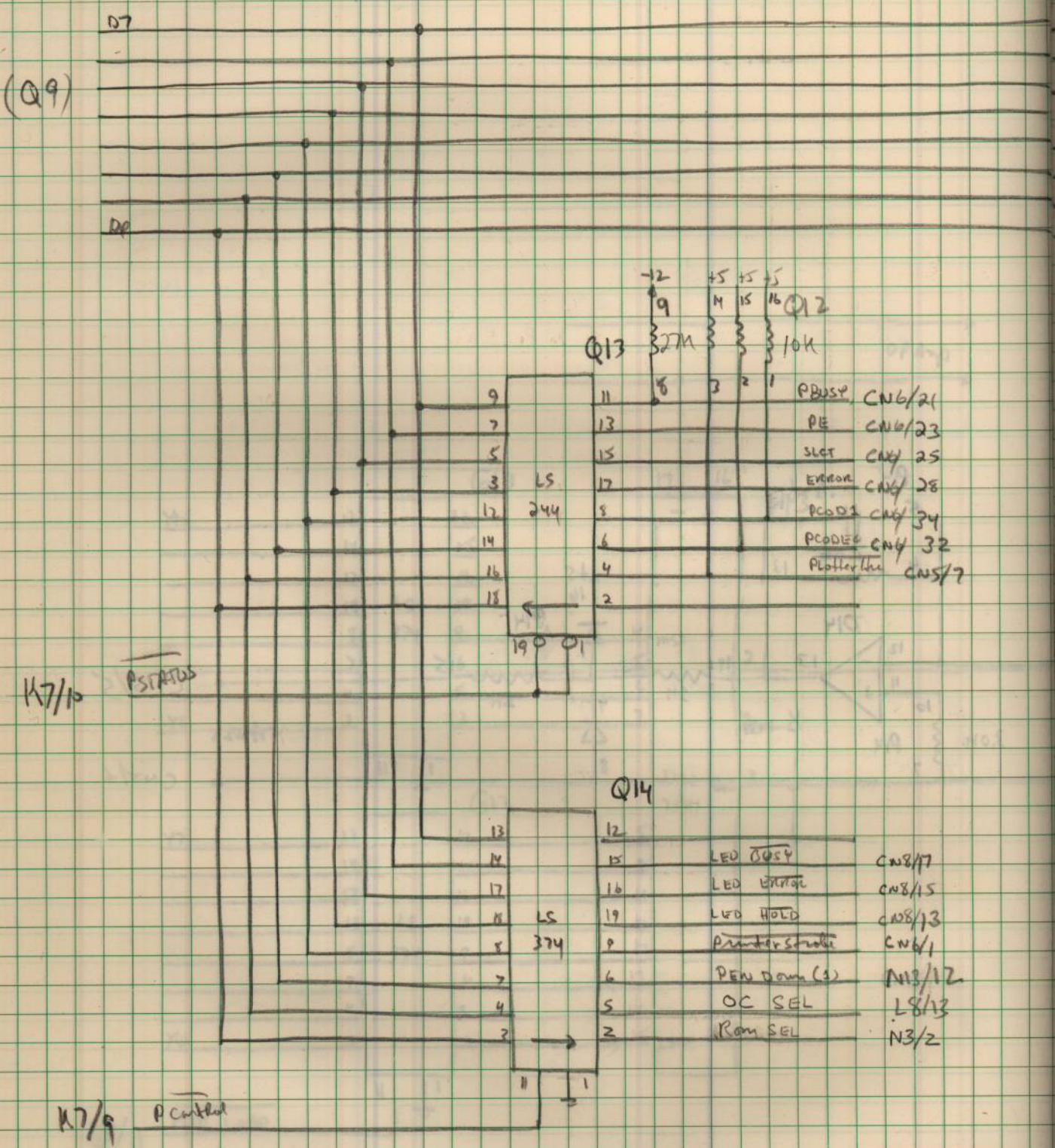
Plotter Y-Axis (Complementary input)

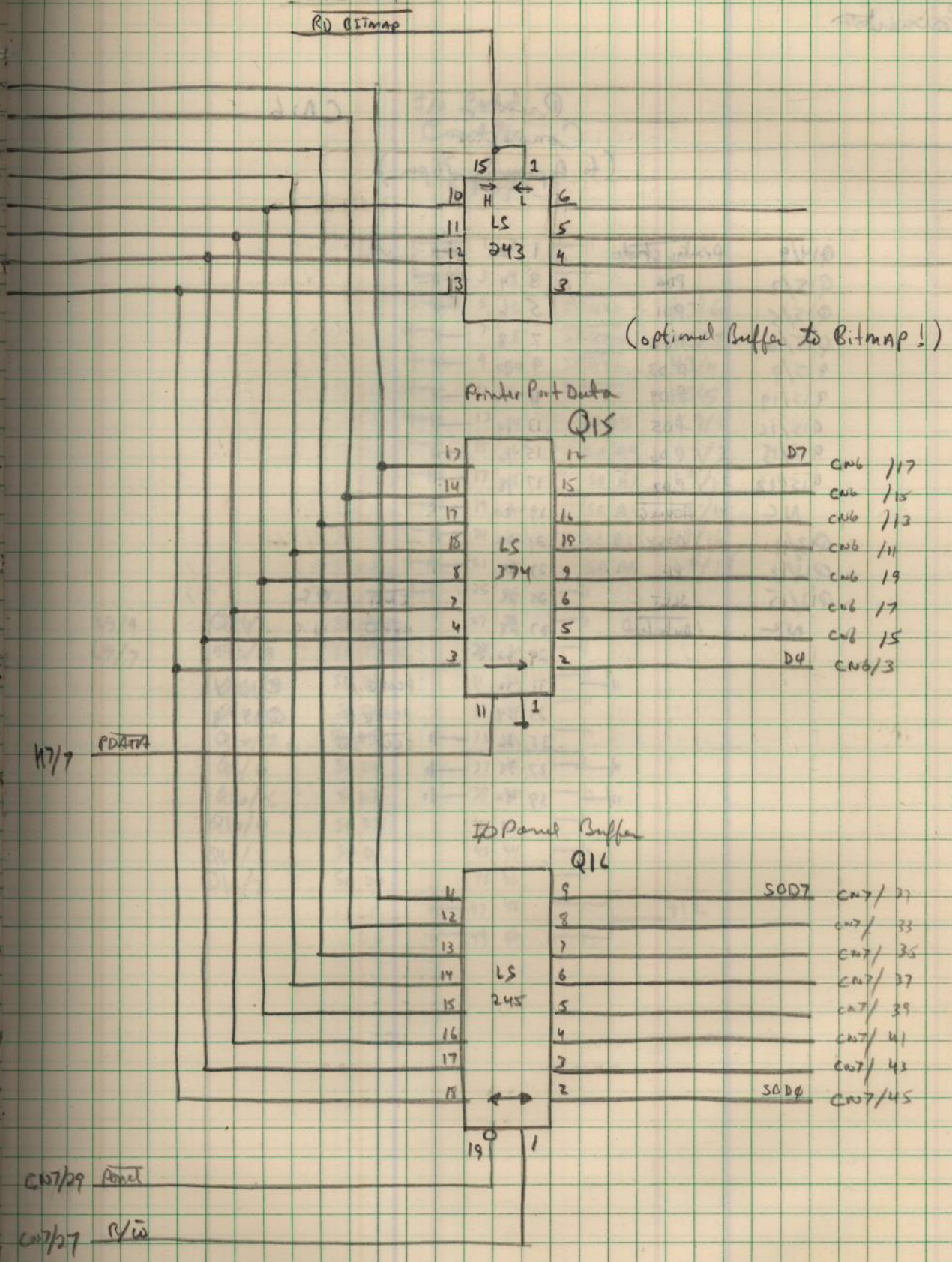



A4
.1 uf
5
3.0M
4
13
O14
12
11
10
7
13
14
1M
3
2
15
1M
+5
16
P14
1
9
8
3.0M
P14
½ 1458
CN5/5
CN5/6

24 Oct 84
ABD

Printer Control / I/O Panel Buffer / Pen Control



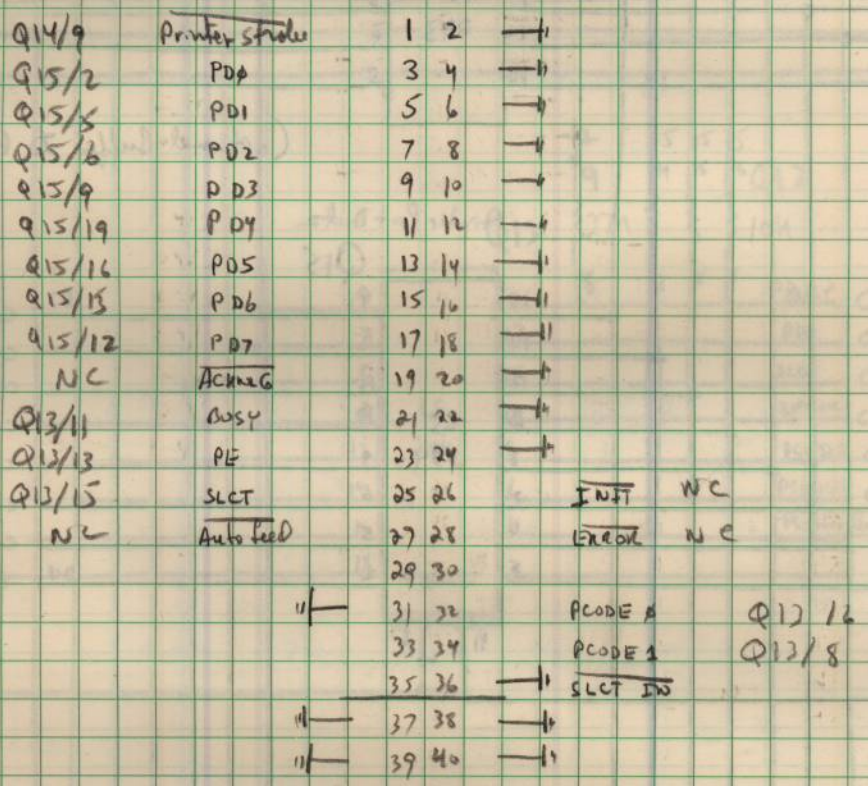

RD BITMAP
15
1
H
L
10
11
12
13
LS
243
6
5
4
3
(optimal Buffer to Bitmap!)
Printer Port Data
Q15
17
14
17
18
8
7
4
3
LS
374
12
15
16
19
9
6
5
2
D7
D6
CN6 /17
CN6 /15
CN6 /13
CN6 /11
CN6 /9
CN6 /7
CN6 /5
CN6/3
11
1
H7/7
PDATA
IO Panel Buffer
Q16
11
12
13
14
15
16
17
18
LS
245
9
8
7
6
5
4
3
2
SOD7
SOD6
CN7/ 37
CN7/ 33
CN7/ 35
CN7/ 37
CN7/ 39
CN7/ 41
CN7/ 43
CN7/45
19
1
CN7/29
PANEL
CN7/27
R/W

24 Oct 84
ABC

Connector

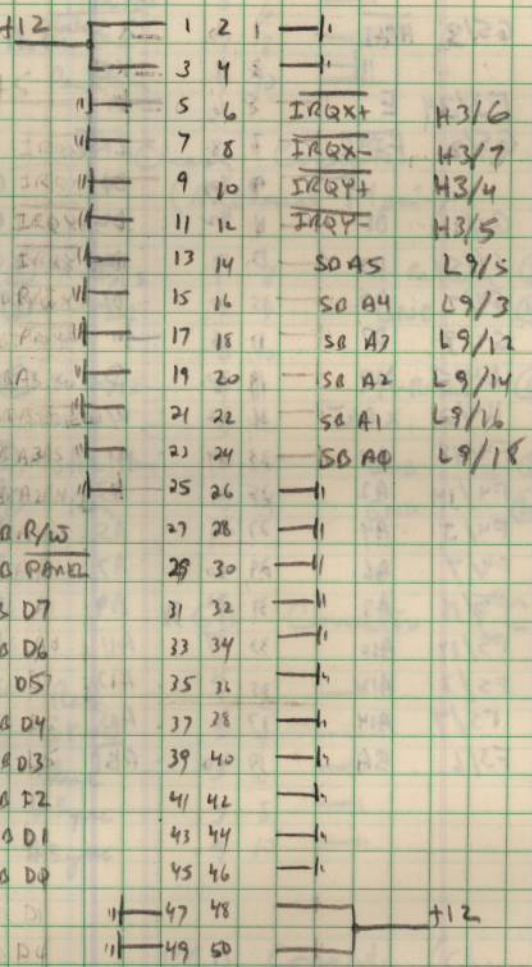
Printer Connector (to Alphanet 36 pin)

CN6



I/O Panel Connector

CN7



24 Out of 84
ARD

Connectors

Diagnostic (6809)

CN3

	G5/3	$\overline{HALT}$	1 2	X Select	G5/4		
		⊣⊢	3 4	XE	H4/11		
	F3/34	E	5 6	⊣⊢			
	G5/2	$\overline{FIRQ}$	7 8	$\overline{IRQ}$	G5/1		
		⊣⊢	9 10	D0	G4/2	F7/11	
F7/12	G4/3	D1	11 12	D2	G4/4	F7/13	
F7/15	G4/5	D3	13 14	D4	G4/6	F7/16	
F7/17	G4/7	D5	15 16	D6	G4/8	F7/18	
F7/19	G4/9	D7	17 18	⊣⊢			
	H4/5	XQ	19 20	Q	F3/35		
	H4/21	X R/$\overline{W}$	21 22	R/$\overline{W}$	F3/32		
F7/10	F4/18	A0	23 24	A1	F4/16	F7/9	
F7/8	F4/14	A2	25 26	A3	F4/12	F7/7	
F7/6	F4/3	A4	27 28	A5	F4/5	F7/5	
F7/4	F4/7	A6	29 30	A7	F4/9	F7/3	
F7/25	F5/18	A8	31 32	A9	F5/16	F7/24	
F7/21	F5/17	A10	33 34	A11	F5/12	F7/23	
F7/2	F5/3	A12	35 36	A13	F5/5	J3/17	
J3/18	F5/7	A14	37 38	A15	F5/9	J3/19	
	F3/6	BA	39 40	$\overline{RST}$	F2/4		

Plotter Connector (CN5)

N14/15	X-Axis	1	2	
		3	4	
P14/15	Y-Axis	5	6	
Q13/4	Plotter Home	7	8	
N13/6	Pen Down	9	10	

Chang Connector (Amphenol 14pin)

(1)	X-Axis	1	8	X-Return (2)
(3)	NC	2	9	GND (4)
		3	10	
(5)	Y-Axis	4	11	Y-Return (6)
		5	12	
(7)	Plotter Home	6	13	GND (8)
(9)	Pen	7	14	GND (10)

RGB Connector (CN4)

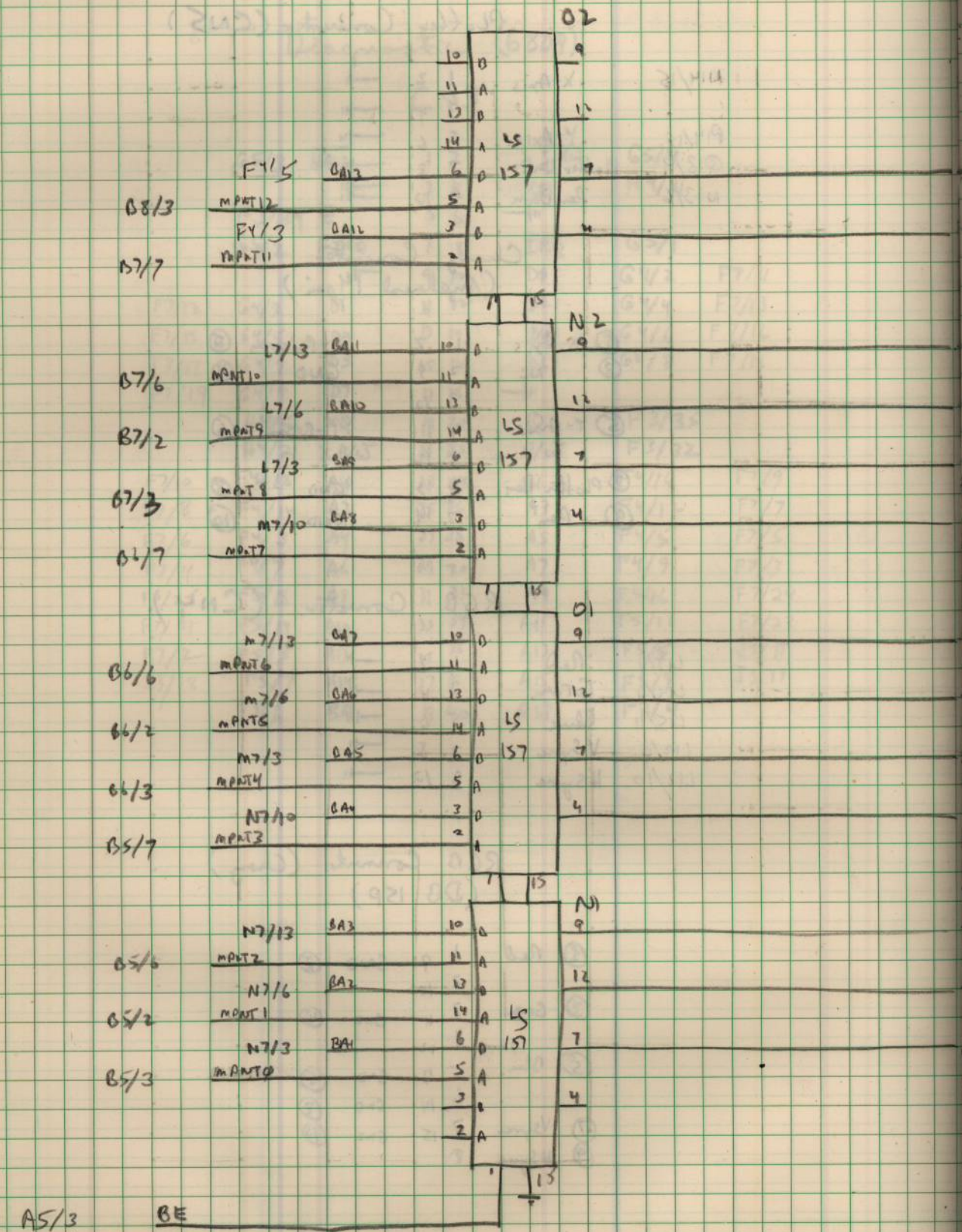
L13/11	Red	1	2	
L14/11	Green	3	4	
L15/11	Blue	5	6	
L12/9	V Sync	7	8	
L12/10	H Sync	9	10	

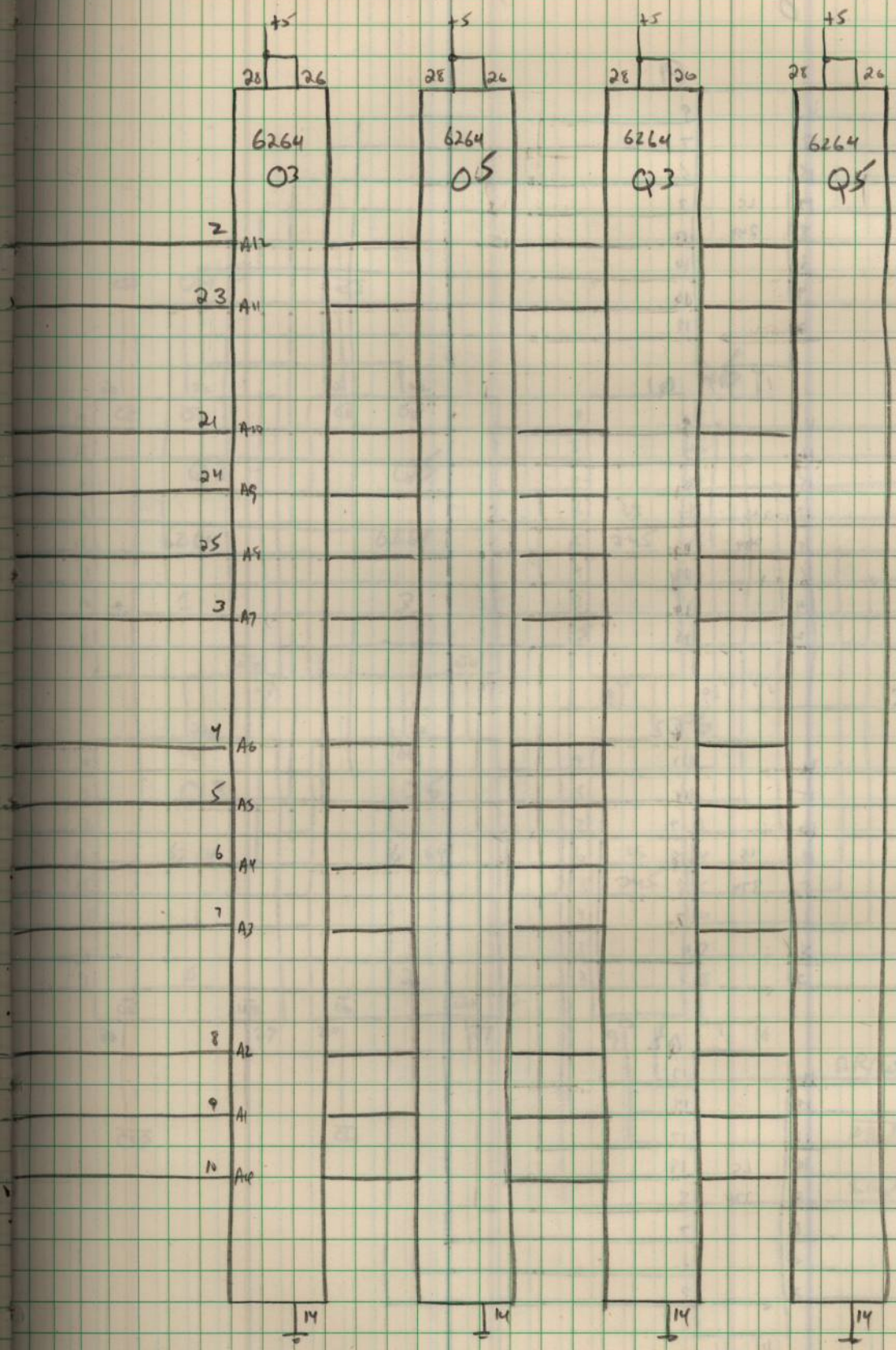
RGB Connector (Chang) (DB 15P)

(1)	Red	1	9	GND	(2)
		2	10		
(3)	Green	3	11	GND	(4)
		4	12		
(5)	Blue	5	13	GND	(6)
		6	14	GND	(8)
(7)	Vsync	7	15	GND	(10)
(9)	H Sync	8			

24 Oct 84
ARD

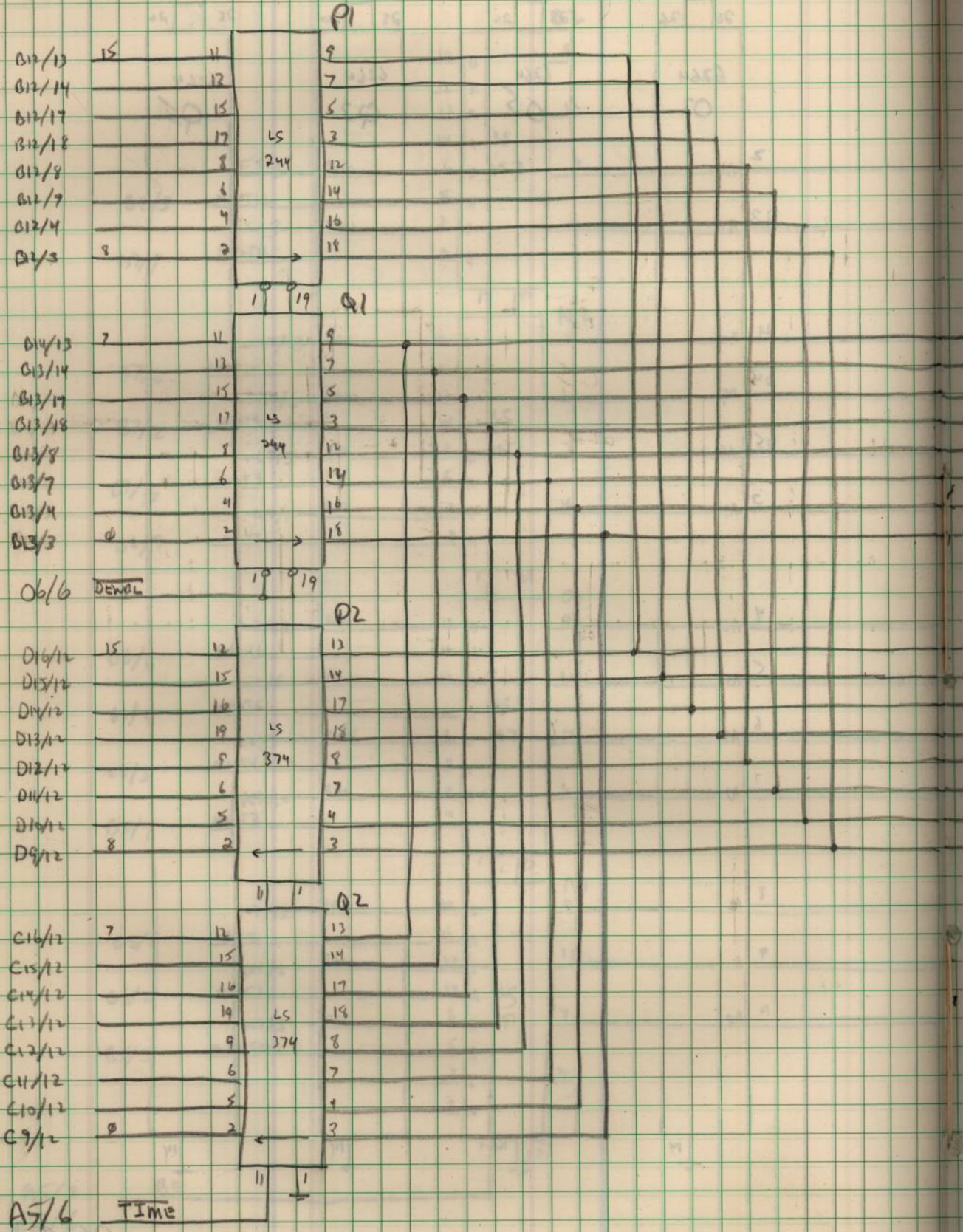
RAM Memory Addressing

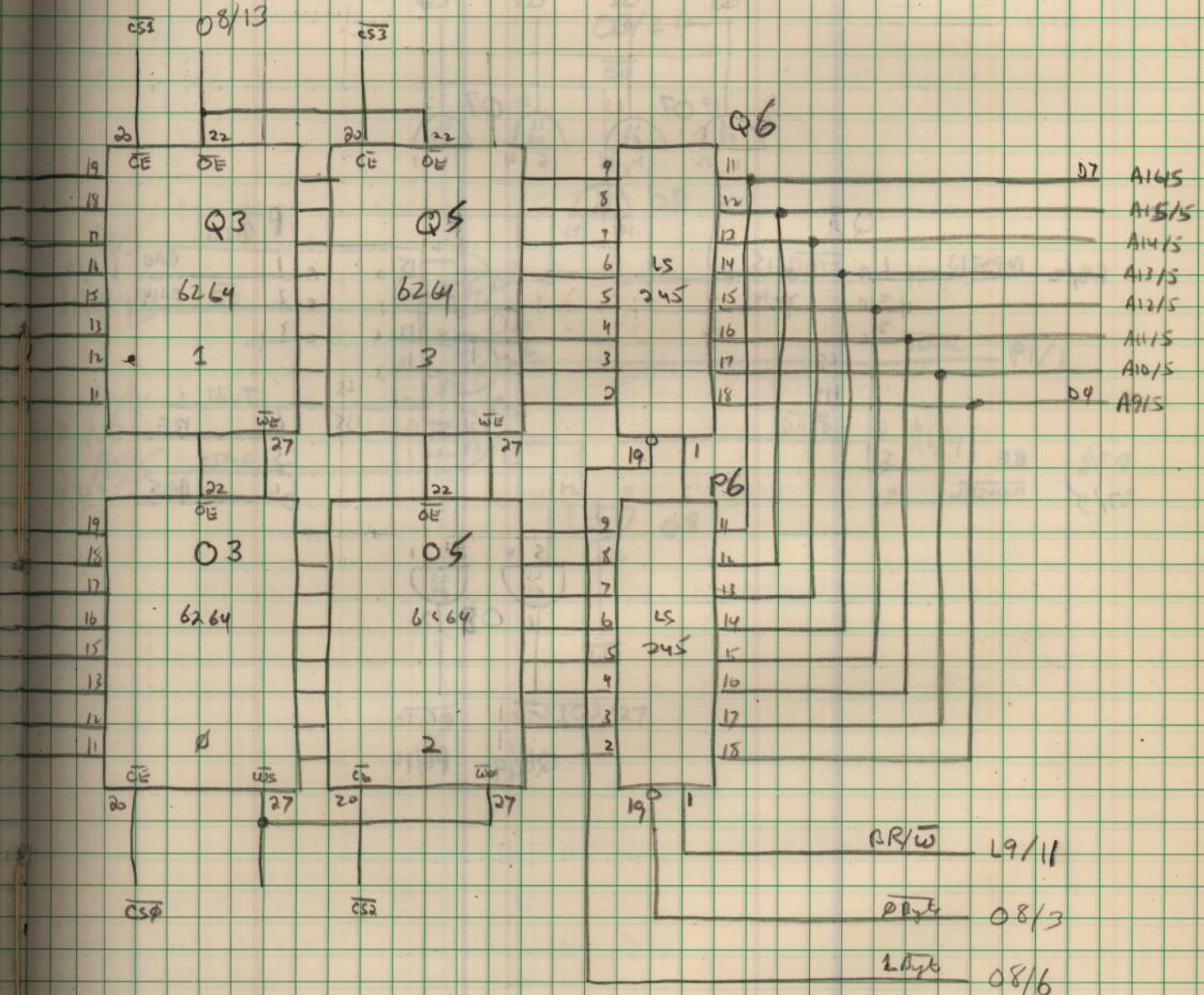



+5
28
26
6264
Q3
Q5
2
A12
23
A11
21
A10
24
A9
25
A8
3
A7
4
A6
5
A5
6
A4
7
A3
8
A2
9
A1
10
A0
14

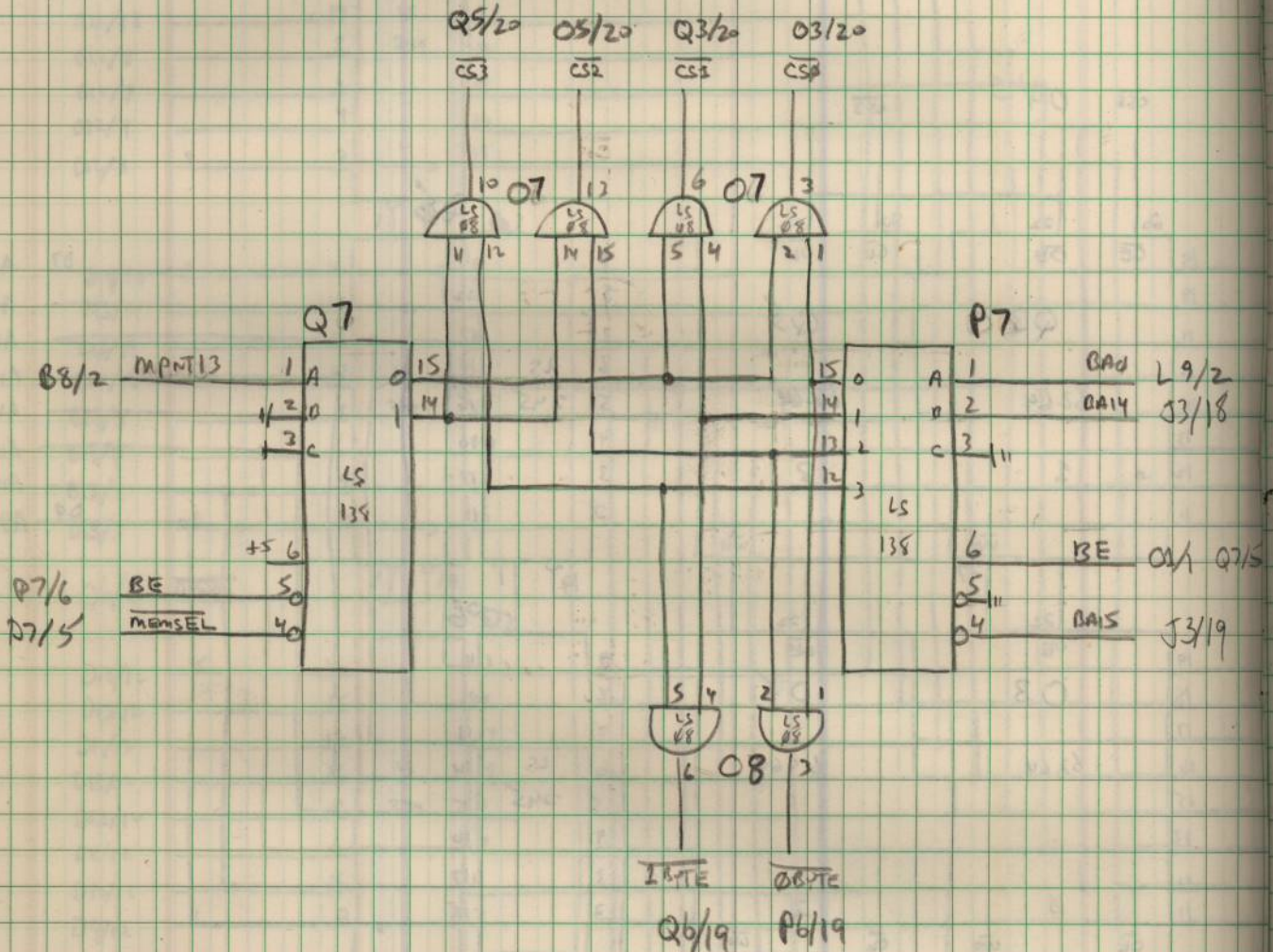
26 Oct 84
BBD

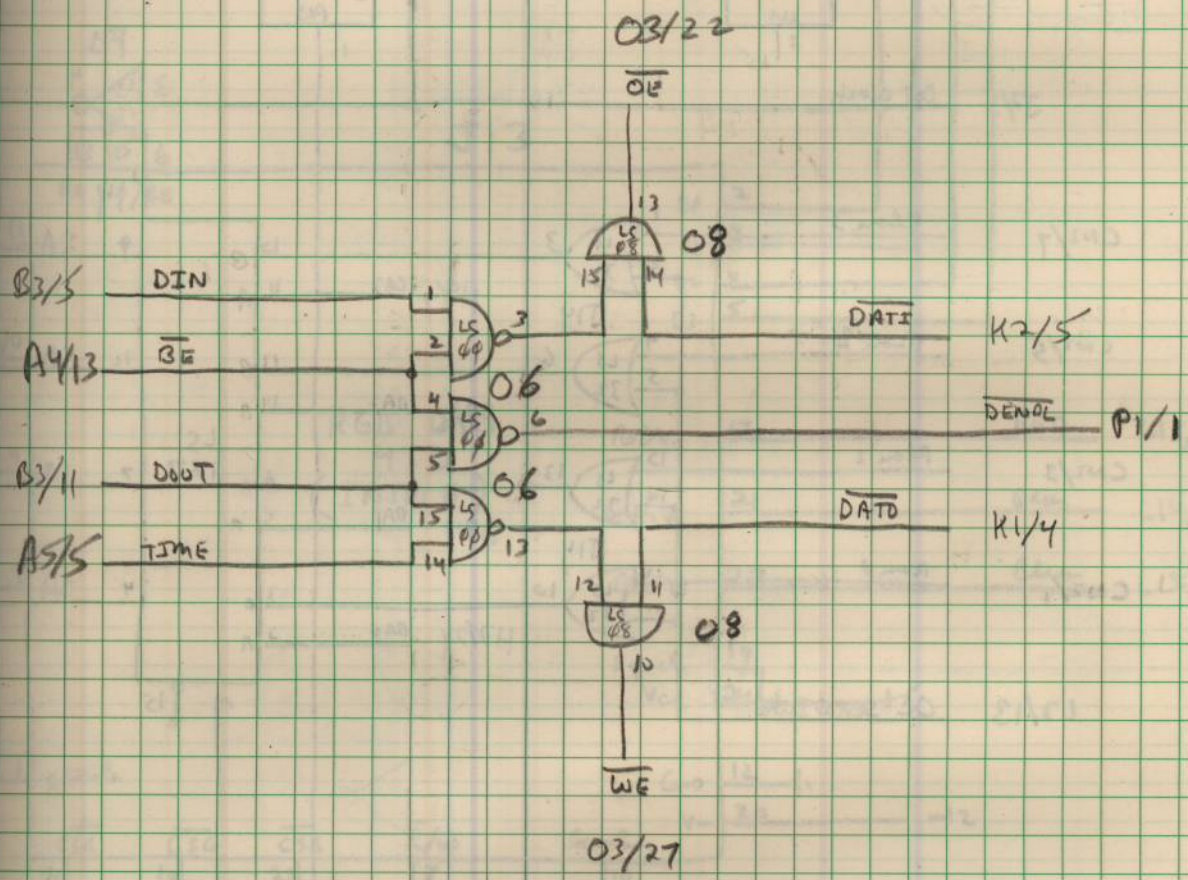
Dual Ported Memory 32K x 8 / 16K x 16




CS1
08/13
CS3
Q3
6264
1
Q5
6264
3
Q6
LS
245
P6
O3
6264
0
O5
6264
2
CS0
CS2
D7
A14/S
A13/S
A12/S
A11/S
A10/S
D0
A9/S
R/W
L9/11
0 Byte
08/3
1 Byte
08/6

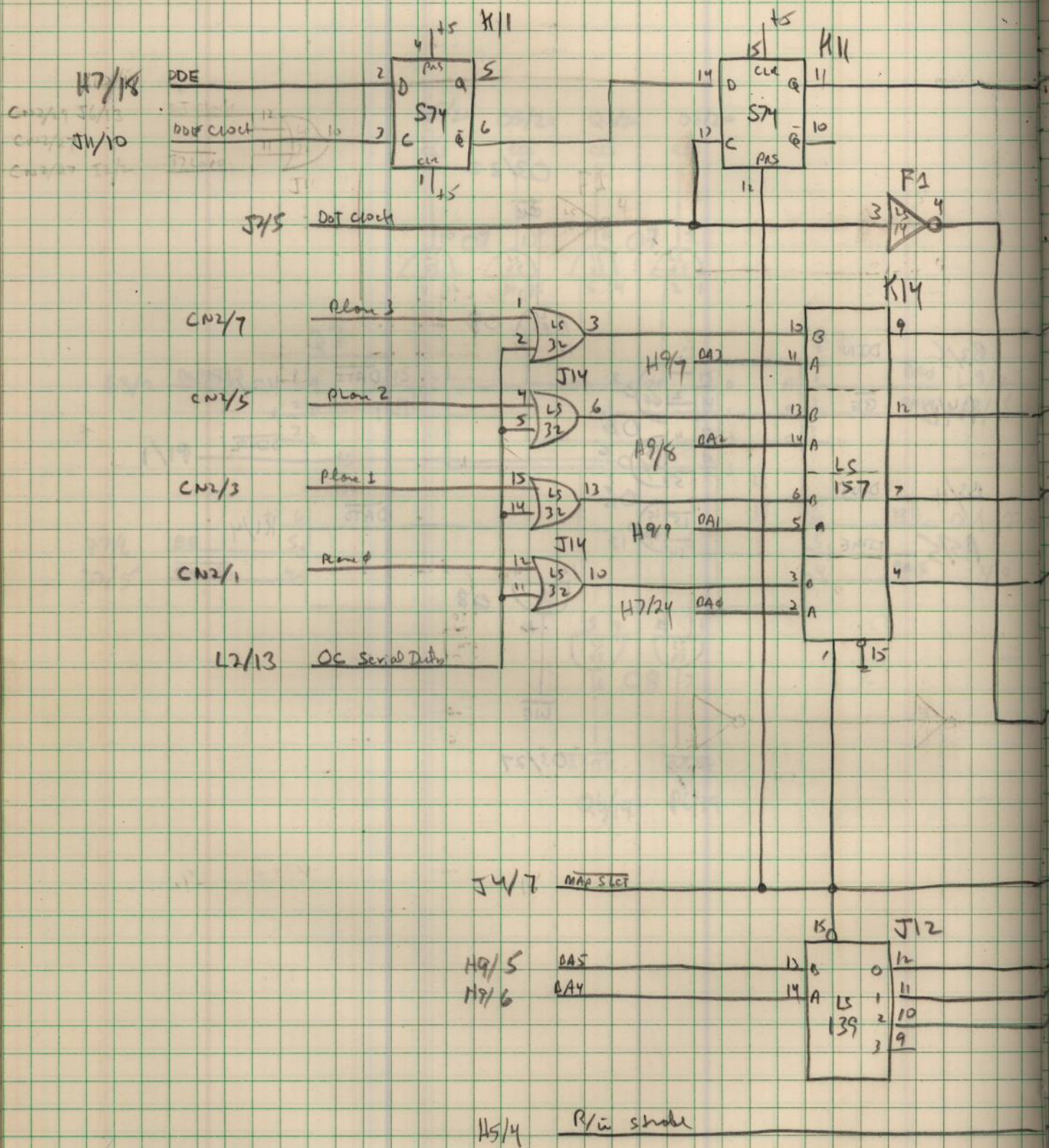
Dual Ported RAM Decoders

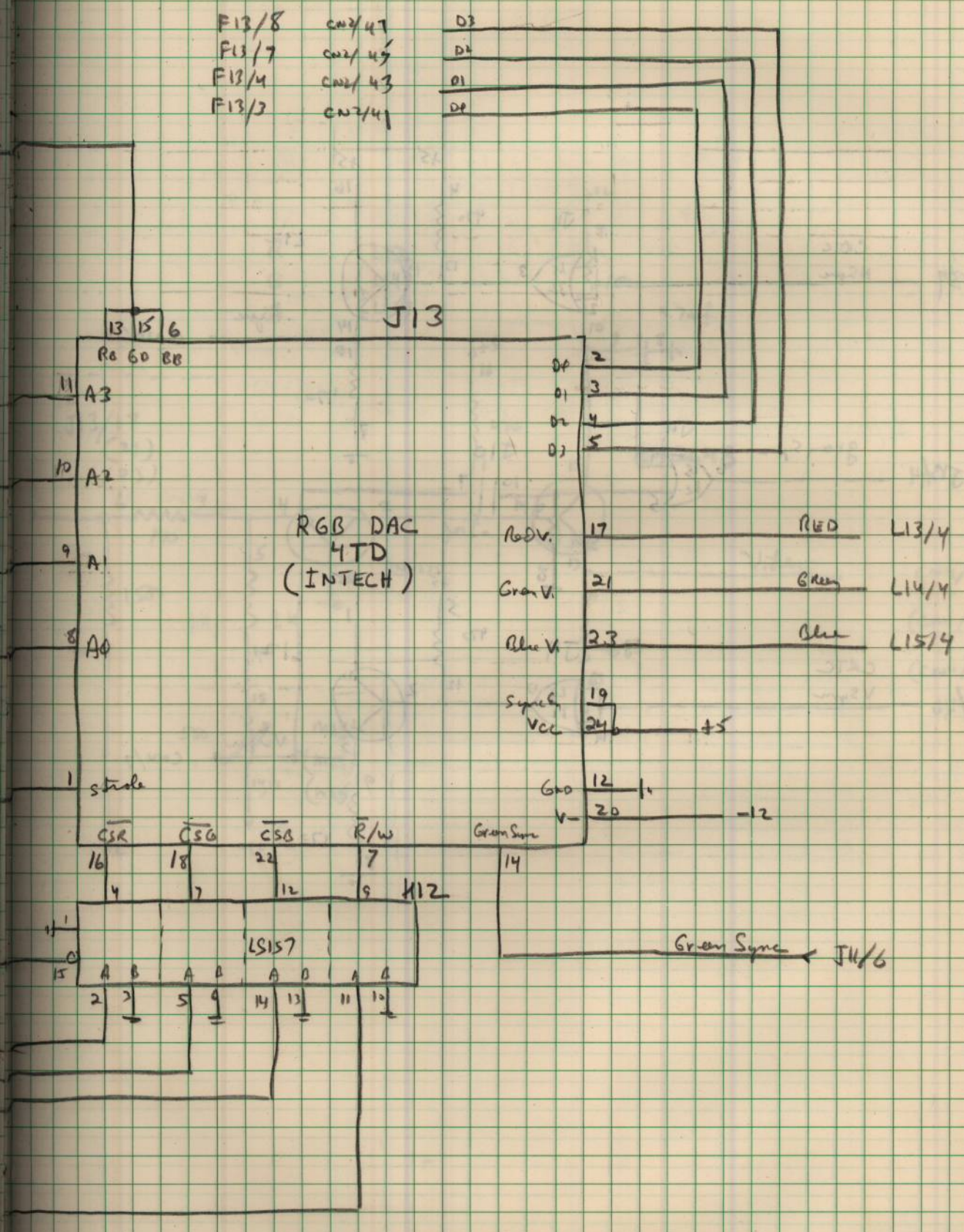



03/22
OE
13
08
15
14
B3/5
DIN
A4/13
BE
06
B3/11
DOUT
06
A5/5
TIME
DATI
K2/5
DENOL
P1/1
DATO
K1/4
12
11
08
10
WE
03/27

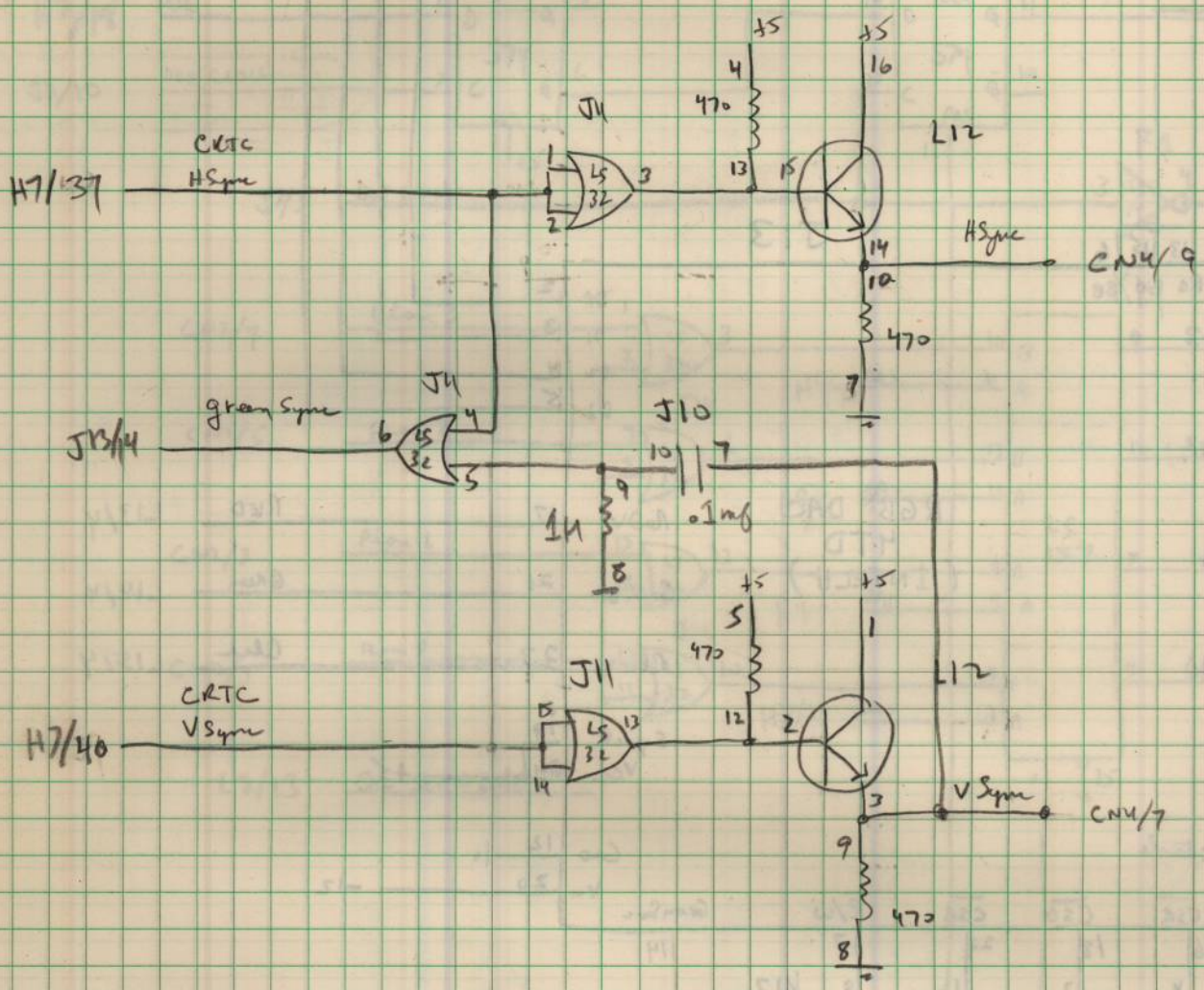
26 Oct 84
ABD

RGB DAC Logic

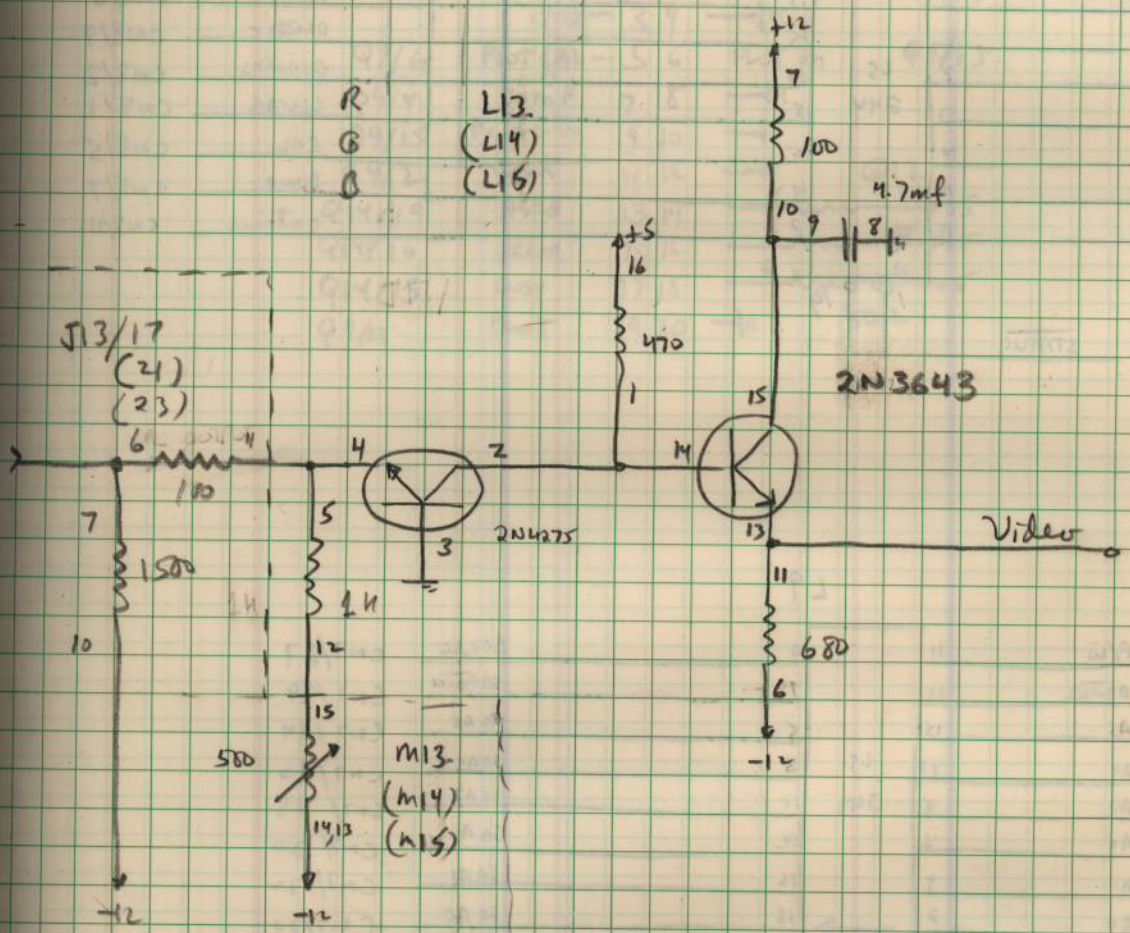



F13/8 CN2/47 D3
F13/7 CN2/45 D2
F13/4 CN2/43 D1
F13/3 CN2/41 D0
J13
13 15 6
R6 G0 B8
11 A3
10 A2
9 A1
8 A0
1 strobe
RGB DAC
4TD
(INTECH)
D0 2
D1 3
D2 4
D3 5
RedV. 17
GreenV. 21
BlueV. 23
Sync 19
Vcc 24 +5
Gnd 12
V- 20 -12
RED L13/4
Green L14/4
Blue L15/4
CSR CSG CSB R/W Green Sync
16 18 22 7 14
4 7 12 9 J12
LS157
15
A B A B A B A B
2 3 5 6 14 13 11 10
Green Sync ← J14/6
26 Oct 84
ROD

Sync Drivers

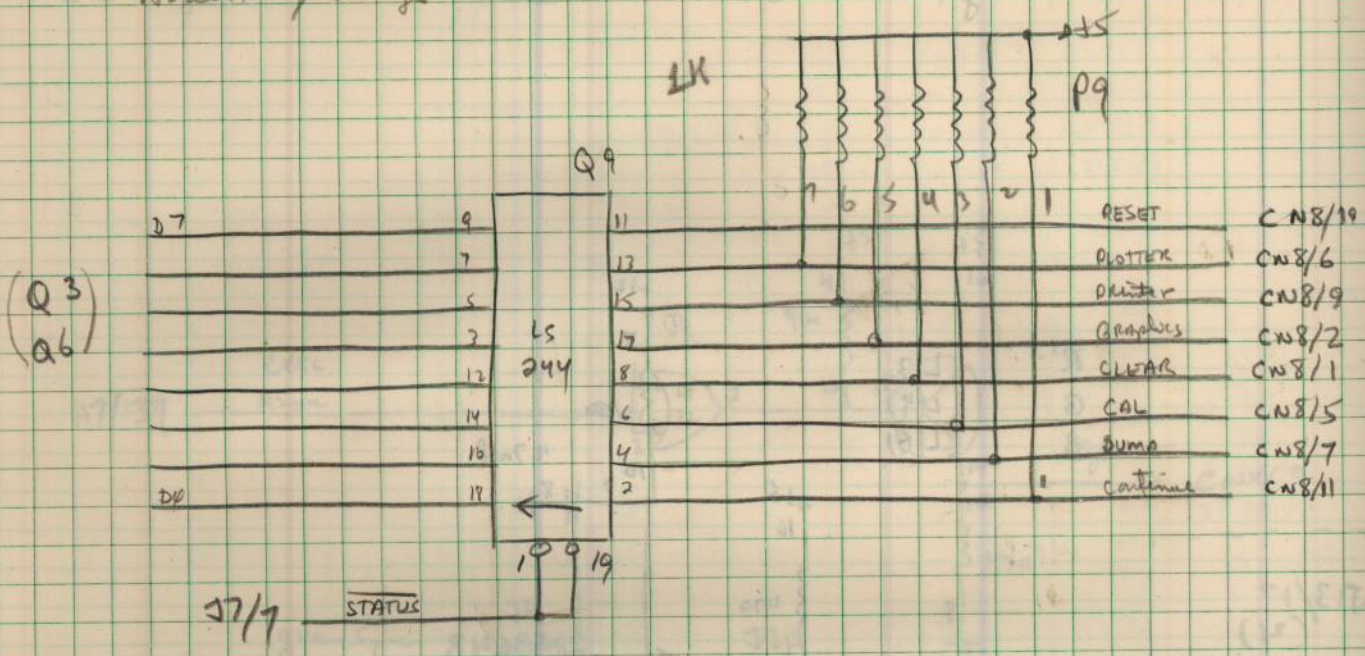


One of Three High Level Video Drivers

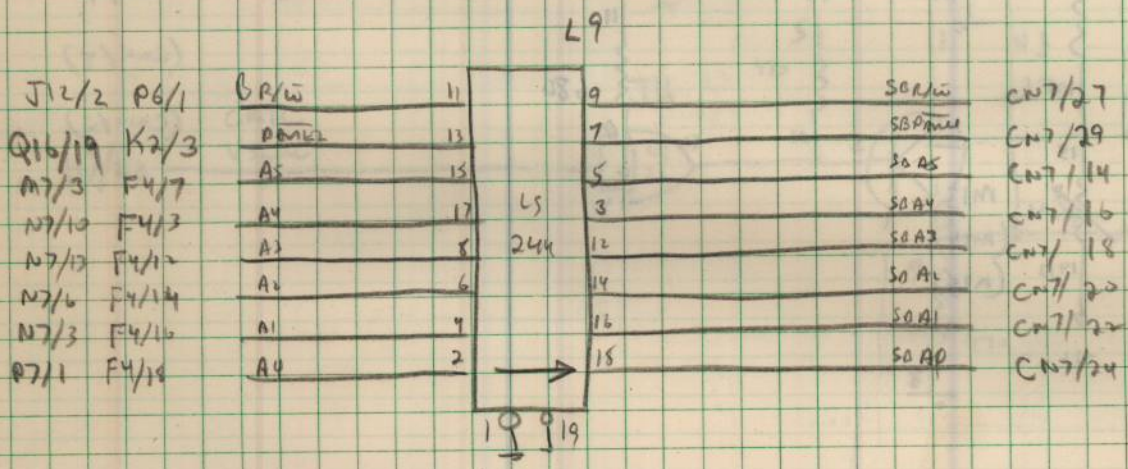


27 Oct 84
ARD

Additions / changes



(Q3 Q6)



J12/2	P6/1
Q16/19	K2/3
A7/3	F4/7
N7/10	F4/3
N7/13	F4/2
N7/6	F4/14
N7/3	F4/16
P7/1	F4/19

PANEL STATUS Connector

CN8

Q9/8	Vid CLR -	1	2	- Vide on	Q9/17
	(Vid)	3	4	(Vid)	
Q9/6	PLOT CAL -	5	6	PLOT on	Q9/13
Q9/4	DUMP	7	8		
Q9/15	Printer on	9	10		
Q9/2	Cont	11	12		
Q14/19	Hold	13	14		
Q14/16	ERROR	15	16		
Q14/15	BUSY	17	18		
Q9/11	Reset	19	20		

Q12
4 → 13 → +5
5 → 12 → +5
6 → 11 → +5
150Ω

9 Nov 84
AHD

IC Index

F 1	LS14	33, 34, 52
F 2	Res	33, 34
F 3	68B09E	35
F 4	LS244	35
F 5	LS244	35
F 6	connector	35
F 7	2764	42
F 8	—	
F 9	2764	42
F 10	LS374	43
F 11	LS374	43
F 12	LS374	43
F 13	LS374	43
F 14	—	

G 1	279	33
G 2	279	33
G 3	—	
G 4	LS245	35
G 5	RES	35
G 6	—	
G 7	—	
G 8	—	
G 9	—	
G 10	LS374	44
G 11	LS374	44
G 12	LS374	45
G 13	LS374	45
G 14	—	

H1	LS74	33
#2	LS30	33
H3	6828	35
H4	LS157	35
H5	LS00	35
#6	LS32	35
H7	68B45	42
H8	—	
H9	2764	42
#10	153	43
H11	153	43
H12	—	
H13	—	
H14	—	

I 1	S124	33
I 2	LS164	33
I 3	—	
I 4	LS133	35
I 5	LS08	34, 35
I 6	LS38	35
I 7	—	
I 8	—	
I 9	—	
I 10	153	43
I 11	153	43
I 12	8T97	33, 43
I 13	8T97	33, 43
I 14	LS257	41

25 Oct 84
AD

J 1	crystal	33
J 2	LS193	33
J 3	TDP1854Z	36
J 4	LS138	39
J 5	LS74	39, 41
J 6	LS74	34
J 7	LS138	39
J 8	———	
J 9	———	
J 10	———	
J 11	LS32	42, 53
J 12	LS139	52
J 13	RGB DAC 4TD	52
J 14	LS32	52

H 1	LS138	39
H 2	LS32	34, 39
H 3	———	
H 4	LS138	40
H 5	LS32	40
H 6	———	
H 7	LS138	39
H 8	———	
H 9	———	
H 10	LS74	42
H 11	S74	52
H 12	LS157	52
H 13	———	
H 14	LS157	52

L 1	LS151	41
L 2	LS166	41
L 3	2764	41
L 4	——	
L 5	6116	40
L 6	LS244	40
L 7	LS157	40
L 8	LS157	40
L 9	LS244	54
L 10	——	
L 11	——	
L 12	RES	53
L 13	RES	53
L 14	RES	53
L 15	RES	53

M 1	LS151	41
M 2	LS166	41
M 3	——	
M 4	——	
M 5	——	
M 6	LS244	40
M 7	LS157	40
M 8	LS157	40
M 9	——	
M 10	——	
M 11	——	
M 12	——	
M 13	RES	53
M 14	RES	53
M 15	RES	53

28JA84
AKO

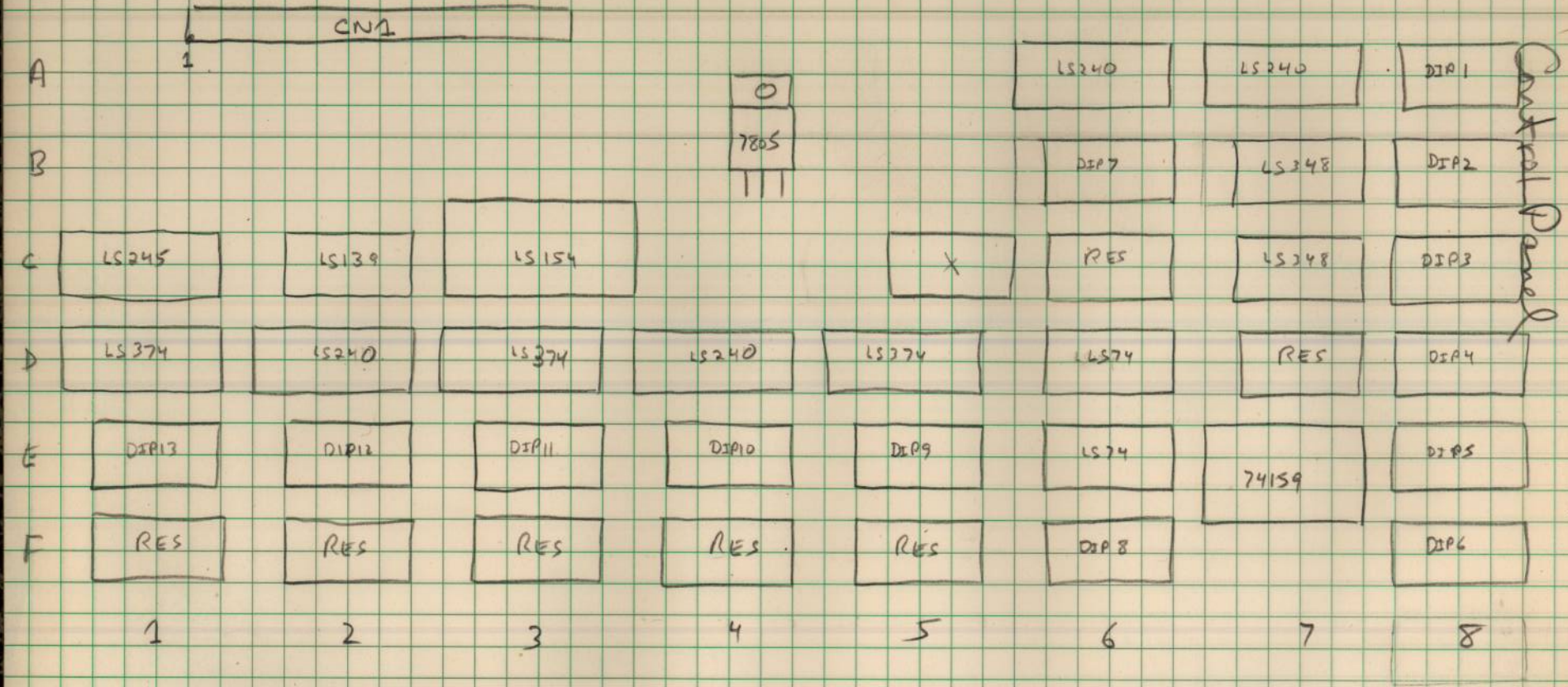
N1	LS157	49
N2	LS157	49
N3	2764	41
N4		
N5	6116	40
N6	LS157	41
N7	LS157	40
N8	LS157	40
N9		
N10		
N11		
N12		
N13	RES	44
N14	RES	44

01	LS157	49
02	LS157	49
03	6264	49, 50
04		
05	6264	49, 50
06	LS00	51
07	LS08	51
08	LS08	51
09		
010		
011		
012		
013	7541	44
014	1458	44, 45

P 1	LS244	50
P 2	LS374	50
P 3	—	
P 4	—	
P 5	—	
P 6	LS245	50
P 7	LS138	51
P 8	—	
P 9	Res	54
P 10	—	
P 11	—	
P 12	—	
P 13	7541	45
P 14	RES	45

Q 1	LS244	50
Q 2	LS 374	50
Q 3	6264	49, 50
Q 4	—	
Q 5	6264	49, 50
Q 6	LS245	50
Q 7	LS 138	51
Q 8	—	
Q 9	LS244	54
Q 10	—	
Q 11	—	
Q 12	RES	46, 54
Q 13	LS244	46
Q 14	LS374	46
Q 15	LS374	46
Q 16	LS245	46

28 04 84
ABO


CN1
1
7805
LS240
LS240
DIP1
DIP7
LS248
DIP2
LS245
LS139
LS154
X
RES
LS248
DIP3
LS374
LS240
LS374
LS240
LS374
LS74
RES
DIP4
DIP13
DIP12
DIP11
DIP10
DIP9
LS74
74159
DIP5
RES
RES
RES
RES
RES
DIP8
DIP6
A
B
C
D
E
F
1
2
3
4
5
6
7
8

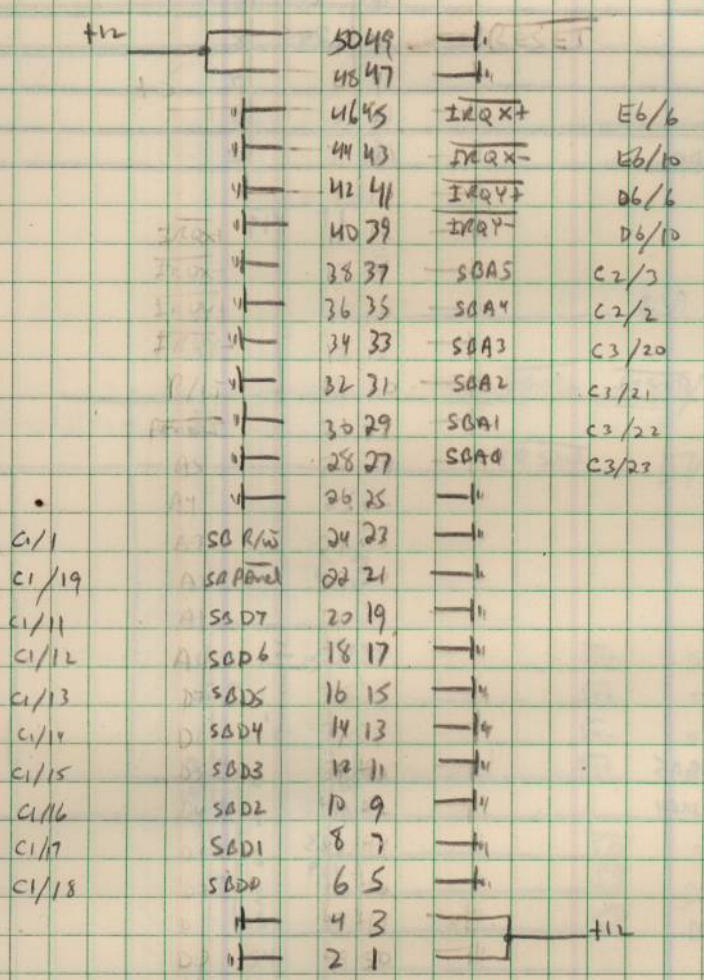
DIP Connectors to Panel

DIP 1	-	Thumbwheel 1
DIP 2	-	Thumbwheel 2
DIP 3	-	Rotary 1
DIP 4	-	Rotary 2
DIP 5	-	Rotary 3
DIP 6	-	Rotary 4
DIP 7	-	Cursor switches
DIP 8	-	Track Ball
DIP 9	-	Toggle LED's
DIP 10	-	Toggle switches
DIP 11	-	Push LED's
DIP 12	-	Push Buttons
DIP 13	-	LED's

15 Nov 84
ABD

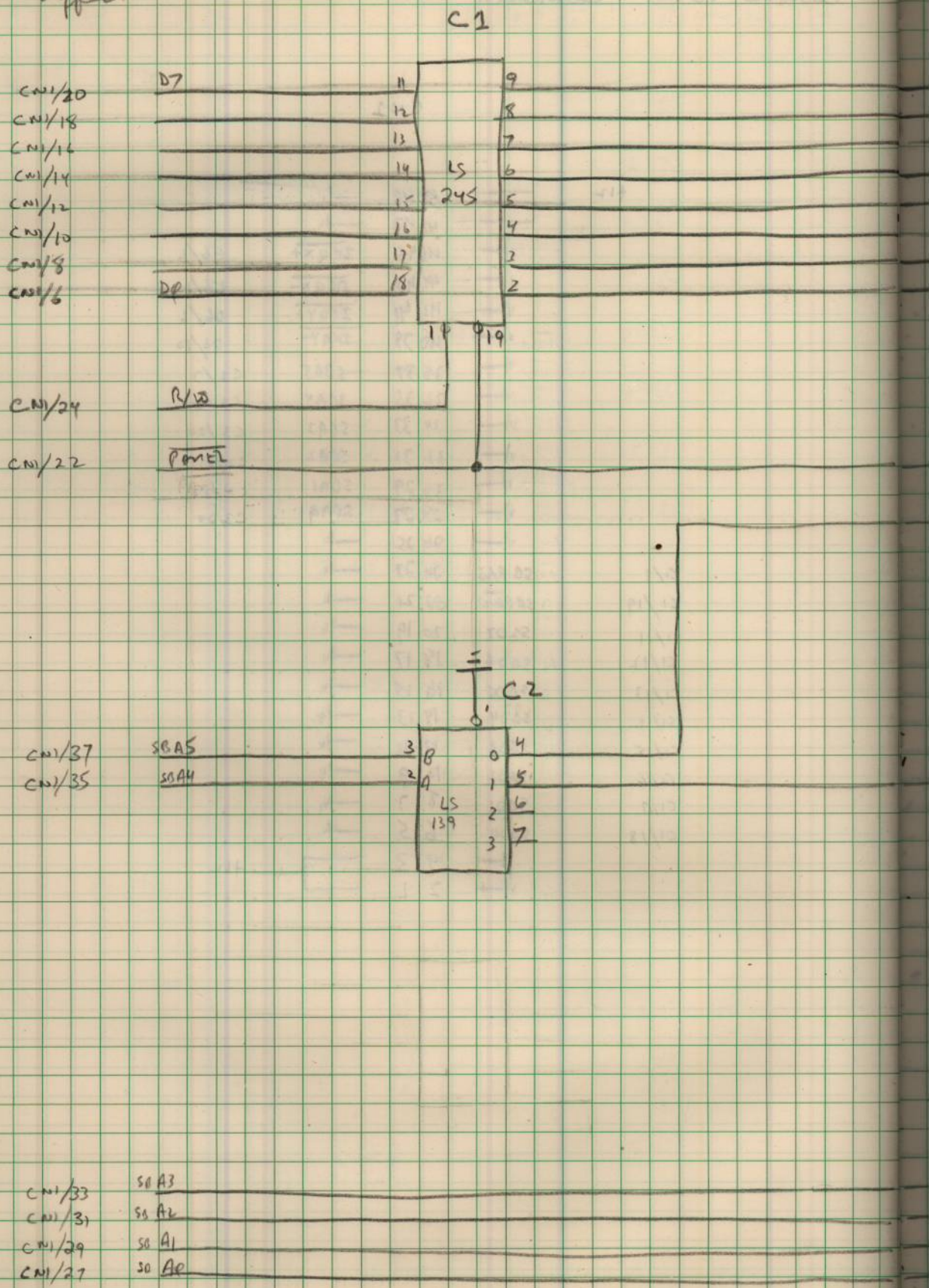
Control Panel Connector

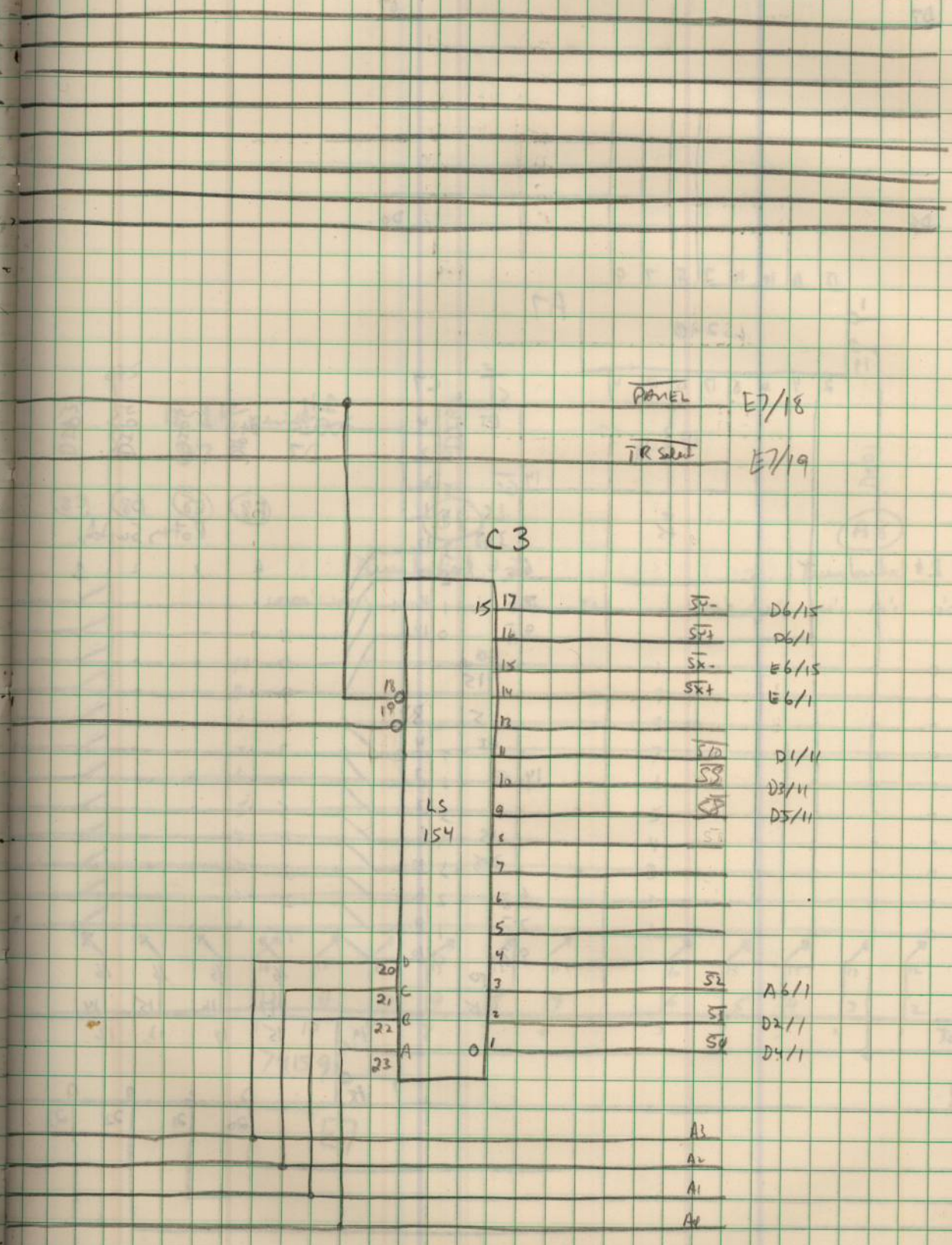
CN1



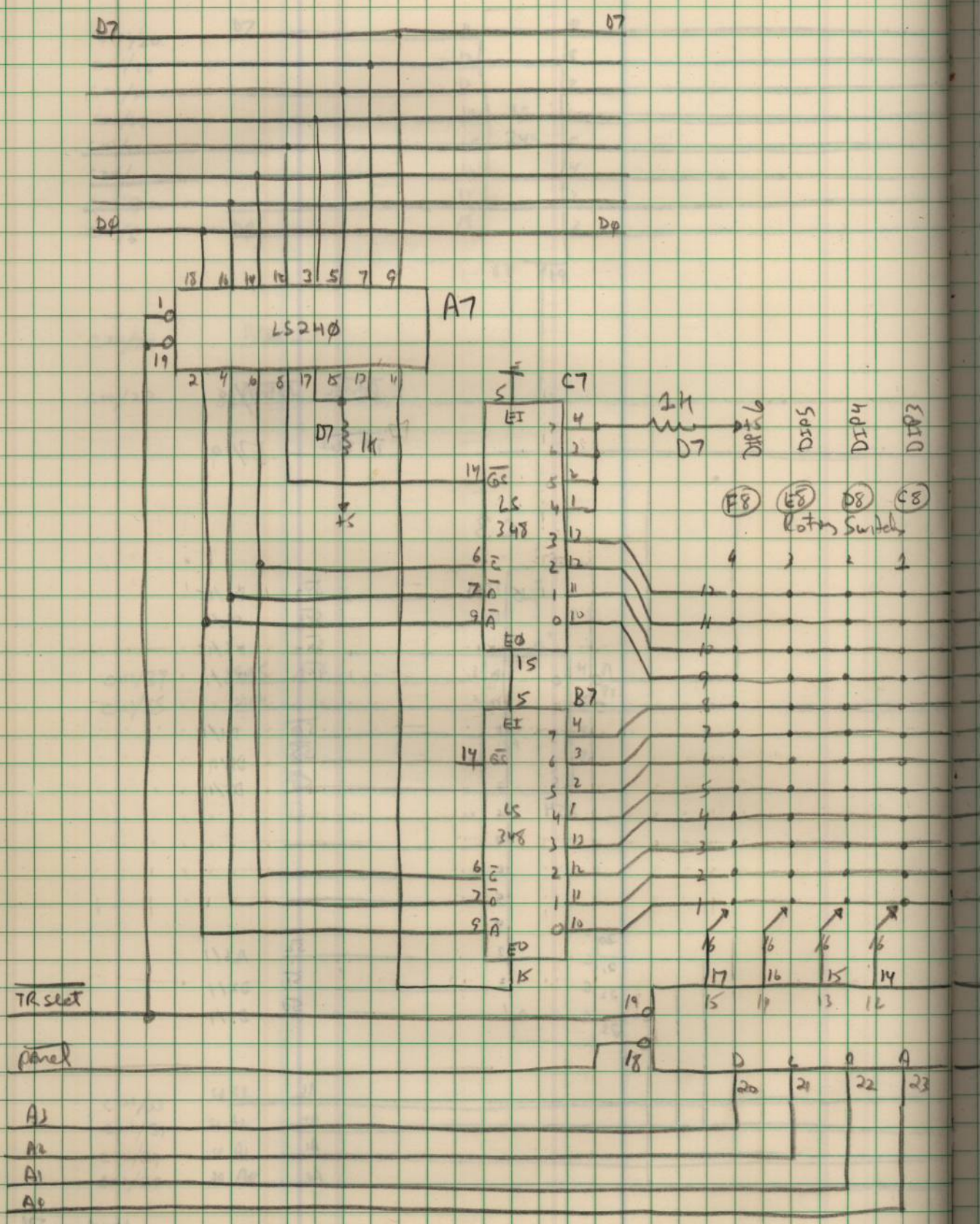
4 Nov 84
DBO

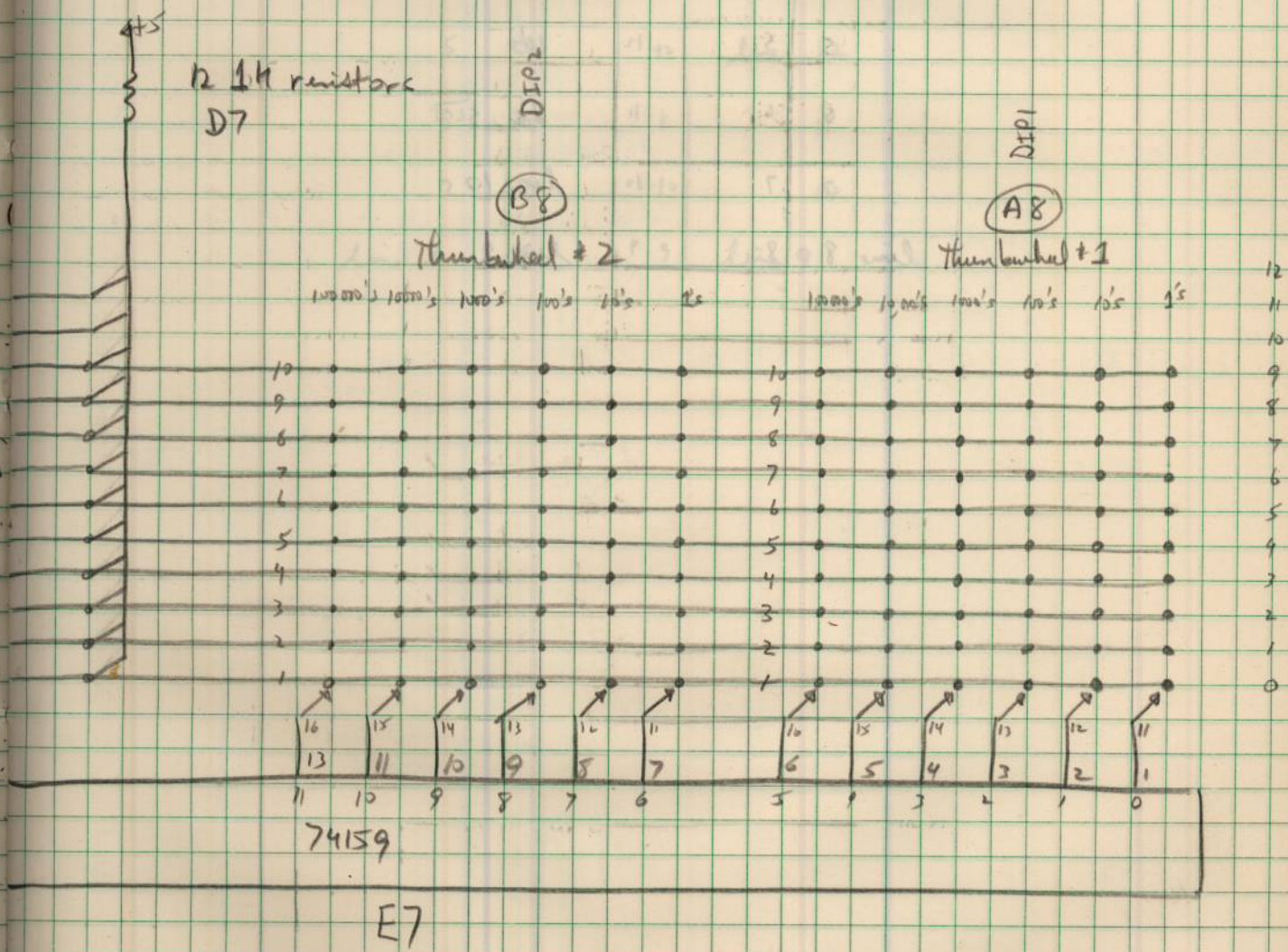
Data Buffer


C1
CN1/20
CN1/18
CN1/16
CN1/14
CN1/12
CN1/10
CN1/8
CN1/6
D7
D0
11
12
13
14
15
16
17
18
LS
245
9
8
7
6
5
4
3
2
1
19
CN1/24
R/W
CN1/22
PMEL
C2
CN1/37
CN1/35
SBA5
SBA4
3
2
B
A
LS
139
0
1
2
3
4
5
6
7
CN1/33
CN1/31
CN1/29
CN1/27
SB A3
SB A2
SB A1
SB A0


PANEL
E7/18
IR select
E7/19
C3
LS
154
15
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
18
19
20
21
22
23
D
C
B
A
$\overline{SY}$-
D6/15
$\overline{SY}$+
D6/1
$\overline{SX}$-
E6/15
$\overline{SX}$+
E6/1
D1/11
D3/11
D5/11
$\overline{S2}$
A6/1
$\overline{S1}$
D2/1
$\overline{S0}$
D4/1
A3
A2
A1
A0
15 Nov 84
ASO

Thumbwheel / Rotary Switch Read Select

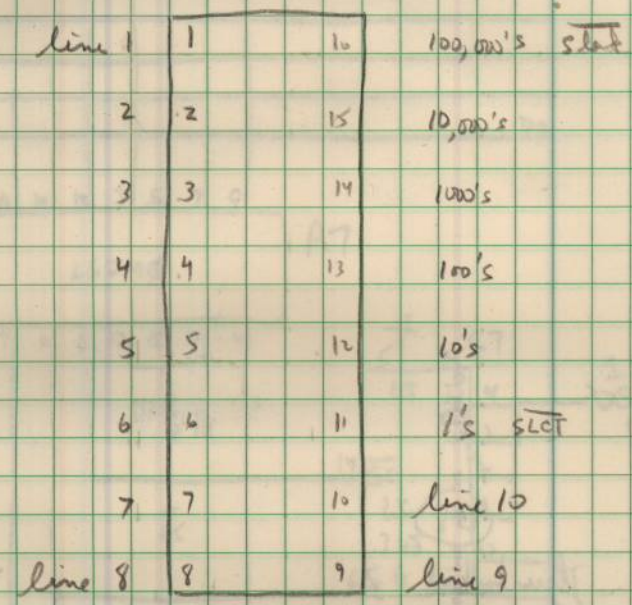



+5
2 1K resistors
D7
DIP2
DIP1
B8
A8
Thumbwheel #2
Thumbwheel #1
10's
1's
74159
E7

5 NOV 84

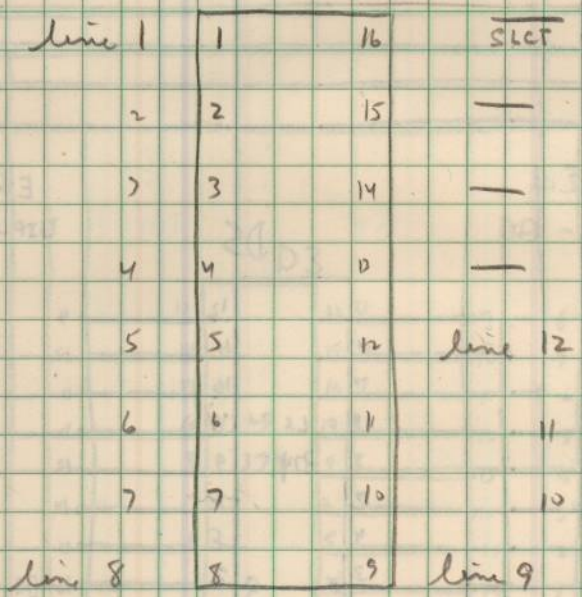
DIP 1 & DIP 2

Thumbwheels DIP connectors



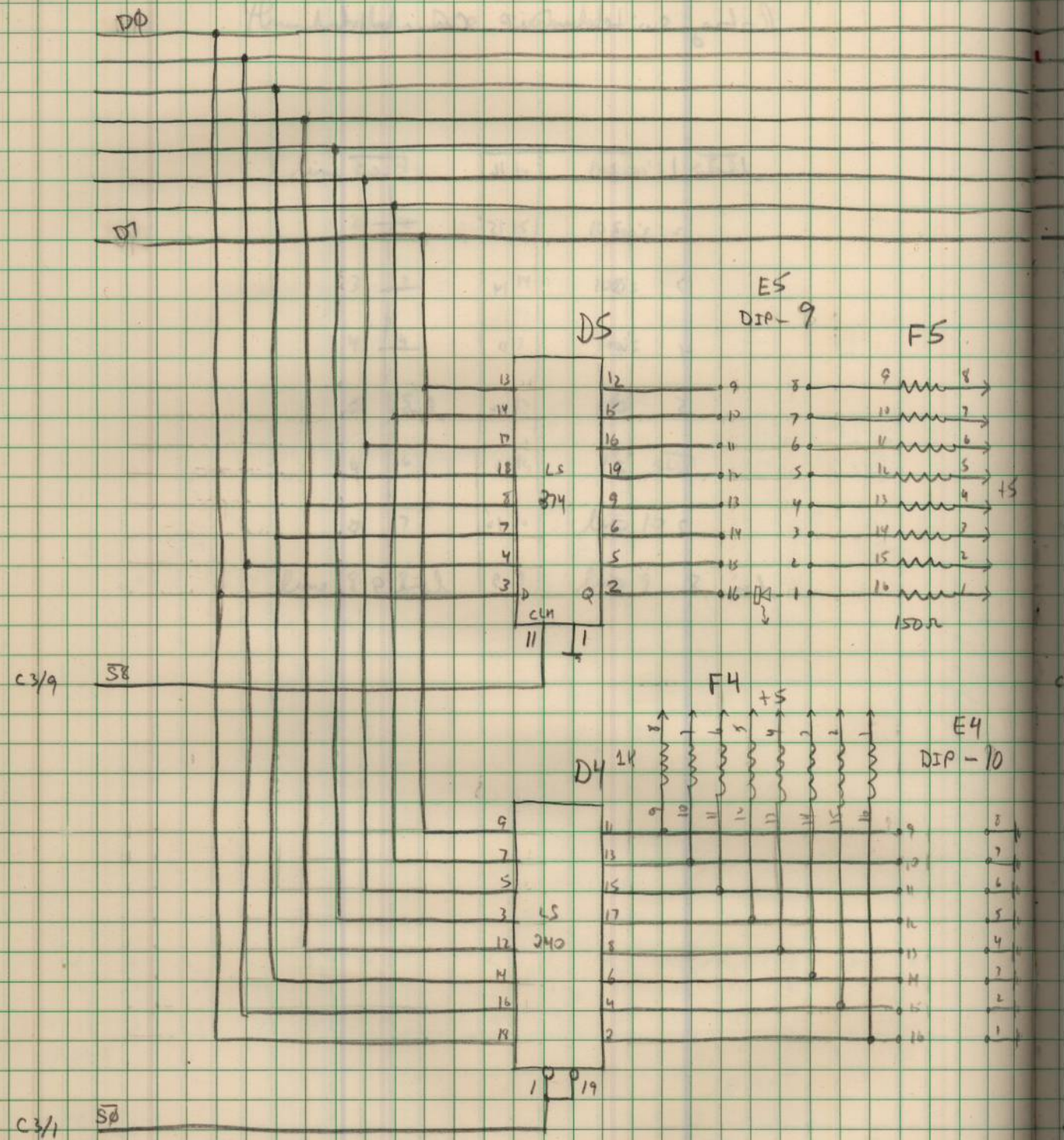
DIP 3 – DIP 6

Rotary switch DIP connectors

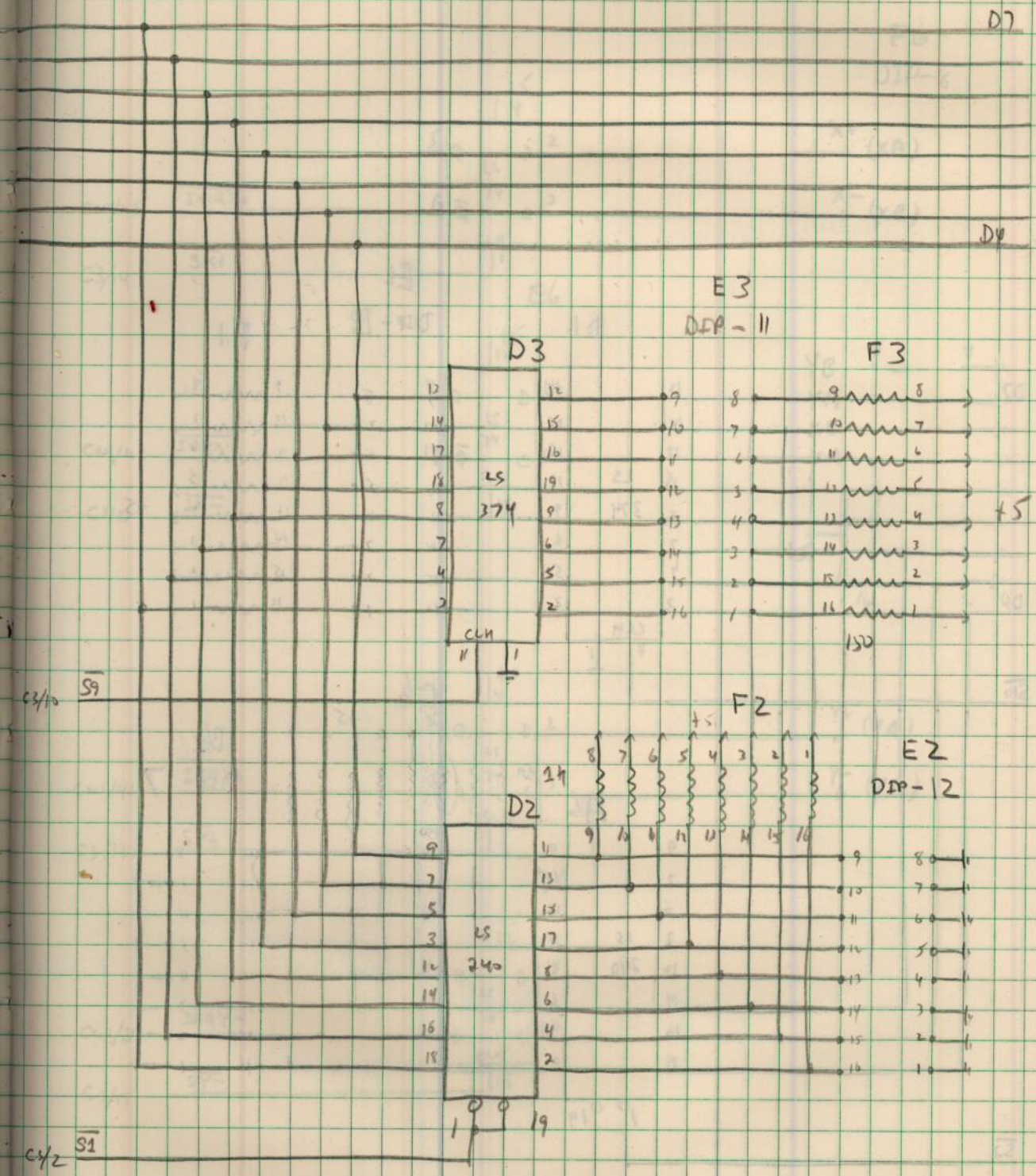


15 Nov 84
ABC

Toggles

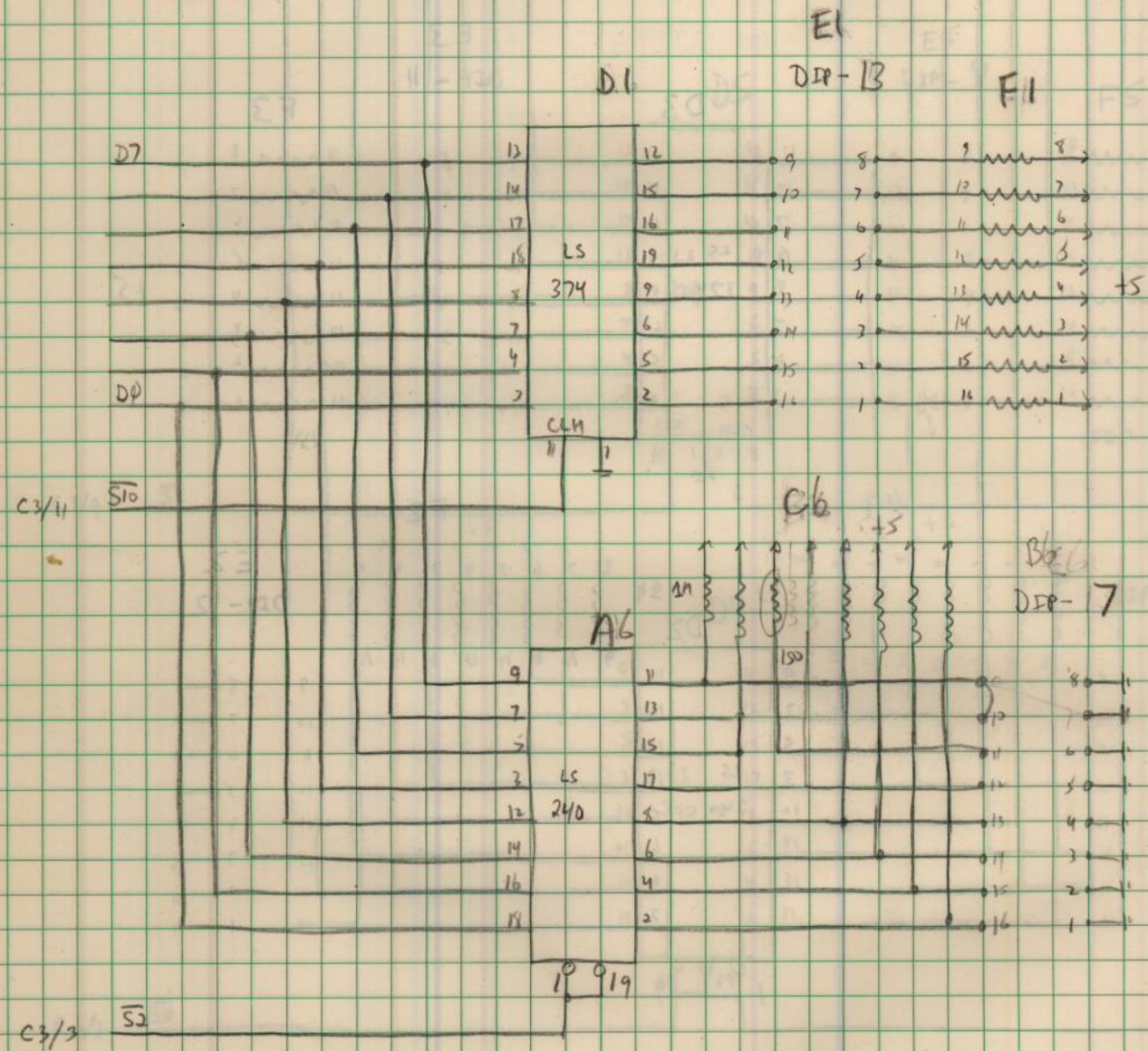

D0
D7
E5
DIP-9
D5
F5
LS
374
CLK
+5
150Ω
C3/9
S8
F4
+5
E4
DIP-10
D4
1K
LS
240
C3/1
S0

Push Buttons

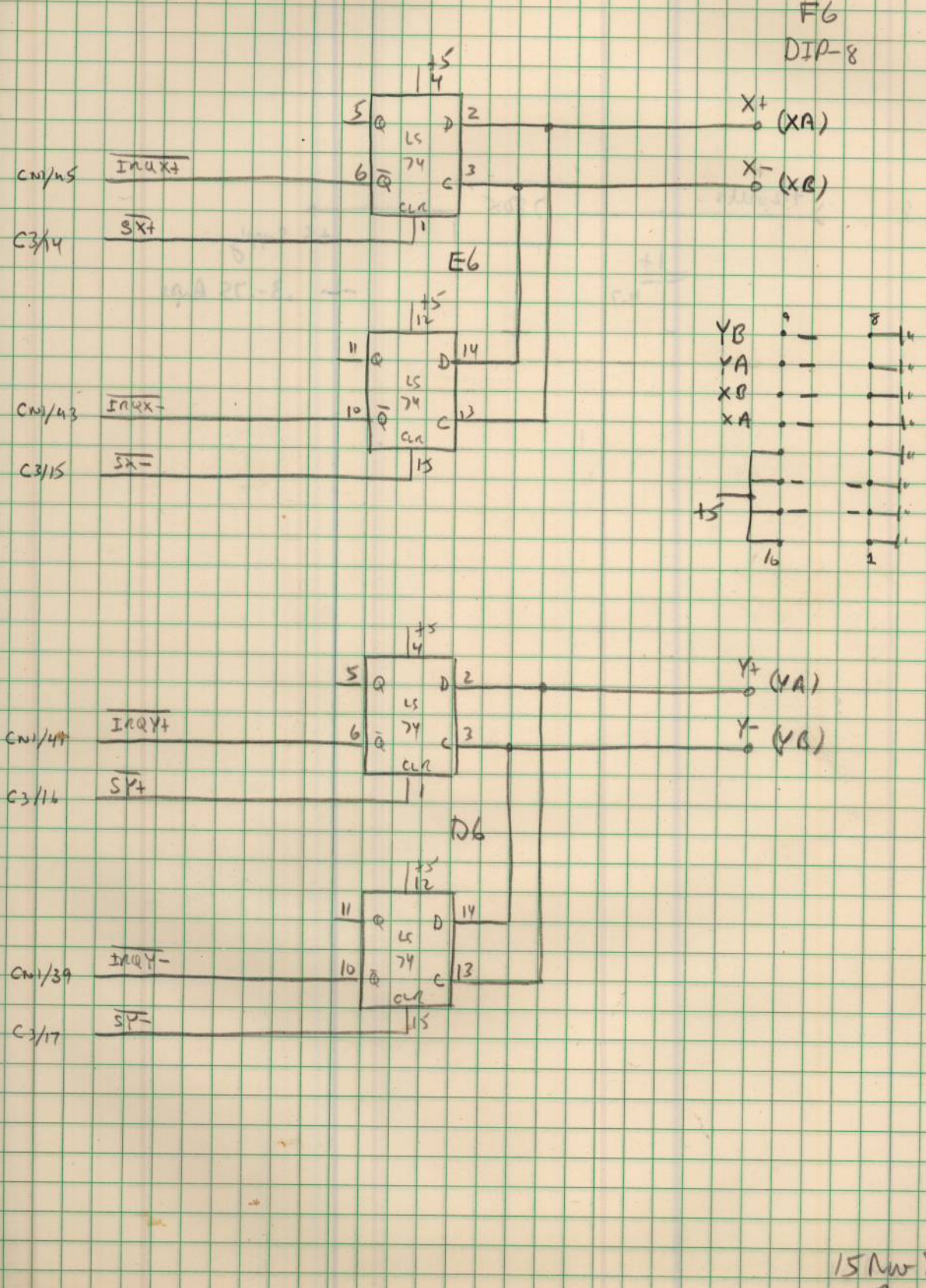


15 Nov 84
DMD

LED'S / CURSOR

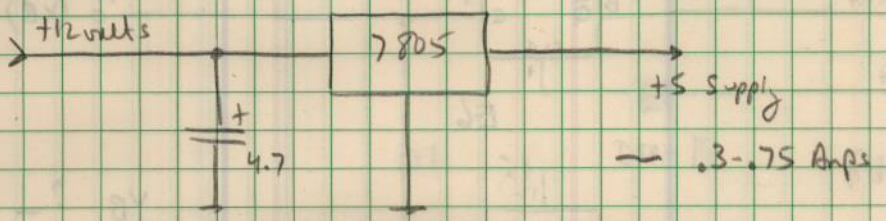

E1
DIP-13
D1
F1
LS
374
CLK
+5
C6
B6
DIP-7
A6
LS
240
D7
D0
C3/11
C3/3
S1
S2

TRACKBALL



15 Nov 84
AHO

Regulator



IC Index

A 6	LS240	–	67
A 7	LS240	–	64
A 8	DIP-1	–	64, 65
B 6	DIP-7	–	67
B 7	LS348	–	64
B 8	DIP-2	–	64, 65
C 1	LS245	–	63
C 2	LS139	–	63
C 3	LS154	–	63
C 5	X	–	
C 6	RES	–	67
C 7	LS348	–	64
C 8	DIP-3	–	64, 65

D1	LS374	—	67
D2	LS240	—	66
D3	LS374	—	66
D4	LS 240	—	66
D5	LS 374	—	66
D6	LS 74	—	67
D7	RES	—	64
D8	DIP-4	—	64, 65

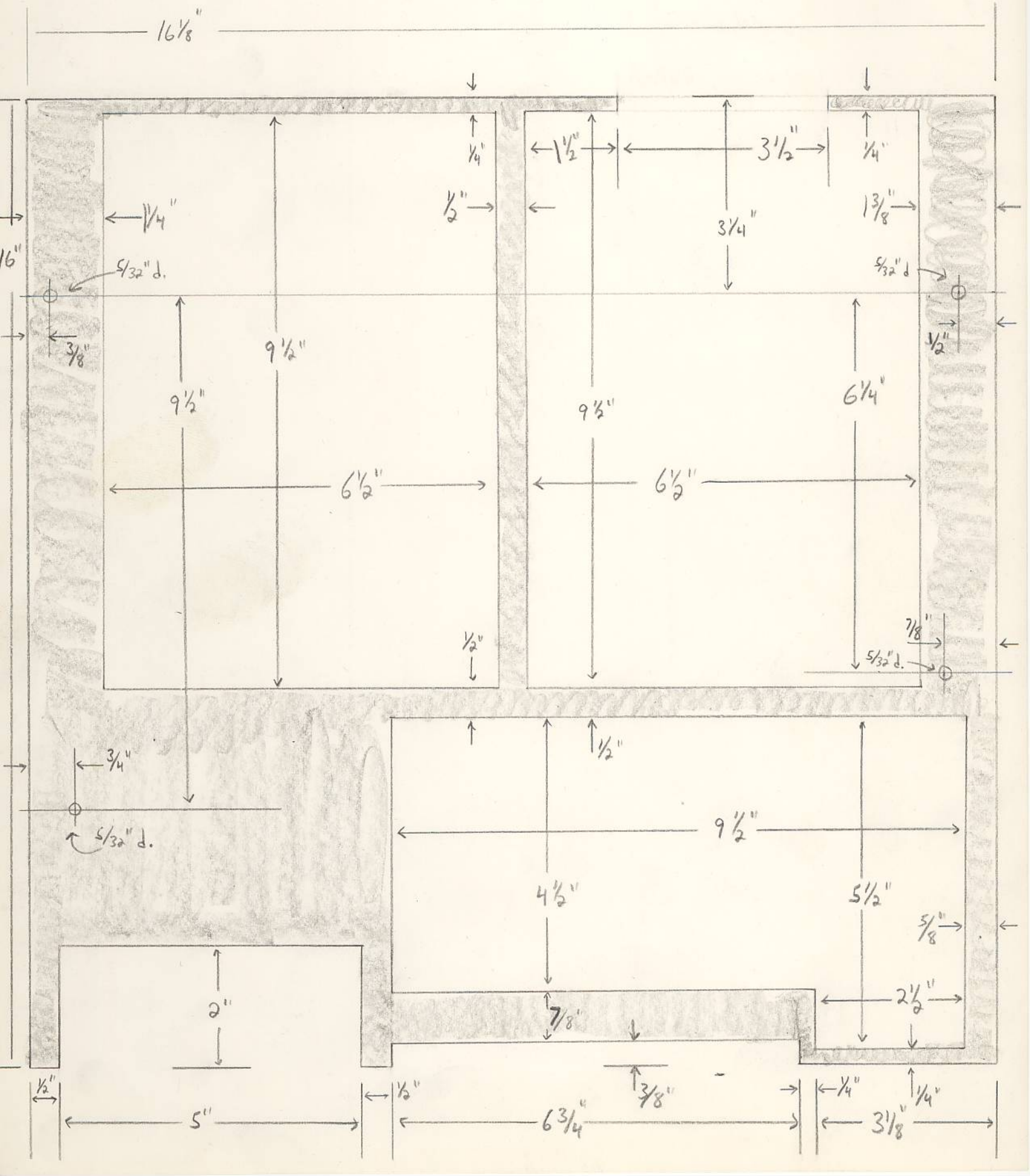
E1	DIP-13	—	67
E2	DIP-12	—	66
E3	DIP-11	—	66
E4	DIP-10	—	66
E5	DIP-9	—	66
E6	LS74	—	67
E7	74159	—	64
E8	DIP-5	—	64, 65

F1	RES	—	67
F2	RES	—	66
F3	RES	—	66
F4	RES	—	66
F5	RES	—	66
F6	DIP-8	—	67
F7	———	—	
F8	DIP-6	—	64, 65

15 Nov 84
ARO

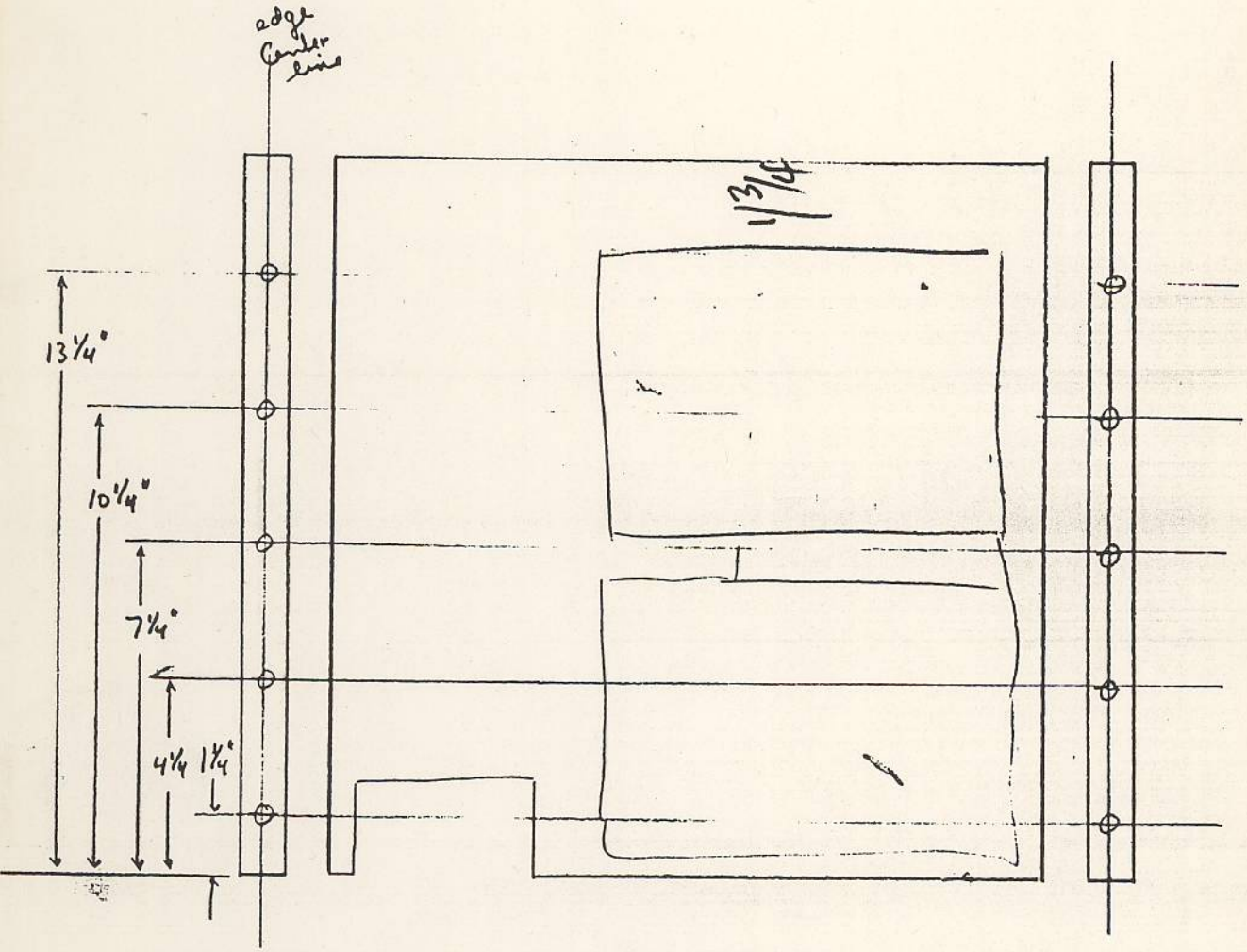
Aluminum Panel

2pcs 1/4" thick 16" x 16 1/8"



#1

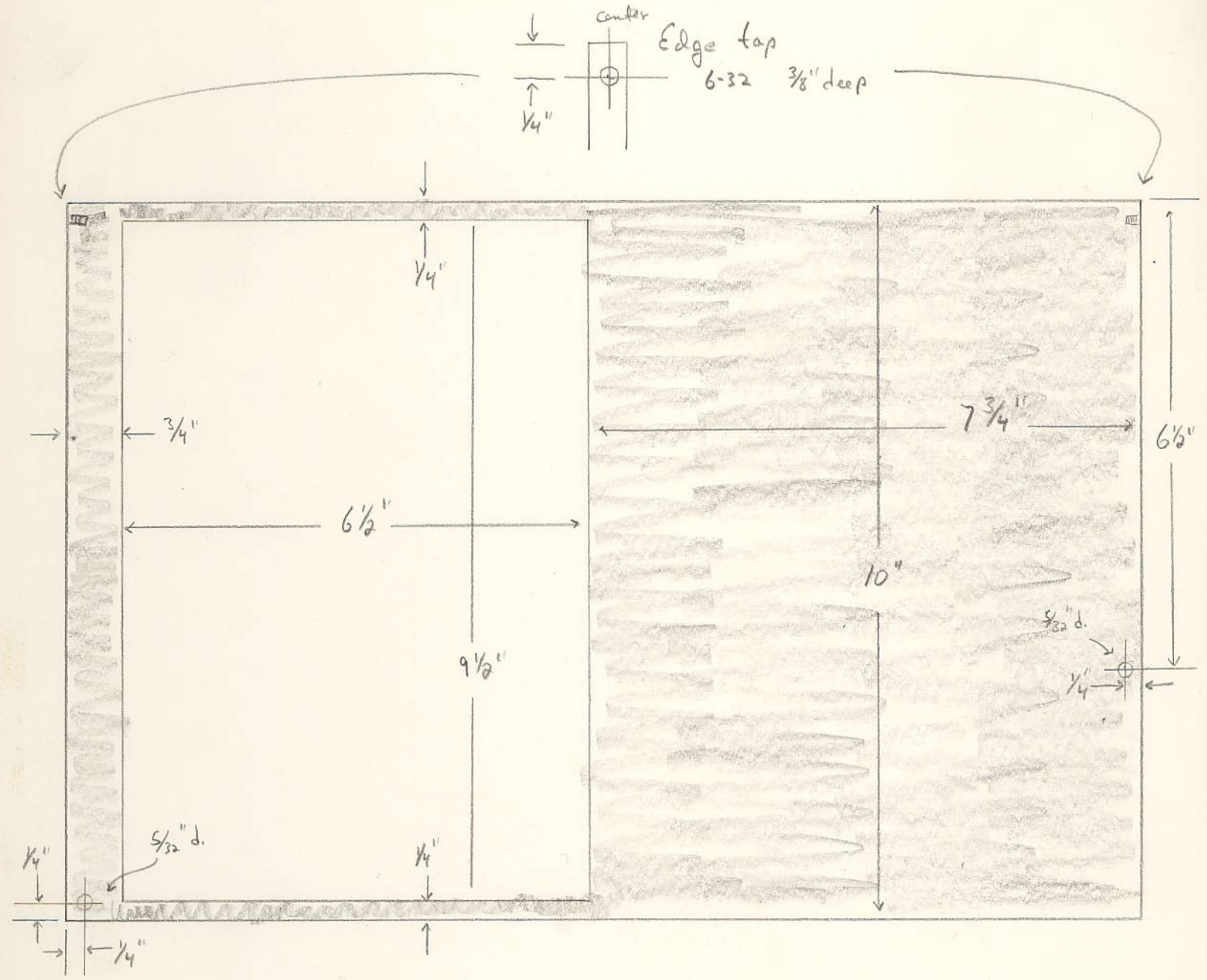
Edge Hole pattern – tapped 6-32
½" minimum depth



Aluminum Panel

2pcs

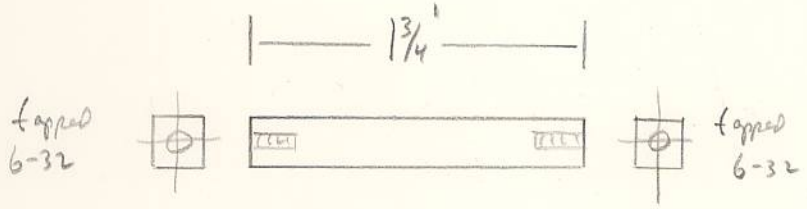
1/4" x 15" x 10"



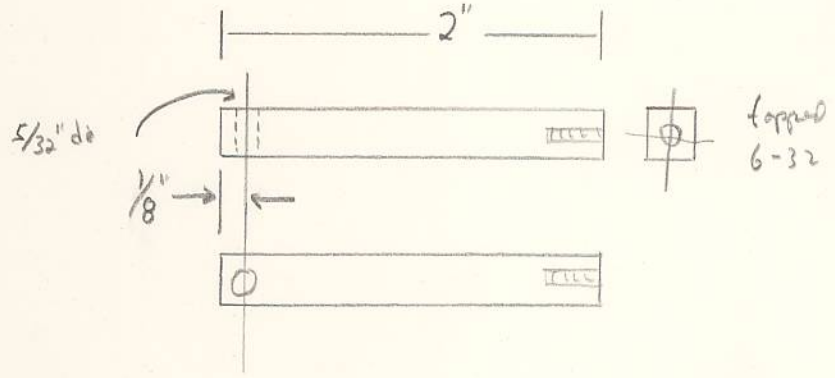
Aluminum POSTS

4pcs each type

1/4" square x 1 3/4" long / both ends tapped 6-32

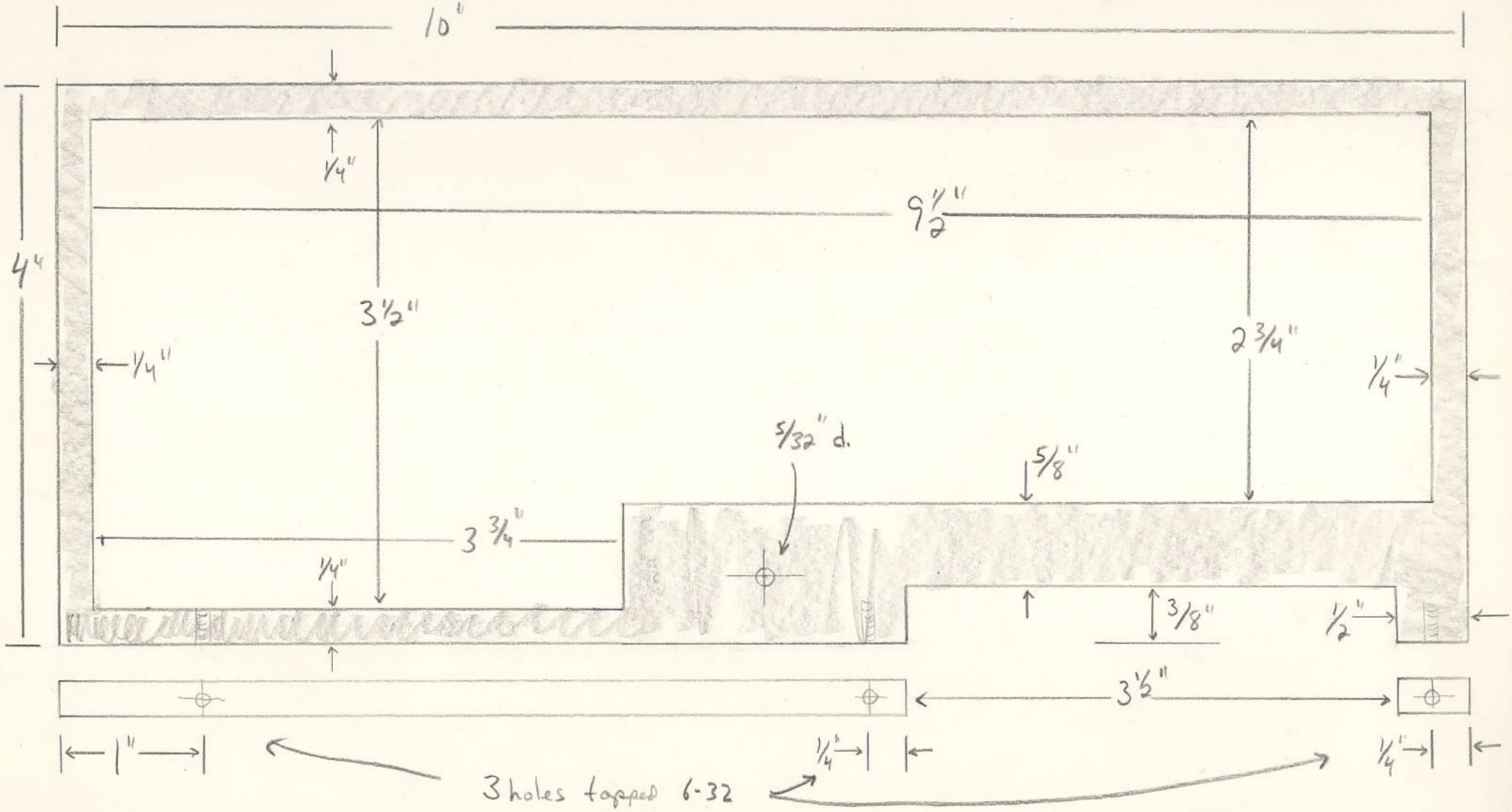


1/4" square x 2" long



Aluminum Panel

2pcs 1/4" thick 10" x 4"



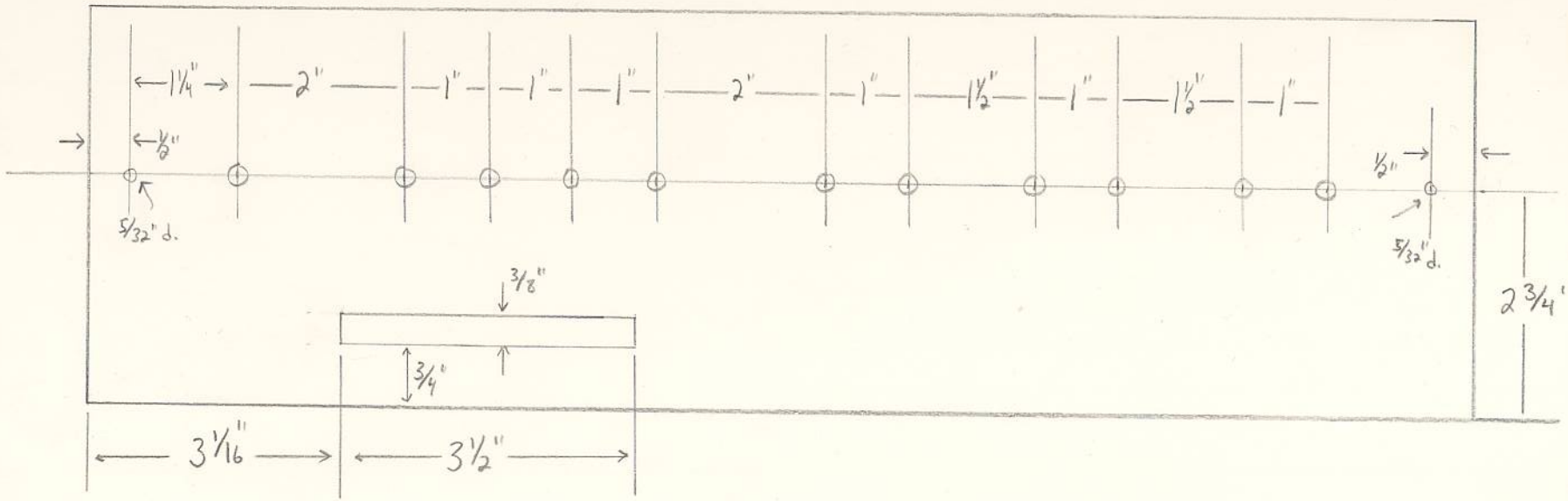
Gray 1/16 Aluminum panel – Supplied

Part A

Do not Marr front Surface

2pcs

All UNmarked Holes = 1/4" d.

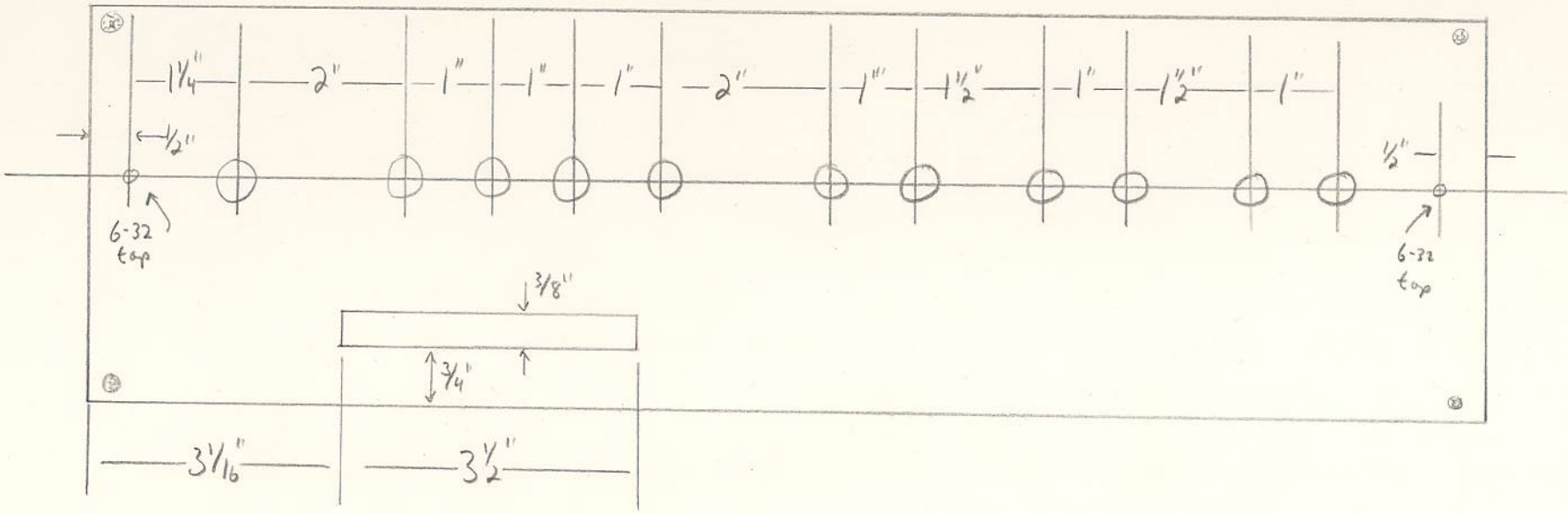


1/8" Aluminum panel – Supplied

Part B

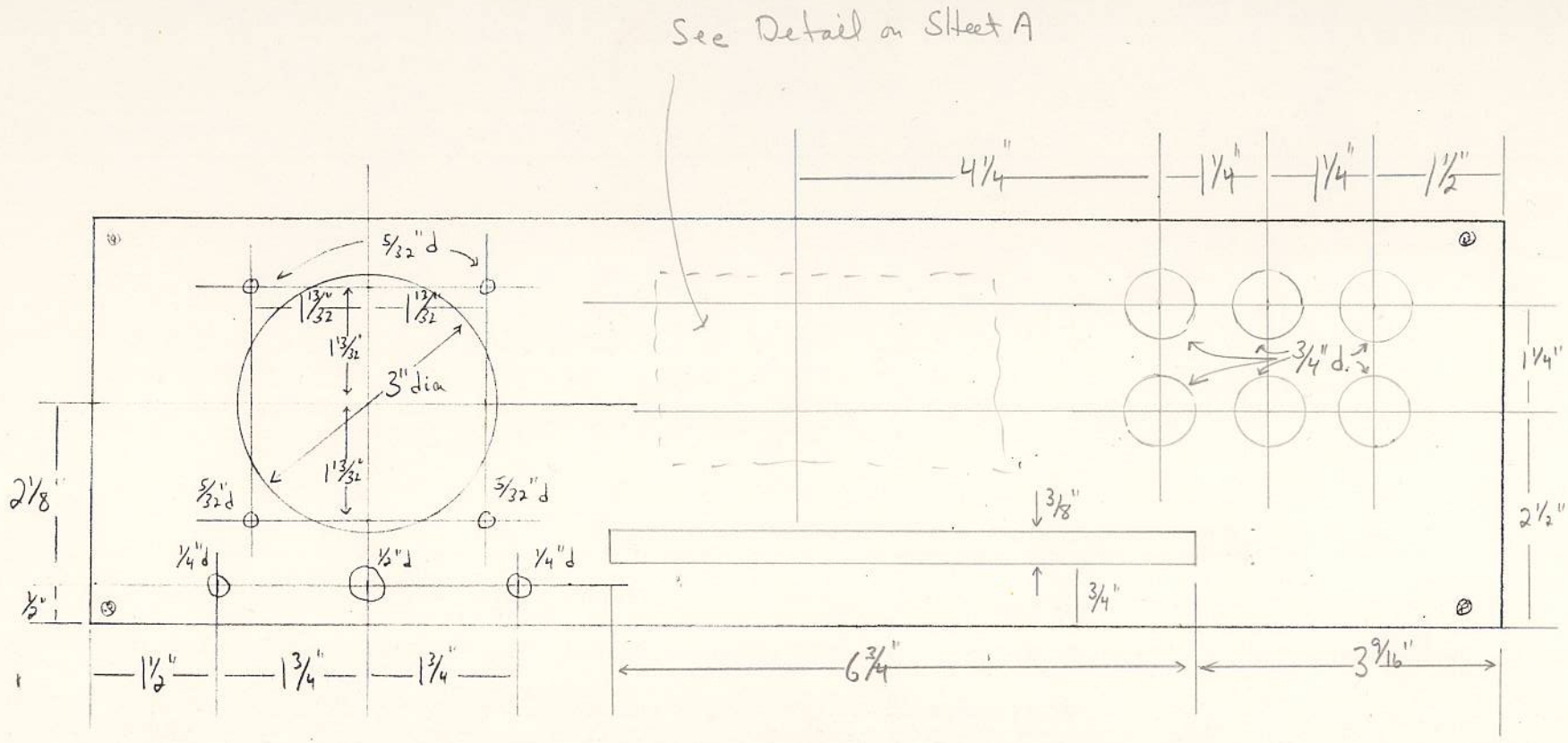
2 pcs

All Unmarked Holes = 3/4" d.

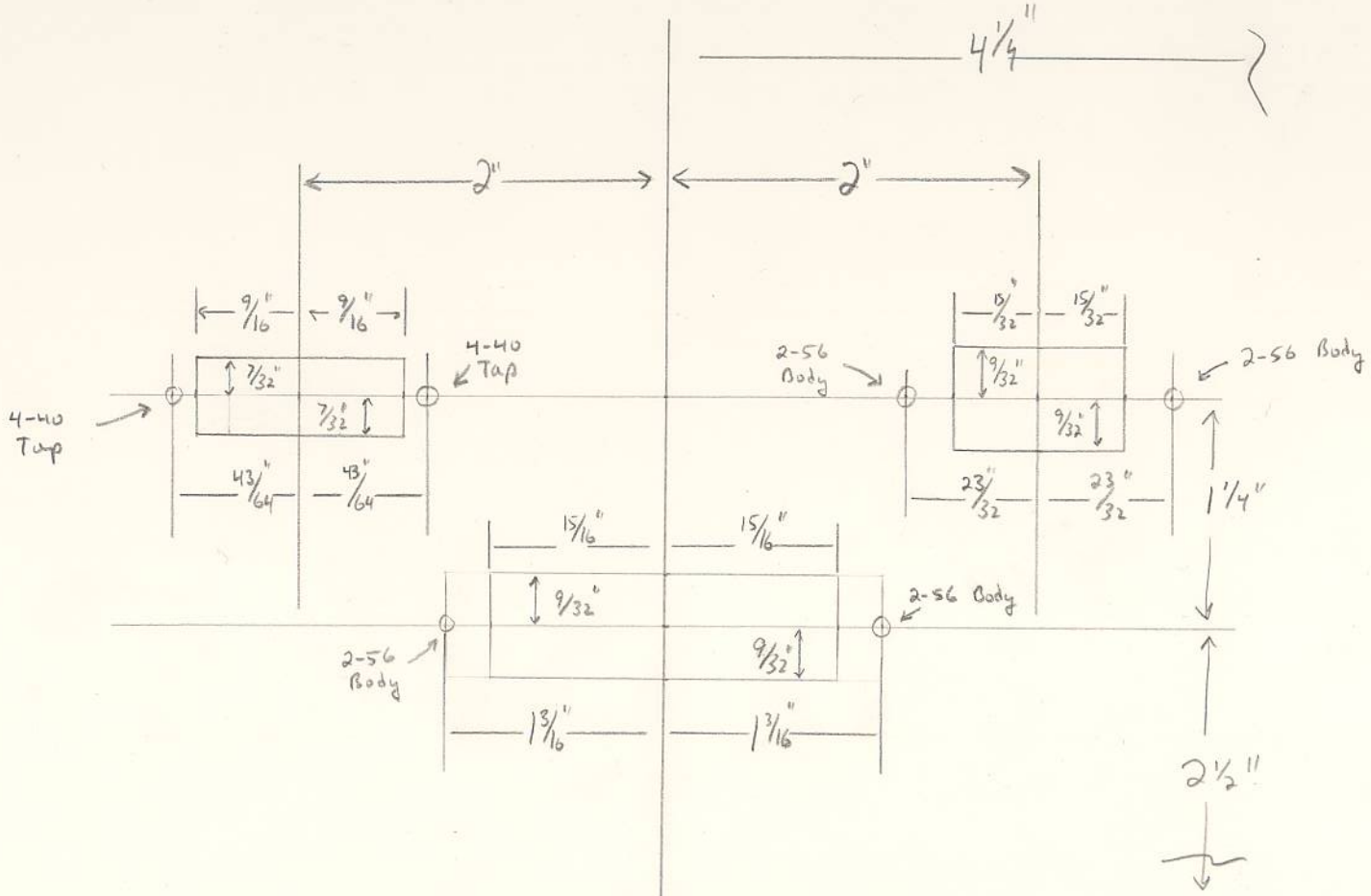


~~Blue~~ 1/8" Aluminum Panel - Supplied Part C

2pcs Gray



Sheet A Detail of Part C

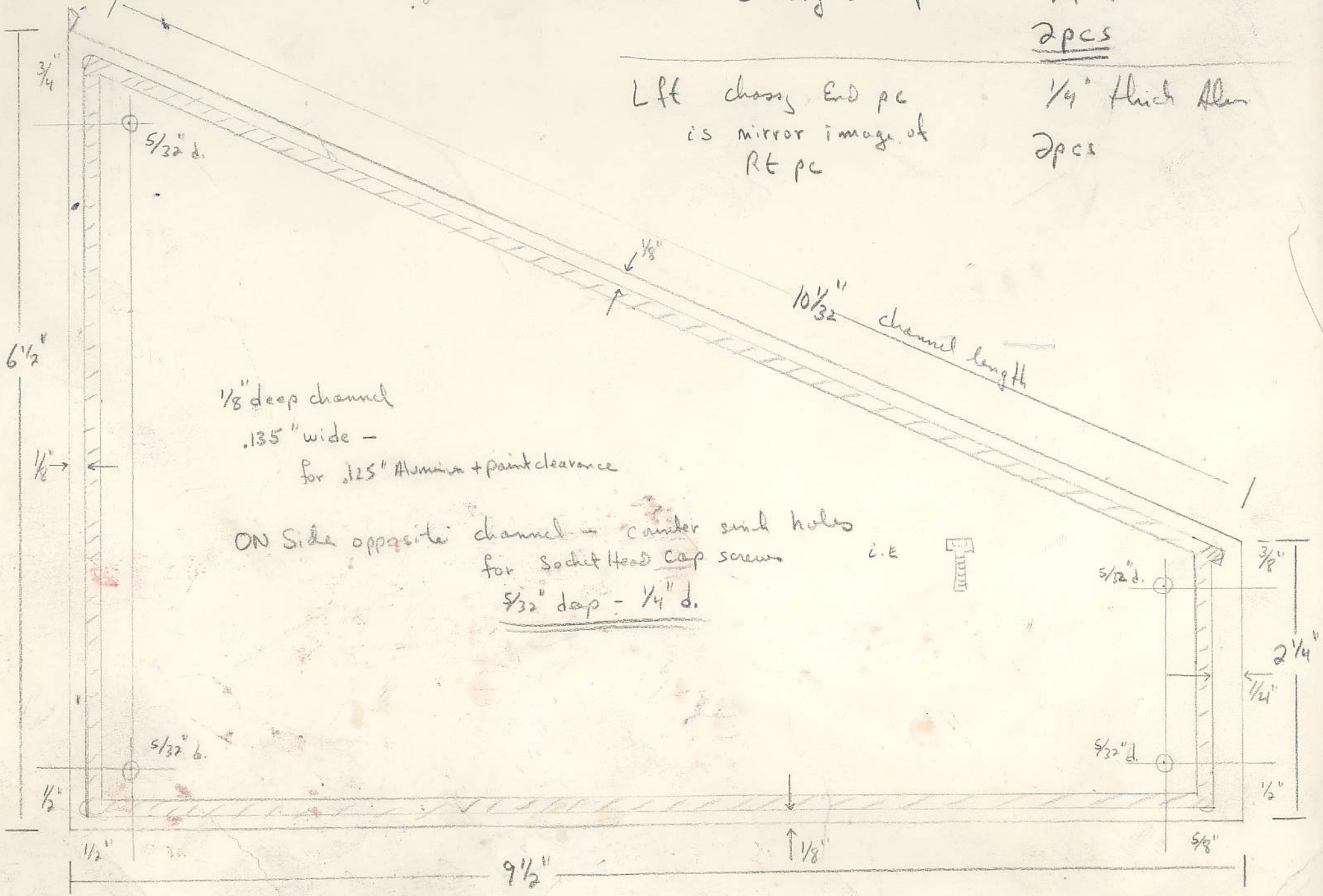


Rt. chassy End pc. 1/4" thick Aluminum

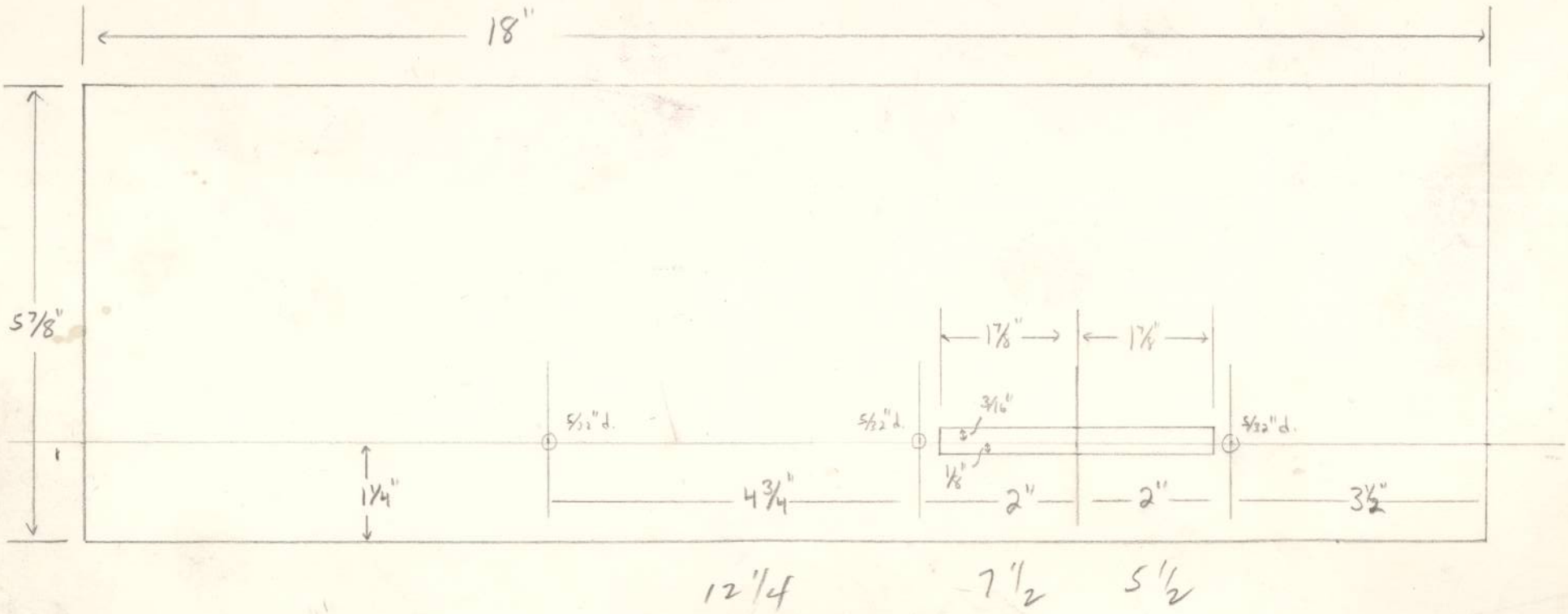
2pcs

Lft chassy End pc 1/4" thick Alum

is mirror image of Rt pc 2pcs



2pcs 1/8" Aluminum 5 7/8" x 18" (+.010 / -.000)

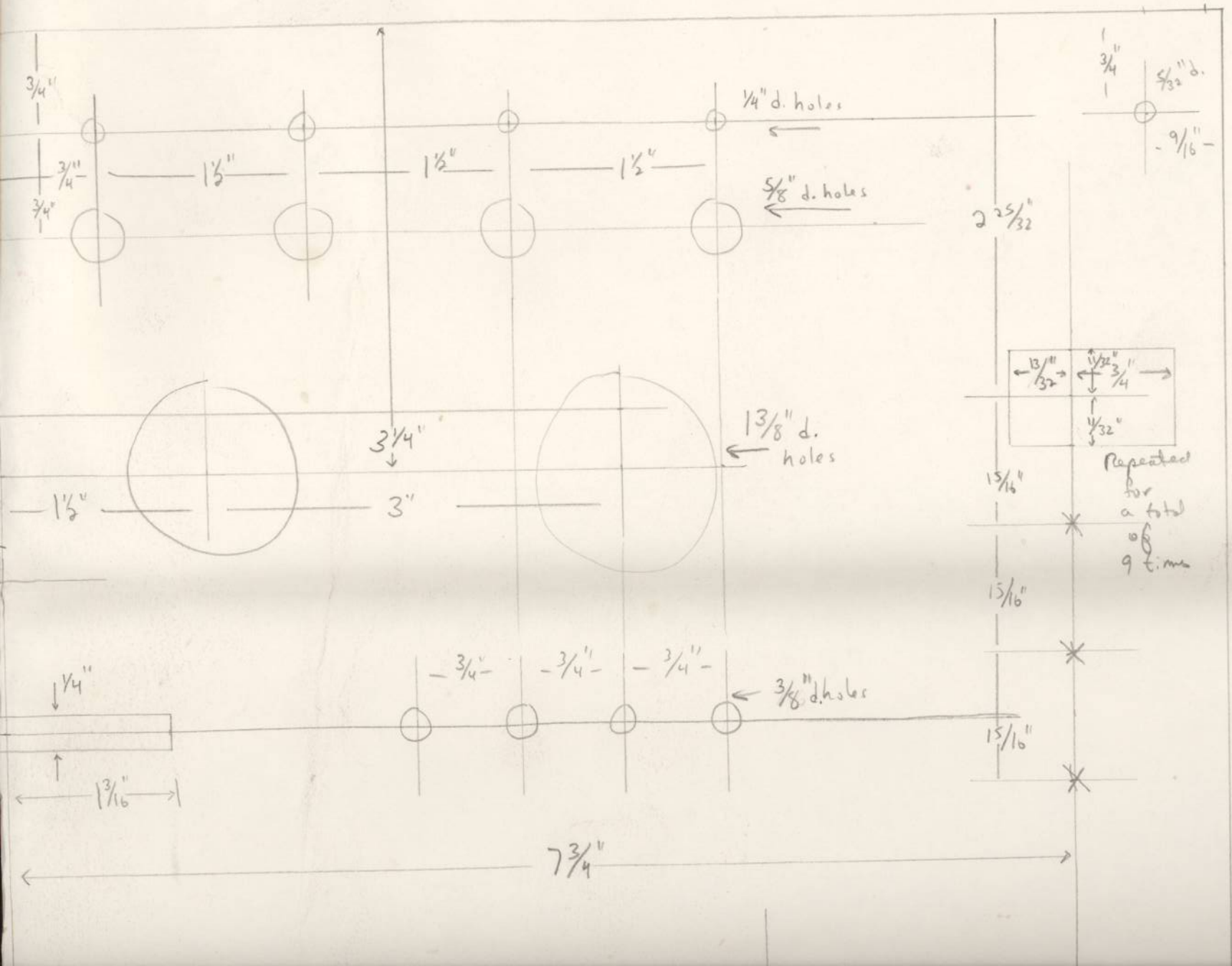


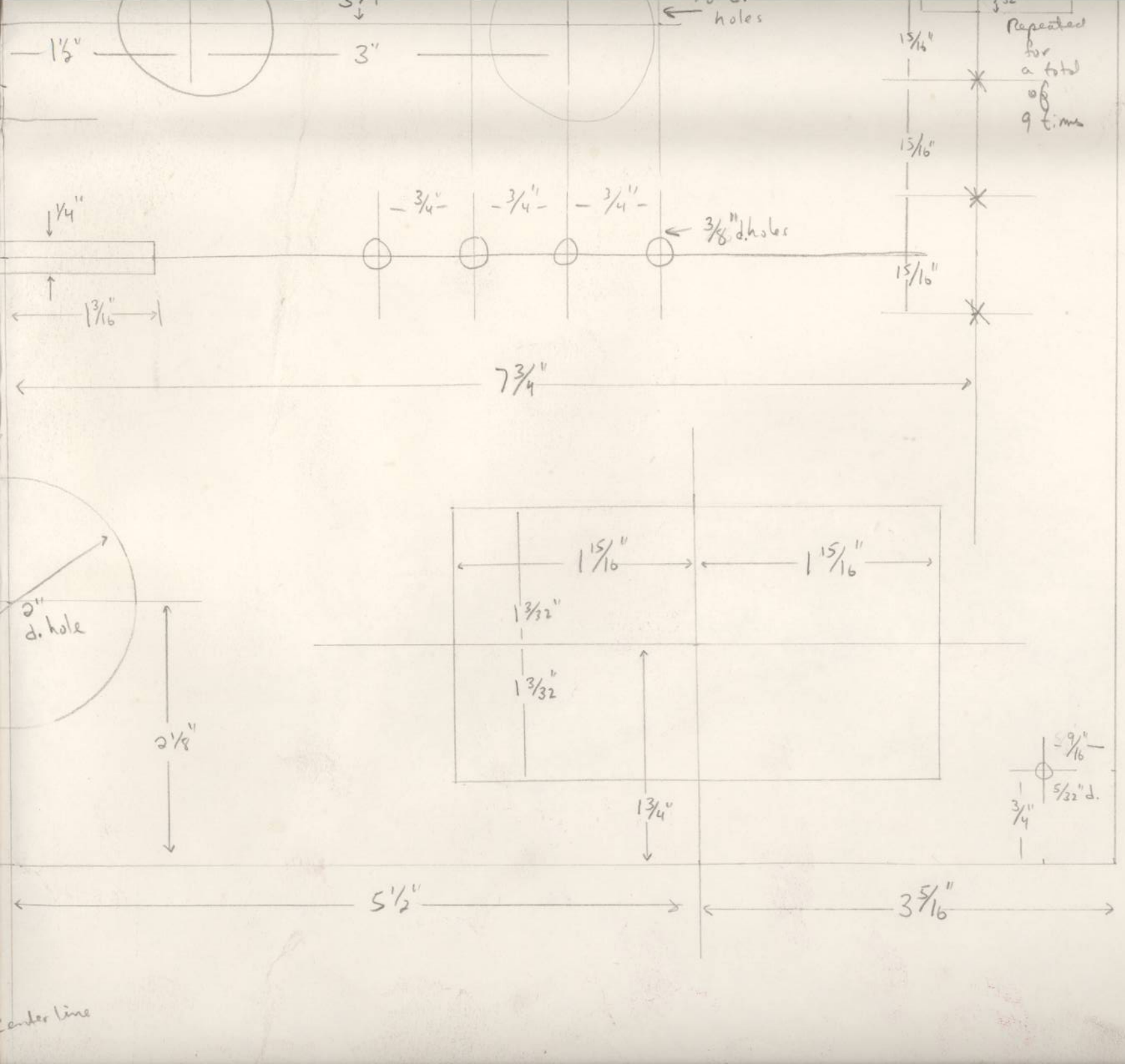
1.875
1.5
5.5
7.375
7

7 3/4"
2 1/4"
1 1/16"
3/4"
3/8" ø hole
3"
1 1/2"
1 1/4"
1 1/4"
3/4"
3/4"
1 5/16"
1 5/16"
1 5/16"
2 25/32"
3/4"
5/16"
3/4"
3/8"

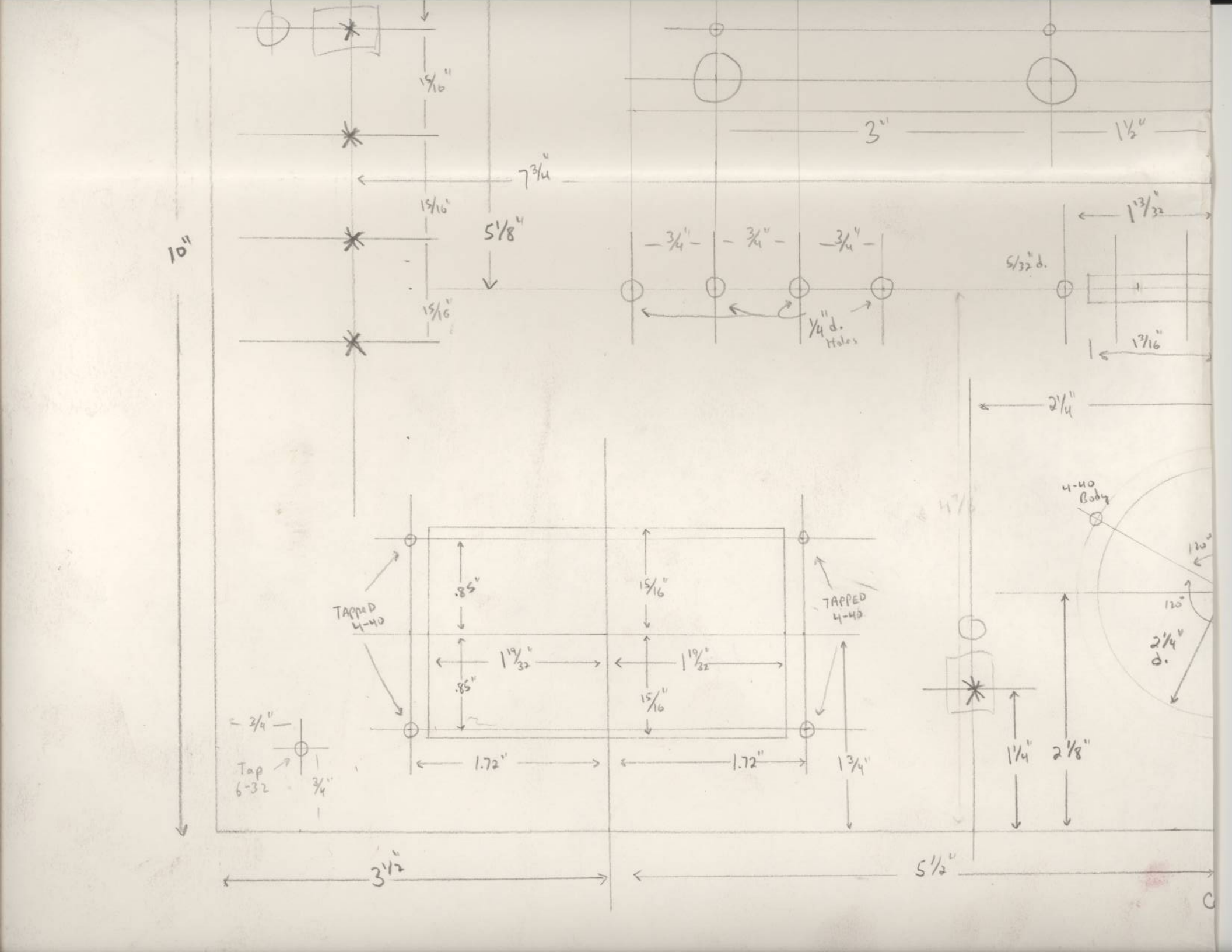
3 5/16"
5 1/2"
9/16"
5/32" d.
3/4"
1 3/4"
1 3/32"
1 15/16"
1 15/16"
7 3/4"
2 1/4"
1 1/4"
3/4"
3/4"
3/4"
3/8" d. hole
3"
1 1/4"
3/4"
15/16"
15/16"
15/16"
5 1/8"

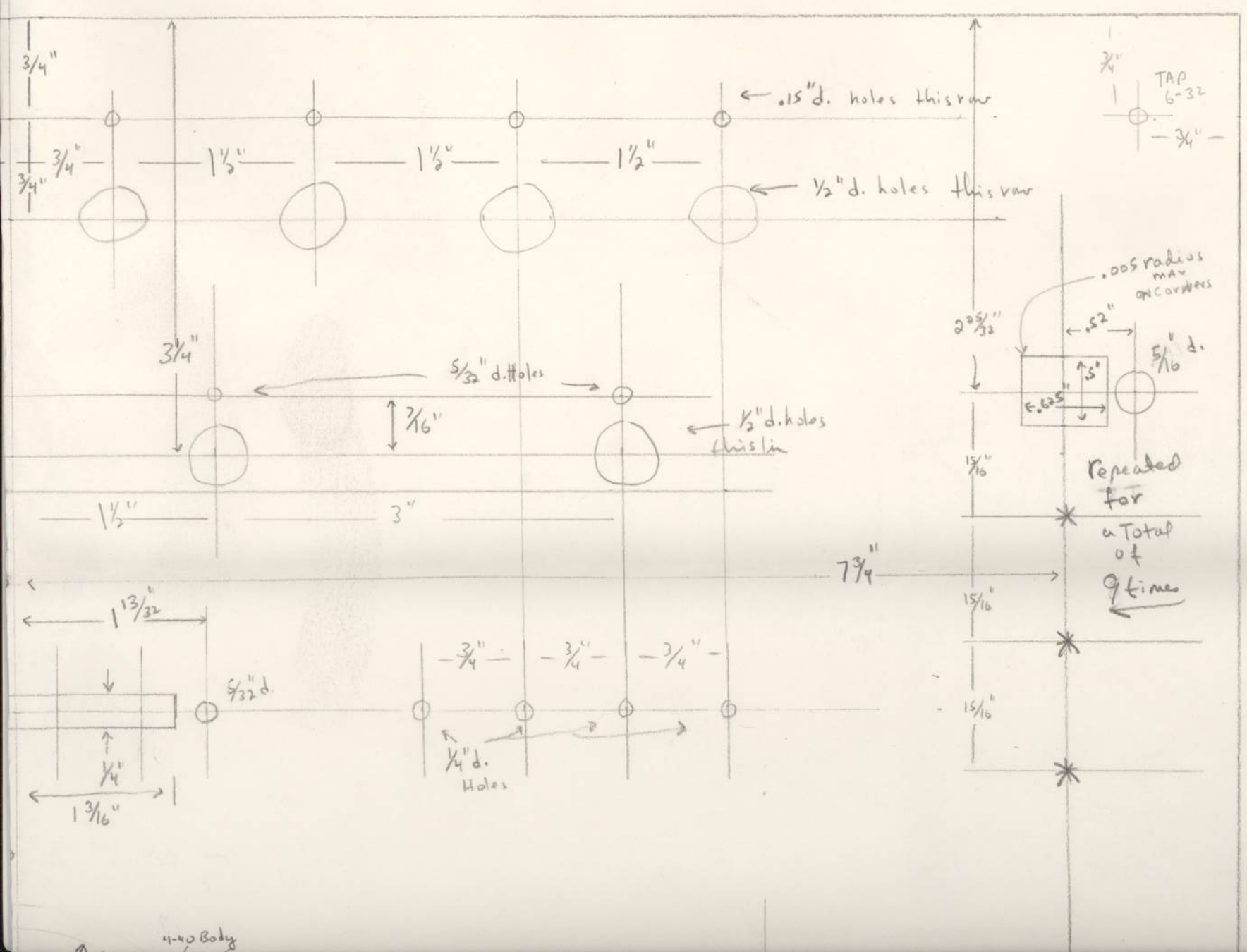
8 8½
562
250

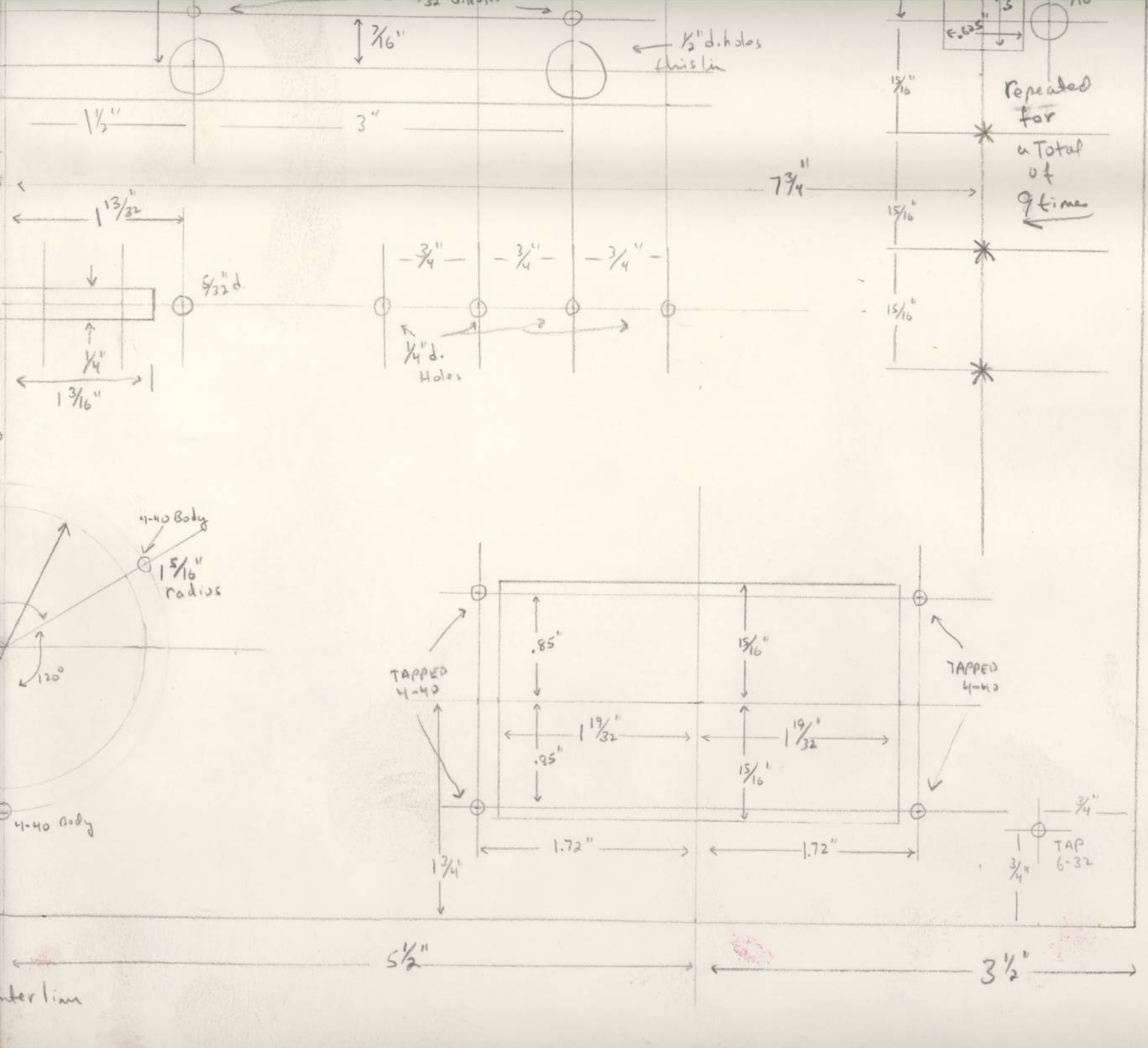

3/4"
3/4"
3/4"
1½"
1½"
1½"
¼" d. holes
5/8" d. holes
3¼"
13/8" d. holes
1½"
3"
¼"
3/4"
3/4"
3/4"
3/8" d. holes
1 3/16"
7 3/4"
3/4"
5/32" d.
9/16"
2 25/32"
13/32"
11/32"
3/4"
11/32"
15/16"
Repeated for a total of 9 times
15/16"
15/16"


1½"
3"
holes
15/16"
Repeated for a total of 9 times
15/16"
15/16"
¼"
3/4"
3/4"
3/4"
3/8" d. holes
1 3/16"
7 3/4"
2" d. hole
2 1/8"
1 15/16"
1 15/16"
1 3/32"
1 3/32"
1 3/4"
9/16"
5/32" d.
3/4"
5 1/2"
3 5/16"
center line

10"
TAP
6-32
3/4"
3/4"
2 23/32"
1 5/16"
1 5/16"
5 1/8"
7 3/4"
1 1/2"
1 1/2"
1 1/2"
3"
1 1/2"
3/4"
3/4"
3/4"
3/4"
1/4" D. Hole
5/32" D.
2 1/4"
1 3/16"
1 3/32"

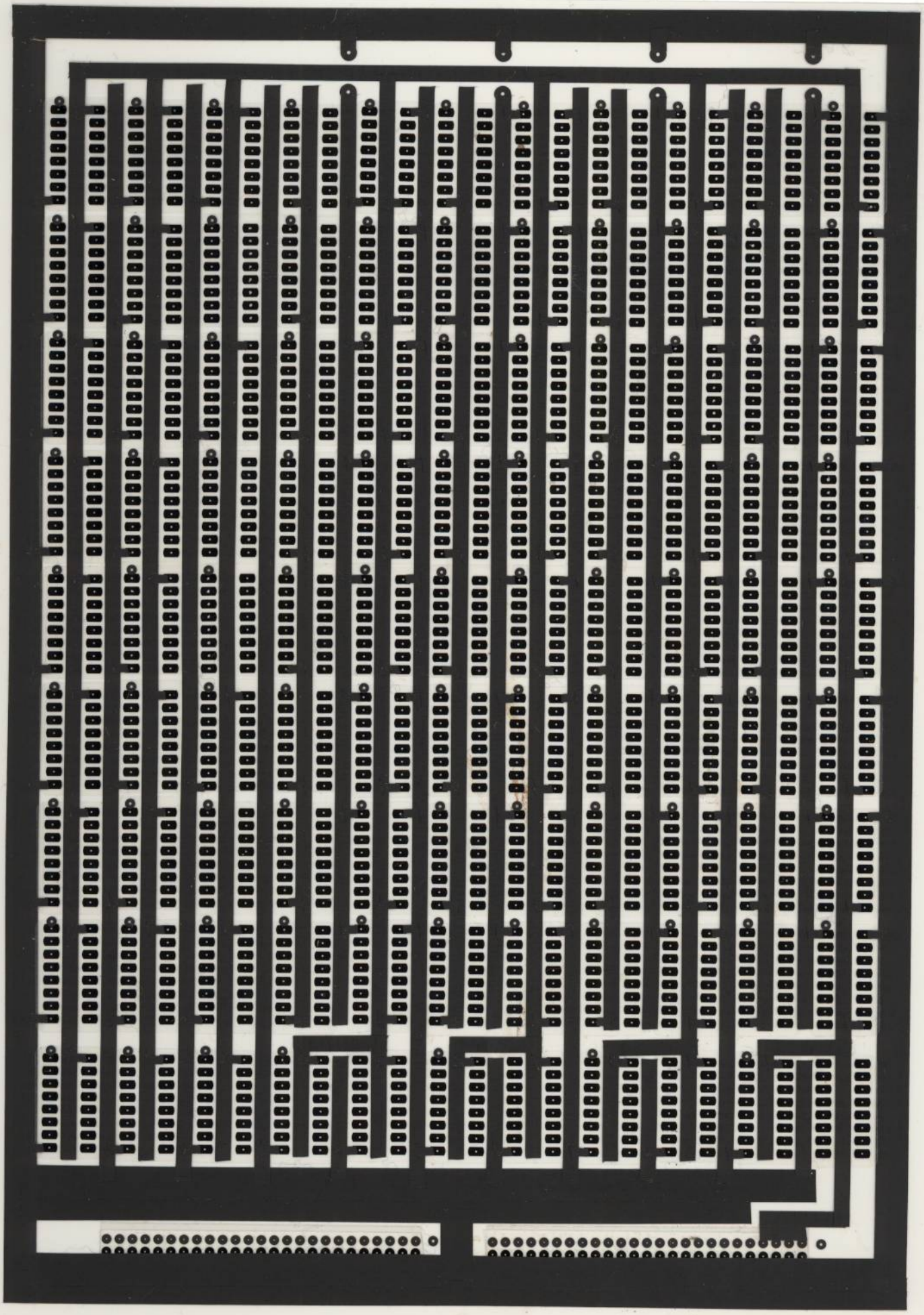

10"
15/16"
15/16"
15/16"
7 3/4"
5 1/8"
3"
1 1/2"
3/4"
3/4"
3/4"
1/4" d.
Holes
1 13/32"
5/32" d.
1 3/16"
2 1/4"
4-40
Body
120°
120°
2 1/4"
d.
TAPPED
4-40
.85"
15/16"
1 19/32"
1 19/32"
.85"
15/16"
TAPPED
4-40
3/4"
Tap
6-32
3/4"
1.72"
1.72"
1 3/4"
1 1/4"
2 1/8"
3 1/2"
5 1/2"


3/4"
3/4"
3/4"
1 1/2"
1 1/2"
1 1/2"
← .15" d. holes this row
← 1/2" d. holes this row
3 1/4"
5/32" d. Holes
7/16"
← 1/2" d. holes this line
1 1/2"
3"
1 13/32"
5/32" d.
1/4"
1 3/16"
3/4"
3/4"
3/4"
1/4" d. Holes
7 3/4"
2 25/32"
.052"
.005 radius MAY OCCUR HERE
.525"
.5"
5/16" d.
15/16"
15/16"
15/16"
Repeated for a Total of 9 times
3/4"
TAP 6-32
3/4"
4-40 Body


7/16"
1/2" d. holes this line
1 1/2"
3"
1 13/32"
5/32" d.
1/4"
1 3/16"
3/4"
3/4"
3/4"
1/4" d. Holes
15/16"
repeated for a Total of 9 times
7 3/4"
15/16"
15/16"
.625"
4-40 Body
1 5/16" radius
120°
4-40 Body
TAPPED 4-40
.85"
15/16"
1 19/32"
.85"
15/16"
1 19/32"
TAPPED 4-40
3/4"
1.72"
1.72"
1 3/4"
TAP 6-32
3/4"
5 1/2"
3 1/2"

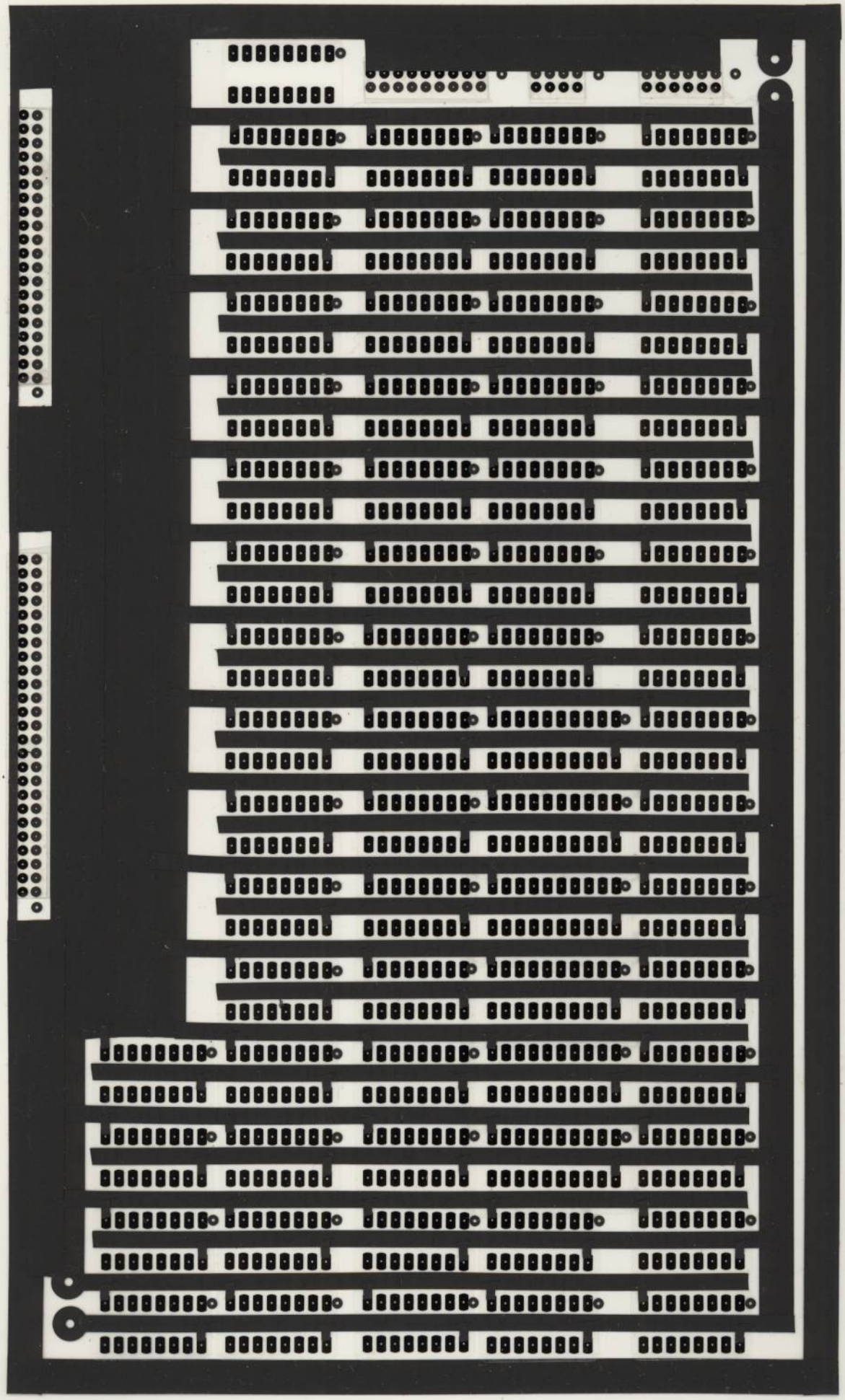
FEED ↑ UP
MODEL 914, 720, 420, 2400

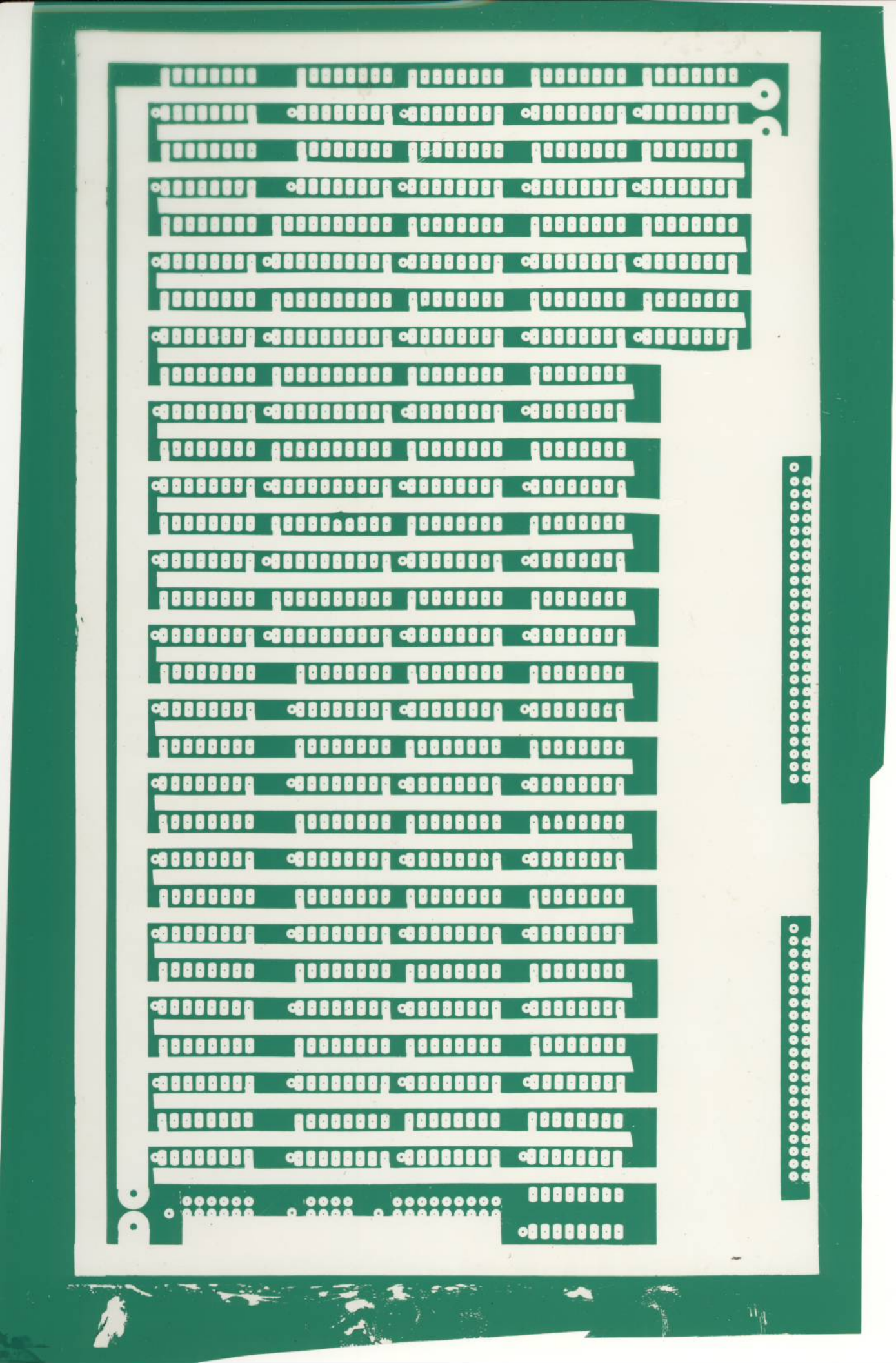
NOTES



FEED ↑ UP
MODEL 914, 720, 420, 2400

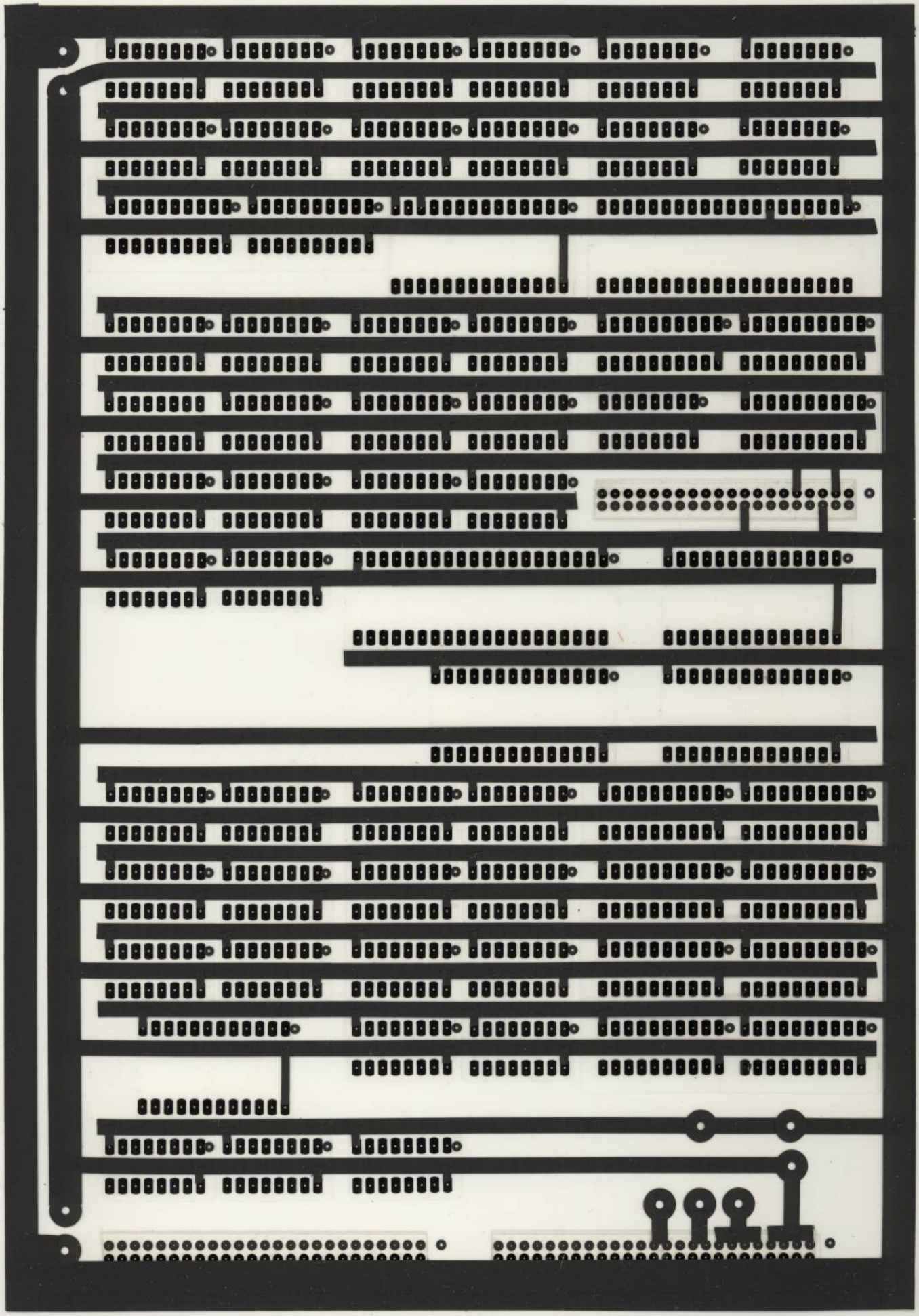
NOTES

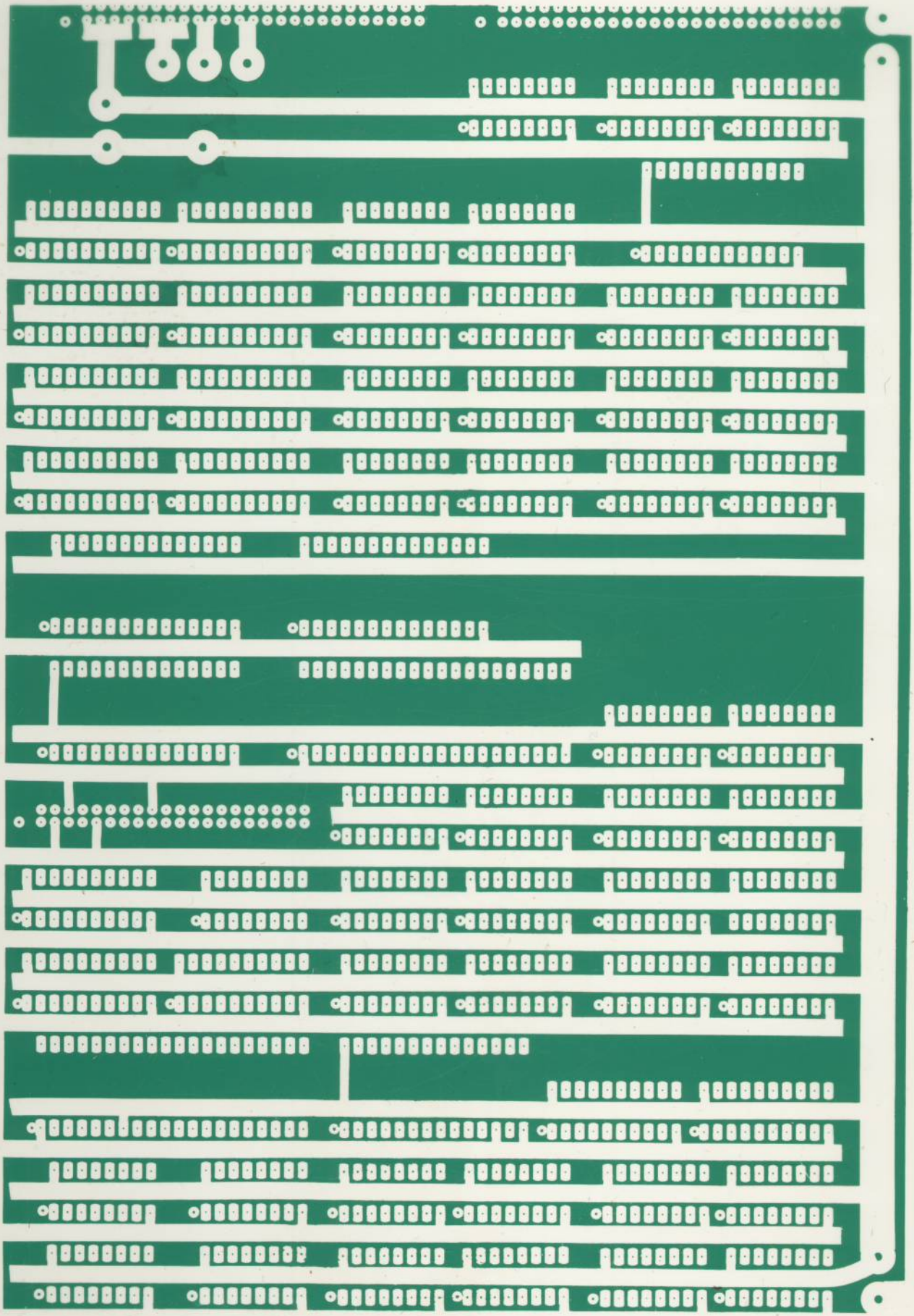




FEED ↑ UP
MODEL 914, 720, 420, 2400

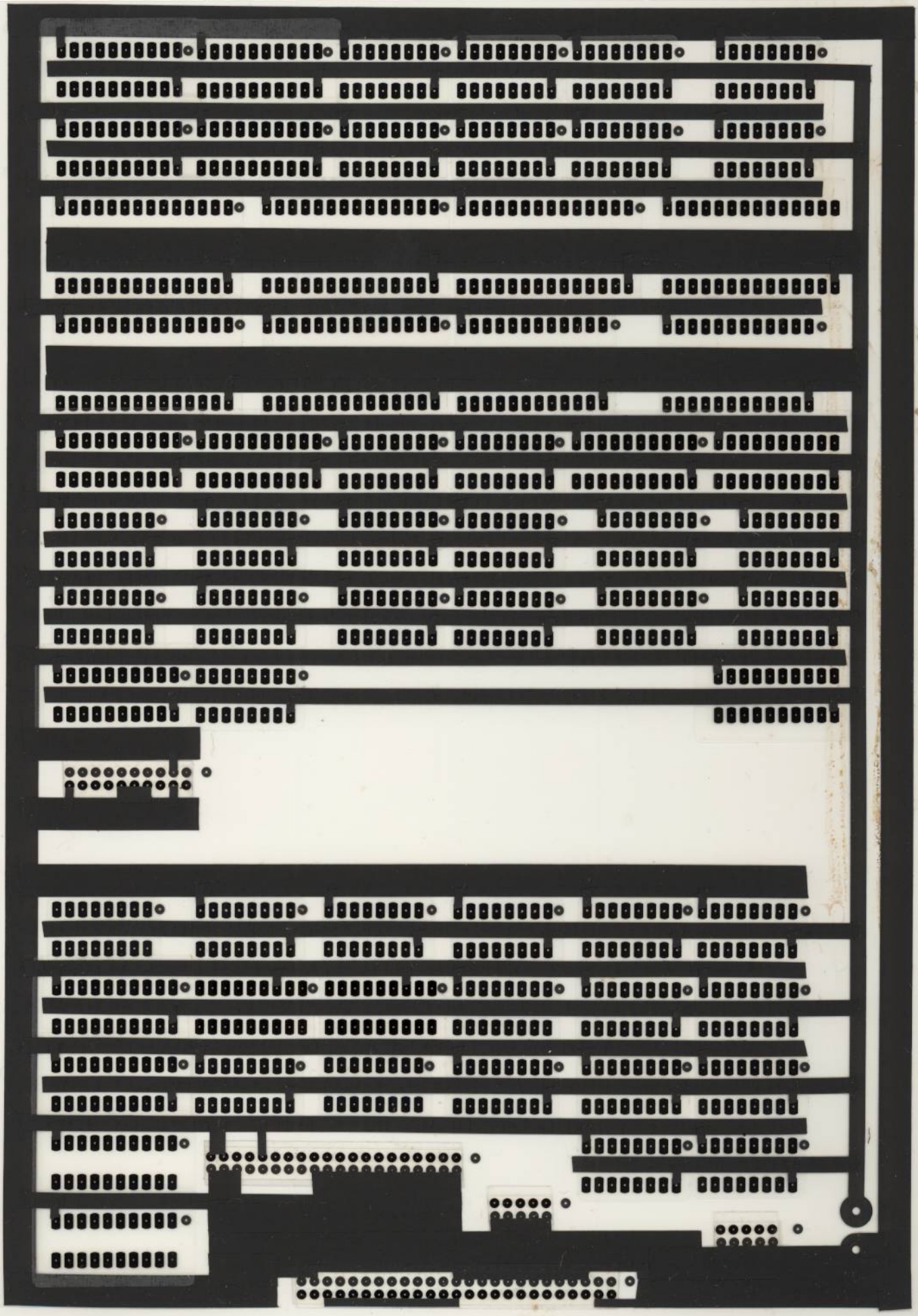
NOTES

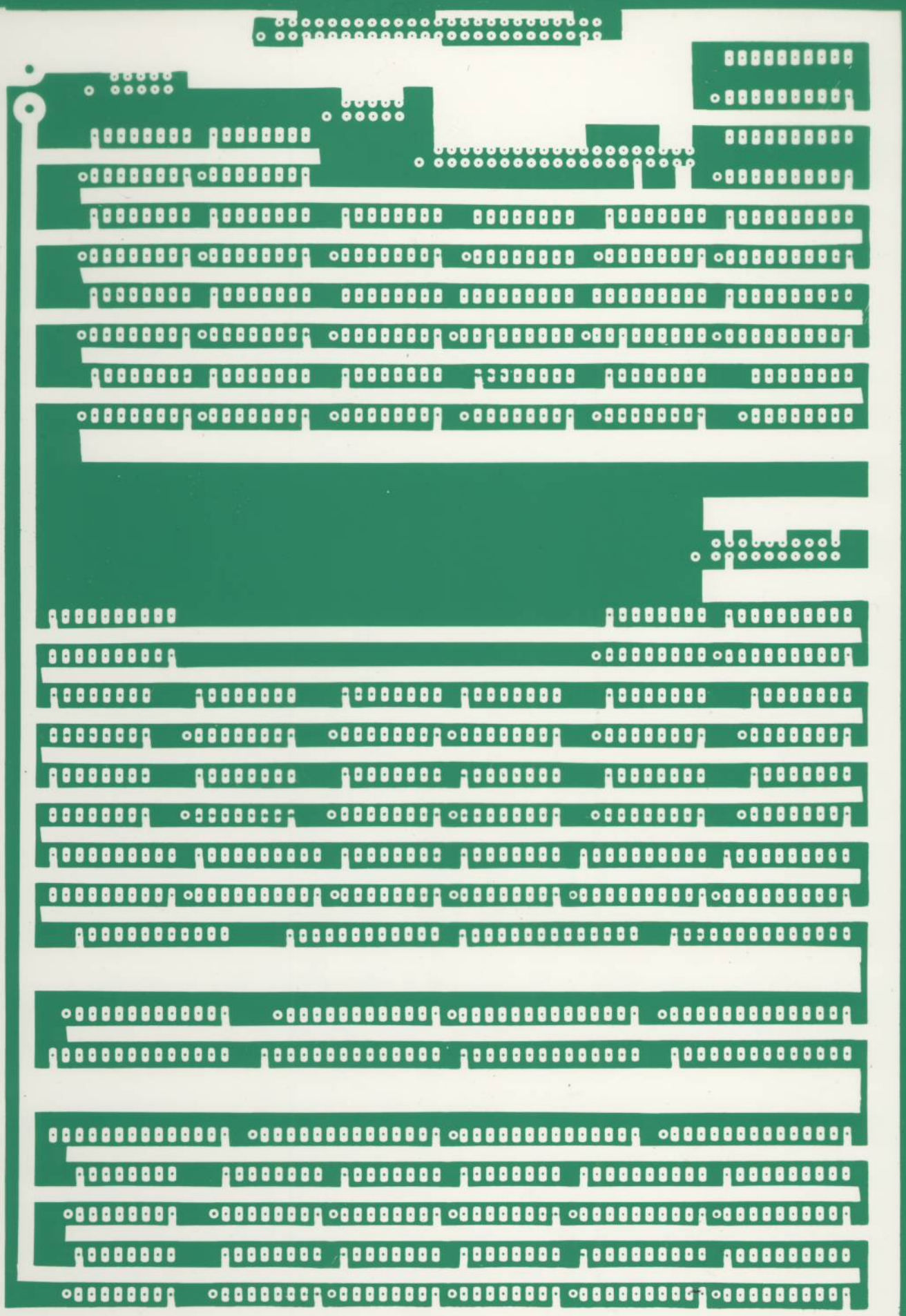




FEED ↑ UP
MODEL 914, 720, 420, 2400

NOTES

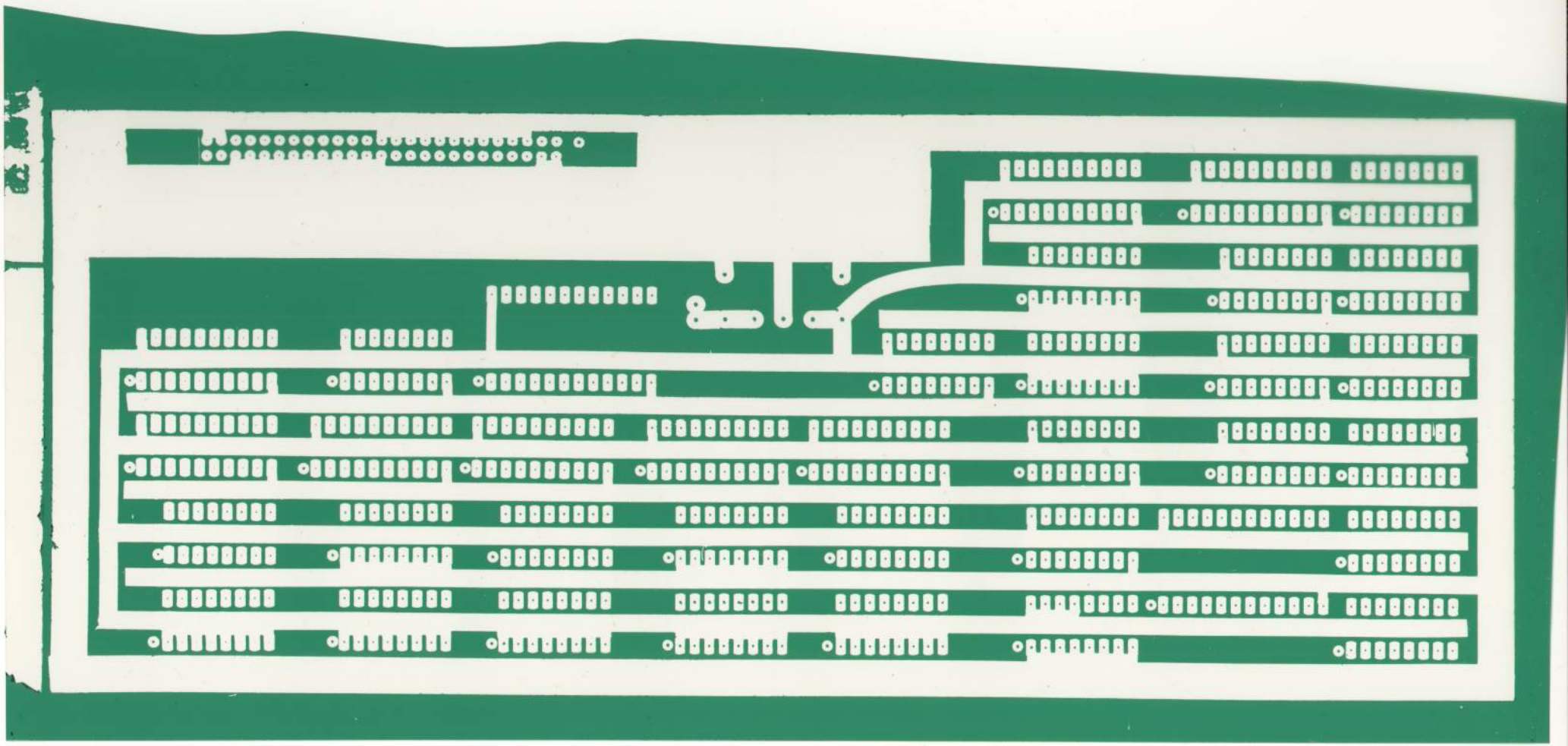


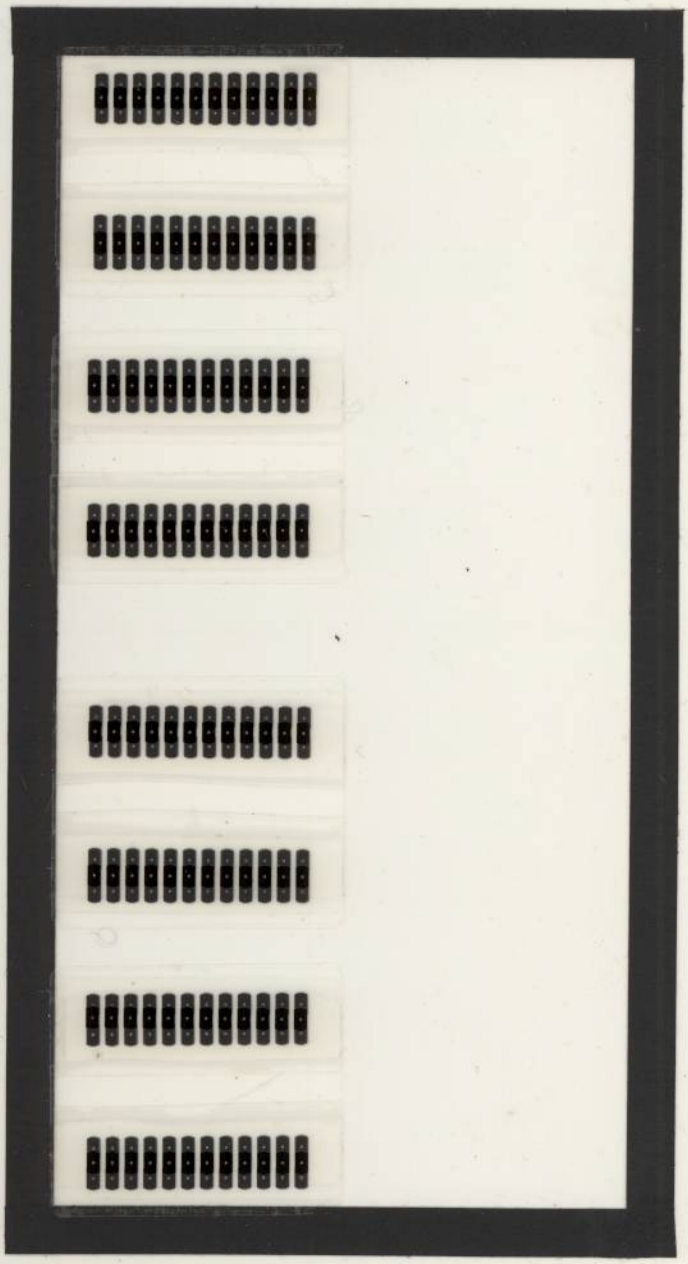


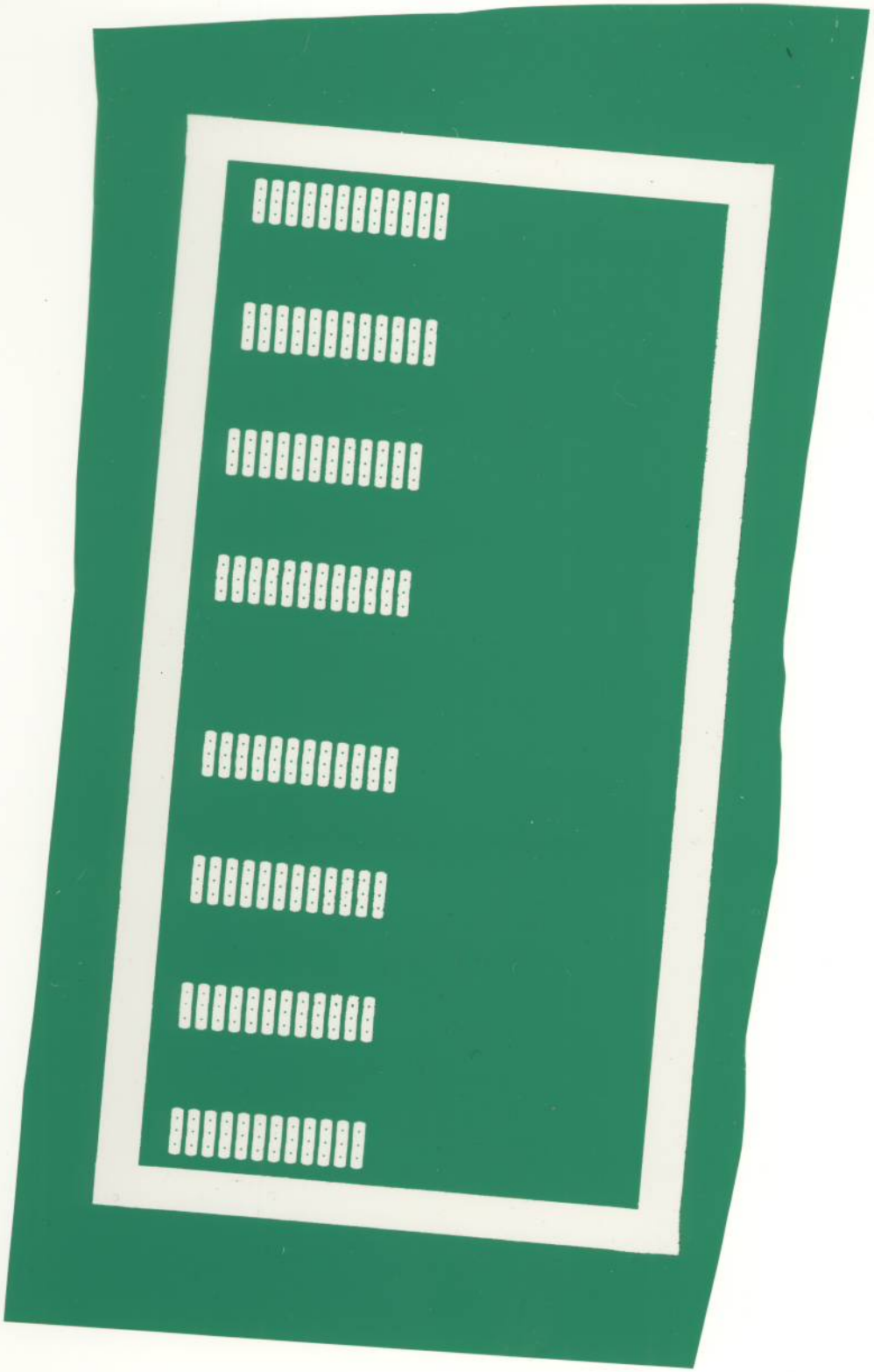
FEED ↑ UP
MODEL 914, 720, 420, 2400

NOTES

FEED ↑ UP
MODEL 818, 088, 330







intech
MICROCIRCUITS DIVISION

RGB DAC 4TD

HYBRID
TRIPLE 4 BIT
COLOR MAPPED VIDEO DAC

DESCRIPTION

This hybrid digital to analog converter provides 3 channels of video output for RGB type video displays with picture element (pixel) rates to 40MHz. The RGB DAC 4TD contains 3 video DAC's and 3 RAM arrays (16 x 14) as well as all control inputs required to generate output waveforms which are compatible with EIA standards RS-170 and RS-343 (see fig. 3).

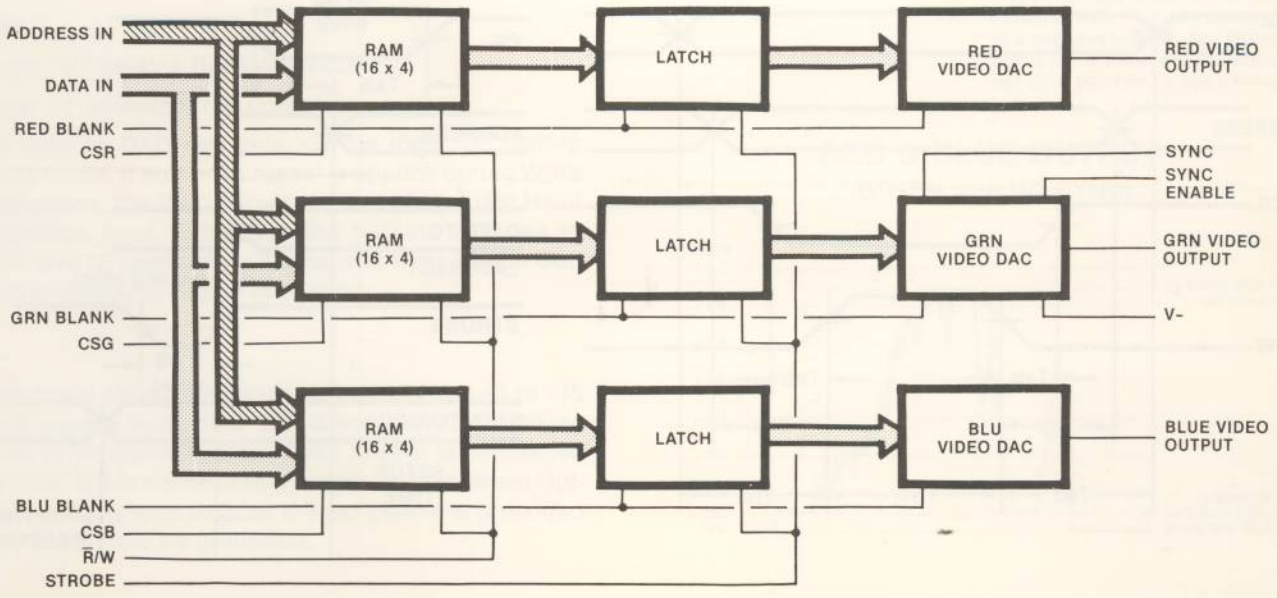
The RGB DAC 4TD provides 4 bits of resolution or 16 levels of "gray scale" per color. This gives the user a $(2^4)^3$ or 4096 color palette. The 16 x 4 ram array per color allows the user a choice of any 16 of the 4096 colors for each sweep. The write speed is also high enough to allow the color map to be updated during the horizontal retrace. The high update rates and the ability to generate a 1 Volt composite video output across the 75 Ohm load makes this DAC ideally suited for color computer graphics using raster scan technology.

FEATURES

- 4096 COLOR PALETTE
- 3 VIDEO DACS
- UPDATE RATES TO 40 MHz
- COMPLETE COMPOSITE VIDEO CAPABILITY
- 16 x 4 BIT COLOR MAP RAM
- TTL COMPATIBLE
- DIRECT 75 OHM RGB OUTPUTS
- CLEAN OUTPUT (Deglitching not required)
- SINGLE +5V SUPPLY
- SMALL SIZE—24 PIN DIP

BLOCK DIAGRAM

FIG. 1



SPECIFICATIONS

(Typical @ +25°C, +5V and 75Ω load unless otherwise stated)

RESOLUTION	4 bits
PALETTE	4096
(max colors per sweep)	16
ANALOG OUTPUTS (each channel)	
Voltage Range (1%)	0 to 0.6 volts
Current (Max)	17 mA
Impedence (±5%)	75 ohms
Compliance	±2 volts
OUTPUT-COMPOSITE SYNC	
Green Only[4] (±5%) (Referred to Blanking Level)	-286 mV
OUTPUT-COMPOSITE BLANKING	
(Referred to Black Level)[4] (±5%)	-50 mV
ACCURACY (Each Channel)	
Absolute	± ½ LSB
Linearity	± ½ LSB
Offset	10 mV
Offset	100 ppm/°C
Gain Tempco	500 ppm/°C
Linearity Tempco	250 ppm/°C
DYNAMIC CHARACTERISTICS (each DAC)	
Settling Time (to 1 LSB)	5 nS
Update Rate	40 MHz
Rise Time	3 nS
Slew Rate	200 V/μs
DATA & ADDRESS INPUTS	
Compatibility	TTL
Coding	BIN

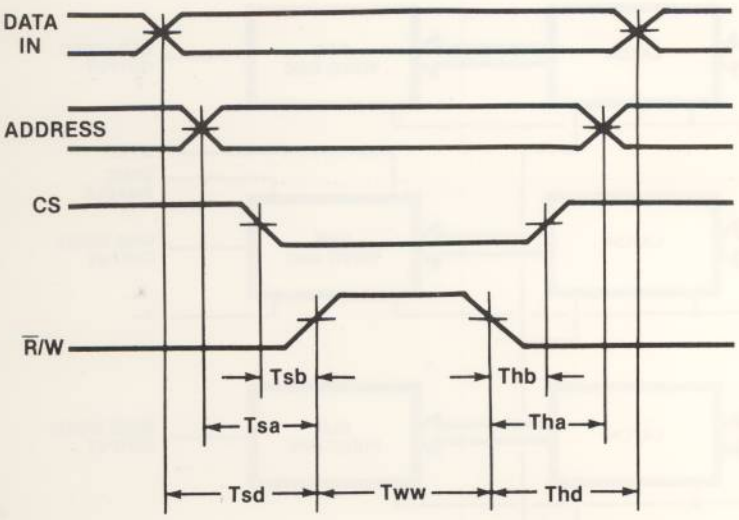
CONTROL INPUTS	
STROBE	
Compatibility	TTL
Set-up Time[1]	25nS
Hold Time (From Valid Address)	0 nS
Propagation	12 nS
COMPOSITE BLANKING SYNC	
Compatibility	TTL
Settling Time	20 nS
DYNAMIC CHARACTERISTICS (each RAM)	
Write Pulse Width (Tww) (min)	25 nS
Address Setup Time[2] (Tsa) (min)	0 nS
Data Setup Time[2] (Tsd)	25 nS
CS Setup Time[3] (Tsb)	25 nS
Address Hold Time[3] (Tha)	0 nS
Data Hold Time[3] (Thd)	0 nS
Chip Select Hold Time[3] (Thb)	0 nS
Address Access Time (Taa) (Max)	20 nS
Chip Select Access Time (Tab)	15 nS
POWER REQUIREMENTS	
Voltage	+4.5 to +5.5 V
Current (Typ)	480 mA
(Max)	580mA
V- Input	
Voltage	-3 to -15 V
Current (Max)	-16 ma
TEMPERATURE RANGE (ambient)	0 to 70 degrees C

NOTES: 1. Time from stable "Read Address' to Strobe Hi to Lo Transition.
2. Lo to Hi R/W Transition
3. Hi to Lo R/W Transition.
4. See Figure 3.

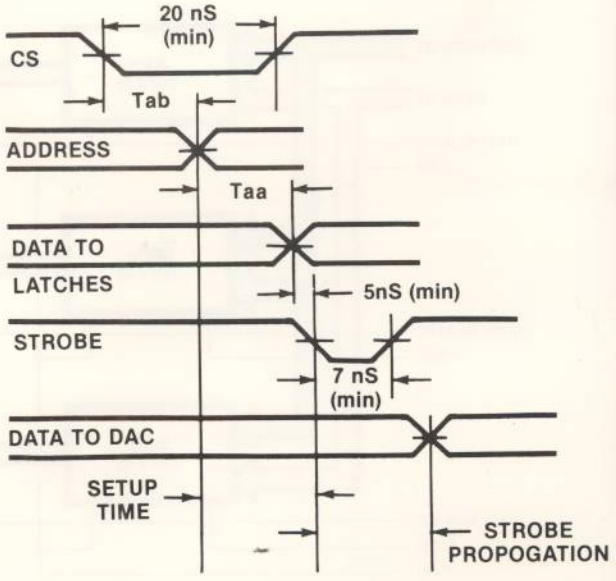
TIMING DIAGRAMS

FIG. 2

WRITE CYCLE



READ CYCLE



PIN DESCRIPTIONS

STROBE

The transition from Logic "1" to Logic "0" transfers data from RAM outputs to the Data Register outputs (DAC inputs). The Strobe is not used for Write operations.

GREEN SYNC (Composite Sync)

A Logic "1" sets the registers to all "0's" (Ref Black) and drives the Green output to a level 286mV more negative than the Reference Blanking level. This signal has priority over Blanking and Data Inputs from the RAM's and is used to synchronize the beam traces in a Raster Scan type display.

BLANKING (Red, Grn, Blu)

A Logic "1" sets the registers to all "0's" (Ref Black) and drives the output to a level 50mV more negative than the Reference Black level. This signal has priority over the Data Inputs from the RAM's and is used to shut off the beam for the darkest display possible.

SYNC ENABLE

This input should be connected to +5V to enable the Sync Signal feature of the Green Output. If this feature is not desired this input should be connected to ground.

ADDRESS-4 Bits-A_0, A_1, A_2, A_3

Determines Read or Write Address for all RAM arrays. These inputs are used as an Address Input to load arrays in the Write mode. During the Read mode display colors are obtained by addressing the appropriate memory locations and transferring the memory data to the DACs.

DATA-4 Bits-D_0, D_1, D_2, D_3

Used to load data to all RAM arrays-Data loaded will be non-inverted to the DAC during Read Mode.

$\overline{R/W}$

Logic "O" selects Read operation.

Logic "1' selects Write operation.

All data to DAC registers will be logic "1" during Write mode. If no strobe signal is applied during Write operations, the DAC outputs will retain the last Read operation level. Otherwise the outputs will rise to REF WHITE level unless Composite Sync (Green Output) or Blanking Signal is applied.

V-

This input should be connected to a stable -3 to -15 Volt supply to restore the Green Output Blanking level to 0V absolute if the Sync Enable is connected to +5V. If it is not necessary to restore the Green Output blanking level to 0V or if Sync Enable is grounded this pin should be grounded.

$\overline{CSR}$, $\overline{CSG}$, $\overline{CSB}$

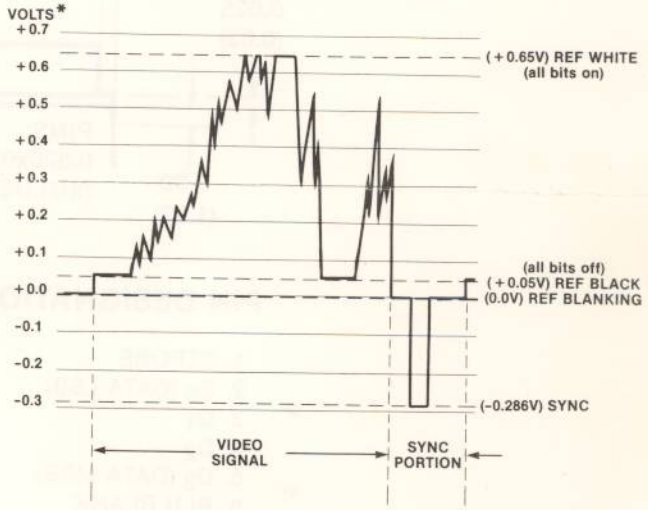
Chip select for each channel (Red, Green, or Blue). A logic "0" selects RAM channel for Read or Write operations and a "1" deselects RAM for any operation. If deselected, the data to the DAC registers will be logic "1". If no strobe signal is applied while the RAM is deselected DAC outputs will retain the last "Read" operation level. Otherwise the output will rise to REF WHITE level unless Composite Sync (Green Output) or Blanking Signal is applied. Normally only one input is enabled (logic "0") to Write data to its corresponding RAM and all inputs are enabled (logic "0") to read data during display.

COMPOSITE VIDEO WAVEFORMS

FIG. 3

GREEN OUTPUT

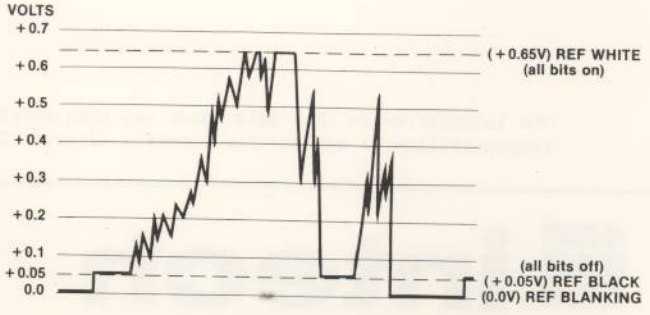
(with SYNC ENABLED)



*NOTE: Voltage levels shown are obtained with "V-" pin connected to a negative (-3V to -15V DC) level. If this pin is grounded the output will shift positive by 286 millivolts.

RED & BLUE OUTPUT

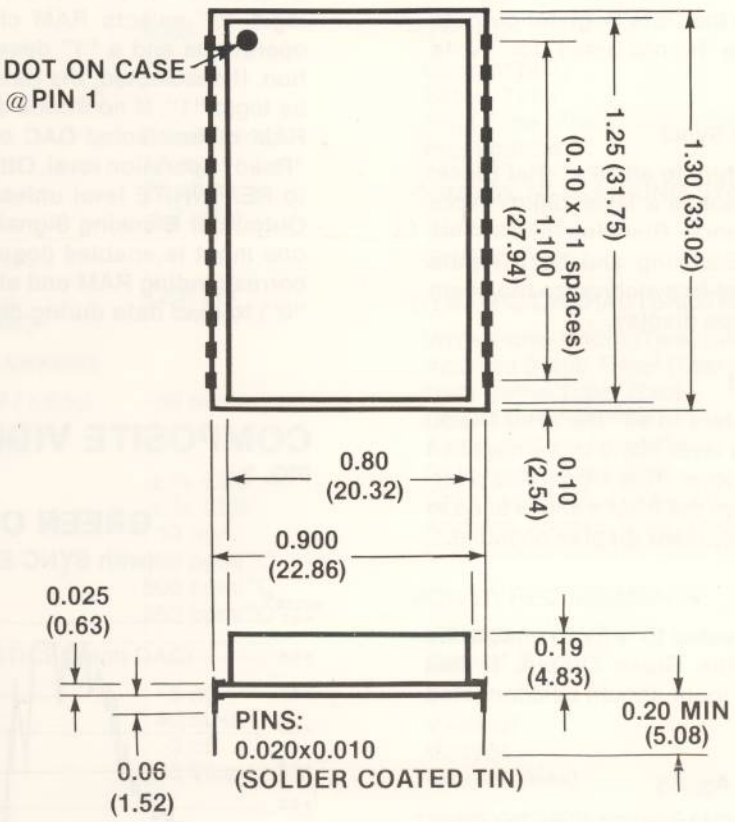
(GREEN with NO SYNC)



RGB DAC 4TD

MECHANICAL OUTLINE

DIMENSIONS IN INCHES
(MM)



PIN DESIGNATIONS

1. STROBE
2. D_0 (DATA LSB)
3. D_1
4. D_2
5. D_3 (DATA MSB)
6. BLU BLANK
7. $\overline{R}/W$
8. A_0 (ADDRESS LSB)
9. A_1
10. A_2
11. A_3 (ADDRESS MSB)
12. GND
13. RED BLANK
14. GRN SYNC
15. $\overline{GRN}$ BLANK
16. $\overline{CSR}$
17. $\overline{RED}$ VIDEO OUT
18. $\overline{CSG}$
19. SYNC ENABLE
20. V-
21. $\overline{GRN}$ VIDEO OUT
22. $\overline{CSB}$
23. BLU VIDEO OUT
24. +5 V

intech
MICROCIRCUITS DIVISION

CUSTOM SERVICE IS OUR STANDARD

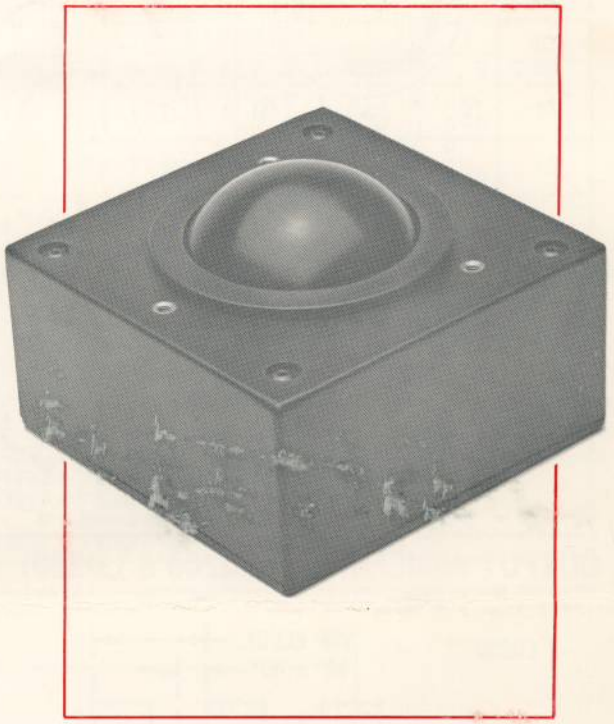
2270 MARTIN AVENUE, SANTA CLARA, CALIFORNIA 95050-2781
TELEPHONE (408) 988-4930 TWX 910-338-2213

Disc Instruments
A Honeywell SUBSIDIARY

200 SERIES

LOW COST 2" TRACKBALL

DISC'S 200 SERIES Trackballs are the low cost answer to enhance computer or graphics terminal performance by providing accurate X, Y cursor positioning on a CRT screen. Replacing conventional joysticks and their A to D conversion electronics to achieve digital outputs, the LD200 and the LQ200 deliver direct digital outputs configured as direction decoded pulses or quadrature square wave that are CMOS/TTL compatible. The LS200 version in the series provides for 3 optional external switch inputs and delivers a serial binary output with RS232C or CMOS/TTL interface. Sized to be keyboard or panel mounted, or used for standalone and remote applications, DISC'S 200 SERIES Trackballs are ideal for business and personal computers, CAD/CAM, video games, graphic terminals, radar monitor systems or anywhere man/machine X, Y interface is required. With a 2" ball, a mere 3.25" square and a slim 1.5" depth, the 200 is a space saving alternative to the mouse or bit pad. Drawing on more than 15 years of optical encoder manufacturing experience, DISC offers trackballs of incomparable performance and features.



SPECIFICATIONS

ELECTRICAL

Interface:	
LD or LQ200	CMOS/TTL
LS200	RS232C or CMOS/TTL
Resolution (PPR)	192 +/−5%
Output Signal:	
LQ200	Quadrature Square Waves
LD200	5μsec+/−30% Pulses, Direction Decoded
LS200	Serial Binary (See Format Description)
Input Power:	
RS232C	+5VDC @ 100 ma. max. and +12VDC @ 30 ma. max. and −12VDC @ 30 ma. max.
CMOS/TTL	+5VDC @ 100 ma. max.
Output Driver:	
RS232C	SN75150
CMOS/TTL	"1" 3.0V min. @ 3 ma. source, "0" 0.4V max. @ 3 ma. sink
Connector (Socket)	8 Pin telephone modular, locking female
Mating Plug	8 Pin telephone modular, locking male (not furnished with unit)

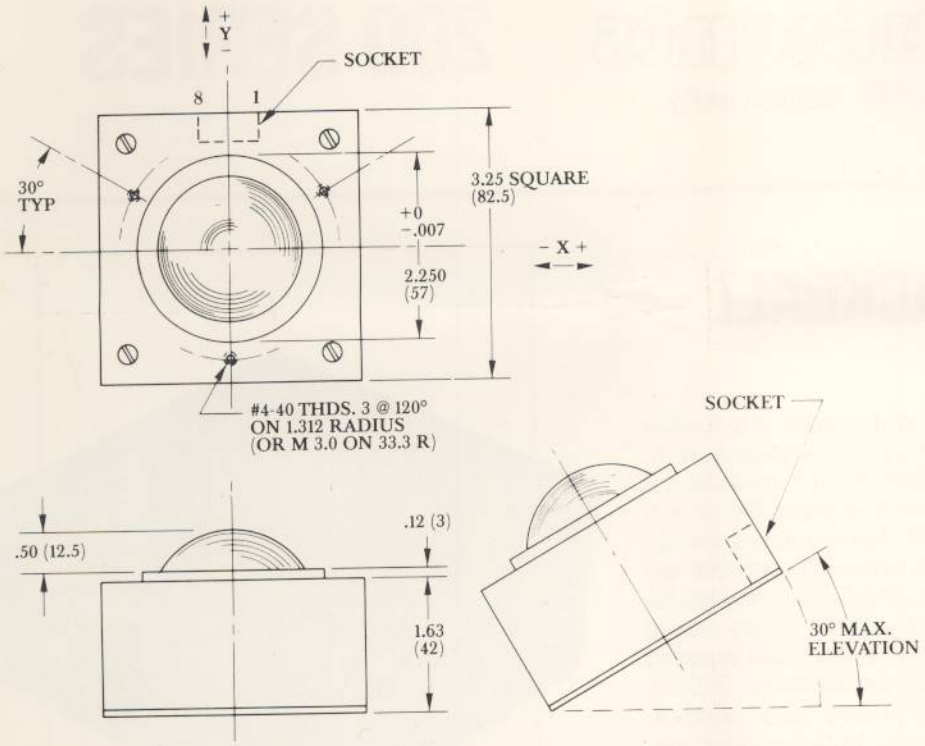
MECHANICAL

Weight	8.0 oz. max. (230 gm)
Ball	Thermoset Phenolic, 2" dia. black (51mm)
Tracking Force	1.5 oz. max., 0.7 oz typ. (20gm)
Ball Load	100 lbs. max. downward (45 Kg)
Ball Rotation	Continuous & reversible any direction
Ball Speed	200 RPM maximum
Housing Material	ABS (Cycolac KJT) Black
Transducers	Photo-optical Encoders with LEDs
Mounting Position	Horizontal to 30 degrees rear elevation

ENVIRONMENTAL

Temperature:	
Storage	−25 degrees C to +60 degrees C
Operating	+5 degrees C to +50 degrees C

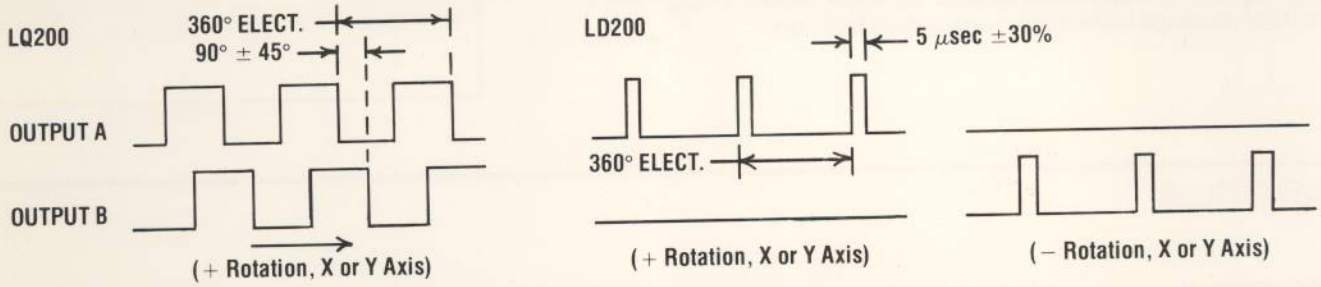
OUTLINE DRAWING AND CONNECTIONS



PIN	LQ200	LD200
1	Y_B	−Y
2	Y_A	+Y
3	+5VDC	+5VDC
4	N/C	N/C
5	X_B	−X
6	X_A	+X
7	GROUND	GROUND
8	GROUND	GROUND

PIN	LS200	
	RS232C	CMOS/TTL
1	S1	S1
2	S2	S2
3	+5VDC	+5VDC
4	+12VDC	N/C
5	−12VDC	N/C
6	OUTPUT	OUTPUT
7	S3	S3
8	GROUND	GROUND

OUTPUT WAVEFORMS (LQ200 & LD200)



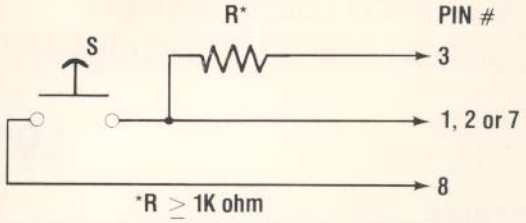
OUTPUT FORMAT DESCRIPTION (LS200)

The LS200 is the serial output version of the 200 SERIES trackball incorporating a custom IC chip that converts conventional incremental quadrature encoder inputs to a single serial binary data stream. Pulses from each axis encoder are counted during a period which is approximately equivalent to the reciprocal of the data block rate. The resultant X and Y counts and (if used) the switch data are loaded into buffer registers during the first output start pulse. The registers then reset to zero and counting is immediately resumed; no pulse goes uncounted.

The output formats available are shown in the timing diagram examples for the 2 word and 3 word options. Blocks of data are transmitted only when an encoder count pulse occurs or a switch input transitions from high to low. When there is no input data change to the serial chip following the last transition there is no output signal and the line remains at logic "1". The X and Y change of position count is a two's complement binary number appearing in the first and second 8 bit words. Word length is always 8 bits but the format differs for 2 word versus 3 word. Each count word consists of either 6 or 5 bits of count followed by 1 bit of direction and then 1 or 2 bits of data indentification respectively. The 3 word format must be used if there are switch inputs. The switch word consists of 3 bits of switch status followed by 3 logic "0" bits and the 2 bits identifying the word as switch data.

The serial chip has a 1.8 MHz clock from which several baud rates can be derived with an accuracy of +/−0.1%. The data block length may be either 128 or 256 bits and is a parameter usually determined by the user's system requirements. Baud rate, count length and data block length are customer specified and set at the factory. The output interface can be either RS232C with negative logic ("1" being −6 volts and "0" being +6 volts) or direct CMOS/TTL in +5 volt positive logic.

EXTERNAL SWITCH CONNECTION AND DATA CHARTS (LS200)



DATA RATE CHART

BAUD RATE	7 BIT COUNT LENGTH		6 BIT COUNT LENGTH	
	DATA BLOCK LENGTH IN BITS			
	128	256	128	256
19200	9300 150	4650 75	4650 150	2325 75
9600	4650 75	2325 37.5	2325 75	1162 37.5
4800	2325 37.5	1162 18.75	1162 37.5	581 18.75
2400	1162 18.75	581 9.38	581 18.75	290 9.38
1200	581 9.38	290 4.69	290 9.38	145 4.69
600	290 4.69	145 2.34	145 4.69	72 2.34
300	145 2.34	72 1.17	72 2.34	36 1.17

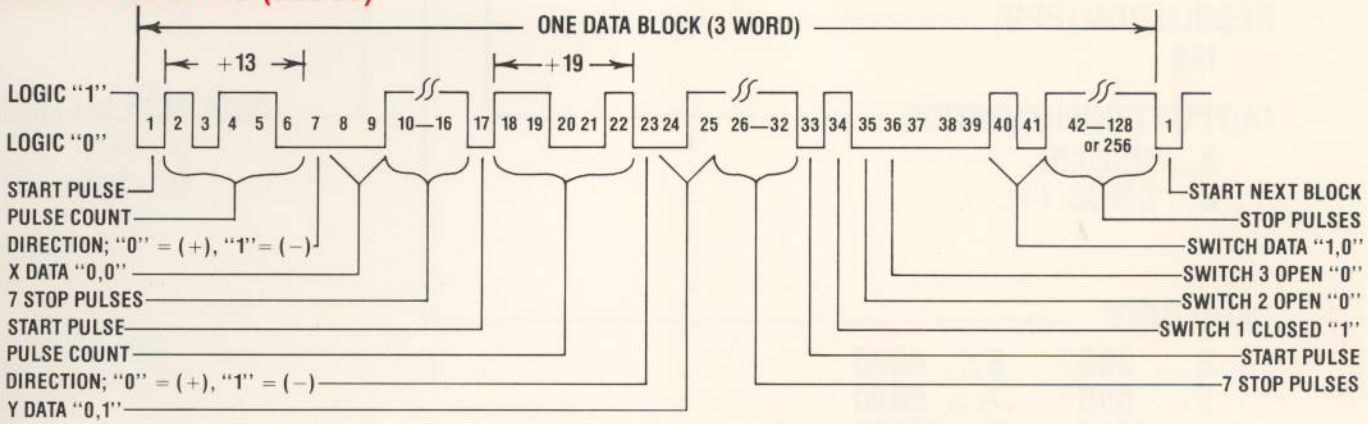
MAXIMUM COUNT RATE
MAXIMUM DATA BLOCK RATE

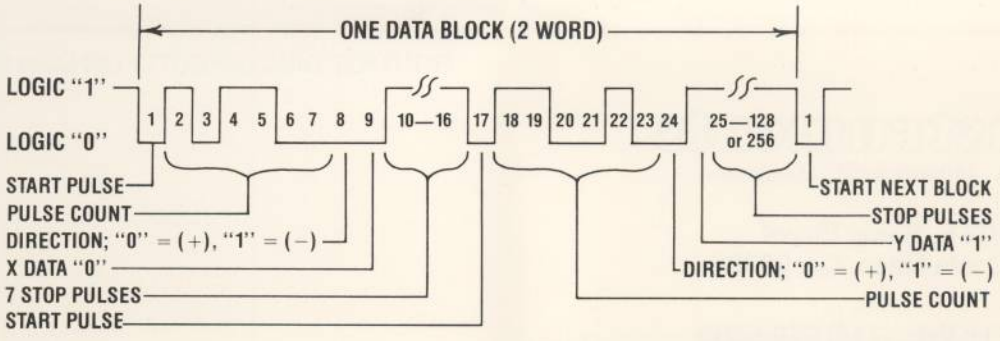
DATA BLOCK FORMAT CHART

BYTE NO.	NO. OF WORDS	BIT NUMBER 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	BIT COUNT
1	2	1	1	1	1	1	1	1	0	D	X5	X4	X3	X2	X1	X0	0	16
1	3	1	1	1	1	1	1	1	0	0	D	X4	X3	X2	X1	X0	0	16
2	2	1	1	1	1	1	1	1	1	D	Y5	Y4	Y3	Y2	Y1	Y0	0	32
2	3	1	1	1	1	1	1	1	1	0	D	Y4	Y3	Y2	Y1	Y0	0	32
3	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	48
3	3	1	1	1	1	1	1	1	0	1	0	0	0	S3	S2	S1	0	48
4-8	2 or 3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	128
9-16	2 or 3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	256

X0 To X5 = X Count Data, 2's Complement Binary
Y0 To Y5 = Y Count Data, 2's Complement Binary
D = Direction; 0 for (+), 1 for (−)
S1 to S3 = Switch Data; 0 for Open, 1 for Closed

TIMING DIAGRAMS (LS200)





(BOTH DIAGRAMS SHOWN AS +LOGIC CMOS/TTL)

MODEL NUMBER DESIGNATION
LD200 and LQ200 (INCREMENTAL)

LD200 ___ — 192 — 5

SERIES
LD200 (Direction Decoded)
LQ200 (Quadrature)

OPTIONAL MOUNTING
BLANK...Standard, #4-40
M...Metric, 3.0 mm

RESOLUTION (PPR)
192

INPUT POWER
5...+5VDC

LS200 (SERIAL BINARY)

LS200 ___ — 192 — A 4 B 6

SERIES
LS200

OPTIONAL MOUNTING
BLANK...Standard, #4-40
M...Metric, 3.0 mm

RESOLUTION (PPR)
192

OUTPUT CONFIGURATION
A...RS232C
B...CMOS/TTL

BAUD RATE
2... 300
3... 600
4...1200
5...2400
6... 4800
7... 9600
8...19200

COUNT LENGTH
6...Switch Data
7...No Switch Data

DATA BLOCK LENGTH
A...128
B...256

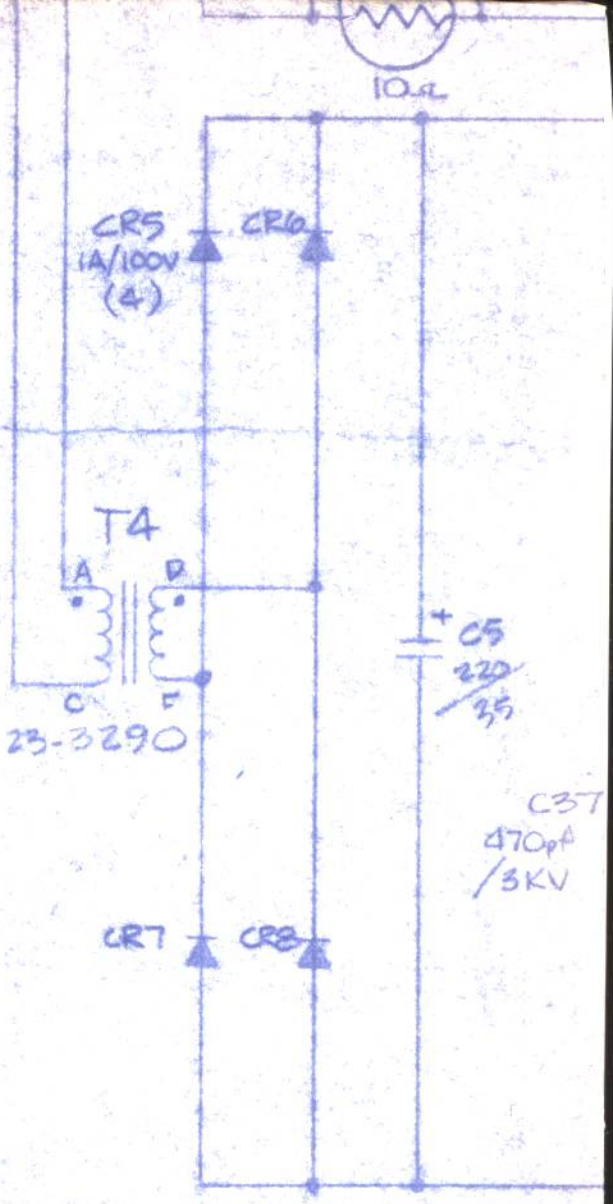
CONTACT DISC DIRECTLY OR OUR REPRESENTATIVE

Disc Instruments
A Honeywell SUBSIDIARY

102 E. Baker Street
Costa Mesa, Ca. 92626

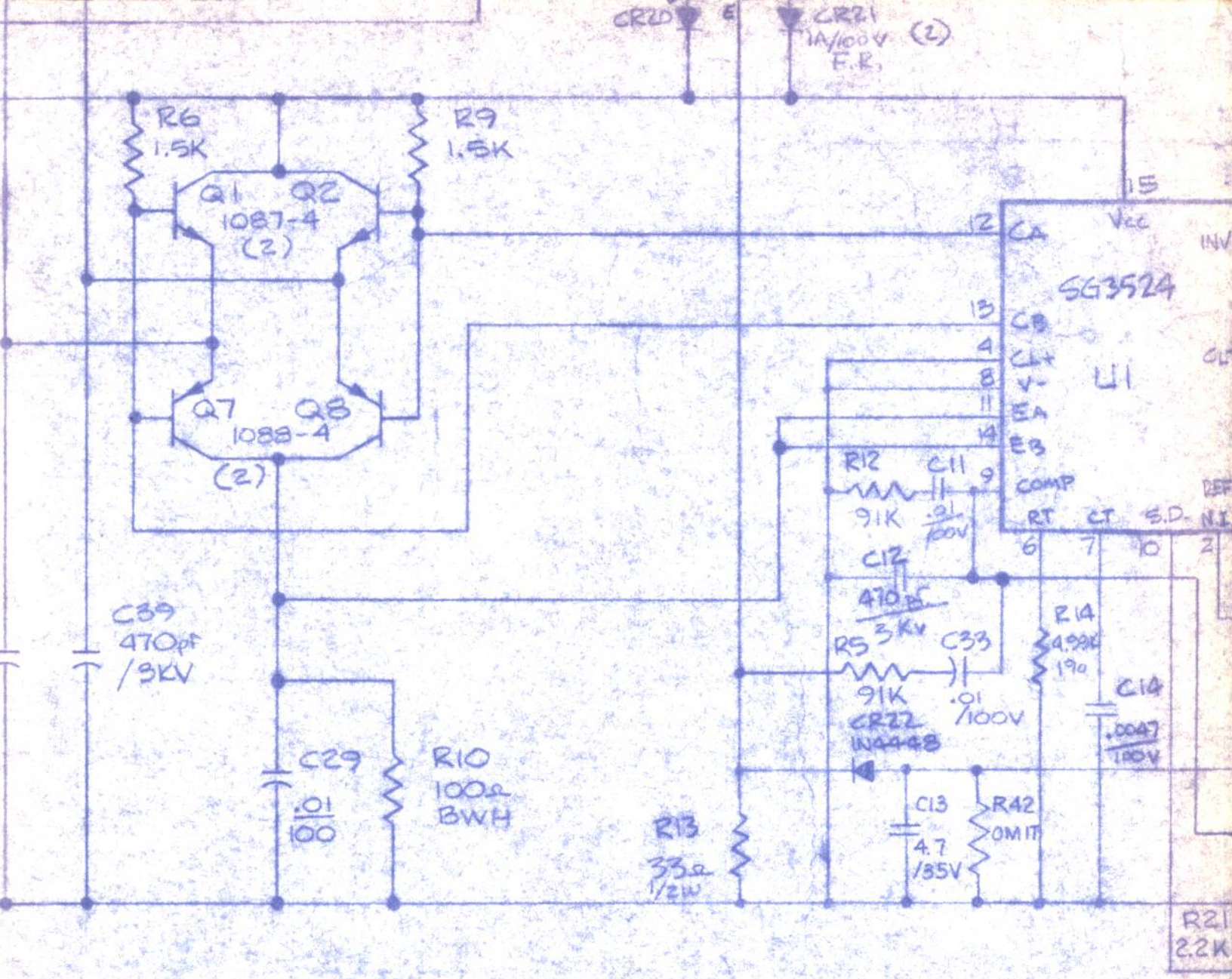
PHONE: (714) 979-5300
TWX: 910-595-1987 DISC CSMA

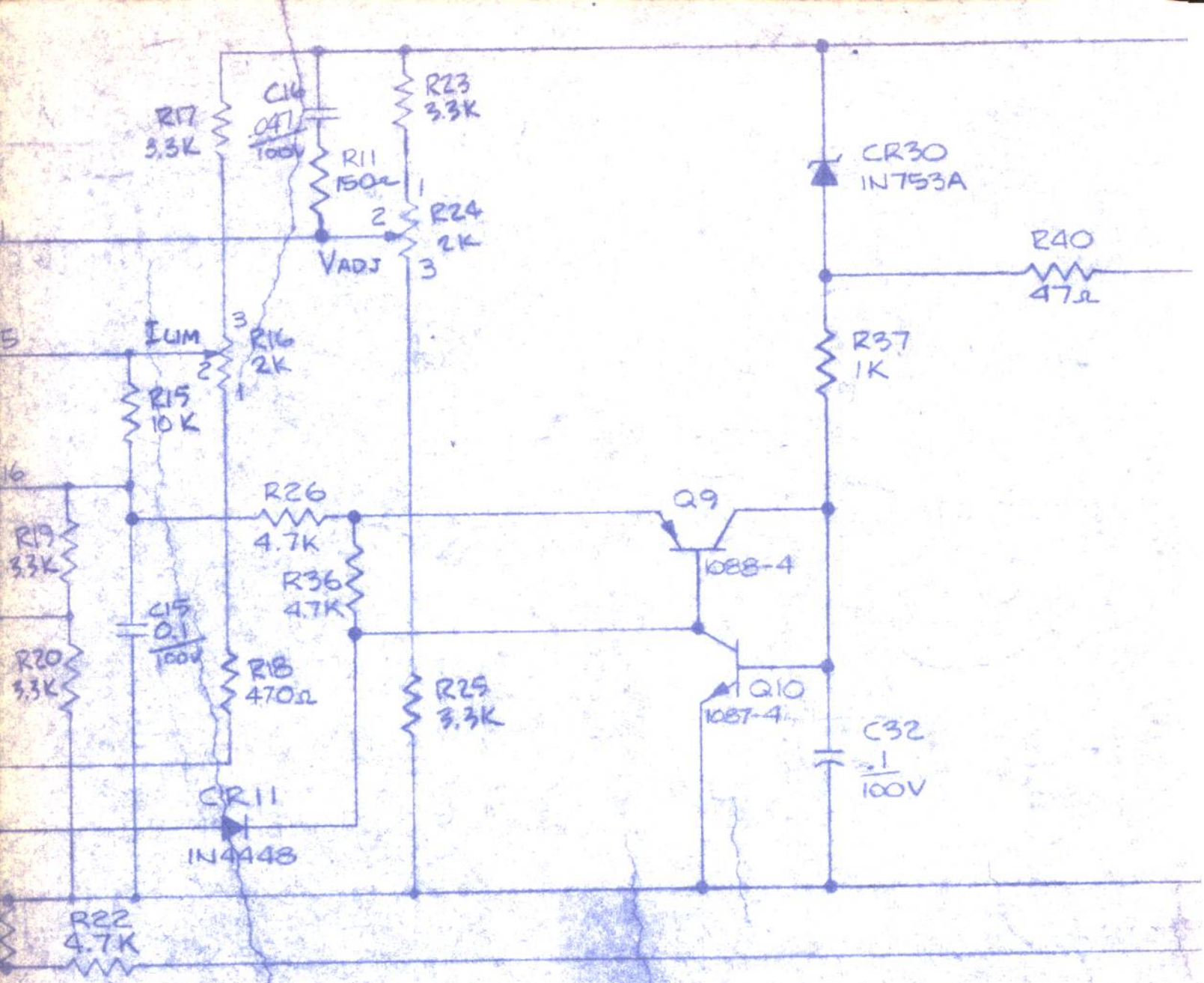
INPUT 7+8
NO JUMPER REQ.



1) JUMPERS W1 THRU W12 ARE ALL REQUIRED ON P.C. BD (NOT SHOWN ON SCHEMATIC)

NOTE: UNLESS OTHERWISE SPECIFIED


CR20
CR21
1A/100V (2)
F.R.
R6
1.5K
R9
1.5K
Q1 Q2
1087-4
(2)
Q7 Q8
1088-4
(2)
15
Vcc
12
CA
INV
SG3524
13
CB
4
CL+
8
V-
U1
11
EA
14
EB
R12
C11
9
COMP
91K
.01
100V
RT
CT
S.D.
REF
N.I.
6
7
10
2
C12
470pf
3KV
C39
470pf
/3KV
R5
C33
R14
4.99K
1%
91K
.01
/100V
CR22
IN4448
C14
.0047
100V
C29
.01
100
R10
100Ω
BWH
R13
33Ω
1/2W
C13
4.7
/35V
R42
OMIT
R21
2.2K



QTY REQD	PART NO. OR TYPE DESIGNATION	SYM

TB1	
U5	
T4	
R44 ET2	R12
C10	
L3	
F1	
C231	
C29	
LAST USED	NOT USED
REF DES	

UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES ON:
DECIMALS X ±.060
ANGLES XX ±.030
XXX ±.010
DIM AND TOL PER MIL-STD-8
MACHINED DIA'S ON COMMON CENTER CONCENTRIC WITHIN .005 T.I.R.
MACHINED SURFACE ROUGHNESS PER MIL-STD-10
REMOVE BURRS, BREAK EDGES .010 MAX
DO NOT SCALE DRAWING
REPORT ERRORS TO ENGINEERING DEPT
MAKE CHANGES WITHOUT PROPER AUTHORIZATION

NEXT ASSY	FINAL ASSY

MATERIAL

FINISH

L3
FERRITE BEAD

CONFIDENTIAL

INHIBIT

NOMENCLATURE OR DESCRIPTION	CODE IDENT	MATERIAL, NOTES AND SPECIFICATIONS	ITEM NO

LIST OF MATERIALS

CONTRACT NO.

ALPHA POWER INC
CANOGA PARK, CA

APPD	K/L	4/6/81
CHECK		
DRAWN	K HOOKE	3/2/81

TITLE
SCHEMATIC -
5BXMP-A

APPROVED

APPROVED

SIZE	CODE IDENT NO.	DRAWING NUMBER	REV
D		00-3105	E

SCALE

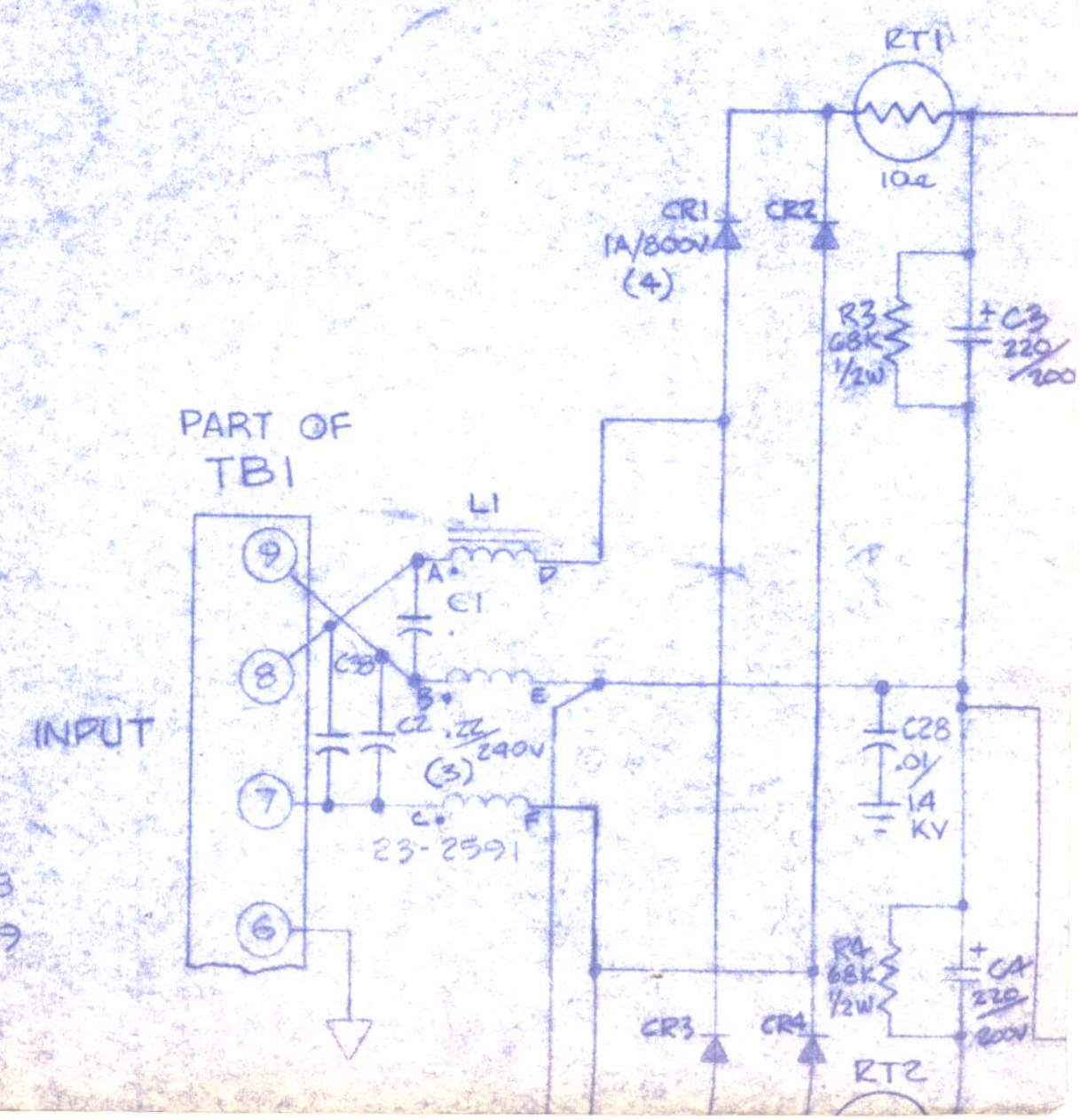
SHEET 1 OF 1

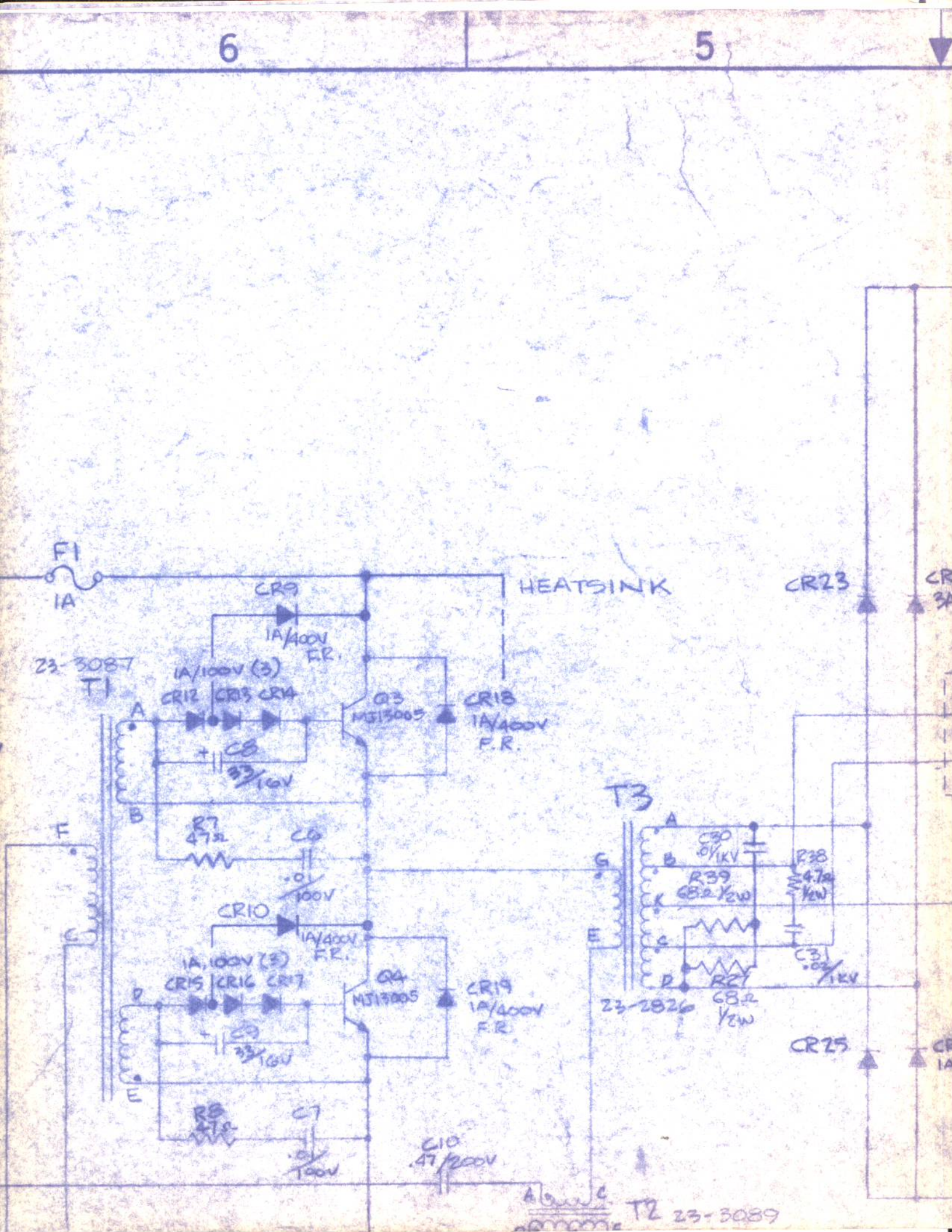
B

A

D

C




F1
1A
HEATSINK
CR9
1A/400V F.R.
Z3-3087
T1
1A/100V (3)
CR12 CR13 CR14
Q3 MJ13005
CR18 1A/400V F.R.
C8 33/16V
R7 47Ω
C6 .01/100V
CR10
1A/400V F.R.
1A, 100V (3)
CR15 CR16 CR17
Q4 MJ13005
CR19 1A/400V F.R.
C9 33/16V
R8 47Ω
C7 .01/100V
C10 .47/200V
T3
C30 .01/1KV
R38 4.7Ω 1/2W
R39 68Ω 1/2W
C31 .02/1KV
R27 68Ω 1/2W
Z3-2826
CR23
CR25
T2 Z3-3089

E	SEE ECO C1677	10-29-82
F	SEE ECO C2095	
G	SEE ECO C2328	

23-2827

L2A

H G

R30
0.22 BWH

Q5
SUES86A

Q6
1075-1

R29
1K

2N3905

R31
47Ω

U3
723

3 Cs
10 Vo
11 INV
4
12 V+
6 Ref
2 CL
5 NI
V- COMP
7
13

R34
1.00K
1%

R35
1.47K
1%

R32
560Ω

R33
10K

C18
220/35V

C21
.0047/100V

CR29
1A/100V

R24
A/100V F.R.
(2)

U2

L2B

C
A
D
B

R711X

R41
220Ω

CR31

R28
68Ω
1/2W

C19
1000/16V

C20
1000/16V

C24
33/16V

R43
1K

7905

U4
-5V/1A

1
2
3

CR27
1A/100V

C17
220/35V

R26
A/100V F.R.
(2)

R44
2.2K

C23
33/16V

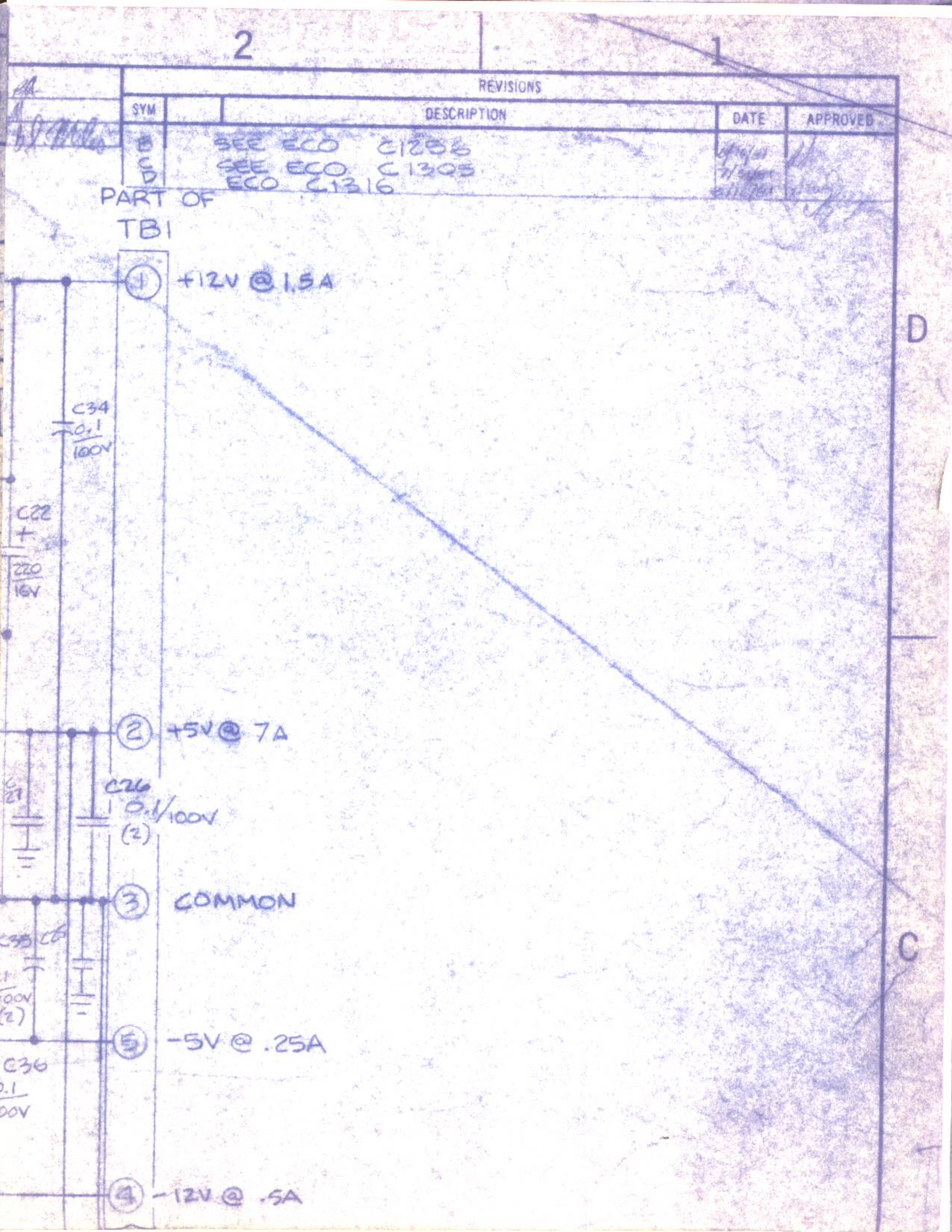
L2C

E F

7912

U5
-12V/1A

CR28
1A/100V


2
1
REVISIONS
SYM
DESCRIPTION
DATE
APPROVED
B
SEE ECO C1256
C
SEE ECO C1305
D
ECO C1316
PART OF
TB1
1
+12V @ 1.5A
C34
0.1
100V
C22
+
220
16V
2
+5V @ 7A
C26
0.1/100V
(2)
3
COMMON
C35
C6
5
-5V @ .25A
C36
4
-12V @ .5A
D
C

```
.title  display system / color graphics controller

;*******************************************************
;*                                                     *
;*              DISPLAY SYSTEM /                       *
;*              COLOR GRAPHICS CONTROLLER FIRMWARE     *
;*                                                     *
;*              WRITTEN BY                             *
;*                      ALAN R. BALDWIN                *
;*                      OTSELIC SPECIALTIES            *
;*                      721 BERKELEY                   *
;*                      KENT, OHIO  44240              *
;*                                                     *
;*              FOR                                    *
;*                      KENT STATE UNIVERSITY          *
;*                      DEPARTMENT OF PHYSICS          *
;*                      KENT, OHIO  44242              *
;*                                                     *
;*                                                     *
;*              COMPLETED                              *
;*      V01        MARCH   1985                        *
;*                                                     *
;*              UPDATES                                *
;*      V01.01     MAY     1985   ($Q,$KR,$AFR ADDED)  *
;*                 JUNE    1985   (FIX $B)             *
;*                 MARCH   1986   (FIX ROM ALLOCATION) *
;*      V01.02     SEP     1986   (ADDED LASERJET)     *
;*                                ($TS PRINTER SELECT) *
;*                                                     *
;*      V02.00     OCT     1986   (ADD ELLIPTIC)       *
;*                                (FUNCTIONS)          *
;*                                (AND LOADABLE)       *
;*                                (PRINTER DRIVER)     *
;*      V02.01     SEP     1987   (REMOVE 8'TH BIT)    *
;*                                (STRIPPING AND)      *
;*                                (CHECKING.)          *
;*                                (XON/XOFF DEFAULTS)  *
;*                                (TO DISABLED.)       *
;*                                ($E SCAN SETUP)      *
;*                                                     *
;*      V02.02     NOV     1988   ($AL AND $AR CHANGED)*
;*                                (TO BINARY MODE AND  *
;*                                (SCANNING CHANGED.)  *
;*                                (ADDED USER FUNCTION)*
;*                                (CALLS $0 - $9.)     *
;*                                ($S1, $S9 FUNCTION)  *
;*                                (AS A BINARY LOADER) *
;*                                                     *
;*******************************************************
.page
;*******************************************************
;*                                                     *
;*                                                     *
;*        THIS PROGRAM IS WRITTEN FOR A 6809           *
;*      MICROPROCESSOR WITH THE FOLLOWING              *
;*      PERIPHERAL DEVICES:                            *
;*                                                     *
;*              1. a)   512 X 480 VIDEO DISPLAY        *
;*                                                     *
;*                 b)   512 X 512 VIDEO DISPLAY        *
;*                                                     *
;*                                                     *
;*              2.      MOTOROLA 6845 GRAPHICS DISPLAY *
;*                      CONTROLLER WITH 4 BIT PLANES   *
;*                      FOR 16 COLORS.  4-BIT DAC'S    *
;*                      ARE COLOR MAPPED TO ALLOW      *
;*                      SELECTION OF 16 OF 4096        *
;*                      COLORS.                        *
;*                                                     *
;*                                                     *
;*              3.      PARALLEL CONTROL PORT          *
;*                      EMULATING A                    *
;*                      DIGITAL EQUIPMENT CORPORATION  *
;*                      DL(V)11 SERIAL LINE.           *
;*                                                     *
;*                                                     *
;*              4.      PARALLEL INTERFACE             *
;*                      FOR AN EPSON FX-80/100         *
```

```
        ;*                      GRAPHICS PRINTER,               *
        ;*                      A TEKTRONIX 4695                *
        ;*                      COLOR GRAPHICS PRINTER,         *
        ;*                      OR A HEWLETT-PACKARD            *
        ;*                      LASERJET(+) PRINTER.            *
        ;*                                                      *
        ;*                                                      *
        ;*              5.      2 12-BIT DAC'S FOR AN           *
        ;*                      ANALOG X-Y PLOTTER              *
        ;*                                                      *
        ;*                                                      *
        ;*              6.      A 'DISPLAY SYSTEM'              *
        ;*                      CONTROL PANEL WITH LED'S,       *
        ;*                      TOGGLE, ROTARY, AND PUSHBUTTON  *
        ;*                      SWITCHES, TWO 6-DIGIT           *
        ;*                      THUMBWHEEL SWITCHES, AND        *
        ;*                      A TRACK-BALL DEVICE.            *
        ;*                                                      *
        ;*                                                      *
        ;*        THE SYSTEM ACCEPTS SIMPLE ASCII COMMANDS      *
        ;*      TO PERFORM POINT PLOTTING, VECTOR DRAWING,      *
        ;*      CHARACTER PLOTTING, AND PEN CONTROL.            *
        ;*                                                      *
        ;******************************************************
        .page
        .sbttl  assembly options

        hplaser =       1       ;assemble with 'laserjet(+)' printer option
        tektronix =     1       ;assemble with 'tektronix' copier option
        epson   =       1       ;assemble with 'epson' printer option
        plotter =       1       ;assemble with analog x-y plotter option

                                ;printer option included
        printer =       hplaser | tektronix | epson


        .page
        .sbttl  system definitions

        .module dspcgc
        .radix  o

;  6809 data and program space

        .area   RAM

$hstor:                         ;display system histo area
                                ;16384 bytes of histogram space

$macro: .blkb   34000           ;macro area (overlays histo area)
        $macnd  =       .       ;14k bytes

        ;       macro buffer variables
$macbf:
q$pchr: .byte   0               ;previous $q processed character
q$end:  .byte   0,0             ;end of macro buffer
q$pntr: .byte   0,0             ;macro pointer
q$cntr: .byte   0,0             ;macro character counter
q$save: .blkb   0d64            ;save status area
q$tble: .blkb   4*0d256         ;macro pointer
q$tend:                         ;and length (4-bytes)
        .blkb   4000-(.-$macbf) ;2k bytes macro tables

$ocbf0: .blkb   4000            ;display system 'oc' display buffer
$ocbf1: .blkb   4000            ;2048 bytes per buffer

af$bgn: .blkb   4000            ;area fill buffer area
        af$end  =       .       ;end of buffer area
                                ;2048 bytes

chram:  .blkb   4000            ;writable character set
                                ;2048 bytes

        optfun  =       .
$xtrn0: .blkb   4               ;4096 bytes allocated
$xtrn1: .blkb   4               ;for optional functions
$xtrn2: .blkb   4
$xtrn3: .blkb   4
```

```
$xtrn4: .blkb   4
$xtrn5: .blkb   4
$xtrn6: .blkb   4
$xtrn7: .blkb   4
$xtrn8: .blkb   4
$xtrn9: .blkb   4

$xtrnl: .blkb   4               ;loadable printer driver $t_ entry
$xscrn: .blkb   4               ;loadable printer driver screen entry
        .blkb   0d4096-(.-optfun)
        $xtend  =       .       ;end of user area

        sclen   =       0d2048
scbuf:  .blkb   sclen           ;scan buffer area
        scmpty  =       0d256
        scfull  =       sclen-0d64
        scvful  =       sclen-0d32
        scend   =       .       ;end of buffer

        dllen   =       0d256
dlbuf:  .blkb   dllen           ;host link character buffer
        dlmpty  =       0d32
        dlfull  =       0d224
        dlend   =       .       ;end of buffer

        .page

        .area   VARSAV

         .setdp 0               ;256 bytes
        spjump  = .+0d254       ;special diagnostic variable

        direct  =       74400   ;direct variable space
         varsav ==      direct

        .area   DPVSAV

        dpstat  =       75000   ;display system status area
         dpvsav ==      dpstat  ;512 bytes

        .area   BUFSAV

        bufsav  ==      76000   ;buffers area
                                ;1024 bytes

extbypls:       .blkb   2       ;loadable system vectors
extbymns:       .blkb   2
extbxpls:       .blkb   2
extbxmns:       .blkb   2
exundfnd:       .blkb   2
exdltint:       .blkb   2
exdlrint:       .blkb   2
exclocki:       .blkb   2

exrsrv:         .blkb   2
exswi3:         .blkb   2
exswi2:         .blkb   2
exfirq:         .blkb   2
exirq:          .blkb   2
exswi:          .blkb   2
exnmi:          .blkb   2

        stack   =       100000  ;top of ram area

        $intrnl =       100000  ;internal i/o space

        $panel  =       104000  ;optional control panel

        $btmap  =       110000  ;display bit map address

        chrom1  ==      120000  ;character tables (8k)
        chrom2  ==      124000
        chrom3  ==      130000
        chrom4  ==      134000

        .area   VECTOR          ;vector characters
chksm0: .byte   0,0             ;for 8k rom
```

```
        .area   R6571A          ;r6571a character rom

        .area   VT1XX           ;vt1xx character rom

        .area   FIGROM          ;figrom character set

        .area   PGMSAV

        pgmsav  ==      140000  ;basic program space (8k)

        .area   MC6845

        .area   EXTSAV

        extsav  ==      160000  ;extension program space (8k)


        .page
        .sbttl  interrupt vector definitions

        .area   IRQINT

        irqint  ==      177510  ;interrupt vector space

irqtbl: .word   tbypls, extbypls        ;table of initial vectors
        .word   tbymns, extbymns
        .word   tbxpls, extbxpls
        .word   tbxmns, extbxmns
        .word   undfnd, exundfnd
        .word   dlrint, exdltint        ;dltint & dlrint
        .word   dlrint, exdlrint        ;interrupts serviced together
        .word   clocki, exclocki

        .word   undfnd, exrsrv
        .word   undfnd, exswi3
        .word   undfnd, exswi2
        .word   undfnd, exfirq
        .word   undfnd, exirq
        .word   undfnd, exswi
        .word   $nmi,   exnmi
        .word   0,      0

1$:     jmp     [extbypls]
2$:     jmp     [extbymns]
3$:     jmp     [extbxpls]
4$:     jmp     [extbxmns]
5$:     jmp     [exundfnd]
6$:     jmp     [exdltint]
7$:     jmp     [exdlrint]
8$:     jmp     [exclocki]

9$:     jmp     [exrsrv]
10$:    jmp     [exswi3]
11$:    jmp     [exswi2]
12$:    jmp     [exfirq]
13$:    jmp     [exirq]
14$:    jmp     [exswi]
15$:    jmp     [exnmi]
        jmp     [exrsrv]

                ;6828 vectored interrupts
        .word   1$      ;priority 0
        .word   2$      ;priority 1
        .word   3$      ;priority 2
        .word   4$      ;priority 3
        .word   5$      ;priority 4
        .word   6$      ;priority 5
        .word   7$      ;priority 6
        .word   8$      ;priority 7
        .word   13$     ;(unused)       checksum requirement

        .blkb   0o26

        .word   9$      ;reserved       (unused)
        .word   10$     ;swi3
        .word   11$     ;swi2
        .word   12$     ;firq vector
        .word   13$     ;irq  vector
```

```
        .word   14$     ;swi
        .word   15$     ;nmi
        .word   pwrup   ;power up entry point

        .page
        .sbttl  copyright notice

        .area   PGMSAV

chksm1: .byte   0,0     ;for 8k rom
        dsp$cr  =       .
        .byte   15,12
        .ascii  /DSPCGC  0001  V02.02/
        .byte   15,12
        dsp$id  =       .-dsp$cr        ;length of version
        .ascii  /COPYRIGHT 1985,1986,1987,1988/
        .byte   15,12
        .ascii  /OTSELIC SPECIALTIES/
        .byte   15,12
        .ascii  /721 BERKELEY/
        .byte   15,12
        .ascii  /KENT, OHIO  44240/
        .byte   15,12
        dsp$cr  =       .-dsp$cr        ;length of notice


undfnd: rti             ;non-existent routines


        .page
        .sbttl  rgb color table
        ;       r       g       b
        ;       -       -       -
clrtbl: .byte   0,      0,      0       ;black
        .byte   17,     0,      0       ;red
        .byte   0,      17,     0       ;green
        .byte   17,     17,     0       ;yellow
        .byte   0,      0,      17      ;blue
        .byte   17,     0,      17      ;magenta
        .byte   0,      17,     17      ;cyan
        .byte   12,     12,     12      ;white
                                        ;
        .byte   6,      6,      6       ;gray
        .byte   12,     5,      5       ;red      (pastels)
        .byte   5,      12,     5       ;green        "
        .byte   12,     12,     5       ;yellow       "
        .byte   5,      5,      12      ;blue         "
        .byte   12,     5,      12      ;magenta      "
        .byte   5,      12,     12      ;cyan         "
        .byte   17,     17,     17      ;very white

        .page
        .sbttl  rom checksum verification

        ;  enter via    ldu     #1$
        ;               jmp     verrom
        ;       1$:     .word   start address
        ;               .word   last address+1
        ;               .word   checksum location
        ;               .byte   error byte
        ;  returns here ---

verrom: ldx     ,u++            ;start address
        clra                    ;init sum
        clrb
1$:     addb    ,x+             ;build sum
        adca    #0
        cmpx    ,u              ;finished ?
        bne     1$              ;loop until all summed
        leau    2,u
        ldx     ,u
        subb    ,x              ;take out checksum itself
        sbca    #0
        subb    1,x
        sbca    #0
        addd    ,x              ;this should give '0'
        bne     verror          ;no - have an error
        jmp     3,u             ;return
```

```
        .page
        .sbttl  ram verify routine

        ;  enter via    ldu     #1$
        ;               jmp     verram
        ;       1$:     .word   start address
        ;               .word   last address+2
        ;               .byte   error byte
        ;  returns here ---
        ;
        ;  this routine is for memory organized as
        ;  16-bit words on an eight bit bus.  every other
        ;  byte is in the same ram.

verram: ldx     ,u++            ;first location
        clra
1$:     sta     ,x++            ;incrementing pattern
        inca
        bne     2$
        inca
2$:     cmpx    ,u              ;finished settup ?
        bne     1$              ;no - loop
        ldx     -2,u            ;first location
        clra
3$:     cmpa    ,x++
        bne     verror          ;bad - error
        inca
        bne     4$
        inca
4$:     cmpx    ,u              ;finished checking ?
        bne     3$              ;no - loop

        ldx     -2,u            ;first address
        clra
5$:     coma
        sta     ,x++            ;complement of incrementing pattern
        coma
        inca
        bne     6$
        inca
6$:     cmpx    ,u              ;finished settup ?
        bne     5$              ;no - loop
        ldx     -2,u            ;first location
        clra
7$:     coma
        cmpa    ,x++
        bne     verror          ;bad - error
        coma
        inca
        bne     8$
        inca
8$:     cmpx    ,u              ;finished checking ?
        bne     7$              ;no - loop
        jmp     3,u             ;finished

        .page
        .sbttl  verify error routine

verror: std     $hstor
        ldb     #0d10           ;10. flashes
1$:     lda     2,u             ;error code
        sta     plout
        ldx     #-0d25000       ;.1 seconds @ 2mHZ
2$:     leax    1,x
        bne     2$
        lda     #370
        sta     plout
        ldx     #-0d25000
3$:     leax    1,x
        bne     3$
        decb
        bgt     1$              ;loop for 10. flashes
        jmp     3,u             ;and return

        .page
;  host 'data link' registers
```

```
        ;         the dl(v)11 equivalent registers

        hrdata  =       100300  ;input data register            (ro)
                                ;data taken (clear irq)         (wo)
        hrcsr   =       100301  ;input control/status register  (r/w)
        htdata  =       100302  ;output data register           (wo)
        htcsr   =       100303  ;output control/status register (r/w)

        ;       control/status registers
        ;
        ;       7 -     port ready
        ;       6 -     interrupt enable bit
        ;
        ;
        ;         2 16-bit input registers
        ;
        inreg0  =       100304
        inreg1  =       100306


;  graphics registers

        ;         bit-plane control registers

        bp$clr  =       100100  ;clear selected bit plane (wo)
        bp$dat  =       100101  ;bit-plane intensity (wo)
        bp$wrt  =       100102  ;bit-plane write enable (wo)
        bp$dsp  =       100103  ;bit-plane display enable (wo)

        ;         the graphics system contains 2 16-bit write only
        ;       registers for the x and y position parameters.

        vidx    =       100400  ;16-bit x-position register (wo)
        vidy    =       100402  ;16-bit y-position register (wo)

        vidplt  =       $btmap  ;read bitmap data / write data in bp$dat

        ld.dac  =       100404  ;load x-y plotter dacs with data
                                ;from vidx & vidy registers (dumby)

        rgbmap  =       100700  ;rgb dac color map

;  clock register

        clkclr  =       100200


;  panel status register

        ;       panel status register is an 8-bit
        ;       read only register with bits assigned
        ;       as follows:

        ;       7-      reset(0)
        ;       6-      plotter off(0)/on(1)
        ;       5-      printer off(0)/on(1)
        ;       4-      graphics off(0)/on(1)
        ;       3-      video clear(0)/normal(1)
        ;       2-      calibrate (0)/normal(1)
        ;       1-      screen dump (0)/normal (1)
        ;       0-      hold  continue(0)/normal(1)

        plstat  =       100107

;  panel output register

        ;       panel output register is an 8-bit
        ;       write only register with the following
        ;       definitions

        ;       7-
        ;       6-      busy led (0)
        ;       5-      error led (0)
        ;       4-      hold led (0)
        ;       3-      printer strobe (0)
        ;       2-      pen up(0) / down(1) [plotter option]
        ;       1-      'oc' character set select
        ;       0-      'oc' buffer select
```

```
        plout   =       100406

;  printer port registers

        pntin   =       100405

        ;       printer & status register
        ;       read only register with bits assigned
        ;       as follows:

        ;       7-      printer busy (1)
        ;       6-      paper out (1)
        ;       5-      slct (1)
        ;       4-      error (0)
        ;       3-
        ;       2-
        ;       1-      plotter option installed (0)
        ;       0-


        pntout  =       100407

        ;       8-bit printer port output data


        .page
        .sbttl  6845 crtc settup tables

        a.6845  =       100500          ;mc6845 crtc

        .radix  d

        ;       table a - ntsc standard 525 line interlaced
        ;               (480 displayed), 15.70 khz / 59.81 hz
        ;       table b - non  standard 553 line interlaced
        ;               (512 displayed), 16.57 khz / 59.96 hz
        ;       table c - european standard 625 line interlaced
        ;               (512 displayed), 15.70 khz / 50.24 hz
        ;       table d - non  standard 533 line interlaced
        ;               (480 displayed), 15.98 khz / 59.96 hz
        ;       table e - non standard european, 627 line interlaced
        ;               (512 displayed), 15.70 khz / 50.08 hz
        ;       table f - non  standard 560 line noninterlaced
        ;               (512 displayed), 21.83 khz / 38.976 hz

        d$len   =       64              ;length of displayed line

        ;       6845 crtc settup table
        ;       clock frequency of 14.31818/8 mhz
        ;       horizontal frequency is 15.70 khz
        ;       vertical frequency is ( 15.70 / 262.5) = 59.81 hz
        ;       total scan lines per frame is 525
        ;       with 480 active lines displayed per frame
a.crtc: .byte   114-1           ;h total
        .byte   d$len           ;displayed characters
        .byte   82              ;sync position
        .byte   8               ;h sync width

        .byte   16-1            ;lines total
        .byte   6               ;adjustment
        .byte   15              ;displayed rows
        .byte   15              ;v sync position
        .byte   3               ;interlaced sync and video
        .byte   16-1            ;max scan line

        .byte   0               ;cursor / no blink
        .byte   10

        .word   2*64            ;top line is 480.
        .word   0               ;cursor not displayed

        ;       6845 crtc settup table
        ;       clock frequency of 14.31818/8 mhz
        ;       horizontal frequency is 16.57 khz
        ;       vertical frequency is (16.57 khz / 276.5) = 59.93 hz
        ;       total scan lines per frame is 553
        ;       with 512 active lines displayed per frame
```

```
b.crtc: .byte   108-1           ;h total
        .byte   d$len           ;displayed characters
        .byte   80              ;sync position
        .byte   8               ;h sync width

        .byte   17-1            ;lines total
        .byte   4               ;adjustment
        .byte   16              ;displayed rows
        .byte   16              ;v sync position
        .byte   3               ;interlaced sync and video
        .byte   16-1            ;max scan line

        .byte   0               ;cursor / no blink
        .byte   10

        .word   0               ;top line is 512.
        .word   0               ;cursor not displayed

        ;       6845 crtc settup table
        ;       clock frequency of 14.31818/8 mhz
        ;       horizontal frequency is 15.70 khz
        ;       vertical frequency is ( 15.70 / 312.5) = 50.24 hz
        ;       total scan lines per frame is 625
        ;       with 512 active lines displayed per frame
c.crtc: .byte   114-1           ;h total
        .byte   d$len           ;displayed characters
        .byte   82              ;sync position
        .byte   8               ;h sync width

        .byte   19-1            ;lines total
        .byte   8               ;adjustment
        .byte   16              ;displayed rows
        .byte   17              ;v sync position
        .byte   3               ;interlaced sync and video
        .byte   16-1            ;max scan line

        .byte   0               ;cursor / no blink
        .byte   10

        .word   0               ;top line is 512.
        .word   0               ;cursor not displayed

        ;       6845 crtc settup table
        ;       clock frequency of 14.31818/8 mhz
        ;       horizontal frequency is 15.98 khz
        ;       vertical frequency is ( 15.98 / 266.5) = 59.96 hz
        ;       total scan lines per frame is 533
        ;       with 480 active lines displayed per frame
d.crtc: .byte   112-1           ;h total
        .byte   d$len           ;displayed characters
        .byte   81              ;sync position
        .byte   8               ;h sync width

        .byte   16-1            ;lines total
        .byte   10              ;adjustment
        .byte   15              ;displayed rows
        .byte   15              ;v sync position
        .byte   3               ;interlaced sync and video
        .byte   16-1            ;max scan line

        .byte   0               ;cursor / no blink
        .byte   10

        .word   2*64            ;top line is 480.
        .word   0               ;cursor not displayed

        ;       6845 crtc settup table
        ;       clock frequency of 14.31818/8 mhz
        ;       horizontal frequency is 15.70 khz
        ;       vertical frequency is ( 15.70 / 313.5) = 50.08 hz
        ;       total scan lines per frame is 627
        ;       with 512 active lines displayed per frame
e.crtc: .byte   114-1           ;h total
        .byte   d$len           ;displayed characters
        .byte   82              ;sync position
        .byte   8               ;h sync width

        .byte   19-1            ;lines total
```

```
        .byte   9               ;adjustment
        .byte   16              ;displayed rows
        .byte   17              ;v sync position
        .byte   3               ;interlaced sync and video
        .byte   16-1            ;max scan line

        .byte   0               ;cursor / no blink
        .byte   10

        .word   0               ;top line is 512.
        .word   0               ;cursor not displayed

        ;       6845 crtc settup table
        ;       clock frequency of 14.31818/8 mhz
        ;       horizontal frequency is 21.83 khz
        ;       vertical frequency is ( 21.83 / 560 ) = 38.98 hz
        ;       total scan lines per frame is 560
        ;       with 512 active lines displayed per frame
f.crtc: .byte   82-1            ;h total
        .byte   d$len           ;displayed characters
        .byte   68              ;sync position
        .byte   8               ;h sync width

        .byte   35-1            ;lines total
        .byte   0               ;adjustment
        .byte   32              ;displayed rows
        .byte   32              ;v sync position
        .byte   0               ;interlaced sync and video
        .byte   16-1            ;max scan line

        .byte   0               ;cursor / no blink
        .byte   10

        .word   0               ;top line is 512.
        .word   0               ;cursor not displayed

        .page
        .sbttl  buffers and variables

        .radix  o

        .area   VARSAV

timflg: .byte   0               ;timing flag
bfstat: .byte   0               ;buffers status
        dl      =       1       ;bit 0 - data link
        trm     =       2       ;bit 1 - terminal (no hardware)
        scn     =       4       ;bit 2 - scan
statpl: .byte   0               ;previous panel status
outpl:  .byte   0               ;panel output status

        ;       host link buffer pointers

dlcntr: .byte   0               ;characters in buffer
dlqntr: .byte   0,0             ;character pointer for output
dlpntr: .byte   0,0             ;character pointer for input

        ;       scan buffer pointers

scntr:  .byte   0,0             ;characters in bufffer
scqntr: .byte   0,0             ;character pointer for output
scpntr: .byte   0,0             ;character pointer for input

        ;       list processor variables

lsbuff: .blkb   9               ;character buffer
lstcnt: .byte   0               ;chars in lsbuff
lpntr:  .byte   0,0             ;pointer to chars
lstat:  .byte   0               ;list processor status
                                ;       7- single character mode
                                ;       6- list scan
                                ;       5- build mode
                                ;       4- special single character mode
                                ;       3- sizing flag x(0)/y(1)
                                ;       2- lower/greek
                                ;       1- scan/plot
                                ;       0- horizontal/vertical
                                ;
```

```
lstatx: .byte   0               ;list processor status extension
                                ;       7- '0' slashed (0) / non-slashed (1)
                                ;       6- clipping (0) / not clipping (1)
                                ;       0- '0' plotter (0) / block (1)
                                ;
format: .byte   0               ;integer format length
gosub:  .byte   0,0             ;address of process
char:   .byte   0               ;current character
pchar:  .byte   0               ;previous character
schflg: .byte   0               ;seen character flag

        ;       evaln variables

ndigit  =       6
nsign:  .byte   0               ;sign of number
number: .blkb   ndigit          ;16-bit result / character buffer

        .page

        ;       graphics variables

xorg:   .byte   0,0             ;x origin
xstep:  .byte   0,0             ;x step size
xval:   .byte   0,0             ;x value
fxval:  .byte   0,0             ;final x value
vectlx: .byte   0,0             ;x vector length
valx:   .byte   0,0,0,0         ;running x value
fractx: .byte   0,0,0,0         ;fractional updating value
signx:  .byte   0               ;sign of vectlx
xsize:  .byte   0               ;x - character sizing
xcnt:   .byte   0               ;x-point position
pxcnt:  .byte   0               ;previous position
svxstp: .byte   0,0             ;save xstep
arcvlx: .byte   0,0             ;x arc value
cxzoom: .byte   0               ;block mode x-zoom
afracx: .byte   0,0,0,0         ;arc fractional update

yorg:   .byte   0,0             ;y origin
ystep:  .byte   0,0             ;y step size
yval:   .byte   0,0             ;y value
fyval:  .byte   0,0             ;final y value
vectly: .byte   0,0             ;y vector length
valy:   .byte   0,0,0,0         ;running y value
fracty: .byte   0,0,0,0         ;fractional updating value
signy:  .byte   0               ;sign of vectly
ysize:  .byte   0               ;y - character sizing
ycnt:   .byte   0               ;y-point position
pycnt:  .byte   0               ;previous position
svystp: .byte   0,0             ;save ystep
arcvly: .byte   0,0             ;y arc value
cyzoom: .byte   0               ;block mode y-zoom
afracy: .byte   0,0,0,0         ;arc fractional update

divcnt: .byte   0               ;division bit counter
dnd:    .byte   0,0             ;dividend
dsr:    .byte   0,0             ;divisor
qt:     .byte   0,0             ;quotient

vecntr: .byte   0,0             ;vector points
pntskp: .byte   0               ;points to skip
chpntr: .byte   0,0             ;character pointer
ptchar: .byte   0,0             ;address of point character table
romset: .byte   0,0             ;last selected rom character set
prow:   .byte   0               ;character dots
svgsub: .byte   0,0             ;gosub save
svstat: .byte   0               ;rstat1 save

        .page
        .sbttl  arc drawing variables

        ; may be used by any $a routine

arcbgn: .byte   0,0             ;arc beginning
arcend: .byte   0,0             ;arc end
angle:  .byte   0,0             ;processing angle
radius: .byte   0,0             ;arc radius
piflag: .byte   0               ;arc/pie flag
stcntr: .byte   0,0             ;update counts in arc
```

```
upcntr: .byte   0,0             ;current counts in arc
fxyarc: .byte   0,0             ;arc drawing parameters
f1arc:  .byte   0,0
f2arc:  .byte   0,0
af1arc: .byte   0,0
af2arc: .byte   0,0
af3arc: .byte   0,0
avarc:  .byte   0
bvarc:  .byte   0
cvarc:  .byte   0
xvarc:  .byte   0,0
yvarc:  .byte   0,0

        .page

        ;       timing constants [plotter option]

        .if     plotter
pltron: .byte   0               ;plotter on flag

        .area   BUFSAV

plptrn: .byte   0,0             ;plotter pattern
waitxy: .byte   0,0             ;x-y recorder wait count
waitup: .byte   0,0             ;pen up wait count
waitdn: .byte   0,0             ;pen down wait count
waitmv: .byte   0,0             ;move wait count
wtlp:   .byte   0               ;move wait count modifier

        .area   VARSAV

        .endif

        ;       'term' & 'link' protocol status

d.stat: .byte   0               ;'link' control status

        ;       7-      xoff'd(1)/else(0) (data from host)
        ;       6-      xoff'd(1)/else(0) (data to host)
        ;       0-      xon(^q)/xoff(^s) enabled(1)/disabled(0)

$dxonf: .byte   0               ;'link' flag character

$xon:   .byte   0               ;xon character (default is ^q)
$xoff:  .byte   0               ;xoff character (default is ^s)

        ;       printer variables

        .if     printer
$pntrs: .byte   0               ;printer select code
$tofst: .byte   0,0             ;offset columns
$tx1:   .byte   0,0             ;first column
$tx2:   .byte   0,0             ;last column
$ty1:   .byte   0,0             ;first row
$ty2:   .byte   0,0             ;last row
$tcntr: .byte   0,0             ;printer special character counter
$tdens: .byte   0               ;low or high density dump
$tzoom: .byte   0               ;printer zoom factor
$tdump: .byte   0               ;screen dump density

$color: .byte   0               ;color result
$bkgnd: .byte   0               ;bacground flag
$anyclr:.byte   0               ;any color flag
$inkclr:.byte   0               ;ink color code
$inkjet:.byte   0               ;ink jet code
$pdens: .byte   0               ;density mode
$drctn: .byte   0               ;memory read direction
$tnsfr: .byte   0               ;read word count

$pzoom: .byte   0               ;display zoom factor
$dots:  .byte   0,0             ;dots per line
$rzoom: .byte   0               ;running x-zoom
$rdots: .byte   0,0             ;running dots left

$pofst: .byte   0,0             ;positional offset
$plen:  .byte   0,0             ;total line length
$pasci: .blkb   5               ;ascii of color/jet code
                                ;and length
```

```
$dotctr:.byte   0               ;dot counter
        .byte   0,0
$cur0:  .byte   0,0             ;variable set 0
$zoom0: .byte   0
$line0: .byte   0,0
        .byte   0,0
$cur1:  .byte   0,0             ;variable set 1
$zoom1: .byte   0
$line1: .byte   0,0
        .byte   0,0
$cur2:  .byte   0,0             ;variable set 2
$zoom2: .byte   0
$line2: .byte   0,0


        .area   BUFSAV

$trmap: .blkb   0d16            ;16-byte transformed map

        .area   VARSAV

        .endif

        .page
        .sbttl  run status definitions

rstat0: .byte   0
rstat1: .byte   0
rstat2: .byte   0

        ;       rstat0 (rstat2 same - low latched)

        ;       7-      reset (1)
        ;       6-      plotter on(1)/off(0)
        ;       5-      printer on(1)/off(0)
        ;       4-      graphics on(1)/off(0)
        ;       3-      clear video (0)
        ;       2-      calibrate (1)
        ;       1-      screen dump (0)
        ;       0-      continue (0)


        ;       rstat1

        ;       7-      running (1)
        ;       6-      relative origin (0) / relative position (1)
        ;       5-      auto x step (1)
        ;       4-      auto y step (1)
        ;       3-      vector (0) / point (1)
        ;       2-      replace (0) / complement (1)
        ;       1-      pen up (0) / pen down (1)
        ;       0-      replace/complement enable(1)



        .page
        .sbttl  color maps and temporaries

zrtmp:  .byte   0               ;rgb values from $z
zgtmp:  .byte   0
zbtmp:  .byte   0
gcolor: .byte   0               ;drawing color
altmap: .byte   0,0             ;current selected map address


        .area   BUFSAV

curmap: .blkb   0d48            ;current color mapping
altmpa: .blkb   0d48            ;values for alternate color maps
altmpb: .blkb   0d48
altmpc: .blkb   0d48
altmpd: .blkb   0d48
ds$map: .blkb   0d48            ;values for display system map
.$ptrn: .blkb   0d2             ;line drawing pattern
        .blkb   0d32            ;and area drawing pattern


        .area   VARSAV
```

```
        .sbttl  current display system status

crxpos: .byte   0,0             ;current vidx contents
crypos: .byte   0,0             ;current vidy contents

$dsdat: .byte   0               ;current writing data

$dswrt: .byte   0               ;current writing code

$blank: .byte   0               ;current blanking code
        .byte   0               ;code for display blanking
        .byte   0               ;code for display enabling

$dsclr: .byte   0               ;current clearing code
        .byte   0               ;code for clearing display
        .byte   0               ;code for not clearing display

$dssts: .byte   0               ;current display system status
                                ;       3 - histo/cursor on
                                ;       2 - drawing histo/cursor
                                ;       1 - 'oc' on
                                ;       0 - loading 'oc'
ds$tim: .byte   0               ;display system .1 second timer
dstate: .byte   0               ;display state counter
occntr: .byte   0               ;'oc' counter
hscntr: .byte   0               ;histo counter
dstmp1: .byte   0               ;temporaries
dstmp2: .byte   0

        .page
        .sbttl  6809 startup

        .area   PGMSAV

pwrup:  orcc    #120            ;disable interrupts
        clr     177776          ;disable 6828 interrupt controller
        lda     #370            ;led's off
        sta     plout

;  verify rom integrity
        ldu     #1$
        jmp     verrom
1$:     .word   120000          ;first address
        .word   140000          ;last address+1
        .word   chksm0          ;location of checksum word
        .byte   350             ;error code

        ldu     #2$
        jmp     verrom
2$:     .word   140000
        .word   160000
        .word   chksm1
        .byte   330

        ldu     #3$
        jmp     verrom
3$:     .word   160000
        .word   000000
        .word   chksm2
        .byte   310

;  verify ram integrity
        ldu     #4$
        jmp     verram
4$:     .word   000000          ;first address
        .word   040000          ;last address+2
        .byte   270             ;error code

        ldu     #5$
        jmp     verram
5$:     .word   000001          ;every other byte is
        .word   040001          ;in the same ram
        .byte   250

        ldu     #6$
        jmp     verram
```

```
6$:     .word   040000          ;verram steps address by 2
        .word   100000
        .byte   230

        ldu     #7$
        jmp     verram
7$:     .word   040001
        .word   100001
        .byte   210

;  clear all ram data
        ldd     #0              ;set all ram to zero
        lds     #0
8$:     std     ,s++
        cmps    #stack
        bne     8$

;  direct page settup
        lda     #>direct
        tfr     a,dp            ;set direct page

;  set interrupt table
        ldx     #irqtbl         ;table pointer
9$:     ldd     ,x++            ;get vector entry
        beq     10$
        std     [,x++]          ;place in vector table
        bra     9$              ;loop

;   set up communication
10$:    ldd     #dlbuf          ;preset host link buffer
        std     *dlqntr
        std     *dlpntr

        lda     #21             ;default xon (^q)
        sta     *$xon
        lda     #23             ;default xoff (^s)
        sta     *$xoff

        ldd     #scbuf          ;preset scan buffer
        std     *scqntr
        std     *scpntr

        lda     #370            ;led's off / strobe(1)
        sta     *outpl
        sta     plout

        lda     #377            ;set panel flags high
        sta     *statpl
        sta     *rstat0
        sta     *rstat2

;  initialize ram character set
        ldy     #r6571a         ;source
        sty     *ptchar
        sty     *romset
        jsr     romcopy         ;to chram

        lda     #0d8            ;initialize character size
        sta     *xsize
        sta     *ysize

;  startup crt controller
        ldx     #a.6845         ;use table a
        ldy     #a.crtc
        jsr     i6845$

;  set draw color to 'white'
        lda     #7
        sta     *gcolor
        sta     $dsdat
        sta     bp$dat

;  initialize rgb dac maps
        ldx     #altmpa
        stx     *altmap
        jsr     iniflt
        ldx     #altmpb
        jsr     iniflt
```

```
        ldx     #altmpc
        jsr     iniflt
        ldx     #altmpd
        jsr     iniflt
        ldx     #ds$map
        jsr     iniflt
        ldx     #clrtbl
        jsr     inirgb

;  default pattern and drawing mode
        ldd     #-1             ;all ones pattern
        ldx     #.$ptrn         ;pattern area

11$:    std     ,x++            ;load default pattern
        cmpx    #.$ptrn+0d34    ;end of pattern area
        bne     11$

;  setup bit-map control
        lda     #17
        ldb     #0

        sta     bp$wrt          ;enable writing to all planes
        sta     *$dswrt         ;current state

        sta     bp$dsp          ;display all planes
        sta     *$blank         ;current state
        stb     *$blank+1       ;blanking
        sta     *$blank+2       ;not blanking

        stb     bp$clr          ;stop clearing
        stb     *$dsclr         ;current state
        sta     *$dsclr+1       ;clearing
        stb     *$dsclr+2       ;not clearing

;  setup default dump button mode
        .if     printer
        lda     #1              ;graphics
        sta     *$tdump
        .endif

        .page
        .sbttl  main routine

main:   orcc    #120            ;disable interrupts
        lda     #>direct
        tfr     a,dp            ;reset direct page
        lds     #stack          ;reset stack pointer
        clr     177740          ;enable 6828 interrupt controller
        andcc   #257            ;now allow interrupts
        lda     *rstat2
        bita    #10             ;video need clearing ?
        bne     2$              ;if not - skip
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~100
        sta     *outpl          ;'busy'
        sta     plout
        puls    cc
        jsr     clear           ;clear screen
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        ora     #100
        sta     *outpl          ;'not busy'
        sta     plout
        puls    cc
1$:     lda     *rstat0
        bita    #10             ;button released ?
        beq     1$              ;loop until released
        lda     *rstat2
        ora     #10
        sta     *rstat2         ;clear entry flag

        .page
        .sbttl  screen dump / plotter calibration

2$:
```

```
        .if     printer
        lda     *rstat2
        bita    #2              ;screen dump ?
        bne     3$              ;no - skip
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~100
        sta     *outpl          ;'busy'
        sta     plout
        puls    cc
        jsr     screen          ;dump screen
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        ora     #100
        sta     *outpl          ;'not busy'
        sta     plout
        puls    cc
        .endif
3$:
        .if     plotter
        jsr     calib           ;check cal button
        .endif

        .page
        .sbttl  character scanner

        ldx     *scntr          ;any characters ?
        beq     6$              ;if not - skip
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~100
        sta     *outpl          ;'busy'
        sta     plout
        puls    cc
        jsr     getchr          ;get character from scan buffer

        .if     printer
        lda     *rstat0
        bita    #40             ;printer on ?
        beq     4$              ;no - skip
        jsr     printc          ;print the character
4$:
        .endif

        jsr     listpr          ;enter list processor
5$:     pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        ora     #100
        sta     *outpl          ;'not busy'
        sta     plout
        puls    cc
6$:     jmp     main            ;keep going

        .page
        .sbttl  clock interrupt routine

clocki: orcc    #120            ;stop other interrupts
        tst     clkclr          ;clear clock latch
        lda     #>direct
        tfr     a,dp            ;set direct page

        ;panel switch scanner

        ldb     plstat          ;get switch data
        lda     *statpl         ;get previous data
        stb     *statpl         ;save new for next time
        eora    *statpl         ;unstable bits = 1
        tab                     ;save
        coma                    ;stable switches = 1
        anda    *statpl         ;mask unstable bits
        andb    *rstat0         ;mask out newly stable bits
        aba                     ;combine old bits with new stable bits
        sta     *rstat0         ;save new status
        anda    *rstat2         ;set up latched low status
```

```
        sta     *rstat2         ;save new status

        dec     *timflg         ;update timing flag
        bgt     1$
        clr     *timflg
1$:

        .if     plotter
        clr     *pltron         ;plotter off
        lda     pntin           ;plotter option installed ?
        bita    #2
        bne     2$              ;no - skip
        lda     *rstat0
        bita    #100            ;plotter on ?
        beq     2$              ;if not - skip
        com     *pltron         ;plotter on !
2$:
        .endif

        .page

;   host 'data link' checks

3$:     lda     *bfstat
        bita    #scn            ;scan buffer need data ?
        bne     5$              ;if not - skip
        lda     #100            ;enable interrupts
        sta     hrcsr
4$:     lda     *d.stat
        bita    #1              ;xon/xoff enabled ?
        beq     5$              ;if not - skip
        tsta                    ;xoff'd ?
        bpl     5$              ;no - skip
        anda    #177            ;clear xoff'd
        sta     *d.stat
        ldb     *$xon           ;get 'xon' character
        stb     *$dxonf         ;place character for xmtr to see
        bra     7$

;  restart xmtr interrupts

5$:     lda     *$dxonf         ;special ?
        bne     7$              ;yes - skip
        lda     *dlcntr         ;any characters ?
        beq     8$              ;no - skip
6$:     lda     *d.stat
        bita    #1              ;xon/xoff enabled ?
        beq     7$              ;no - skip
        bita    #100            ;xoff'd ?
        bne     8$              ;yes - skip
7$:     lda     #100            ;enable xmtr
        sta     htcsr
8$:     jmp     $dsrvc          ;jump to display system checks

        .page
        .sbttl  host data link receiver interrupt handler

dlrint: orcc    #120            ;stop other interrupts
        lda     #>direct
        tfr     a,dp            ;set direct page
        tst     hrcsr           ;data ?
        lbpl    8$              ;no - skip

        ldb     hrdata          ;get data (request is not cleared)

        .if     1-printer
        andb    #177            ;7-bit ascii
        .endif

        lda     *d.stat
        bita    #1              ;xon/xoff enabled ?
        beq     3$              ;if not - skip
;  check for xon/xoff characters
        cmpb    *$xon           ;'xon' ?
        bne     1$              ;if not - skip
        anda    #277            ;clear xoff
        sta     *d.stat
        bra     2$              ;exit
```

```
1$:     cmpb    *$xoff          ;'xoff' ?
        bne     3$              ;if not - skip
        ora     #100            ;set xoff
        sta     *d.stat
2$:     stb     hrdata          ;clear request (dumby write)
        bra     8$              ;exit

3$:     lda     hrcsr           ;check for interrupt enabled
        cmpa    #300            ;and a character
        bne     8$              ;no - skip
        stb     hrdata          ;clear request (dumby write)

        ldx     *scpntr         ;get pointer
        stb     ,x+             ;place character
        cmpx    #scend          ;at end of buffer ?
        bne     4$              ;if not - skip
        ldx     #scbuf          ;reset pointer
4$:     stx     *scpntr         ;save new pointer
        ldd     *scntr          ;get count
        addd    #1              ;update count
        std     *scntr          ;save count

        cmpd    #scfull         ;full ?
        bcs     8$              ;if not - skip
        cmpd    #scvful         ;too full ?
        bcs     5$              ;if not - skip
        clr     hrcsr           ;disable interrupts
5$:     lda     *bfstat
        ora     #scn
        sta     *bfstat         ;say full
        lda     *d.stat
        bita    #1              ;xon/xoff enabled ?
        beq     8$              ;if not - exit
        tsta                    ;already xoff'd ?
        bpl     6$              ;if not - skip
        ldb     *scntr+1        ;get count
        andb    #7              ;after 8'th character do another xoff
        bne     8$              ;exit

6$:     ora     #200            ;xoff'd
        sta     *d.stat
        ldb     *$xoff          ;get 'xoff' character
        tst     htcsr           ;output ready ?
        bpl     7$              ;no - exit
        stb     htdata          ;transmit data
        bra     8$

7$:     stb     *$dxonf         ;place character for xmtr to see
8$:                             ;fall through to dltint

        .page
        .sbttl  host data link transmitter interrupt handler

dltint: tst     htcsr           ;output ready ?
        bpl     8$              ;no - exit

        ldb     *$dxonf         ;special ?
        bne     7$              ;yes - skip
1$:     lda     *d.stat
        bita    #1              ;xon/xoff enabled ?
        beq     2$              ;no - skip
        bita    #100            ;xoff'd ?
        bne     3$              ;yes - exit
2$:     ldb     *dlcntr         ;get character count
        bne     4$              ;if characters left - skip
3$:     clr     htcsr           ;terminate transfers
        bra     8$              ;finished

4$:     decb                    ;one less character
        stb     *dlcntr         ;save count
        cmpb    #dlmpty         ;buffer emptying ?
        bcc     5$              ;if not - skip
        lda     *bfstat
        anda    #~dl
        sta     *bfstat         ;else say - need data
5$:     ldx     *dlqntr         ;get pointer to character
        ldb     ,x+             ;get character
        cmpx    #dlend          ;at end of buffer ?
```

```
        bne     6$              ;if not - skip
        ldx     #dlbuf          ;reset pointer
6$:     stx     *dlqntr         ;save new pointer
7$:     stb     htdata          ;transmit data
        clr     *$dxonf         ;clear special
8$:     rti                     ;finished

        .page
        .sbttl  place data in data link buffer

plcdl:  pshs    cc
        orcc    #120            ;hold interrupts
        ldx     *dlpntr         ;get pointer
        stb     ,x+             ;save character
        inc     *dlcntr         ;update counter
        cmpx    #dlend          ;at end of buffer ?
        bne     1$              ;if not - skip
        ldx     #dlbuf          ;reset pointer
1$:     stx     *dlpntr         ;save new pointer
        ldb     *dlcntr         ;get count
        cmpb    #dlfull         ;buffer full ?
        bcs     2$              ;if not - branch
        lda     *bfstat
        ora     #dl
        sta     *bfstat         ;set full
        puls    cc
        sec                     ;full
        rts                     ;finished
2$:     puls    cc
        clc                     ;not full
        rts                     ;finished

        .page
        .sbttl  get character from scan buffer

getchr: pshs    cc
        orcc    #120            ;hold interrupts
        ldx     *scqntr         ;get pointer
        lda     ,x+             ;get character
        sta     *char           ;save character
        cmpx    #scend          ;at end of buffer ?
        bne     1$              ;if not skip
        ldx     #scbuf          ;reset pointer
1$:     stx     *scqntr         ;save pointer
        ldd     *scntr          ;update count
        subd    #1
        std     *scntr
        cmpd    #scmpty         ;empty ?
        bcc     2$              ;if not - skip
        lda     *bfstat
        anda    #~scn
        sta     *bfstat         ;say empty
2$:     puls    cc
        rts                     ;finished

        .page
        .sbttl  list processor


        ; 1.    if in single char mode, then:
        ;               clear single char mode
        ;               if not in special mode, then:
        ;                       scan for control characters, if found:
        ;                               clear gosub
        ;                               end
        ;               do gosub (if defined)
        ;               if character used - end
        ;               else go to 2.

listpr: lda     *lstat          ;get status
        bpl     list.d          ;not in single - skip
        anda    #177            ;clear single mode
        sta     *lstat          ;save status
        bita    #20             ;special character mode ?
        beq     1$              ;not special mode - skip
        anda    #357            ;clear special
        sta     *lstat          ;and save
        lda     *char           ;get any character
```

```
        bra     list.a          ;and go
1$:     lda     *char           ;get character
        cmpa    #40             ; a control ?
        bcc     list.a          ;if not - skip
        ldx     #0              ;else clear gosub
        stx     *gosub
        bra     list.b
list.a: ldx     *gosub          ;get process address
        beq     list.b          ;if undefined - skip
        jsr     ,x              ;do it
        bcc     list.d          ;character not used
list.b: rts                     ;finished

        ; 2.    do command scanner
        ;       if command is found - finished
        ;
list.d: jsr     cmdscn          ;scan for commands
        bcs     list.e          ;command found - done
        lda     *rstat1         ;are we running
        bmi     list.f          ;if so - skip ahead
list.e: rts                     ;finished

        ; 3.    check scan mode, if set then:
        ;               if char is (+) then:
        ;                       clear scan
        ;                       set build
        ;                       reset list buffer
        ;                       end

list.f: lda     *lstat
        bita    #100            ;in scan mode ?
        beq     list.j          ;if not - skip
        ldb     *char           ;get character
        cmpb    #'+             ;is it a (+)  ?
        bne     list.g          ;if not - skip out
        anda    #277            ;else clear scan mode
        ora     #40             ;set build
        sta     *lstat          ;save new status
        clr     *schflg         ;clear seen flag
        clr     *lstcnt         ;reset buffer
        ldx     #lsbuff
        stx     *lpntr
list.g: rts                     ;finished

        ; 4.    check build, if not set then:
        ;               reset buffer
        ;               if char is (0-9) or (-)
        ;                  or free format   then:
        ;                       set build
        ;                       put char in buffer
        ;                       goto lcheck
        ;                  else: set scan
        ;                        go scan

list.j: lda     *lstat
        bita    #40             ;building ?
        bne     list.o          ;if so - skip
        clr     *schflg         ;clear seen flag
        clr     *lstcnt         ;reset buffer
        ldx     #lsbuff
        stx     *lpntr
        ldb     *format         ;in free format ?
        beq     list.k          ;if so - set up build
        ldb     *char           ;get character
        cmpb    #'-             ;is it a (-) ?
        beq     list.k          ;if so - skip
        cmpb    #'0             ;bcd character ?
        bcs     list.l          ;<0 - skip
        cmpb    #'9             ;>9 ?
        bhi     list.l          ;not a number - skip
list.k: ora     #40             ;set build
        sta     *lstat          ;save status
        bra     list.o          ;now process this character
list.l: ora     #100            ;set scan
        sta     *lstat          ;save status
        bra     list.f          ;go scan

        ; 5.    build is set
```

```
        ;               put character in buffer
        ;               check bufffer

list.o: lda     *format         ;check for free format
        beq     list.q          ;if so - skip
        lda     *char           ;get character
        cmpa    #40             ;any controls ?
        bcc     list.x          ;if not  skip
list.p: clr     *lstcnt         ;reset buffer
        clr     *schflg         ;seen character flag
        ldx     #lsbuff
        stx     *lpntr
        lda     *lstat          ;set status for scan
        anda    #17             ;save lower for graphics
        ora     #100            ;scanning
        sta     *lstat
        rts                     ;finished

list.q: lda     *schflg         ;seen a char ?
        bne     list.u          ;if so - skip
        lda     *char           ;else scan this character
        cmpa    #'-             ;a - sign ?
        beq     list.r          ;if so - save it
        cmpa    #'+             ;a + sign ?
        beq     list.t          ;strip +'s
        cmpa    #40             ;space ?
        beq     list.t          ;strip spaces
        cmpa    #'0             ;only want numerals
        bcs     list.p
        cmpa    #':
        bcc     list.p
        bsr     list.r          ;save character
list.s: inc     *schflg         ;character seen
list.t: rts                     ;finished

list.r: ldx     *lpntr          ;get pointer
        sta     ,x+             ;save character
        stx     *lpntr
        inc     *lstcnt         ;one more character
        rts                     ;and return

list.u: lda     *char           ;get character
        cmpa    #'-             ;a - sign ?
        beq     list.w          ;if so - skip
        cmpa    #'+             ;a + sign ?
        beq     list.w          ;if so - skip
        cmpa    #'0             ;want only numerals
        bcs     list.w
        cmpa    #':
        bcc     list.w
        ldx     *lpntr          ;get pointer
        sta     ,x              ;save character
        cmpx    #lsbuff+0d8     ;at end of buffer ?
        beq     list.v          ;if so - skip update
        leax    1,x             ;else update pointer
        stx     *lpntr
        inc     *lstcnt         ;update counter
list.v: rts                     ;finshed
list.w: bsr     lche.a          ;go evaluate and process
        clr     *schflg         ;clear seen flag
        bra     list.q          ;rescan last character

list.x: ldx     *lpntr          ;get pointer
        sta     ,x+             ;save character
        stx     *lpntr
        inc     *lstcnt         ;update count


lcheck: lda     *lstcnt         ;get count
        cmpa    *format         ;enough characters ?
        bne     lche.b          ;if not - skip
lche.a: jsr     evaln           ;go evaluate data
        clr     *lstcnt         ;clear buffer
        ldx     #lsbuff
        stx     *lpntr
        ldx     *gosub          ;get process
        beq     lche.b          ;if undefined - skip
        jmp     ,x              ;else do process
```

```
lche.b: rts                     ;finished

        .page
        .sbttl  command scanner

cmdscn: ldb     *char           ;get character
        lda     *pchar          ;get previous character
        stb     *pchar          ;save new character
        cmpa    #'$             ;old a ($) ?
        bne     6$              ;if not - skip
        cmpb    #141            ;allow lower case commands
        bcs     1$              ;upper case - skip
        subb    #40             ;make lower into upper
1$:     subb    #'A             ;a=0,z=31
        cmpb    #21             ;r ?
        beq     2$              ;yes - skip
        cmpb    #22             ;s ?
        beq     2$              ;yes - skip
        lda     *rstat1
        bita    #200            ;running ?
        beq     6$              ;if not - skip
        tstb                    ;a=0,z=31
        bmi     3$              ;not an internal command
        cmpb    #32             ;not a-z ?
        bcc     3$              ;not an internal command
2$:     ldx     #cmdtbl         ;table pointer
        aslb                    ;offset into table
        bsr     4$              ;reset controls
        jsr     [b,x]           ;and do command
        sec                     ;found command
        rts                     ;finished

3$:     ldb     *char           ;check if user function
        subb    #'0             ;make bcd
        bmi     5$              ;if not - bad command
        cmpb    #0d9            ;must be a digit
        bhi     5$              ;if not - bad command
        aslb
        aslb
        ldx     #$xtrn0
        leax    b,x             ;add offset into entry table
        ldd     ,x              ;a function there ?
        beq     5$              ;no - skip
        subd    2,x             ;verify entry
        cmpd    #0x5AA5         ;check code
        bne     5$              ;invalid - skip
        bsr     4$              ;reset controls
        jsr     [,x]            ;and goto user function
        sec                     ;found command
        rts                     ;finished

4$:     lda     *lstat
        anda    #~360
        sta     *lstat          ;all new list control
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        ora     #40
        sta     *outpl          ;clear format error
        sta     plout
        puls    cc
        rts

        .page

;  bad command service
badcom=.
        leas    2,s             ;pop return
        ldu     #0              ;no service
        stu     *gosub
5$:     pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~40
        sta     *outpl          ;format error
        sta     plout
        puls    cc
        sec                     ;found bad command
```

```
        rts                     ;finished
6$:     clc                     ;no command
        rts                     ;finished


cmdtbl: .word   .$a
        .word   .$b
        .word   .$c
        .word   .$d
        .word   .$e
        .word   .$f
        .word   .$g
        .word   .$h
        .word   .$i
        .word   badcom
        .word   .$k
        .word   .$l
        .word   .$m
        .word   .$n
        .word   .$o
        .word   .$p
        .word   .$q
        .word   .$r
        .word   .$s

        .if     printer
        .word   .$t
        .else
        .word   badcom
        .endif

        .word   .$u
        .word   .$v

        .if     plotter
        .word   .$w
        .else
        .word   badcom
        .endif

        .word   .$x
        .word   .$y
        .word   .$z

        .page
        .sbttl  evaluate number routine

evaln:  ldy     #nsign          ;point to variables
1$:     clr     ,y+             ;clear byte
        cmpy    #number+ndigit  ;end of area ?
        bne     1$              ;loop until all cleared
        ldx     *lpntr          ;string pointer
2$:     ldb     ,-x             ;get character
        cmpb    #'-             ; a (-) ?
        bne     3$              ;if not - skip
        lda     #377            ;sign is negative
        sta     *nsign
        bra     5$              ;skip ahead
3$:     cmpy    #number         ;got enough ?
        beq     7$              ;if so - skip
        cmpb    #40             ;a space ?
        beq     5$              ;if so - skip
        subb    #'0             ;make bcd
        bmi     4$              ;if not -  skip
        cmpb    #0d9            ;must be a digit
        bls     6$              ;if so - skip
4$:     pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~40
        sta     *outpl          ;set format error
        sta     plout
        puls    cc
5$:     clrb
6$:     stb     ,-y             ;save character
7$:     dec     *lstcnt         ;any more ?
        bgt     2$              ;loop until done
```

```
        .page
        .sbttl  bcd to binary conversion

bcdbin: ldd     #0              ;init result
        bra     2$              ;go to entry
1$:     aslb                    ;*2
        rola
        std     ,--s            ;save
        aslb                    ;*8
        rola
        aslb
        rola
        addd    ,s++            ;*2+*8 -> *10
2$:     addb    ,y+             ;add in digit
        adca    #0
        cmpy    #number+ndigit  ;end ?
        bne     1$

;  now use sign
        tst     *nsign          ;negative ?
        beq     3$              ;if positive - skip
        coma
        comb
        addd    #1
3$:     std     *number         ;save result
        rts                     ;finished

        .page
        .sbttl  next character transfer routines

nxtchr: lda     *lstat
        ora     #200
        sta     *lstat          ;character mode
        sec                     ;character used
nxtgos: ldu     ,s++            ;next entry
        stu     *gosub
        rts                     ;finished

errgos: pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~40
        sta     *outpl          ;undefined code error
        sta     plout
        puls    cc

endgos: ldu     #0              ;no entry
        stu     *gosub
        rts                     ;finished

;  table driven dispatcher

$dispatch:
        cmpa    #141            ;allow lower case options
        bcs     1$              ;skip if upper
        cmpa    #173
        bcc     1$              ;not lower - skip
        suba    #40             ;make upper
1$:     ldu     ,s++            ;table address
2$:     clc                     ;character unused if exits
        tst     ,u              ;end of table ?
        bmi     3$              ;terminate scan and do service
        beq     endgos          ;end of service
        cmpa    ,u              ;match table entry ?
        beq     3$              ;yes - exit
        leau    3,u             ;update table pointer
        bra     2$              ;and loop

3$:     jmp     [1,u]           ;go to service routine

        .page
        .sbttl  $r command

.$r:    lda     *rstat0
        bita    #20             ;graphics on ?
        bne     1$              ;if so - skip
        lda     *rstat1
```

```
        anda    #~200
        sta     *rstat1         ;else not
        rts                     ;finished
1$:     lda     *rstat1
        ora     #200
        sta     *rstat1         ;say running
        clr     *format         ;set for free format
        jsr     nxtchr          ;get character

        jsr     $dispatch
        .byte   'V
        .word   2$              ;version select
        .byte   'C
        .word   3$              ;copyright select
        .byte   -1
        .word   6$              ;end dispatch

2$:     lda     #dsp$id         ;dsp$id character version #
        bra     4$

3$:     lda     #dsp$cr         ;dsp$cr copyright version #

4$:     sta     *number
        ldx     #chksm1+2       ;version string
5$:     ldb     ,x+             ;get character
        stx     *qt             ;save pointer
        bsr     plcbuf          ;send character
        ldx     *qt             ;get pointer
        dec     *number         ;more ?
        bne     5$              ;yes - loop
        lda     *lstat
        ora     #200
        sta     *lstat          ;character mode
        rts                     ;and finished

6$:     suba    #'0             ;make bcd
        bcs     7$              ;if not - skip
        cmpa    #0d9            ;bcd ?
        bhi     7$              ;if not - skip
        sta     *format         ;save new format
        sec                     ;character used
        bra     8$
7$:     clc                     ;character not used
8$:     jmp     endgos          ;don't come back

plcbuf: lda     *bfstat
        bita    #dl             ;buffer full ?
        bne     plcbuf          ;yes - loop
        jsr     plcdl           ;put char in host data link buffer
        rts                     ;finished

        .page
        .sbttl  $o command

.$o:    jsr     nxtgos          ;set up for next entry

        ldx     *number         ;get value
        stx     *xorg           ;save xorigin
        jsr     nxtgos

        ldx     *number         ;get value
        stx     *yorg           ;save yorigin
        bra     .$o

        .page
        .sbttl  $m command

.$m:    lda     *rstat1
        anda    #~160
        sta     *rstat1         ;relative to origin
1$:     jsr     nxtgos          ;go set up for next entry

        ldx     *number         ;get value
        stx     *xstep          ;save as x
        jsr     nxtgos

        ldx     *number         ;get value
        stx     *ystep          ;save as y
```

```
        jsr     proc            ;go process
        bra     1$

        .page
        .sbttl  $l command

.$l:    lda     *rstat1         ;get status
        anda    #217            ;clear modes
        ora     #100            ;set relative to current position
        sta     *rstat1         ;save
1$:     jsr     nxtgos

        ldx     *number         ;get value
        stx     *xstep          ;save as x
        jsr     nxtgos

        ldx     *number         ;get value
        stx     *ystep          ;save as y
        jsr     proc            ;go process
        bra     1$

        .page
        .sbttl  $d, $u, $v, and $p commands

.$d:    lda     *rstat1
        ora     #2
        sta     *rstat1         ;indicate pen down
        jsr     video           ;try video
.$dx:
        .if     plotter
        tst     *pltron         ;plotter on ?
        beq     1$              ;no - skip
        lda     *outpl
        bita    #4              ;is pen down ?
        bne     1$              ;if so - don't wait
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        ora     #4
        sta     *outpl          ;drop pen
        sta     plout
        puls    cc
.$dw=.
        ldx     waitdn          ;get down wait
        bra     .$ua
        .endif

1$:     rts                     ;finished


.$u:    lda     *rstat1
        anda    #~2
        sta     *rstat1         ;indicate pen up
.$ux:
        .if     plotter
        tst     *pltron         ;plotter on ?
        beq     3$              ;no - skip
        lda     *outpl
        bita    #4              ;is pen up ?
        beq     3$              ;if so - skip
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~4
        sta     *outpl
        sta     plout
        puls    cc
.$uw=.
        ldx     waitup          ;get up wait
.$ua=.
        leax    1,x             ;at least 1
1$:     lda     #0d38           ;100 microseconds per x
2$:     deca
        bne     2$
        leax    -1,x
        bne     1$
        .endif
```

```
3$:     rts                     ;finished


.$v:    lda     *rstat1
        anda    #~10
        sta     *rstat1         ;vector mode
        rts                     ;finished


.$p:    lda     *rstat1
        ora     #10
        sta     *rstat1         ;point mode
        rts                     ;finished


        .page
        .sbttl  $w command

        .if     plotter
.$w:    ldx     #0              ;init waits to minimum
        stx     waitxy
        stx     waitup
        stx     waitdn
        stx     waitmv
        jsr     nxtgos

        ldx     *number         ;save waitxy
        stx     waitxy
        jsr     nxtgos

        ldx     *number         ;save waitup
        stx     waitup
        jsr     nxtgos

        ldx     *number         ;save waitdn
        stx     waitdn
        jsr     nxtgos

        ldx     *number         ;save waitmv
        stx     waitmv
        jmp     endgos          ;end of input
        .endif

        .page
        .sbttl  $s command

        ;       $s loader variables

        .area   BUFSAV

$scntr: .byte   0               ;character counter
$schksm:.byte   0               ;running checksum
$svalu: .byte   0,0             ;binary value of hex
$srbc:  .byte   0               ;running byte count
$sladdr:.byte   0,0             ;load address

        .area   PGMSAV

.$s:    lda     *rstat1
        anda    #~200
        sta     *rstat1         ;not running
        jsr     nxtchr          ;get character

        jsr     $dispatch
        .byte   '1
        .word   1$
        .byte   '9
        .word   1$
        .byte   -1
        .word   endgos


1$:     clra
        sta     $schksm         ;clear checksum

        lda     #2
        sta     $scntr          ;for rbc
2$:     jsr     nxtchr
```

```
        jsr     $sghex          ;convert hex to binary
        lbcs    errgos          ;note input error
        dec     $scntr
        bgt     2$
        stb     $srbc           ;save byte count
        addb    $schksm         ;update checksum
        stb     $schksm

        lda     #4
        sta     $scntr          ;for address
3$:     jsr     nxtchr
        jsr     $sghex          ;convert hex to binary
        lbcs    errgos          ;note input error
        dec     $scntr
        bgt     3$
        std     $sladdr         ;save load address
        aba
        adda    $schksm         ;update checksum
        sta     $schksm
        dec     $srbc
        dec     $srbc

4$:     dec     $srbc           ;more data ?
        ble     7$              ;no skip to checksum

        lda     #2
        sta     $scntr          ;for data bytes
5$:     jsr     nxtchr
        jsr     $sghex          ;convert hex to binary
        lbcs    errgos          ;note input error
        dec     $scntr
        bgt     5$

        ldx     $sladdr         ;load data

        .if     0               ;let user beware (no address checks)
        cmpx    #$xtrn0         ;address check
        blo     6$              ;not in range - skip
        cmpx    #$xtend         ;address check
        bhis    6$              ;not in range - skip
        .endif

        stb     ,x+
        stx     $sladdr         ;save address
6$:     addb    $schksm         ;update checksum
        stb     $schksm
        bra     4$

7$:     lda     #2
        sta     $scntr          ;for checksum byte
8$:     jsr     nxtchr
        jsr     $sghex          ;convert hex to binary
        lbcs    errgos          ;note input error
        dec     $scntr
        bgt     8$

        addb    $schksm         ;verify checksum
        incb
        lbeq    endgos          ;good data
        lbra    errgos          ;note checksum error

        .page
        .sbttl  Input Hex to Binary Conversion

$sghex: lda     *char           ;get character
        suba    #'0             ;convert to binary
        bmi     2$              ;error - skip
        cmpa    #0d9
        blos    1$              ;0-9 skip
        suba    #7
        cmpa    #0d15
        bhi     2$              ;error - skip
1$:     pshs    a               ;save value
        ldd     $svalu          ;update value
        aslb
        rola
        aslb
        rola
```

```
        aslb
        rola
        aslb
        rola
        orb     ,s+             ;or in new nibble
        std     $svalu          ;save result
        clc
        rts                     ;good return
2$:     sec
        rts                     ;error on input

        .page
        .sbttl  $k command

.$k:    lda     *lstat
        ora     #220
        sta     *lstat          ;special mode
        sec
        jsr     nxtgos          ;wait for character

        jsr     $dispatch
        .byte   'R
        .word   2$              ;clear a rectangular region
        .byte   -1
        .word   1$

1$:     lda     *rstat2
        anda    #~10
        sta     *rstat2         ;set flag - allow main to do job
        clc                     ;character not used
        rts                     ;finished

        ;rectangular area clear

2$:     sec
        jsr     nxtgos
        ldd     *number         ;compute screen position
        addd    *xorg           ;x0
        anda    #17
        std     *fractx

        jsr     nxtgos
        ldd     *number         ;y0
        addd    *yorg
        coma
        comb
        anda    #17
        std     *fracty

        jsr     nxtgos
        ldd     *number         ;x1
        addd    *xorg
        anda    #17
        std     *fractx+2
        cmpd    *fractx         ;order x0 and x1
        bhs     3$              ;x1 > x0
        ldx     *fractx
        std     *fractx
        stx     *fractx+2       ;now - x1 > x0

3$:     jsr     nxtgos
        ldd     *number         ;y1
        addd    *yorg
        coma
        comb
        anda    #17
        std     *fracty+2
        cmpd    *fracty         ;order y0 and y1
        bhs     4$              ;y1 > y0
        ldx     *fracty
        std     *fracty
        stx     *fracty+2       ;now - y1 > y0

4$:     clrb                    ;set to color '0'
        stb     *$dsdat ;current state
        stb     bp$dat
        ldy     *fracty         ;load coordinants
5$:     sty     *crypos
```

```
        sty     vidy
        ldx     *fractx
6$:     stx     *crxpos
        stx     vidx
        stb     vidplt          ;write pixel
        leax    0d8,x           ;update x
        cmpx    *fractx+2       ;finished ?
        ble     6$              ;no - loop
        leay    0d8,y           ;update y
        cmpy    *fracty+2       ;finished ?
        ble     5$              ;no - loop
        bra     2$              ;loop for another region


clear:  pshs    cc
        orcc    #120            ;inhibit interrupts
        ldb     *$blank+1       ;blank screen
        stb     *$blank         ;current state
        stb     bp$dsp
        ldb     *$dsclr+1       ;clear display
        stb     *$dsclr         ;current state
        stb     bp$clr
        puls    cc
        ldb     #3              ;wait at least 1 vertical frame
        stb     *timflg         ;reset timing flag
1$:     tst     *timflg         ;done ?
        bne     1$
        pshs    cc
        orcc    #120            ;inhibit interrupts
        ldb     *$dsclr+2       ;end clearing display
        stb     *$dsclr         ;current state
        stb     bp$clr
        ldb     *$blank+2       ;enable display
        stb     *$blank
        stb     bp$dsp
        puls    cc
        rts                     ;finished

        .page
        .sbttl  $h and $i commands
.$h:    bsr     .$hi
        bne     .$h             ;loop until reset button hit
        rts


.$i:    jsr     nxtgos          ;wait for data

        ldd     *number         ;get .1 second increments
        pshs    a,b             ;save count
1$:     lda     #6
        sta     *timflg         ;start time interval
2$:     bsr     .$hi
        beq     3$              ;exit wait if cont hit
        tst     *timflg         ;.1 second yet ?
        bne     2$              ;no - loop
        ldd     ,s              ;get count
        subd    #1              ;finished ?
        std     ,s
        bgt     1$              ;no - loop
3$:     leas    2,s             ;pop counter
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        ora     #20
        sta     *outpl          ;turn lamp off
        sta     plout
        puls    cc
        rts

        .page

.$hi:   pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~20
        sta     *outpl          ;turn hold lamp on
        sta     plout
        puls    cc
```

```
1$:     lda     *rstat2
        bita    #10             ;video need clearing ?
        bne     3$              ;if not - skip
        jsr     clear           ;clear screen
2$:     lda     *rstat0
        bita    #10             ;button released ?
        beq     2$              ;loop until released
        lda     *rstat2
        ora     #10
        sta     *rstat2         ;clear entry flag
3$:
        .if     plotter
        jsr     calib           ;calibrate check
        .endif

        .if     printer
        lda     *rstat2
        bita    #2              ;screen dump ?
        bne     4$              ;no - skip
        jsr     screen          ;go dump screen
        .endif


4$:     lda     *rstat0
        bita    #1              ;cont button pushed ?
        bne     6$              ;no - exit
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        ora     #20
        sta     *outpl          ;turn lamp off
        sta     plout
        puls    cc
5$:     lda     *rstat0
        bita    #1              ;button released ?
        beq     5$              ;loop until released
        clra                    ;=0 indicates button hit
6$:     rts                     ;finished

        .page
        .sbttl  $x command

.$x:    lda     *rstat1         ;get status
        anda    #217            ;clear other modes
        ora     #40             ;set xstep mode
        sta     *rstat1         ;save
        jsr     nxtgos

        ldx     *number         ;get value
        stx     *xstep          ;save
        jsr     nxtgos

        ldx     *number         ;get number
        stx     *ystep          ;save
        jmp     proc            ;do process

        .page
        .sbttl  $y command

.$y:    lda     *rstat1         ;get status
        anda    #217            ;clear all modes
        ora     #20             ;set ystep mode
        sta     *rstat1         ;save
        jsr     nxtgos

        ldx     *number         ;get value
        stx     *ystep          ;save
        jsr     nxtgos

        ldx     *number         ;get value
        stx     *xstep          ;save
        jmp     proc            ;do process

        .page
        .sbttl  $z command

.$z:    jsr     nxtchr          ;get character
```

```
1$:     jsr     $dispatch       ;dispatch accordingly
        .byte   'L
        .word   3$              ;load selected alterable map
        .byte   'R
        .word   4$              ;load default map
        .byte   'S
        .word   6$              ;select alterable map
        .byte   -1              ;end of dispatch
        .word   2$              ;go here

2$:     jsr     nxtgos          ;wait for number

        ldb     *number+1       ;get value
        andb    #17             ;only 4 bits
        stb     *gcolor         ;save the color
        jsr     nxtgos          ;wait for parameters

        ldb     *number+1       ;get red value
        stb     *zrtmp
        jsr     nxtgos

        ldb     *number+1       ;get green value
        stb     *zgtmp
        jsr     nxtgos

        ldb     *number+1       ;get blue value
        stb     *zbtmp

        clra
        ldb     *gcolor         ;color #
        aslb
        addb    *gcolor         ;3-bytes per color
        addd    *altmap         ;rgb alternate graphics map
        tfr     d,y
        ldx     #zrtmp          ;new rgb values

        ldb     ,x+             ;get values
        stb     ,y+             ;and store in table
        ldb     ,x+
        stb     ,y+
        ldb     ,x+
        stb     ,y+
        bra     2$              ;and go wait for more

3$:     ldx     *altmap         ;<l> load alterable map
        bra     5$

4$:     ldx     #clrtbl         ;<r> load default map

5$:     bsr     ldrgb           ;load dac table
        bra     .$z

6$:     jsr     nxtchr          ;<s> select mode

        jsr     $dispatch       ;dispatch accordingly
        .byte   'A
        .word   7$              ;select map 'a' (alterable ram)
        .byte   'B
        .word   8$              ;select map 'b' (alterable ram)
        .byte   'C
        .word   9$              ;select map 'c' (alterable ram)
        .byte   'D
        .word   10$             ;select map 'd' (alterable ram)
        .byte   'R
        .word   12$             ;restore selected map to default map
        .byte   -1              ;end of dispatch
        .word   1$              ;loop to check main line options

7$:     ldx     #altmpa         ;<a> select alterable map 'a'
        bra     11$

8$:     ldx     #altmpb         ;<b> select alterable map 'b'
        bra     11$

9$:     ldx     #altmpc         ;<c> select alterable map 'c'
        bra     11$
```

```
10$:    ldx     #altmpd         ;<d> select alterable map 'd'

11$:    stx     *altmap         ;save map address
        bra     13$

12$:    bsr     lddflt          ;restore alterable map to default

13$:    lda     *lstat
        ora     #200
        sta     *lstat          ;character mode
        sec                     ;character used
        rts

;  copy default color table to alterable color table

lddflt: ldx     *altmap
iniflt: ldy     #clrtbl
cpytbl: lda     #0d48           ;triple 4-bit dac

1$:     ldb     ,y+
        stb     ,x+             ;user map
        deca                    ;finished ?
        bgt     1$              ;no - loop
2$:     rts

;  rgb dac map loader
;
;       the color table address must be in x
;
ldrgb:  inc     *timflg         ;wait for retrace
1$:     tst     *timflg
        bne     1$              ;loop until retrace time

inirgb: pshs    cc              ;use this to restore cc
        orcc    #120            ;hold interrupts
        lda     *$dssts
        bita    #12             ;display or 'oc' on ?
        lbeq    10$             ;no - skip

        tfr     x,y             ;y is source address
        ldx     #curmap         ;copy new data into curmap
        bsr     cpytbl
        bra     1$              ;then save a copy of curmap

setmap=.
        pshs    cc              ;use this to restore cc
        orcc    #120            ;hold interrupts
        lda     *$dssts
        bita    #12             ;histo/cursor or 'oc' on ?
        bne     2$              ;yes - then restore curmap

1$:     ldy     #curmap         ;else save copy of table
        ldx     #ds$map
        bsr     cpytbl
        bra     3$

rstmap=.
        pshs    cc              ;save
        ldx     #ds$map
        lda     *$dssts
        bita    #12             ;histo/cursor or 'oc' on ?
        beq     10$             ;no - restore everything

2$:     ldy     #ds$map         ;else restore map
        ldx     #curmap         ;for reprocessing
        bsr     cpytbl

3$:     lda     *$dssts
        bita    #14             ;histo/cursor on ?
        beq     6$              ;no - skip
        ldu     #hs$clr         ;check histo/cursor color
        ldb     1,u             ;red
        orb     3,u             ;green
        orb     5,u             ;blue
        bne     4$              ;if already defined - skip
        ldb     #17             ;else use default of white
        stb     1,u
        stb     3,u
```

```
        stb     5,u
4$:     ldx     #curmap+6       ;start at color #2
        lda     #0d14           ;through #15
5$:     ldb     1,u
        stb     ,x+
        ldb     3,u
        stb     ,x+
        ldb     5,u
        stb     ,x+
        deca
        bgt     5$

6$:     lda     *$dssts
        bita    #3              ;'oc' on ?
        beq     8$              ;no - skip
        ldx     #curmap+0d45    ;color #15
        ldu     #oc$clr         ;check 'oc' color
        ldb     1,u             ;red
        orb     3,u             ;green
        orb     5,u             ;blue
        bne     7$              ;if already defined - skip
        ldb     #17             ;else use default of green
        stb     3,u
7$:     ldb     1,u             ;color #15
        stb     ,x+
        ldb     3,u
        stb     ,x+
        ldb     5,u
        stb     ,x+

8$:     ldx     #curmap         ;copy current map to hardware
        ldy     #rgbmap         ;hardware map address
        lda     #0d16           ;4-bit dacs

9$:     ldb     ,x+
        stb     ,y+             ;r map
        ldb     ,x+
        stb     0d15,y          ;g map
        ldb     ,x+
        stb     0d31,y          ;b map

        deca                    ;more ?
        bgt     9$              ;yes - loop
        puls    cc              ;restore interrupt status
        rts                     ;else finished

10$:    ldy     #rgbmap         ;hardware map address
        ldu     #curmap         ;current map
        lda     #0d16           ;4-bit dacs

11$:    ldb     ,x+
        stb     ,y+             ;r map
        stb     ,u+
        ldb     ,x+
        stb     0d15,y          ;g map
        stb     ,u+
        ldb     ,x+
        stb     0d31,y          ;b map
        stb     ,u+

        deca                    ;more ?
        bgt     11$             ;yes - loop
        puls    cc              ;restore interrupt status
        rts                     ;else finished

        .page
        .sbttl  process routine

proc:   lda     *rstat1
        bita    #40             ;auto x mode ?
        beq     proc.a          ;if not - skip
        bsr     val.x           ;fxval=xval+xstep
        bsr     org.y           ;fyval=yorg+ystep
        bra     proc.d

proc.a: bita    #20             ;auto y mode ?
        beq     proc.b          ;if not - skip
        bsr     org.x           ;fxval=xorg+xstep
```

```
        bsr     val.y           ;fyval=yval+ystep
        bra     proc.d

proc.b: bita    #100            ;relative to current ?
        beq     proc.c          ;if not - skip
        bsr     val.x           ;fxval=xval+xstep
        bsr     val.y           ;fyval=yval+ystep
        bra     proc.d

proc.c: bsr     org.x           ;fxval=xorg+xstep
        bsr     org.y           ;fyval=yorg+ystep

proc.d: jsr     vector          ;set up vector lengths
        lda     *rstat1
        bita    #10             ;in vector mode ?
        bne     proc.e          ;if not - skip vector drawing
        jsr     vect.1          ;go do vector calculations
        jsr     plotvc          ;and then plot

        .if     plotter
        lda     *rstat1
        bita    #2              ;pen down ?
        beq     proc.e          ;no - skip
        jsr     .$dx            ;drop pen
        .endif

proc.e: ldx     *fxval          ;xval=fxval
        stx     *xval
        stx     *valx           ;valx=fyval
        ldx     *fyval          ;yval=fyval
        stx     *yval
        stx     *valy           ;valy=fyval
        jsr     video           ;plot last video point

        .if     plotter
        lda     *rstat1
        bita    #10             ;point mode ?
        beq     proc.f          ;if not - finished
        jmp     plotr           ;try plotter
        .endif

proc.f: rts                     ;finished


val.x:  ldx     #xorg
        bra     1$
val.y=.
        ldx     #yorg
1$:     ldd     4,x
        bra     3$

org.x=.
        ldx     #xorg
        bra     2$
org.y=.
        ldx     #yorg
2$:     ldd     ,x
3$:     addd    2,x
        std     6,x
        rts

        .page
        .sbttl  vector generation routine

vector: ldx     #xorg
        jsr     absd            ;abs [ fxval-xval ]
        ldx     #yorg
        jmp     absd            ;abs [ fyval-yval ]
                                ;vector lengths computed
vect.1: ldx     *vectlx         ;vectlx=vectly ?
        cmpx    *vectly
        bne     vect.2          ;if not - skip
        stx     *vecntr         ;save steps in vector
        ldx     #xorg           ;point at x
        lda     #1              ;update = 1
        jsr     vsetc           ;set update value
        jsr     setsgn          ;set sign of updating
        ldx     #yorg           ;point to y
```

```
        lbra    vep.nd

vect.2: ldy     *vectlx         ;is vectlx=0 ?
        bne     vect.3          ;if not - skip
        ldx     *vectly         ;save as vector count
        stx     *vecntr
        ldx     #xorg           ;point to x
        clra                    ;update = 0
        jsr     vsetc           ;set update value
        ldx     #yorg           ;point to y
        lbra    vep.nd

vect.3: ldy     *vectly         ;=0 ?
        bne     vect.4          ;if not - skip
        ldx     *vectlx         ;save as count
        stx     *vecntr
        ldx     #yorg           ;point to y
        clra                    ;update = 0
        jsr     vsetc           ;set update value
        ldx     #xorg           ;point to x
        bra     vep.nd

vect.4: ldd     *vectlx         ;which is longer ?
        subd    *vectly
        bcs     vect.5          ;if vectly>vectlx - skip
        ldx     *vectlx         ;vectlx is larger
        stx     *vecntr
        stx     *dsr            ;save as divisor
        ldx     *vectly
        stx     *dnd            ;save dividend
        jsr     divide          ;do 16-bit division
        ldx     #yorg           ;point to y
        clra                    ;set update = 0
        jsr     vsetc           ;set update
        ldd     qt              ;get 16-bit fraction
        std     *fracty+2       ;save fractional update
        jsr     setsgn          ;set sign of update
        ldx     #xorg           ;point to x
        bra     vep.nd

vect.5: ldx     *vectly         ;vectly is larger
        ldy     *vectlx
        stx     *vecntr         ;save as count
        stx     *dsr            ;and as divisor
        ldx     *vectlx
        stx     *dnd            ;save as dividend
        jsr     divide          ;do 16-bit division
        ldx     #xorg           ;point to x
        clra                    ;update = 0
        jsr     vsetc           ;set update value
        ldd     *qt             ;get 16-bit fraction
        std     *fractx+2       ;save fractional update
        jsr     setsgn          ;set sign of update
        ldx     #yorg           ;point to y
vep.nd: lda     #1              ;update = 1
        jsr     vsetc           ;set update value
        jmp     setsgn          ;set sign of update
        rts                     ;finished

        .page
        .sbttl  absolute vector length calculation

absd:   clr     22,x            ;set sign to +
        ldd     6,x             ;fval
        subd    4,x             ;- val
        bcc     1$              ;if positive - skip
        com     22,x            ;sign is -
        coma                    ;and make positive
        comb
        addd    #1
1$:     std     10,x            ;vectl
        rts                     ;finished

        .page
        .sbttl  set vector updating routine

vsetc:  clrb
        stb     16,x            ;set fract
```

```
        sta     17,x
        stb     20,x
        stb     21,x
        stb     15,x            ;set val to
        ldb     #200            ;_val + .5
        stb     14,x
        ldd     4,x
        std     12,x
        rts                     ;finished

        .page
        .sbttl  set sign of updating value

setsgn: tst     22,x            ;- sign ?
        beq     1$              ;if + done
        com     21,x
        com     20,x
        com     17,x
        com     16,x
        inc     21,x
        bne     1$
        inc     20,x
        bne     1$
        inc     17,x
        bne     1$
        inc     16,x
1$:     rts                     ;finished

        .page
        .sbttl  16-bit divide

divide: lda     #20             ;bit counter
        sta     *divcnt
        ldd     *dnd            ;get dividend
        aslb                    ;initial shift
        rola
1$:     subd    *dsr            ;subtract divisor
        bcc     2$              ;if carry clear - good test
        addd    *dsr            ;else restore dividend
2$:     rol     *qt+1           ;shift in computed bit
        rol     *qt
        aslb                    ;shift dnd
        rola
        dec     *divcnt         ;16 bits generated yet ?
        bne     1$              ;loop until done
        com     *qt+1           ;get true result
        com     *qt
        rts                     ;finished

        .page
        .sbttl  plot vector scanner

plotvc:
        .if     plotter
        tst     *pltron         ;plotter on ?
        bne     plotvp          ;yes - skip
        .endif

        ;fast video only plotting

        ldd     *vecntr         ;divide count by 8
        asra
        rorb
        lsra
        rorb
        lsra
        rorb
        std     *vecntr         ;save result
        ldx     #fractx         ;point to x fraction
        bsr     mul8            ;multiply by 8
        ldx     #fracty         ;point to y fraction
        bsr     mul8            ;multiply by 8
        ldx     *vecntr         ;points to plot
        leax    1,x             ;is vecntr+1
        stx     *vecntr
        bra     2$              ;go to entry

1$:     ldx     #xorg           ;point to x
```

```
        bsr     update          ;update x axis
        ldx     #yorg           ;point to y
        bsr     update          ;update y axis
2$:     jsr     video           ;try video
        ldx     *vecntr         ;get points to plot
        leax    -1,x            ;any more ?
        stx     *vecntr
        bne     1$              ;keep going
        rts                     ;finished

        .if     plotter
plotvp: ldx     *vecntr         ;points to plot
        leax    1,x             ;is vecntr+1
        stx     *vecntr
        jsr     video           ;do video first
        bra     5$              ;go to entry

1$:     lda     *rstat1
        bita    #2              ;pen 'down' ?
        beq     4$              ;no - skip
        andb    #7              ;time for video ?
        bne     4$              ;no - skip
        tst     plptrn          ;check pattern
        bpl     2$              ;pen 'up'
        jsr     .$dx            ;pen 'down'
        bra     3$
2$:     jsr     .$ux
3$:     jsr     video           ;then do video and rotate pattern
4$:     ldx     #xorg           ;point to x
        bsr     update          ;update x axis
        ldx     #yorg           ;point to y
        bsr     update          ;update y axis
5$:     ldd     *valx           ;get x axis
        std     *crxpos
        std     vidx            ;store
        ldd     *valy           ;get y axis
        comb                    ;complementary binary
        coma
        std     *crypos
        std     vidy            ;store
        jsr     plotr           ;try plotter
        ldd     *vecntr         ;get points to plot
        addd    #-1             ;any more ?
        std     *vecntr
        bne     1$              ;keep going
        rts                     ;finished
        .endif

        ;updating routine

update: ldd     14,x            ;32-bit add of val_ and fract_
        addd    20,x
        std     14,x
        ldd     12,x
        adcb    17,x
        adca    16,x
        std     12,x
        rts                     ;finished

mul8:   lda     #3              ;3 shifts
1$:     asl     3,x
        rol     2,x
        rol     1,x
        rol     ,x
        deca
        bne     1$              ;loop until finished
        rts                     ;finished


        .page
        .sbttl  video graphics handler

video:  ldd     *valx           ;get x axis
        std     *crxpos
        std     vidx            ;store
        ldd     *valy           ;get y axis
        comb                    ;complementary binary
        coma
```

```
        std     *crypos
        std     vidy            ;store
        lda     *rstat1
        bita    #2              ;pen up ?
        beq     7$              ;if so - don't plot
        lda     *rstat1
        bita    #10             ;vector/point ?
        bne     1$              ;point - skip
        ldd     .$ptrn          ;rotate pattern
        rorb
        ror     .$ptrn
        ror     .$ptrn+1
        rolb
        bra     2$
1$:     ldd     #-1
2$:
        .if     plotter
        std     plptrn          ;for plotter
        .endif

        rorb                    ;draw this bit ?
        bcc     7$              ;no - skip

        clrb                    ;init write color
        lda     *rstat1
        bita    #1              ;replace/complement mode ?
        beq     3$              ;no - skip

        ldb     vidplt          ;get color at point
        lda     *rstat1
        bita    #4              ;replace or complement ?
        beq     3$              ;replace - skip
        eorb    *gcolor         ;complement selected planes
        bra     4$

3$:     orb     *gcolor         ;set selected planes
4$:     stb     *$dsdat         ;current state
        stb     bp$dat
5$:     lda     *lstatx
        bita    #100            ;clipping ?
        bne     6$              ;no - skip
        ldd     *crypos
        cmpd    #-0d4096
        blo     7$
        ldd     *crxpos
        cmpd    #0d4096
        bhs     7$
        ldb     *$dsdat         ;get data
6$:     stb     vidplt          ;write pixel

7$:     rts                     ;finished

        .page
        .sbttl  plotter handler (optional)

        .if     plotter
plotr:  tst     *pltron         ;plotter on ?
        beq     4$              ;no - skip

        ;       x-y recorder handler

        lda     *rstat1
        bita    #10             ;in vector mode ?
        bne     5$              ;if not - point mode

        ;       vector mode

        lda     *lstatx
        bita    #100            ;clipping ?
        bne     1$              ;no - skip
        ldd     *crypos
        cmpd    #-0d4096
        blo     2$
        ldd     *crxpos
        cmpd    #0d4096
        bhs     2$
1$:     tst     ld.dac          ;load dacs with data
                                ;from vidx & vidy
```

```
2$:     ldx     waitxy          ;get x-y wait count
        leax    1,x             ;at least one
3$:     leax    -1,x
        bne     3$
4$:     rts                     ;finished

        ;       point mode

5$:     lda     *vectly         ;get vector lengths
        adda    *vectlx
        asla                    ;shift 3 places
        asla
        asla
        ldb     #5
6$:     asla                    ;find msb set
        bcs     7$
        decb
        bgt     6$
7$:     incb
        stb     wtlp            ;save loop time factor

        lda     *rstat1
        bita    #2              ;pen down ?
        beq     plotf           ;if not - skip
        jsr     .$u             ;move pen up
        bsr     plotf           ;set dacs and wait
        jsr     .$d             ;pen down
        rts                     ;finished
plotf:  lda     *lstatx
        bita    #100            ;clipping ?
        bne     1$              ;no - skip
        ldd     *crypos
        cmpd    #-0d4096
        blo     2$
        ldd     *crxpos
        cmpd    #0d4096
        bhs     2$
1$:     tst     ld.dac          ;load dacs
2$:     ldb     wtlp            ;get loop factor
3$:     ldx     waitmv          ;wait awhile
        jsr     .$ua
        decb                    ;finished factors ?
        bgt     3$              ;if not - loop
        rts                     ;finished

        .page
        .sbttl  plotter calibration routine

calib:  tst     *pltron         ;plotter on ?
        lbeq    7$              ;no - skip
        lda     *rstat2
        bita    #4              ;calibration ?
        lbne    7$              ;if not - skip
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl          ;get pen control
        tfr     a,b
        andb    #24             ;mask pen / hold
        anda    #353            ;lift pen / turn hold lamp on
        sta     *outpl
        sta     plout
        puls    cc
        stb     ,-s             ;save for restoration
        jsr     .$uw            ;lift pen wait
        ldx     *crxpos         ;save current position
        ldy     *crypos
        pshs    x,y
1$:     ldx     #0              ;x=y=0
        ldy     #0d4095
        bsr     2$
        ldx     #0d4095         ;x=y=4095
        ldy     #0
        bsr     2$
        bra     1$
2$:     stx     *crxpos
        stx     vidx
        sty     *crypos
        sty     vidy
```

```
        tst     ld.dac          ;load dacs
3$:     lda     *rstat0
        bita    #4              ;button released ?
        beq     3$              ;loop until released
        lda     *rstat2
        ora     #4
        sta     *rstat2         ;clear entry flag
4$:     lda     *rstat2
        bita    #4              ;another calibrate ?
        bne     5$              ;if not skip
        rts
5$:     lda     *rstat0
        bita    #1              ;reset yet ?
        bne     4$              ;if not just loop
        leas    2,s             ;pop return address
        puls    x,y
        stx     *crxpos         ;restore pen position
        stx     vidx
        sty     *crypos
        sty     vidy
        lda     #5              ;set up wait modifier
        sta     wtlp
        jsr     plotf           ;set dacs and wait for move
        puls    a               ;get previous pen / hold
        pshs    cc
        orcc    #120            ;hold interrupts
        ora     *outpl
        sta     *outpl
        sta     plout
        puls    cc
        jsr     .$dw            ;drop pen wait
6$:     lda     *rstat0
        bita    #1              ;button released ?
        beq     6$              ;loop until released
        lda     *rstat2
        ora     #1
        sta     *rstat2         ;clear entry flag
7$:     lda     *rstat2
        ora     #4
        sta     *rstat2         ;always clear cal flag
        rts                     ;finished
        .endif

        .page
        .sbttl  $f command

.$f:    lda     *lstat          ;re-enable character mode
        ora     #200
        anda    #367            ;x size first
        sta     *lstat
        lda     *rstat1
        anda    #~5
        sta     *rstat1         ;default pixel mode
        ldd     #-1             ;default pattern
        ldx     #.$ptrn
        stx     *qt             ;temporary pointer
1$:     std     ,x++
        cmpx    #.$ptrn+0d34
        bne     1$              ;loop until all set to default
        jsr     nxtgos

        jsr     $dispatch       ;dispatch accordingly
        .byte   'H
        .word   2$
        .byte   'V
        .word   3$
        .byte   'L
        .word   4$
        .byte   'G
        .word   5$
        .byte   '/
        .word   6$
        .byte   '\\
        .word   7$
        .byte   'C
        .word   8$
        .byte   'R
        .word   9$
```

```
        .byte   'P
        .word   10$
        .byte   'B
        .word   11$
        .byte   -1              ;end dispatch scan
        .word   12$             ;and go here


2$:     lda     *lstat
        anda    #~1
        sta     *lstat          ;horizontal mode
        lbra    16$

3$:     lda     *lstat
        ora     #1
        sta     *lstat          ;vertical mode
        bra     16$

4$:     lda     *lstat
        anda    #~4
        sta     *lstat          ;lower case
        bra     16$

5$:     lda     *lstat
        ora     #4
        sta     *lstat          ;substitute greek for lower case
        bra     16$

6$:     lda     *lstatx
        anda    #~200
        sta     *lstatx         ;slashed '0'
        bra     16$

7$:     lda     *lstatx
        ora     #200
        sta     *lstatx         ;non-slashed '0'
        bra     16$

8$:     lda     *rstat1
        ora     #5
        sta     *rstat1         ;complement selected planes mode
        bra     16$

9$:     lda     *rstat1
        ora     #1
        sta     *rstat1         ;replace selected planes mode
        bra     16$

10$:    lda     *lstatx
        anda    #~1
        sta     *lstatx         ;plotter mode sizing
        bra     16$

11$:    lda     *lstatx
        ora     #1
        sta     *lstatx         ;block mode sizing
        bra     16$

12$:    lda     *char           ;get character
        suba    #'0             ;make into bcd
        bcs     17$             ;error - skip
        cmpa    #0d9
        bhi     17$             ;error - skip
        pshs    a               ;save

        ldb     *lstatx         ;plotter/block mode ?
        bitb    #1
        beq     13$             ;plotter - skip
        inca
        asla
        asla
        asla                    ;*8
        bra     14$
13$:    ldx     #xytbl          ;plotter sizing table
        lda     a,x             ;get size specification
14$:    ldb     *lstat          ;x or y
        bitb    #10
        bne     15$
```

```
        orb     #10             ;next is y
        stb     *lstat          ;save
        sta     *xsize          ;save new xsize
        puls    a
        inca
        sta     *cxzoom         ;and block x-zoom
        bra     16$
15$:    andb    #367            ;next is x
        stb     *lstat          ;save
        sta     *ysize          ;save new ysize
        puls    a
        inca
        sta     *cyzoom         ;and block y-zoom

16$:    lda     *lstat
        ora     #200
        sta     *lstat          ;re-enable character mode
        sec                     ;character used
        rts                     ;and wait for another character

17$:    clc                     ;character not used
        jsr     nxtgos          ;wait for pattern data

        ldd     *number         ;get pattern
        ldx     *qt
        std     ,x++            ;save pattern
        cmpx    #.$ptrn+0d34
        bne     18$
        ldx     #.$ptrn
18$:    stx     *qt
        rts

xytbl:  .byte   0d4, 0d6, 0d8, 0d10, 0d12, 0d14, 0d16, 0d20, 0d26, 0d32

        .page
        .sbttl  $c command

.$c:    jsr     .$u             ;always raise pen
        lda     *rstat1         ;get mode status
        anda    #217            ;clear current mode
        ora     #100            ;$l mode
        sta     *rstat1         ;save
1$:     lda     *lstat          ;get status
        ora     #200            ;enable character mode
        anda    #375            ;character scan
        sta     *lstat
        sec                     ;character used
        jsr     nxtgos

2$:     jsr     $dispatch       ;dispatch accordingly
        .byte   'L
        .word   11$             ;load new character
        .byte   'R
        .word   10$             ;restore character set
        .byte   'S
        .word   3$              ;select character set
        .byte   -1              ;end dispatch scan
        .word   chrscn          ;and go here

3$:     jsr     nxtchr          ;get character

        jsr     $dispatch       ;dispatch accordingly
        .byte   'A
        .word   4$              ;character set 'a' (alterable)
        .byte   'B
        .word   5$              ;character set 'b' (rom)
        .byte   'C
        .word   6$              ;character set 'c' (rom)
        .byte   'D
        .word   7$              ;character set 'd' (rom)
        .byte   -1              ;end of dispatch
        .word   2$

4$:     ldd     #chram          ;character set 'a'
        bra     9$

5$:     ldd     #r6571a         ;character set 'b'
        bra     8$              ;this is default character set
```

```
6$:     ldd     #vt1xx          ;character set 'c'
        bra     8$

7$:     ldd     #figrom         ;character set 'd'
8$:     std     *romset         ;rom selection

9$:     std     *ptchar
        bra     3$

10$:    ldy     *romset         ;source character rom
        jsr     romcopy         ;restore ram
        bra     1$

11$:    sec                     ;character used
        jsr     nxtgos          ;wait for character code

        lda     *number+1       ;get character number
        anda    #177            ;128 characters in ram
        ldb     #0d16           ;*16.
        mul
        addd    #chram          ;base address
        std     *chpntr         ;save address
        jsr     nxtgos          ;wait for bytes

        lda     *number+1       ;get byte
        ldx     *chpntr         ;and address
        sta     ,x+             ;update data and address
        cmpx    #chram+0d2048   ;at end ?
        bne     12$
        ldx     #chram
12$:    stx     *chpntr         ;save address
        rts                     ;and wait for next byte

13$:    cmpb    #'+             ;'+' ?
        bne     14$             ;no - exit
        ora     #2              ;now plotting
        sta     *lstat
        jsr     .$u             ;always raise pen
        sec                     ;character used
        bra     15$
14$:    clc                     ;character not used
15$:    jsr     nxtgos          ;wait for next character
chrscn=.
        ldb     *char           ;get character
        lda     *lstat
        ora     #200
        sta     *lstat          ;character mode
        bita    #2              ;plotting ?
        beq     13$             ;no - branch

        lda     *lstat
        bita    #4              ;lower/greek
        beq     chrs.d          ;if lower - skip
        cmpb    #140            ;in range of greeks ?
        bcs     chrs.d          ;if not - skip
        bne     chrs.c          ;if so - skip
        ldb     #37             ;out of place character
        bra     chrs.d
chrs.c: subb    #141            ;get to greeks
chrs.d: lda     *rstat1
        bita    #10             ;vector mode ?
        bne     chrs.l          ;if not - point mode
chrscx: cmpb    #'0             ;a '0' ?
        bne     1$              ;if not - skip
        lda     *lstatx
        bita    #200            ;non-slashed '0' ?
        beq     1$              ;if not - skip
        ldb     #0d138          ;pointer to 'nsv0'
1$:     clra                    ;find entry in table
        aslb                    ;two bytes per pointer
        rola
        ldx     #table
        ldx     d,x             ;get address of string
chrs.e: lda     ,x+             ;get vector
        stx     *chpntr         ;save address
        cmpa    #end            ;end ?
        beq     chrs.i          ;end of this character
```

```
        cmpa    #dwn            ;a pen control ?
        bne     chrs.f          ;if not skip
        jsr     .$d             ;move pen down
        bra     chrs.h
chrs.f: cmpa    #up             ;pen control ?
        bne     chrs.g          ;if not - skip
        jsr     .$u             ;lift pen
        bra     chrs.h
chrs.g: jsr     getxst          ;get x vector
        jsr     getyst          ;get y vector
        jsr     proc            ;go process
chrs.h: ldx     *chpntr         ;get vector pointer
        bra     chrs.e          ;get next vector
chrs.i: sec                     ;used character
        rts                     ;finished

        ;point mode

chrs.l: cmpb    #'0             ;a '0' ?
        bne     1$              ;if not - skip
        lda     *lstatx
        bita    #200            ;non-slashed '0' ?
        beq     1$              ;if not - skip
        ldd     table+0d278     ;address of 'nsp0'
        addd    #0d16
        bra     2$
1$:     lda     #0d16           ;compute address in ptchar
        mul
        addd    #0d16
        addd    *ptchar
2$:     std     *chpntr
        lda     *rstat1
        ora     #2
        sta     *rstat1         ;say pen down (physically its up !)
        lda     #10             ;preset pxcnt
        sta     *pxcnt
        lda     #0d9            ;preset pycnt
        sta     *pycnt
        lda     #0d16           ;preset ycnt
        sta     *ycnt
chrs.m: ldx     *chpntr         ;get pointer to points
        lda     ,-x             ;get dot pattern
        stx     *chpntr         ;save pointer
        tsta                    ;any points ?
        beq     chrs.p          ;if no points - skip
        sta     *prow           ;save pattern
        lda     #10             ;dot count
        sta     *xcnt
chrs.n: asl     *prow           ;dot here ?
        bcc     chrs.o          ;if not skip
        jsr     pgtxst          ;get xstep value
        jsr     pgtyst          ;get ystep value
        jsr     proc            ;go process point
chrs.o: dec     *xcnt           ;done yet ?
        bgt     chrs.n          ;loop until row done
chrs.p: dec     *ycnt           ;all rows done ?
        bgt     chrs.m          ;loop until done
        jsr     .$u             ;lift pen
        clr     *xcnt           ;set up for final move
        jsr     pgtxst          ;set xstep value
        lda     #0d9            ;set up ycnt
        sta     *ycnt
        jsr     pgtyst          ;set ystep value
        jsr     proc            ;go process
        sec                     ;character used
        rts                     ;finished

        .page
        .sbttl  x/y get routines

xget:   ldb     -1,x            ;get vector
        asrb                    ;shift into position
        asrb
        asrb
        asrb
        bra     1$
yget=.
        ldb     -1,x            ;get vector
```

```
1$:     andb    #17             ;mask junk
        bitb    #10             ;negative vector ?
        beq     2$              ;if not - finished
        andb    #7              ;save only magnitude
        negb                    ;two's complement
2$:     rts                     ;finished

        .page
        .sbttl  generate x step routine

        ;point mode

pgtxst: lda     *xcnt           ;get location to plot
        ldb     *pxcnt          ;previous point
        sta     *pxcnt          ;save for next time
        subb    *xcnt           ;pxcnt-xcnt
        bra     getx.x          ;go finish

        ;vector mode

getxst: bsr     xget            ;get x vector
getx.x: lda     *xsize
        bsr     multp           ;go multiply <a>*<b>
        ror     *lstat          ;horizontal ?
        bcs     1$              ;if not - skip
        rol     *lstat          ;restore lstat
        std     *xstep          ;save result
        rts
1$:     rol     *lstat          ;restore lstat
        std     *ystep          ;save result
        rts                     ;finished

        .page
        .sbttl  generate y step routine

        ;point mode

pgtyst: lda     *ycnt           ;current position to plot
        ldb     *pycnt          ;previous position
        sta     *pycnt          ;save for next time
        subb    *ycnt           ;find relative move
        bra     gety.x

        ;vector mode

getyst: bsr     yget            ;get y vector
gety.x: lda     *ysize          ;get y size
        bsr     multp           ;go multiply <a>*<b>
        ror     *lstat          ;horizontal ?
        bcs     1$              ;if not - skip
        rol     *lstat          ;restore lstat
        std     *ystep          ;save result
        rts                     ;finished
1$:     rol     *lstat          ;restore lstat
        coma                    ;need negative
        comb
        addd    #1
        std     *xstep          ;save result
        rts                     ;finished

        .page
        .sbttl  multiply routine

multp:  stb     ,-s             ;save sign
        bpl     1$
        negb
1$:     mul                     ;<a,b>=<a>*<b>
        tst     ,s+             ;sign ?
        bpl     2$              ;plus - skip
        coma
        comb
        addd    #1
2$:     rts                     ;finished - result <a,b>

        .page
        .sbttl  rom copy routine

romcopy:                        ;enter with y = source
```

```
        ldx     #0d1024         ;1024. words
        ldu     #chram          ;destination
1$:     ldd     ,y++            ;copy
        std     ,u++
        leax    -1,x
        bne     1$
        rts


        .page
        .sbttl  $b command

        ;       block character variables

        blnctr  =       pxcnt   ;line counter
        bdtctr  =       pycnt   ;dot counter
        bxzoom  =       xcnt    ;running x-zoom
        byzoom  =       ycnt    ;running y-zoom
        bvalx   =       svxstp  ;running valx
        bvaly   =       svystp  ;running valy

.$b:    lda     *rstat1         ;get mode status
        anda    #217            ;clear current mode
        ora     #100            ;$l
        sta     *rstat1         ;save
1$:     lda     *lstat          ;character mode
        ora     #200
        anda    #375            ;scanning
        sta     *lstat
        sec                     ;character used
        jsr     nxtgos

2$:     jsr     $dispatch       ;dispatch accordingly
        .byte   'L
        .word   11$             ;load new character definitions
        .byte   'R
        .word   10$             ;restore character rom
        .byte   'S
        .word   3$              ;select character set
        .byte   -1              ;end of dispatch
        .word   16$

3$:     jsr     nxtchr          ;get character

        jsr     $dispatch       ;dispatch accordingly
        .byte   'A
        .word   4$              ;character set 'a' (alterable)
        .byte   'B
        .word   5$              ;character set 'b' (rom)
        .byte   'C
        .word   6$              ;character set 'c' (rom)
        .byte   'D
        .word   7$              ;character set 'd' (rom)
        .byte   -1              ;end of dispatch
        .word   2$              ;loop to check main options

4$:     ldd     #chram          ;character set 'a'
        bra     9$

5$:     ldd     #r6571a         ;character set 'b'
        bra     8$              ;default character set

6$:     ldd     #vt1xx          ;character set 'c'
        bra     8$

7$:     ldd     #figrom         ;character set 'd'
8$:     std     *romset         ;rom selection

9$:     std     *ptchar
        bra     3$

10$:    ldy     *romset         ;source character rom
        jsr     romcopy         ;restore ram
        bra     1$

11$:    sec                     ;character used
        jsr     nxtgos          ;wait for character code
```

```
        lda     *number+1       ;get character number
        anda    #177            ;128 characters in ram
        ldb     #0d16           ;*16.
        mul
        addd    #chram          ;base address
        std     *chpntr         ;save address
        jsr     nxtgos          ;wait for bytes

        lda     *number+1       ;get byte
        ldx     *chpntr         ;and address
        sta     ,x+             ;update data and address
        cmpx    #chram+0d2048   ;at end ?
        bne     12$
        ldx     #chram
12$:    stx     *chpntr         ;save address
        rts                     ;and wait for next byte

13$:    cmpb    #'+             ;'+' ?
        bne     14$             ;no - exit
        ora     #2              ;now plotting
        sta     *lstat
        jsr     .$u             ;always raise pen
        sec                     ;character used
        bra     15$
14$:    clc                     ;character not used
15$:    jsr     nxtgos          ;wait for next character

16$:    ldb     *char           ;get character
        lda     *lstat
        ora     #200
        sta     *lstat          ;character mode
        bita    #2              ;plotting ?
        beq     13$             ;no - branch
        lda     *lstat
        bita    #4              ;lower/greek
        beq     18$             ;lower - skip
        cmpb    #140            ;in greek range ?
        bcs     18$             ;no - skip
        bne     17$             ;yes - skip
        ldb     #37             ;out of place character
        bra     18$
17$:    subb    #141            ;make greek
18$:    cmpb    #'0             ;a '0' ?
        bne     19$             ;no - skip
        lda     *lstatx
        bita    #200            ;non-slashed zero ?
        beq     19$             ;no - skip
        ldd     table+0d278     ;address of 'nsp0'
        bra     20$
19$:    lda     #0d16           ;compute data address
        mul
        addd    *ptchar
20$:    std     *chpntr         ;save address of character data

        ldd     *xval           ;get x axis
        std     *bvalx
        ldd     *yval           ;get y axis
        comb                    ;complementary binary
        coma
        std     *bvaly

        ldb     #0d16           ;character line counter
        stb     *blnctr

        lda     #0d8*0d8        ;character width offset
        ldb     *cxzoom
        mul
        std     *xstep          ;update value
        std     *ystep

        lda     #0d8*0d9        ;character height offset
        ldb     *cyzoom
        mul
        subd    #0d8
        std     ,--s

        lda     *lstat
        bita    #1              ;horizontal / vertical
```

```
        lbne    29$             ;vertical - skip

        ;       horizontal mode

        ldd     *bvaly
        subd    ,s++
        std     *bvaly           ;top of character y-value
        ldy     *bvaly
        ldd     #0
        std     *ystep          ;no final y-update
21$:    lda     *cyzoom         ;reset y-zoom
        sta     *byzoom
22$:    ldx     *chpntr         ;get character
        lda     ,x
        sta     *char
        sty     *crypos
        sty     vidy            ;y position
        ldx     *bvalx          ;reset x-position
        lda     #0d8            ;reset dot counter
        sta     *bdtctr
23$:    lda     *cxzoom         ;reset x-zoom
        sta     *bxzoom
24$:    stx     *crxpos
        stx     vidx            ;x-position
        lda     *char           ;check character bit
        rola                    ;draw this bit ?
        bcc     28$             ;no - skip

        clrb                    ;init write color
        lda     *rstat1
        bita    #1              ;replace/complement mode ?
        beq     25$             ;no - skip

        ldb     vidplt          ;get color at point
        lda     *rstat1
        bita    #4              ;replace or complement ?
        beq     25$             ;replace - skip
        eorb    *gcolor         ;complement selected planes
        bra     26$

25$:    orb     *gcolor         ;set selected planes
26$:    stb     *$dsdat         ;current state
        stb     bp$dat
27$:    stb     vidplt          ;write pixel

28$:    leax    0d8,x           ;update x-position
        dec     *bxzoom         ;more zoom ?
        bgt     24$             ;yes - loop
        rol     *char           ;next character dot position
        dec     *bdtctr         ;more dots in this row ?
        bgt     23$             ;yes loop
        leay    0d8,y           ;update y-position
        dec     *byzoom         ;more zoom ?
        bgt     22$             ;yes - loop
        ldx     *chpntr         ;else - next row
        leax    1,x
        stx     *chpntr
        dec     *blnctr         ;more lines to do ?
        bgt     21$             ;yes - loop
        ldd     *xval           ;final update
        addd    *xstep
        std     *fxval
        std     *xval
        std     *valx
        sec                     ;character used
        rts                     ;wait for next character

        ;       vertical mode

29$:    ldd     *bvalx
        subd    ,s++
        std     *bvalx           ;top of character x-value
        ldx     *bvalx
        ldd     #0
        std     *xstep          ;no final x-update
30$:    lda     *cxzoom         ;reset x-zoom
        sta     *bxzoom
31$:    ldy     *chpntr         ;get character
```

```
        lda     ,y
        sta     *char
        stx     *crxpos
        stx     vidx            ;x-position
        ldy     *bvaly          ;reset y-position
        lda     #0d8            ;reset dot counter
        sta     *bdtctr
32$:    lda     *cyzoom         ;reset y-zoom
        sta     *byzoom
33$:    sty     *crypos
        sty     vidy            ;y-position
        lda     *char           ;check character bit
        rola                    ;draw this bit ?
        bcc     37$             ;no - skip

        clrb                    ;init write color
        lda     *rstat1
        bita    #1              ;replace/complement mode ?
        beq     34$             ;no - skip

        ldb     vidplt          ;get color at point
        lda     *rstat1
        bita    #4              ;replace or complement ?
        beq     34$             ;replace - skip
        eorb    *gcolor         ;complement selected planes
        bra     35$

34$:    orb     *gcolor         ;set selected planes
35$:    stb     *$dsdat         ;current state
        stb     bp$dat
36$:    stb     vidplt          ;write pixel

37$:    leay    -0d8,y          ;update y-position
        dec     *byzoom         ;more zoom ?
        bgt     33$             ;yes - loop
        rol     *char           ;next character dot position
        dec     *bdtctr         ;more dots in this row ?
        bgt     32$             ;yes loop
        leax    0d8,x           ;update x-position
        dec     *bxzoom         ;more zoom ?
        bgt     31$             ;yes - loop
        ldy     *chpntr         ;else - next row
        leay    1,y
        sty     *chpntr
        dec     *blnctr         ;more lines to do ?
        bgt     30$             ;yes - loop
        ldd     *yval           ;final update
        addd    *ystep
        std     *fyval
        std     *yval
        std     *valy
        sec                     ;character used
        rts                     ;wait for next character

        .page
        .sbttl  $g command

.$g:    ldx     *gosub          ;get jump location
        stx     *svgsub         ;save for restoration
        jsr     nxtchr          ;get character

        ldx     *svgsub         ;restore jump address
        stx     *gosub
        ldb     *char           ;get character
        subb    #'0             ;make bcd
        bcs     1$              ;bad character - done
        cmpb    #0d9
        bhi     1$              ;bad character - done
        pshs b                  ;save character
        jsr     .$u             ;raise pen
        ldx     *xstep          ;save xstep
        stx     *svxstp
        ldx     *ystep          ;save ystep
        stx     *svystp
        lda     *rstat1         ;get status
        sta     *svstat         ;save for restoration
        anda    #207            ;make vector
        ora     #100            ;and relative ($l)
```

```
        sta     *rstat1         ;save new status
        puls b                  ;get character
        addb    #200            ;get to gch's
        jsr     chrscx          ;go do plotting
        lda     *svstat         ;get old status
        sta     *rstat1         ;restore mode
        ldx     *svystp         ;restore ystep
        stx     *ystep
        ldx     *svxstp         ;restore xstep
        stx     *xstep
        sec                     ;character used
        rts                     ;finished
1$:     clc                     ;character not used
        rts                     ;finished

        .page
        .sbttl  $e command

        ;  $e controller communication setup

.$e:    jsr     nxtchr          ;get character

        jsr     $dispatch       ;dispatch accordingly
        .byte   '0
        .word   1$
        .byte   '1
        .word   2$
        .byte   '2
        .word   3$
        .byte   '3
        .word   4$
        .byte   '4
        .word   5$
        .byte   '5
        .word   6$
        .byte   '6
        .word   7$
        .byte   '7
        .word   8$
        .byte   '8
        .word   9$
        .byte   '9
        .word   10$
        .byte   'A
        .word   11$
        .byte   'B
        .word   12$
        .byte   'C
        .word   13$
        .byte   'D
        .word   14$
        .byte   'E
        .word   15$
        .byte   'F
        .word   16$
        .byte   'K
        .word   18$
        .byte   'W
        .word   19$
        .byte   'Y
        .word   23$
        .byte   'Z
        .word   20$
        .byte   0               ;end of table

1$:     ;enable xon/xoff 'term' ;no hardware !!!
2$:     ;disable xon/xoff 'term'
3$:     ;enable hardware 'term'
4$:     ;disable hardware 'term'
        bra     .$e

5$:     lda     *d.stat
        ora     #1
        sta     *d.stat         ;enable xon/xoff 'link'
        bra     .$e
6$:     lda     *d.stat
        anda    #~1
        sta     *d.stat         ;disable xon/xoff 'link'
```

```
        bra     .$e

7$:     ;enable hardware 'link' ;automatic !!
8$:     ;disable hardware 'link'
        bra     .$e

9$:     sec                     ;character used
        jsr     nxtgos          ;come back with data
        ldb     *number+1       ;low order
        andb    #177            ;only 7-bit
        stb     *$xon           ;new 'xon' character
        jmp     endgos

10$:    sec                     ;character used
        jsr     nxtgos          ;come back with data
        ldb     *number+1       ;low order
        andb    #177            ;only 7-bit
        stb     *$xoff          ;new 'xoff' character
        jmp     endgos

11$:    ldy     #a.crtc ;512 x 480 display
        bra     17$             ;525 line interlaced

12$:    ldy     #b.crtc ;512 x 512 display
        bra     17$             ;553 line interlaced

13$:    ldy     #c.crtc ;512 x 512 display
        bra     17$             ;625 line interlaced

14$:    ldy     #d.crtc ;512 x 480 display
        bra     17$             ;533 line interlaced

15$:    ldy     #e.crtc ;512 x 512 display
        bra     17$             ;627 line interlaced

16$:    ldy     #f.crtc ;512 x 512 display
                                ;560 line non interlaced

17$:    ldx     #a.6845
        jsr     i6845$
        lbra    .$e

18$:    lda     *lstatx
        anda    #~100
        sta     *lstatx         ;clipping enabled
        lbra    .$e

19$:    lda     *lstatx
        ora     #100
        sta     *lstatx         ;wrap around enabled
        lbra    .$e

20$:    lda     #0d12           ;set up loop counter
        sta     *divcnt         ;for bytes of data
        ldx     #24$            ;buffer location
        stx     *qt
        sec                     ;character used

21$:    jsr     nxtgos          ;come back with data
        ldb     *number+1       ;get byte
        ldx     *qt             ;place byte
        stb     ,x+             ;and update buffer pointer
        stx     *qt
        dec     *divcnt
        bgt     21$             ;loop for byte data

        lda     #2              ;word data count
        sta     *divcnt

22$:    jsr     nxtgos          ;come back with data
        ldd     *number         ;get word
        ldx     *qt             ;place word
        std     ,x++            ;and update buffer pointer
        stx     *qt
        dec     *divcnt
        bgt     22$

23$:    ldy     #24$            ;pointer to table
```

```
        lda     ,y              ;have parameters ?
        lbeq    .$e             ;no - finished
        bra     17$

        .area   BUFSAV

24$:    .blkb   0d16            ;crtc data

        .area   PGMSAV


        .page
        .sbttl  $n command

.$n:    jsr     nxtchr          ;get character

        jsr     $dispatch       ;dispatch accordingly
        .byte   'B
        .word   1$              ;blank screen
        .byte   'U
        .word   2$              ;unblank screen
        .byte   0               ;end of dispatch

1$:     pshs    cc
        orcc    #120            ;inhibit interrupts
        lda     *$blank+1       ;<b>-blank screen
        bra     3$

2$:     pshs    cc
        orcc    #120            ;inhibit interrupts
        lda     *$blank+2       ;<u>-unblank screen
3$:     sta     *$blank         ;current state
        sta     bp$dsp          ;set display enable control
        puls    cc
        bra     .$n

        .page
        .sbttl  $t command

        .if     printer
.$t:    jsr     nxtchr          ;get character

        jsr     $dispatch       ;dispatch accordingly
        .byte   'L
        .word   .$tl
        .byte   'H
        .word   .$th
        .byte   'G
        .word   .$tg
        .byte   'C
        .word   .$tc
        .byte   'D
        .word   .$td
        .byte   'V
        .word   .$tv
        .byte   'S
        .word   .$ts
        .byte   'T
        .word   .$tt
        .byte   -1
        .word   errgos

.$tl:   clra                    ;low density
        bra     1$
.$th=.
        lda     #-1             ;high density
        bra     1$
.$tg=.
        lda     #1              ;plotter graphics
1$:     sta     *$tdens
        sec
        jsr     nxtgos          ;come back with number

        ldb     *number+1       ;zoom factor
        beq     2$
        decb
        andb    #17             ;maximum zoom = 16
2$:     stb     *$tzoom
```

```
        jsr     nxtgos

        ldd     *number         ;column offset
        std     *$tofst
        jsr     nxtgos

        ldd     *number ;first x
        std     *$tx1
        jsr     nxtgos

        ldd     *number ;last x
        std     *$tx2
        jsr     nxtgos

        ldd     *number ;first y
        std     *$ty1
        jsr     nxtgos

        ldd     *number ;last y
        std     *$ty2

        ldu     #$trmap         ;u is pointer most of the time

        ldd     *$tofst         ;offset in picture units
        jsr     tscale
        std     *$pofst

        lda     *$tzoom         ;set zoom
        sta     *$pzoom
        sta     *$zoom0
        sta     ,-s             ;push 2 copies of the
        sta     ,-s             ;zoom factor

        ldd     *$ty2           ;compute y lines
        subd    *$ty1
        jsr     tscale          ;scale
        std     *$line0         ;save lines
        bra     4$
3$:     addd    *$line0
4$:     dec     ,s              ;more zoom ?
        bge     3$              ;loop
        std     *$line0         ;save result
        leas    1,s             ;pop zoom counter

        ldd     *$tx2           ;compute x dots
        subd    *$tx1
        jsr     tscale          ;scale
        std     *$dots          ;save dots in line
        bra     6$
5$:     addd    *$dots
6$:     dec     ,s              ;more zoom ?
        bge     5$              ;loop
        std     *$dots          ;save result
        leas    1,s             ;pop zoom counter

        addd    *$pofst         ;add offset
        std     *$plen          ;and total dots in line

        jsr     tcursr          ;load cursor position

        lda     *$tdens         ;set density
        sta     *$pdens

        lda     *$pntrs         ;dispatcher
        ldx     #9$
        jsr     [a,x]
        jmp     endgos
7$:     ldd     $xtrnl          ;a driver there ?
        beq     8$              ;no - skip
        subd    $xtrnl+2        ;verify entry
        cmpd    #0x5AA5         ;check code
        bne     8$              ;invalid entry
        jsr     [$xtrnl]        ;else use driver
        lda     #>direct        ;restore direct pointer
        tfr     a,dp
        jmp     endgos
8$:     leas    2,s             ;pop return
        jmp     badcom          ;no handler
```

```
9$:                             ;printer options
        .if     hplaser
        .word   $hp
        .endif

        .if     tektronix
        .word   $tktnx
        .endif

        .if     epson
        .word   $epson
        .endif

        .if     hplaser
        .else
        .word   8$
        .endif

        .if     tektronix
        .else
        .word   8$
        .endif

        .if     epson
        .else
        .word   8$
        .endif

        .word   7$


.$tc:   sec                     ;wait for number
        jsr     nxtgos

        ldd     *number         ;character count
        std     *$tcntr
        lble    endgos          ;exit if '0' characters
        lda     *char           ;check for '+'
1$:     cmpa    #'+
        beq     2$              ;yes - skip
        lda     *lstat
        ora     #200
        sta     *lstat          ;enable character mode
        clc                     ;character not used
        jsr     nxtgos          ;and wait for next character
        bra     1$

2$:     jsr     nxtchr          ;get character

        lda     *rstat0
        bita    #40             ;printer on ?
        beq     3$              ;no - skip
        bita    #4              ;stripping ?
        beq     4$              ;no - skip
3$:     jsr     printc          ;print character
4$:     ldd     *$tcntr         ;update character count
        subd    #1
        std     *$tcntr
        bgt     2$              ;loop for more
        jmp     endgos          ;else finished

.$td:   sec                     ;wait for data
        jsr     nxtgos

        ldb     *number+1       ;get low order data
        jmp     printb          ;print it and
                                ;wait for another

.$tv:   jsr     screen          ;dump screen as displayed
        sec                     ;character used
        jmp     endgos

        ;  set printer type

.$ts:   sec                     ;wait for data
        jsr     nxtgos
```

```
        ldb     *number+1       ;get low order
        andb    #3              ;only 4 options
        aslb                    ;scale option
        stb     *$pntrs         ;save selected printer
        jmp     endgos          ;finish

        ;  set dump button default print mode

.$tt:   jsr     nxtchr          ;get character

        jsr     $dispatch       ;dispatch accordingly
        .byte   'L
        .word   1$
        .byte   'G
        .word   2$
        .byte   'H
        .word   3$
        .byte   -1
        .word   errgos

1$:     clra                    ;low density
        bra     4$
2$:     lda     #1              ;graphics density
        bra     4$
3$:     lda     #-1             ;high density
4$:     sta     *$tdump         ;save dump mode
        sec                     ;character used
        jmp     endgos          ;finished

        .page
        .sbttl  printer print character routines

        ; for centronix compatible printers

printc: ldb     *char           ;get the character

printb: tst     pntin           ;printer ready ?
        bmi     printb          ;loop until ready
        stb     pntout          ;dump character
        pshs    cc
        orcc    #120            ;hold interrupts during strobe
        lda     *outpl          ;current output state
        anda    #367            ;strobe bit
        sta     plout
        ora     #10
        sta     plout
        puls    cc
        rts                     ;finished

1$:     bsr     printb
printm=.
        ldb     ,y+             ;get character
        bpl     1$              ;on character - loop
        rts                     ;finished

        .page
        .sbttl  panel selected screen dump

screen:
        lda     *rstat0
        bita    #2              ;button released ?
        beq     screen          ;loop until released
        lda     *rstat2
        ora     #2
        sta     *rstat2         ;clear start flag
        ldu     #$trmap         ;u is pointer most of the time

        clra                    ;no screen zoom
        sta     *$pzoom
        sta     *$zoom0

        lda     *$pntrs         ;dispatcher
        ldx     #3$
        jmp     [a,x]
1$:     ldd     $xscrn          ;a driver there ?
        beq     2$              ;no - skip
        subd    $xscrn+2        ;verify entry
        cmpd    #0x5AA5         ;check code
```

```
        bne     2$              ;invalid entry
        jsr     [$xscrn]        ;else use driver
        lda     #>direct        ;restore direct pointer
        tfr     a,dp
2$:     rts                     ;and exit
3$:                             ;printer options
        .if     hplaser
        .word   hpscrn
        .endif

        .if     tektronix
        .word   tkscrn
        .endif

        .if     epson
        .word   epscrn
        .endif

        .if     hplaser
        .else
        .word   2$
        .endif

        .if     tektronix
        .else
        .word   2$
        .endif

        .if     epson
        .else
        .word   2$
        .endif

        .word   1$

        .page
        .sbttl  printer line scaling routine

tscale:
        lsra
        rorb
        lsra
        rorb
        lsra
        rorb
        addd    #1
        rts                     ;and exit

        .page
        .sbttl  build cursor address for printer routines

tcursr: ldd     *$tx1           ;x-position
        anda    #17
        andb    #360
        std     *$cur0-2

        ldd     *$ty2           ;y-position
        coma
        comb
        anda    #17
        andb    #360
        std     *$cur0
        rts

        .page
        .sbttl  printer line update routine

tupd0:
        ldy     #$cur0
        bra     1$
tupd1=.
        ldy     #$cur1
        bra     1$
tupd2=.
        ldy     #$cur2          ;address
1$:     lda     *$pzoom         ;get zoom factor
        ldx     3,y             ;running line count
2$:     dec     2,y             ;update running zoom
```

```
        bge     3$              ;not time - skip
        sta     2,y             ;reset zoom factor
        pshs    a,b             ;save a & b
        ldd     ,y              ;update line position
        addd    #0d8
        std     ,y
        puls    a,b             ;restore a & b
3$:     leax    -1,x
        decb                    ;end of updating ?
        bgt     2$              ;no - loop
        stx     3,y             ;save running line count
        rts                     ;and exit

        .page
        .sbttl  general reset loop variables

tmv01:  ldx     #$cur0-2        ;compute address
        bra     1$
tmv12=.
        ldx     #$cur1-2        ;compute address
1$:     ldd     ,x
        std     7,x
        lda     2,x
        sta     11,x
        ldd     3,x
        std     12,x
        ldd     5,x
        std     14,x
        rts

        .page
        .sbttl  general bound parameter routine

        ;       enter with line length limit in <d>

tbound: tfr     d,x             ;save limit
        cmpd    *$plen          ;ok ?
        bge     3$              ;yes - skip
        std     *$plen          ;set maximum length
        subd    *$pofst         ;allow offset
        bgt     2$              ;and remainder to $dots
        ldd     #0
        std     *$pofst         ;else zero offset
        ldd     *$dots          ;check dot count
        cmpx    *$dots
        bgt     1$              ;ok - skip
        tfr     x,d             ;else use maximum
1$:     std     *$plen
2$:     std     *$dots
3$:     rts

        .page
        .sbttl  'epson' printer handler

        .if     epson

        ; 'epson' printer messages

epmsg0: .byte   15,12           ; <cr><lf>
epmsg1: .byte   15,12,200       ; <cr><lf> <end>
epmsg2: .byte   15,200          ; <cr> <end>
epmsg4: .byte   33,'K,200       ; 60 dots/inch
epmsg9: .byte   33,'*,5,200     ; 72 dots/inch
epmsg5: .byte   33,'L,200       ; 120 dots/inch
epmsg6: .byte   33,'2,200       ; 1/6" line spacing
epmsg7: .byte   33,'3,1,200     ; 1/216" line spacing
epmsg8: .byte   33,'3,27,200    ; 23/216" line spacing
epmsg3: .byte   33,'3,30,200    ; 24/216" line spacing

        .page
        .sbttl  panel selected epson screen dump

epscrn: lda     *$tdump         ;screen dump mode
        sta     *$pdens
        bne     1$              ;not low - skip
        ldd     #0d512          ;lines in display
        std     *$line0
        ldd     #0              ;offset
```

```
        std     *$pofst
        ldd     #0              ;y-top
        std     *$cur0
        ldd     #0d128          ;center of x range
        std     *$cur0-2
        ldd     #0d480          ;printer width
        bra     3$

1$:     cmpa    #1
        bne     2$              ;not graphic - skip
        ldd     #0d512          ;lines in display
        std     *$line0
        ldd     #0d35           ;offset
        std     *$pofst
        ldd     #0              ;y-top
        std     *$cur0
        ldd     #0              ;x-left
        std     *$cur0-2
        ldd     #0d512          ;printer width
        bra     3$

2$:     ldd     #0d512          ;lines in display
        std     *$line0
        ldd     #0d200          ;offset
        std     *$pofst
        ldd     #0              ;y-top
        std     *$cur0
        ldd     #0              ;x-left
        std     *$cur0-2
        ldd     #0d512          ;printer width

3$:     std     *$dots
        addd    *$pofst
        std     *$plen          ;total width
        bsr     $epson          ;and print full screen
        ldy     #epmsg0         ;two <cr><lf>'s
        jsr     printm

epdone: lda     *rstat0
        bita    #2              ;button released ?
        beq     epdone          ;loop until released
        lda     *rstat2
        ora     #2
        sta     *rstat2         ;clear stop flag
        ldy     #epmsg6         ;reset line spacing
        jmp     printm          ;finished

        .page
        .sbttl  epson color mapping and dispatch

$epson: ldx     #curmap         ;current rgb mapping
        ldy     #$trmap         ;$trmap result table
        lda     ,x+             ;build background flag
        ora     ,x+
        ora     ,x+
        beq     1$
        ora     #377
1$:     sta     *$bkgnd         ;save background flag
        sta     ,y+             ;and first color
        lda     #0d15           ;15 more colors to transform
2$:     ldb     ,x+             ;transform 'red'
        orb     ,x+             ;transform 'green'
        orb     ,x+             ;transform 'blue'
        beq     3$
        orb     #377
3$:     tst     *$bkgnd         ;background ?
        beq     4$              ;no - skip
        comb
4$:     stb     ,y+             ;save transformed code
        deca
        bgt     2$

        lda     *$pdens         ;check density
        beq     ep60            ;plot low density
        bgt     ep72            ;plotter graphics density
        bra     ep120           ;plot high density

        .page
```

```
        .sbttl  'epson' low density dump routine

ep60:   ldd     #0d900          ;maximum line length
        jsr     tbound          ;bound parameters
        clr     *$pdens         ;low density
        ldy     #epmsg2         ;go to left margin
        jsr     printm
        ldy     #epmsg3         ;8/72" line spacing
1$:     jsr     printm
        lda     *rstat2
        bita    #2              ;stop dump ?
        lbeq    epdone          ;yes - exit
        ldy     #epmsg4         ;load dots/inch
        jsr     epline          ;dump a line to printer
        ldb     #0d8            ;update position
        jsr     tupd0
        lble    epdone          ;exit if end
        ldy     #epmsg1         ;advance to next line
        bra     1$              ;and loop

        .page
        .sbttl  'epson' plotter graphics dump routine

ep72:   ldd     #0d1080         ;maximum line length
        jsr     tbound          ;bound parameters
        clr     *$pdens         ;low density
        ldy     #epmsg2         ;go to left margin
        jsr     printm
        ldy     #epmsg3         ;8/72" line spacing
1$:     jsr     printm
        lda     *rstat2
        bita    #2              ;stop dump ?
        lbeq    epdone          ;yes - exit
        ldy     #epmsg9         ;load dots/inch
        jsr     epline          ;dump a line to printer
        ldb     #0d8            ;update position
        jsr     tupd0
        lble    epdone          ;exit if end
        ldy     #epmsg1         ;advance to next line
        bra     1$              ;and loop

        .page
        .sbttl  'epson' high density dump routine

ep120:  ldd     #0d1800         ;maximum line length
        jsr     tbound          ;bound parameters
        lda     #-1             ;high density
        sta     *$pdens
        ldy     #epmsg2         ;move to left margin
1$:     jsr     printm
        lda     *rstat2
        bita    #2              ;stop dump ?
        lbeq    epdone          ;yes - exit
        ldy     #epmsg7         ;1/216" line spacing
        jsr     printm
        ldy     #epmsg5         ;load dots/inch
        jsr     epline          ;dump a line
        ldb     #1              ;update position
        jsr     tupd0
        ldy     #epmsg1         ;<cr><lf>
        jsr     printm
        ldy     #epmsg8         ;23/216" line spacing
        jsr     printm
        ldy     #epmsg5         ;load line length
        jsr     epline          ;dump line
        ldb     #0d15           ;update position
        jsr     tupd0
        lble    epdone          ;exit if finished
        ldy     #epmsg1         ;<cr><lf>
        bra     1$              ;loop for next line

        .page
        .sbttl  'epson' dump line to printer

epline:
        jsr     printm
        ldb     *$plen+1
        jsr     printb
```

```
        ldb     *$plen
        jsr     printb

        ldd     *$dots          ;init dot counter
        coma
        comb
        std     *$rdots
        lda     *$pzoom         ;and x-zoom
        sta     *$rzoom

        inc     *$rdots+1       ;update dots left
        bne     1$
        inc     *$rdots
        lbpl    17$             ;exit if done

1$:     ldy     *$pofst         ;now dump offset nulls
        beq     3$              ;if none - skip
        clrb                    ;null
2$:     jsr     printb
        leay    -1,y
        bne     2$

3$:     jsr     tmv01           ;load variable set 1
        jsr     tmv12           ;load variable set 2

4$:     lda     #0d8            ;8. rows in swath
        sta     *$dotctr
        ldx     *$cur2-2        ;load x-address
        stx     *crxpos
        stx     vidx
        ldx     *$cur2          ;load y-address
        bra     10$             ;entry point

5$:     ldx     *$line2         ;running line count
        ldb     *$zoom2         ;zoom factor
        tst     *$pdens         ;low - skip
        beq     6$
        leax    -1,x            ;one less line
        dec     *$zoom2
6$:     leax    -1,x            ;one less line
        stx     *$line2         ;save count
        ble     12$             ;past last line
        dec     *$zoom2
        bge     13$             ;not time - skip
        lda     *$pzoom         ;restore zoom value
        ldx     *$cur2          ;buffer pointer
        tst     *$pdens         ;low - skip
        beq     8$
        tsta                    ;high density zoomed ?
        beq     7$              ;no zoom - skip
        decb                    ;update zoom factor
        bge     8$
        tfr     a,b             ;reset zoom factor
        bra     8$
7$:     leax    10,x            ;next row
8$:     leax    10,x            ;next row
        decb                    ;update zoom factor
        bge     9$
        tfr     a,b             ;reset zoom factor
9$:     stb     *$zoom2         ;and save
10$:    stx     *$cur2          ;updated buffer pointer

        ;       $cur2-2 contains the x-address of the data
        ;       $cur2   contains the y-address of the data

        ;       'transformed color' left in $color

        stx     *crypos
        stx     vidy            ;y position
        lda     vidplt          ;read data
        anda    *$blank         ;use selected bits
        bita    #20             ;oc buffer ?
        beq     11$             ;no - skip
        ora     #17             ;color 17 for oc
11$:    anda    #17
        lda     a,u             ;$trmap transformed color
        sta     *$color         ;save color
        bra     14$
```

```
12$:    lda     *$bkgnd         ;else background
        bra     14$
13$:    lda     *$color         ;get color
14$:    rora                    ;shift bit into 'c'
        rol     *char           ;and into char
        dec     *$dotctr        ;loop for all 8. rows
        bgt     5$
15$:    jsr     printc          ;then print the character

        inc     *$rdots+1       ;update dots left
        bne     16$
        inc     *$rdots
        bpl     17$             ;exit if done

16$:    dec     *$rzoom         ;update x-zoom
        bge     15$             ;loop if more x-zoom

        lda     *$pzoom         ;reset zoom
        sta     *$rzoom
        ldd     *$cur1-2        ;update column position
        addd    #0d8
        std     *$cur1-2
        jsr     tmv12           ;reload variable set 2
        lbra    4$

17$:    rts                     ;and finished

        .endif                  ;epson

        .page
        .sbttl  'tektronix' color graphics copier handler

        .if     tektronix

        ; 'tektronix' copier messages

tkmsg0: .byte   30              ; cancel
        .byte   33,'4,200       ; reset page width and margins
tkmsg1: .byte   33,'I,200       ; start a graphics image line
tkmsg2: .byte   33,'A,200       ; print a "microline"
tkmsg3: .byte   15,12,15,12,200

tkscrn: lda     *$tdump         ;check dump mode
        sta     *$pdens
        bgt     2$              ;high - skip
        ldd     #0d512          ;lines in display
        std     *$line0
        ldd     #0              ;offset
        std     *$pofst
        ldd     #0              ;y-top
        std     *$cur0
        ldd     #0              ;x-left
        std     *$cur0-2
        ldd     #0d512          ;printer width
        bra     1$

2$:     ldd     #0d512          ;lines in display
        std     *$line0
        ldd     #0d256          ;offset
        std     *$pofst
        ldd     #0              ;y-top
        std     *$cur0
        ldd     #0              ;x-left
        std     *$cur0-2
        ldd     #0d512          ;printer width

1$:     std     *$dots
        addd    *$pofst
        std     *$plen          ;total width
        bsr     $tktnx          ;dump screen
        ldy     #tkmsg3         ;two <cr><lf>'s
        jmp     printm

        .page
        .sbttl  tektronix color mapping and dispatch

$tktnx: ldx     #curmap         ;current rgb mapping
        ldy     #tkctbl         ;transform table address
```

```
        lda     ,x              ;build background flag
        ora     1,x             ;$trmap address is in <u>
        ora     2,x
        sta     ,-s             ;save background flag
        sta     *$bkgnd
        lda     #0d16           ;16 colors to transform
1$:     ldb     ,x+             ;transform 'red'
        andb    #14
        aslb
        aslb
        orb     ,x+             ;transform 'green'
        andb    #74
        aslb
        aslb
        orb     ,x+             ;transform 'blue'
        lsrb
        lsrb
        ldb     b,y             ;get transformed color
        tst     ,s              ;background ?
        beq     2$              ;no - skip
        lsrb
        lsrb
        lsrb
        lsrb
2$:     stb     ,u+             ;save transformed code
        deca
        bgt     1$
        leas    1,s             ;pop background flag
        ldu     #$trmap         ;restore pointer
        lda     *$pdens         ;check density
        bgt     tk120           ;high density dump
                                ;low/graphics density dump

        .page
        .sbttl  low and graphics density

tk60:   ldd     *$dots          ;low/graphics density
        addd    *$dots
        std     *$dots          ;2*dots
        addd    *$pofst
        std     *$plen          ;new length
        ldd     *$line0
        addd    *$line0
        std     *$line0         ;2*lines
        sec                     ;new zoom
        rol     *$pzoom
        sec
        rol     *$zoom0

        .sbttl  high density

tk120:  ldd     #0d1024         ;maximum line length
        jsr     tbound          ;bound parameters
        ldy     #tkmsg0         ;cancel and
        jsr     printm          ;clear width and margins
        jsr     bnasc3          ;convert $plen to required ascii code
4$:     jsr     tkline          ;dump a "microline"
        lda     *rstat2
        bita    #2              ;stop dump ?
        beq     5$              ;yes - exit
        ldb     #4              ;four real lines per "microline"
        jsr     tupd0           ;update position
        bgt     4$              ;loop for all lines
5$:     lda     *rstat0
        bita    #2              ;button released ?
        beq     5$              ;loop until released
        lda     *rstat2
        ora     #2
        sta     *rstat2         ;clear stop flag
        rts                     ;finished

        .page
        .sbttl  'tektronix' line routine

tkline: lda     #3
        sta     *$inkjet        ;start with top row of jets
        jsr     tmv01           ;load variable set 1
```

```
1$:     jsr     tmv12           ;reload variable set 2
        lda     #black
        sta     *$inkclr        ;start with black jet
        sta     *$anyclr
        ldy     *$cur1          ;load y-address
        sty     *crypos
        sty     vidy
        bra     3$              ;and start up

2$:     lda     *$anyclr        ;get color status
        bita    *$inkclr        ;any of this color ?
        lbeq    15$             ;no - skip
        jsr     tmv12           ;reload variable set 2
3$:     ldx     *$cur2-2        ;load x-address
        stx     *crxpos
        stx     vidx
        ldy     #tkmsg1         ;begin a row
        jsr     printm
        lda     *$inkclr        ;generate color/jet code
        ldb     #20
4$:     subb    #4
        asra
        bne     4$
        subb    *$inkjet
        addb    #'3
        stb     *$pasci         ;load code
        ldy     #$pasci         ;ascii byte length
        jsr     printm

        ldd     *$dots          ;load dots/line
        coma                    ;-dots-1
        comb
        std     *$rdots
        lda     *$pzoom         ;load zoom
        sta     *$rzoom
        ldx     *$cur2-2        ;load x-address

        clr     *char           ;zero byte
        ldd     *$pofst         ;load offset
5$:     subd    #0d8            ;update count
        bmi     6$
        pshs    a,b
        jsr     printc          ;dump a blank 8. dots
        puls    a,b
        bra     5$              ;and loop
6$:     negb                    ;set up dot counter
        stb     *$dotctr
        inc     *$rdots+1       ;update dots left
        bne     9$              ;go to entry point
        inc     *$rdots
        bmi     9$
        bra     13$             ;exit if done

7$:     inc     *$rdots+1       ;update dots left
        bne     8$
        inc     *$rdots
        bpl     13$             ;exit if done
8$:     dec     *$rzoom         ;update zoom
        bge     11$
        lda     *$pzoom         ;restore zoom
        sta     *$rzoom
        ldx     *$cur2-2        ;get buffer position
        leax    10,x            ;update column position
        stx     *$cur2-2

        ;       $cur2-2 contains the x-address of the data
        ;       $cur2   contains the y-address of the data

        ;       'transformed color' left in $color
        ;        color content of line left in $anyclr

9$:     stx     *crxpos
        stx     vidx            ;load x-address
        lda     vidplt          ;read data
        anda    *$blank         ;use selected bits
        bita    #20             ;oc buffer ?
        beq     10$             ;no - skip
        ora     #17             ;color 17 for oc buffer
```

```
10$:    anda    #17
        ldb     a,u             ;$trmap transformed color code
        stb     *$color         ;and save
        orb     *$anyclr        ;full line color status
        stb     *$anyclr

11$:    ldb     *$color         ;get color
        clra
        andb    *$inkclr        ;mask color jet entry
        beq     12$
        inca
12$:    rora
        rol     *char           ;build dots

        dec     *$dotctr        ;finished with byte ?
        bgt     7$              ;no - loop

        jsr     printc          ;print byte of dots
        lda     #0d8            ;reset dot counter
        sta     *$dotctr
        bra     7$              ;and loop for more

13$:    lda     *$dotctr        ;finish off byte
        cmpa    #0d8
        beq     15$             ;done - skip
14$:    asl     *char           ;fill char with blank bits
        deca
        bgt     14$
        jsr     printc          ;print final dots

15$:    asr     *$inkclr        ;next color (black/magenta/yellow/cyan)
        lbne    2$              ;loop for next color
        dec     *$inkjet        ;next row of ink jets
        blt     16$             ;exit if finished
        ldb     #1              ;update line
        jsr     tupd1
        lbgt    1$              ;and loop if more

16$:    ldy     #tkmsg2         ;print a "microline"
        jmp     printm          ;finished

        .page
        .sbttl  'tektronix' color transform tables

tkctbl: .byte   200, 125, 125, 125, 063, 021, 125, 125
        .byte   063, 063, 021, 125, 063, 063, 063, 021
        .byte   146, 104, 125, 125, 042, 010, 125, 125
        .byte   063, 063, 021, 125, 063, 063, 063, 021
        .byte   146, 146, 104, 125, 146, 146, 104, 125
        .byte   042, 042, 010, 125, 063, 063, 063, 021
        .byte   146, 146, 146, 104, 146, 146, 146, 104
        .byte   146, 146, 146, 104, 042, 042, 042, 010


        ;  the above table was built by the following macro definition

        black   =       10
        magenta =       4
        yellow  =       2
        cyan    =       1

;       ...r    =       0
;       .rept   4.
;       ...g    =       0
;       .rept   4.
;       ...b    =       0
;       .rept   4.
;
;       ...big  =       ...r
;       .iif    gt,...g-...big  ...big=...g
;       .iif    gt,...b-...big  ...big=...b
;
;       ...rc   =       ...r-...big
;       ...gc   =       ...g-...big
;       ...bc   =       ...b-...big
;
;       ...clr  =       0
;       .iif    eq,...rc                ...clr=1
```

```
;       .iif    eq,...gc                ...clr=...clr!2
;       .iif    eq,...bc                ...clr=...clr!4
;       .if     eq,...clr-7
;        .iif   eq,...r+...g+...b       ...clr=0
;       .endc
;
;       .iif    eq,...clr-0             ...clr=20*<black>
;       .iif    eq,...clr-1             ...clr=21*<yellow+magenta>
;       .iif    eq,...clr-2             ...clr=21*<yellow+cyan>
;       .iif    eq,...clr-3             ...clr=21*<yellow>
;       .iif    eq,...clr-4             ...clr=21*<magenta+cyan>
;       .iif    eq,...clr-5             ...clr=21*<magenta>
;       .iif    eq,...clr-6             ...clr=21*<cyan>
;       .iif    eq,...clr-7             ...clr=01*<black>
;
;       .nlist
;       .byte   ...clr          ;color code
;       .list
;
;       ...b    =       ...b+1
;       .endr
;       ...g    =       ...g+1
;       .endr
;       ...r    =       ...r+1
;       .endr

        .endif                  ;tektronix

        .page
        .sbttl  'hp laserjet printer handler'

        .if     hplaser

        ; 'laserjet' copier messages

hpmsg0: .byte   33,'*,'t,'7,'5,'R,200           ; 75  dpi resolution
hpmsg1: .byte   33,'*,'t,'1,'5,'0,'R,200        ; 150 dpi resolution
hpmsg2: .byte   33,'&,'a,'7,'2,'0,'H            ; fast 75 dpi absolute
        .byte   33,'&,'a,'1,'0,'8,'0,'V,200     ; horizontal/vertical position
hpmsg3: .byte   33,'&,'a,'1,'9,'4,'4,'H         ; fast 150 dpi absolute
        .byte   33,'&,'a,'1,'0,'8,'0,'V,200     ; horizontal/vertical position
hpmsg4: .byte   33,'*,'r,'1,'A,200              ; start raster graphics
hpmsg5: .byte   33,'*,'b,200                    ; transfer a line
hpmsg6: .byte   33,'*,'r,'B,200                 ; end graphics

hpscrn: lda     *$tdump         ;check dump mode
        sta     *$pdens
        blt     2$              ;high - skip

        ldy     #hpmsg2         ;absolute screen location
        jsr     printm
        ldd     #0d512          ;lines in display
        std     *$line0
        ldd     #0              ;offset (set by hpmsg2)
        std     *$pofst
        ldd     #0              ;y-top
        std     *$cur0
        ldd     #0              ;x-left
        std     *$cur0-2
        ldd     #0d512          ;printer width
        bra     1$

2$:     ldy     #hpmsg3         ;absolute screen location
        jsr     printm
        ldd     #0d512          ;lines in display
        std     *$line0
        ldd     #0              ;offset (set by hpmsg3)
        std     *$pofst
        ldd     #0              ;y-top
        std     *$cur0
        ldd     #0              ;x-left
        std     *$cur0-2
        ldd     #0d512          ;printer width

1$:     std     *$dots
        addd    *$pofst
        std     *$plen          ;total width
        bsr     $hp             ;dump screen
```

```
        ldb     #14             ;'form'
        jmp     printb

        .page
        .sbttl  hp color mapping and dispatch

$hp:    ldx     #curmap         ;current rgb mapping
        ldy     #$trmap         ;$trmap result table
        lda     ,x+             ;build background flag
        ora     ,x+
        ora     ,x+
        beq     1$
        ora     #377
1$:     sta     *$bkgnd         ;save background flag
        sta     ,y+             ;and first color
        lda     #0d15           ;15 more colors to transform
2$:     ldb     ,x+             ;transform 'red'
        orb     ,x+             ;transform 'green'
        orb     ,x+             ;transform 'blue'
        beq     3$
        orb     #377
3$:     tst     *$bkgnd         ;background ?
        beq     4$              ;no - skip
        comb
4$:     stb     ,y+             ;save transformed code
        deca
        bgt     2$

        lda     *$pdens         ;check density
        blt     5$

        ldy     #hpmsg0         ;plot low/graphics density
        jsr     printm
        ldd     #0d600          ;maximum line length
        bra     6$

5$:     ldy     #hpmsg1         ;plot high density
        jsr     printm
        ldd     #0d1200         ;maximum line length

6$:     jsr     tbound          ;bound parameters
        jsr     bnasc3          ;convert $plen to required ascii code
        ldy     #hpmsg4         ;start raster graphics
        jsr     printm
7$:     jsr     hpline          ;dump a line
        lda     *rstat2
        bita    #2              ;stop dump ?
        beq     8$              ;yes - exit
        ldb     #1              ;one real line per image line
        jsr     tupd0           ;update position
        bgt     7$              ;loop for all lines
8$:     lda     *rstat0
        bita    #2              ;button released ?
        beq     8$              ;loop until released
        lda     *rstat2
        ora     #2
        sta     *rstat2         ;clear stop flag
        ldy     #hpmsg6         ;end raster graphics
        jsr     printm
        rts                     ;finished

        .page
        .sbttl  'laserjet' line routine

hpline: ldy     *$cur0          ;load y-address
        sty     *crypos
        sty     vidy
        ldx     *$cur0-2        ;load x-address
        stx     *crxpos
        stx     vidx

        ldy     #hpmsg5         ;begin a row
        jsr     printm
        ldy     #$pasci+1       ;byte count
        jsr     printm
        ldb     #'W             ;end of message
        jsr     printb
```

```
        ldd     *$dots          ;load dots/line
        coma                    ;-dots-1
        comb
        std     *$rdots
        lda     *$pzoom         ;load zoom
        sta     *$rzoom
        lbeq    hpfast          ;do it fast if no zoom

        clr     *char           ;zero byte
        ldd     *$pofst         ;load offset
1$:     subd    #0d8            ;update count
        bmi     2$
        pshs    a,b
        jsr     printc          ;dump a blank 8. dots
        puls    a,b
        bra     1$              ;and loop
2$:     negb                    ;set up dot counter
        stb     *$dotctr
        inc     *$rdots+1       ;update dots left
        bne     5$              ;go to entry point
        inc     *$rdots
        bmi     5$
        bra     8$              ;exit if done

3$:     inc     *$rdots+1       ;update dots left
        bne     4$
        inc     *$rdots
        bpl     8$              ;exit if done
4$:     dec     *$rzoom         ;update zoom
        bge     7$
        lda     *$pzoom         ;restore zoom
        sta     *$rzoom
        leax    10,x            ;update column position

        ;       $cur0-2 contains the x-address of the data
        ;       $cur0   contains the y-address of the data

        ;       'transformed color' left in $color

5$:     stx     *crxpos
        stx     vidx            ;load x-address
        lda     vidplt          ;read data
        anda    *$blank         ;use selected bits
        bita    #20             ;oc buffer ?
        beq     6$              ;no - skip
        ora     #17             ;color 17 for oc buffer
6$:     anda    #17
        ldb     a,u             ;$trmap transformed color code
        stb     *$color         ;and save

7$:     ldb     *$color         ;get color
        rorb
        rol     *char           ;build dots

        dec     *$dotctr        ;finished with byte ?
        bgt     3$              ;no - loop

        jsr     printc          ;print byte of dots
        lda     #0d8            ;reset dot counter
        sta     *$dotctr
        bra     3$              ;and loop for more

8$:     lda     *$dotctr        ;finish off byte
        cmpa    #0d8
        beq     10$             ;done - skip
9$:     asl     *char           ;fill char with blank bits
        deca
        bgt     9$
        jsr     printc          ;print final dots

10$:    rts                     ;finished

hpfast: clr     *char           ;zero byte
        ldd     *$pofst         ;load offset
1$:     subd    #0d8            ;update count
        bmi     2$
        pshs    a,b
        jsr     printc          ;dump a blank 8. dots
```

```
        puls    a,b
        bra     1$              ;and loop
2$:     negb                    ;set up dot counter
        stb     *$dotctr
        cmpb    #0d8            ;full byte
        beq     hploop          ;yes - skip
        jsr     hpchar          ;else do partial

hploop: ldd     *$rdots         ;update dots left               (5)
        lbeq    hpfini          ;none - finished                (5/6)
        addd    #0d8            ;8 bits at a time               (4)
        lbgt    hpchar          ;only partial                   (5/6)
        std     *$rdots         ;                       (5)

        ; this is the fastest coding (but long)

        pshs    cc              ;                               (6)
        orcc    #120            ;hold interrupts                (3)
hpbit7: ;-------------------------------------------------------
        stx     vidx            ;load x-address                 (6)
        leax    10,x            ;update column position         (5)
        lda     vidplt          ;read data                      (5)
        anda    *$blank         ;use selected bits              (4)
        bita    #20             ;oc buffer ?                    (2)
        beq     1$              ;no - skip                      (3)
        ora     #17             ;color 17 for oc buffer         (2)
1$:     anda    #17             ;mask code                      (2)
        lda     a,u             ;$trmap transformed color code  (5)
        rora                    ;                               (2)
        rolb                    ;build dots                     (2)
hpbit6: ;-------------------------------------------------------
        stx     vidx            ;load x-address                 (6)
        leax    10,x            ;update column position         (5)
        lda     vidplt          ;read data                      (5)
        anda    *$blank         ;use selected bits              (4)
        bita    #20             ;oc buffer ?                    (2)
        beq     1$              ;no - skip                      (3)
        ora     #17             ;color 17 for oc buffer         (2)
1$:     anda    #17             ;mask code                      (2)
        lda     a,u             ;$trmap transformed color code  (5)
        rora                    ;                               (2)
        rolb                    ;build dots                     (2)
hpbit5: ;-------------------------------------------------------
        stx     vidx            ;load x-address                 (6)
        leax    10,x            ;update column position         (5)
        lda     vidplt          ;read data                      (5)
        anda    *$blank         ;use selected bits              (4)
        bita    #20             ;oc buffer ?                    (2)
        beq     1$              ;no - skip                      (3)
        ora     #17             ;color 17 for oc buffer         (2)
1$:     anda    #17             ;mask code                      (2)
        lda     a,u             ;$trmap transformed color code  (5)
        rora                    ;                               (2)
        rolb                    ;build dots                     (2)
hpbit4: ;-------------------------------------------------------
        stx     vidx            ;load x-address                 (6)
        leax    10,x            ;update column position         (5)
        lda     vidplt          ;read data                      (5)
        anda    *$blank         ;use selected bits              (4)
        bita    #20             ;oc buffer ?                    (2)
        beq     1$              ;no - skip                      (3)
        ora     #17             ;color 17 for oc buffer         (2)
1$:     anda    #17             ;mask code                      (2)
        lda     a,u             ;$trmap transformed color code  (5)
        rora                    ;                               (2)
        rolb                    ;build dots                     (2)
hpbit3: ;-------------------------------------------------------
        stx     vidx            ;load x-address                 (6)
        leax    10,x            ;update column position         (5)
        lda     vidplt          ;read data                      (5)
        anda    *$blank         ;use selected bits              (4)
        bita    #20             ;oc buffer ?                    (2)
        beq     1$              ;no - skip                      (3)
        ora     #17             ;color 17 for oc buffer         (2)
1$:     anda    #17             ;mask code                      (2)
        lda     a,u             ;$trmap transformed color code  (5)
        rora                    ;                               (2)
        rolb                    ;build dots                     (2)
```

```
hpbit2: ;-------------------------------------------------------
        stx     vidx            ;load x-address                 (6)
        leax    10,x            ;update column position         (5)
        lda     vidplt          ;read data                      (5)
        anda    *$blank         ;use selected bits              (4)
        bita    #20             ;oc buffer ?                    (2)
        beq     1$              ;no - skip                      (3)
        ora     #17             ;color 17 for oc buffer         (2)
1$:     anda    #17             ;mask code                      (2)
        lda     a,u             ;$trmap transformed color code  (5)
        rora                    ;                               (2)
        rolb                    ;build dots                     (2)
hpbit1: ;-------------------------------------------------------
        stx     vidx            ;load x-address                 (6)
        leax    10,x            ;update column position         (5)
        lda     vidplt          ;read data                      (5)
        anda    *$blank         ;use selected bits              (4)
        bita    #20             ;oc buffer ?                    (2)
        beq     1$              ;no - skip                      (3)
        ora     #17             ;color 17 for oc buffer         (2)
1$:     anda    #17             ;mask code                      (2)
        lda     a,u             ;$trmap transformed color code  (5)
        rora                    ;                               (2)
        rolb                    ;build dots                     (2)
hpbit0: ;-------------------------------------------------------
        stx     *crxpos         ;save position                  (6)
        stx     vidx            ;load x-address                 (6)
        leax    10,x            ;update column position         (5)
        lda     vidplt          ;read data                      (5)
        anda    *$blank         ;use selected bits              (4)
        bita    #20             ;oc buffer ?                    (2)
        beq     1$              ;no - skip                      (3)
        ora     #17             ;color 17 for oc buffer         (2)
1$:     anda    #17             ;mask code                      (2)
        lda     a,u             ;$trmap transformed color code  (5)
        rora                    ;                               (2)
        rolb                    ;build dots                     (2)
hpdump: ;-------------------------------------------------------
        tst     pntin           ;printer ready ?                (7)
        bmi     hpdump          ;loop until ready               (3)
        stb     pntout          ;dump character                 (5)
        lda     *outpl          ;current output state           (4)
        anda    #367            ;strobe bit                     (2)
        sta     plout           ;                               (5)
        ora     #10             ;                               (2)
        sta     plout           ;                               (5)
        puls    cc              ;                               (6)

        lda     #0d8            ;reset dot counter              (2)
        sta     *$dotctr        ;                               (4)
        jmp     hploop          ;and loop for more              (4)

hpchar: ldb     *char           ;get bits
        inc     *$rdots+1       ;update dots left
        bne     3$              ;go to entry point
        inc     *$rdots
        bmi     3$
        bra     5$              ;exit if done

1$:     inc     *$rdots+1       ;update dots left               (6)
        bne     2$              ;                               (3)
        inc     *$rdots         ;                               (6)
        bpl     5$              ;exit if done                   (3)
2$:     leax    10,x            ;update column position         (5)
        stx     *crxpos         ;                               (5)
        stx     vidx            ;load x-address                 (6)
3$:     lda     vidplt          ;read data                      (5)
        anda    *$blank         ;use selected bits              (4)
        bita    #20             ;oc buffer ?                    (2)
        beq     4$              ;no - skip                      (3)
        ora     #17             ;color 17 for oc buffer         (2)
4$:     anda    #17             ;mask code                      (2)
        lda     a,u             ;$trmap transformed color code  (5)
        rora                    ;                               (2)
        rolb                    ;build dots                     (2)

        dec     *$dotctr        ;finished with byte ?           (6)
        bgt     1$              ;no - loop                      (3)
```

```
        bra     7$

5$:     lda     *$dotctr        ;finish off byte
        cmpa    #0d8
        beq     hpfini          ;done - skip
6$:     aslb                    ;fill char with blank bits
        deca
        bgt     6$
7$:     jsr     printb          ;dump byte
        lda     #0d8            ;reset dot counter
        sta     *$dotctr
hpfini: rts                     ;finished

        .endif                  ;hplaser

        .page
        .sbttl  binary --> 3-digit ascii conversion routine

        .if     tektronix | hplaser


        ;data format for routine
        ;       0       #-------digit counter
        ;
        ;       1       *msb
        ;       2       *lsb----binary input
        ;
        ;       3       #msb
        ;       4       #lsb----conversion temporary
        ;
        ;       5       %-------bit in digit counter
        ;
        ;       6       *-------fraction

bnasc3: leas    -7,s            ;pointer to stack area
        ldx     #bintbl         ;address of conversion table
        ldy     #$pasci+1       ;pointer to result area
        ldd     *$plen          ;load binary input
        std     1,s
        lda     #3              ;set digit counter
        sta     ,s
        clr     6,s             ;clear fraction
1$:     ror     1,s             ;bytes = $plen/8
        ror     2,s
        ror     6,s             ;fractional result
        deca
        bgt     1$
2$:     clr     ,y              ;init digit
        ldd     ,x++            ;get two bytes
        std     3,s             ;place in temporary
        lda     #4              ;get bit count
        sta     5,s             ;save count
3$:     asl     ,y              ;assume /c='0'
        ldd     1,s             ;do subtraction
        subd    3,s
        bcs     4$              ;branch if ok
        inc     ,y              ;else - /c='1'
        std     1,s             ;save new binary
4$:     dec     5,s             ;digit done ?
        beq     5$              ;branch if yes
        lsr     3,s             ;do a 1 bit shift on 2 bytes
        ror     4,s
        bra     3$              ;loop again
5$:     lda     #'0             ;convert to ascii
        adda    ,y
        sta     ,y+             ;and save
        dec     ,s              ;finished ?
        bne     2$              ;loop until done
        lda     #200            ;end of message
        sta     ,y
        tst     6,s             ;a fraction ?
        beq     6$              ;no - exit
        lda     #'9             ;do fractional update
        ldb     #'0
        inc     ,-y             ;increment
        cmpa    ,y              ;check
        bge     6$              ;ok - skip
        stb     ,y              ;overflowed digit
```

```
        inc     ,-y             ;next digit
        cmpa    ,y              ;check
        bge     6$              ;ok - skip
        stb     ,y              ;overflowed digit
        inc     ,-y             ;last digit
6$:     leas    7,s             ;pop area from stack
        rts


        ;  binary --> bcd conversion table

bintbl: .byte   3,40    ;800.
        .byte   0,120   ;80.
        .byte   0,10    ;8.

        .endif                  ;tektronix | hplaser

        .endif                  ;printer

        .page
        .sbttl  $q command

        .area   EXTSAV

chksm2: .byte   0,0             ;for 8k rom
        .byte   15,12
        .ascii  /DSPCGC  0001  V02.02/
        .byte   15,12
        .ascii  /COPYRIGHT 1985,1986,1987,1988/
        .byte   15,12
        .ascii  /OTSELIC SPECIALTIES/
        .byte   15,12
        .ascii  /721 BERKELEY/
        .byte   15,12
        .ascii  /KENT, OHIO  44240/
        .byte   15,12


.$q:    jsr     nxtchr          ;get character

        jsr     $dispatch       ;dispatch accordingly
        .byte   'D
        .word   1$
        .byte   'E
        .word   2$
        .byte   'R
        .word   8$
        .byte   'S
        .word   10$
        .byte   'X
        .word   12$
        .byte   'Z
        .word   20$
        .byte   0               ;end of table

1$:     sec
        jsr     nxtgos          ;<d> - delete a macro

        jsr     q$dlet          ;delete the macro
        bra     1$

2$:     sec
        jsr     nxtgos          ;<e> - enter a new macro

        jsr     q$dlet          ;delete old macro
        ldx     q$pntr          ;macro table pointer
        lbeq    errgos          ;bad number specified
        ldd     q$end
        std     ,x              ;save beginning address of macro

        lda     *char           ;check for '+'
3$:     cmpa    #'+
        beq     5$              ;yes - skip
        lda     *lstat
        ora     #200
        sta     *lstat          ;enable character mode
        clc                     ;character not used
        jsr     nxtgos          ;and wait for next character
```

```
        bra     3$

4$:     pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~40
        sta     *outpl          ;buffer overflow error
        sta     plout
        puls    cc
5$:     lda     *lstat
        ora     #220
        sta     *lstat          ;special character mode
        sec
        jsr     nxtgos          ;wait for character

        ldb     *char           ;get new character
        lda     q$pchr          ;get previous character
        stb     q$pchr          ;save new character
        cmpd    #"$Q            ;end of sequence ?
        beq     6$              ;yes
        cmpd    #"$q
        bne     7$              ;no - skip
6$:     ldx     q$pntr          ;update for end of sequence
        ldd     2,x
        subd    #1
        std     2,x
        ldd     q$end
        subd    #1
        std     q$end
        lbra    .$q

7$:     ldx     q$pntr
        ldd     ,x
        addd    2,x             ;current position
        cmpd    #$macnd         ;at the end of the table ?
        bhs     4$              ;yes - ignore characters

        tfr     d,u
        ldb     *char           ;save this character
        stb     ,u

        ldd     2,x             ;update count
        addd    #1
        std     2,x

        ldd     q$end           ;update total length
        addd    #1
        std     q$end
        bra     5$              ;loop for next character

8$:     ldx     #q$save         ;<r> - restore drawing status
        cmpx    ,x++            ;check flag
        lbne    .$q             ;nothing to restore

        ldd     ,x++            ;restore x & y status
        std     *xorg
        ldd     ,x++
        std     *xval
        ldd     ,x++
        sta     *xsize
        stb     *cxzoom
        ldd     ,x++
        std     *yorg
        ldd     ,x++
        std     *yval
        ldd     ,x++
        sta     *ysize
        stb     *cyzoom

        ldd     ,x++            ;restore drawing parameters
        sta     *format
        stb     *gcolor
        ldd     ,x++
        sta     *lstat
        stb     *lstatx
        lda     ,x+
        sta     *rstat1
```

```
        ldd     ,x++            ;character selections
        std     *ptchar
        ldd     ,x++
        std     *romset

        ldu     #.$ptrn         ;restore drawing patterns
        ldb     #0d34
9$:     lda     ,x+
        sta     ,u+
        decb
        bgt     9$

        lbra    .$q             ;finished

10$:    ldx     #q$save         ;<s> - save drawing status
        stx     ,x++            ;check flag

        ldd     *xorg           ;save x & y status
        std     ,x++
        ldd     *xval
        std     ,x++
        lda     *xsize
        ldb     *cxzoom
        std     ,x++
        ldd     *yorg
        std     ,x++
        ldd     *yval
        std     ,x++
        lda     *ysize
        ldb     *cyzoom
        std     ,x++

        lda     *format         ;save drawing parameters
        ldb     *gcolor
        std     ,x++
        lda     *lstat
        anda    #17
        ldb     *lstatx
        std     ,x++
        lda     *rstat1
        anda    #217
        sta     ,x+

        ldd     *ptchar         ;character selections
        std     ,x++
        ldd     *romset
        std     ,x++

        ldu     #.$ptrn         ;save drawing patterns
        ldb     #0d34
11$:    lda     ,u+
        sta     ,x+
        decb
        bgt     11$

        lbra    .$q             ;finished

12$:    sec
        jsr     nxtgos          ;<x> - execute a macro

        jsr     q$mcrn          ;check macro number
        beq     12$             ;no macro - loop for next

        lda     *char           ;save last character
        sta     ,-s
        lda     *lstat          ;current list control
        anda    #360
        sta     ,-s             ;save $q status
        lda     *lstat
        anda    #~360
        sta     *lstat          ;all new list control
        ldd     #0              ;clear service address
        std     *gosub

13$:    lda     ,u+             ;get the character
        sta     *char
        stu     q$pntr          ;save macro address
```

```
        .if     printer
        lda     *rstat0
        bita    #40             ;printer on ?
        beq     15$             ;no - skip
14$:    jsr     printc          ;print the character
15$:
        .endif

        jsr     listpr          ;enter list processor

        ;  normal system services have to be checked

16$:    lda     *rstat2
        bita    #10             ;video need clearing ?
        bne     18$             ;if not - skip
        jsr     clear           ;clear screen
17$:    lda     *rstat0
        bita    #10             ;button released ?
        beq     17$             ;loop until released
        lda     *rstat2
        ora     #10
        sta     *rstat2         ;clear entry flag
18$:

        .if     printer
        lda     *rstat2
        bita    #2              ;screen dump ?
        bne     19$             ;no - skip
        jsr     screen          ;dump screen
19$:
        .endif

        .if     plotter
        jsr     calib           ;check cal button
        .endif

        ldu     q$pntr          ;get macro address
        ldd     q$cntr
        subd    #1
        std     q$cntr          ;save macro length
        lbgt    13$             ;loop for the whole macro

        lda     *lstat          ;restore status
        anda    #17
        ora     ,s+
        sta     *lstat
        lda     ,s+             ;restore last character
        sta     *char
        lbra    12$             ;loop for another macro

20$:    ldx     #q$tble         ;<z> - zero macro table
        ldd     #0
21$:    std     ,x++            ;clear entries
        std     ,x++
        cmpx    #q$tend
        bne     21$
        ldd     #$macro         ;reset end
        std     q$end
        lbra    .$q


q$dlet: bsr     q$mcrn          ;check macro number
        beq     4$              ;length = zero - done
        addd    ,x              ;beginning of next macro in buffer
        tfr     d,y

        ldd     q$end           ;end of buffer
        subd    q$cntr
        subd    ,x              ;last macro in buffer ?
        bls     4$              ;yes - skip
        tfr     d,x
        stu     ,--s            ;save beginning of macro to delete

1$:     ldb     ,y+             ;squeeze byte by byte
        stb     ,u+
        leax    -1,x
        bne     1$
```

```
        ldx     #q$tble         ;update table pointers
2$:     ldd     ,x              ;get beginning
        cmpd    ,s              ;after deleted macro ?
        bls     3$              ;no - skip
        subd    q$cntr          ;length moved
        std     ,x              ;new beginning
3$:     leax    4,x             ;next macro
        cmpx    #q$tend         ;end of macro pointer table ?
        bne     2$              ;no - loop
        leas    2,s             ;pop deleted macro address

4$:     ldx     q$pntr          ;valid entry ?
        beq     5$              ;no - skip
        ldd     #0
        std     ,x              ;clear entry
        std     2,x
5$:     ldd     q$end           ;new total length
        subd    q$cntr
        std     q$end
        rts                     ;exit

q$mcrn: ldd     *number         ;get macro number
        subd    #1              ;0. - 255.
        cmpd    #0d256
        bhs     1$              ;bad number
        lslb                    ;compute pointer address
        rola
        lslb
        rola
        ldx     #q$tble
        leax    d,x             ;address of macro entry
        stx     q$pntr          ;save pointer
        ldu     ,x              ;beginning of macro
        ldd     2,x             ;length of macro
        std     q$cntr
        rts

1$:     ldd     #0              ;no macro
        std     q$pntr
        std     q$cntr
        rts                     ;and exit

        .page
        .sbttl  $a command
;
;  variable usage
;
;         entry xval-xorg and yval-yorg are saved for restoration
;       in svxstp and svystp respectively.
;         val_, fract_, and sign_ are used for temporaries by
;       the sincos and arcalc routines.
;
.$a:    ldd     #0              ;init for circles
        std     *arcbgn         ;beginning of arc
        std     *arcend         ;end of arc
        sta     *piflag         ;pie chart flag
        lda     *rstat1
        anda    #~160
        sta     *rstat1         ;new mode
        jsr     nxtchr          ;get character

        jsr     $dispatch       ;dispatch accordingly
        .byte   'A
        .word   3$              ;arcs
        .byte   'C
        .word   1$              ;circles
        .byte   'P
        .word   2$              ;pies
        .byte   'E
        .word   $aelps          ;ellipses
        .byte   'F
        .word   $afill          ;area fills
        .byte   'L
        .word   $aload          ;pixel load function
        .byte   'R
        .word   $aread          ;pixel read function
        .byte   -1
        .word   errgos
```

```
1$:     sec                     ;character used
        jsr     nxtgos          ;wait for radius

        jsr     absnum          ;absolute value of number
        std     *radius         ;save
        jsr     $afrac          ;set fractional updates
        jsr     arcplot         ;draw the circle
        bra     1$              ;and loop

2$:     sta     *piflag         ;pie chart
3$:     sec                     ;character used
4$:     jsr     nxtgos

        jsr     absnum          ;absolute value of number
        std     *radius         ;save radius
        jsr     $afrac          ;set fractional updates
        jsr     nxtgos

        ldd     *number
        std     *arcbgn         ;save beginning of arc
        jsr     nxtgos

        ldd     *number
        std     *arcend         ;save end of arc
        jsr     arcplot         ;draw arc or pie chart
        bra     4$              ;and loop

        .page
        .sbttl  elliptic functions

$aelps: jsr     nxtchr          ;get character

        jsr     $dispatch       ;dispatch accordingly
        .byte   'A
        .word   3$              ;elliptic arcs
        .byte   'C
        .word   1$              ;elliptic circles
        .byte   'P
        .word   2$              ;elliptic pies
        .byte   -1
        .word   errgos

1$:     sec                     ;character used
        jsr     nxtgos          ;wait for x-radius

        jsr     absnum          ;absolute value of number
        std     *radius         ;save x-radius

        jsr     nxtgos          ;wait for y-radius
        jsr     absnum          ;absolute value of number
        bsr     $afrac          ;calculate fractional updates
        jsr     arcplot         ;draw the ellipse
        bra     1$              ;and loop

2$:     sta     *piflag         ;pie chart
3$:     sec                     ;character used
4$:     jsr     nxtgos

        jsr     absnum          ;absolute value of number
        std     *radius         ;save x-radius
        jsr     nxtgos          ;wait for y-radius

        jsr     absnum          ;absolute value of number
        bsr     $afrac          ;calculate fractional updates
        jsr     nxtgos          ;wait for arc position

        ldd     *number
        std     *arcbgn         ;save beginning of arc
        jsr     nxtgos

        ldd     *number
        std     *arcend         ;save end of arc
        jsr     arcplot         ;draw elliptic arc or elliptic pie chart
        bra     4$              ;and loop

        .page
        .sbttl  elliptic fractional update calculation
```

```
$afrac: ldd     #0              ;init for circles
        std     *afracx+2       ;fractional update of 1.0
        std     *afracy+2
        ldd     #1
        std     *afracx
        std     *afracy

        ldd     *radius         ;radius=number ?
        cmpd    *number
        beq     3$              ;default to circle
        bcs     1$              ;if number>radius - skip

        std     *dsr            ;save x-axis as divisor
        ldd     *number
        std     *dnd            ;save y-axis as dividend
        jsr     divide          ;do 16-bit division
        ldx     #afracy
        bra     2$

1$:     std     *dnd            ;save x-axis as dividend
        ldd     *number
        std     *dsr            ;save y-axis as divisor
        std     *radius         ;and radius
        jsr     divide          ;do 16-bit division
        ldx     #afracx

2$:     ldd     #0              ;set fractional update
        std     ,x
        ldd     qt              ;get 16-bit fraction
        std     2,x
3$:     rts                     ;finished

        .page
        .sbttl  1/4 sine wave table

sintbl: .word   000000,000311,000622,001133,001444,001755,002266,002577
        .word   003110,003421,003732,004243,004554,005065,005376,005707
        .word   006220,006531,007042,007353,007663,010174,010505,011015
        .word   011326,011637,012147,012460,012770,013300,013611,014121
        .word   014431,014741,015251,015561,016071,016401,016711,017221
        .word   017530,020040,020347,020657,021166,021475,022004,022313
        .word   022622,023131,023440,023747,024255,024564,025072,025401
        .word   025707,026215,026523,027031,027337,027644,030152,030457
        .word   030765,031272,031577,032104,032411,032715,033222,033526
        .word   034033,034337,034643,035147,035452,035756,036262,036565
        .word   037070,037373,037676,040201,040503,041006,041310,041612
        .word   042114,042416,042717,043221,043522,044023,044324,044624
        .word   045125,045425,045726,046226,046525,047025,047324,047624
        .word   050123,050422,050720,051217,051515,052013,052311,052607
        .word   053104,053401,053676,054173,054470,054764,055260,055554
        .word   056050,056344,056637,057132,057425,057720,060212,060504
        .word   060776,061270,061561,062052,062343,062634,063124,063415
        .word   063705,064174,064464,064753,065242,065531,066017,066305
        .word   066573,067061,067346,067634,070120,070405,070671,071155
        .word   071441,071725,072210,072473,072755,073240,073522,074004
        .word   074265,074546,075027,075310,075570,076050,076330,076610
        .word   077067,077345,077624,100102,100360,100636,101113,101370
        .word   101645,102121,102375,102651,103124,103377,103652,104124
        .word   104376,104650,105121,105372,105643,106114,106364,106633
        .word   107103,107352,107620,110067,110335,110602,111050,111315
        .word   111561,112025,112271,112535,113000,113243,113505,113747
        .word   114211,114452,114713,115154,115414,115654,116113,116353
        .word   116611,117050,117306,117543,120000,120235,120472,120726
        .word   121161,121414,121647,122102,122334,122565,123017,123250
        .word   123500,123730,124160,124407,124636,125064,125312,125540
        .word   125765,126212,126436,126662,127106,127331,127554,127776
        .word   130220,130441,130662,131103,131323,131542,131762,132201
        .word   132417,132635,133052,133270,133504,133720,134134,134347
        .word   134562,134775,135207,135420,135631,136042,136252,136462
        .word   136671,137100,137306,137514,137721,140126,140333,140537
        .word   140743,141146,141350,141552,141754,142155,142356,142556
        .word   142756,143155,143354,143552,143750,144146,144343,144537
        .word   144733,145126,145321,145514,145706,146077,146270,146461
        .word   146651,147040,147227,147416,147604,147771,150156,150343
        .word   150527,150713,151076,151260,151442,151623,152004,152165
        .word   152345,152524,152703,153062,153237,153415,153572,153746
        .word   154122,154275,154450,154622,154774,155145,155316,155466
```

```
        .word   155635,156005,156153,156321,156467,156634,157000,157144
        .word   157307,157452,157614,157756,160117,160260,160420,160557
        .word   160716,161055,161213,161350,161505,161641,161775,162130
        .word   162263,162415,162547,162700,163030,163160,163307,163436
        .word   163564,163712,164037,164163,164307,164433,164556,164700
        .word   165022,165143,165263,165403,165523,165642,165760,166076
        .word   166213,166330,166444,166557,166672,167004,167116,167227
        .word   167340,167450,167557,167666,167775,170102,170207,170314
        .word   170420,170523,170626,170730,171032,171133,171234,171333
        .word   171433,171531,171630,171725,172022,172116,172212,172305
        .word   172400,172472,172563,172654,172745,173034,173123,173212
        .word   173300,173365,173452,173536,173621,173704,173766,174050
        .word   174131,174211,174271,174351,174427,174505,174563,174640
        .word   174714,174770,175043,175115,175167,175240,175311,175361
        .word   175431,175500,175546,175613,175660,175725,175771,176034
        .word   176076,176140,176202,176243,176303,176342,176401,176440
        .word   176475,176533,176567,176623,176656,176711,176743,176774
        .word   177025,177055,177105,177134,177162,177210,177235,177262
        .word   177306,177331,177354,177376,177417,177440,177461,177500
        .word   177517,177536,177553,177570,177605,177621,177634,177647
        .word   177661,177673,177703,177714,177723,177732,177741,177746
        .word   177754,177760,177764,177767,177772,177774,177775,177776
        .word   177777

        .page
        .sbttl  sine/cosine look up

        ;               value of sine left in valy+2
        ;               sign of sine left in signy

        ;               value of cosine left in valx+2
        ;               sign of cosine left in signx

sincos: std     *angle          ;save angle
        anda    #17             ;mod 4096.
        andb    #376            ;even table increments by 2
        ldx     #yorg           ;sin goes with y
        bsr     1$
        ldd     *angle          ;get angle
        addd    #0d1024         ;cos(a)=sin(a+90)
        anda    #17             ;mod 4096.
        andb    #376            ;even table increments by 2
        ldx     #xorg           ;cos goes with x

1$:     clr     22,x            ;sign of result in sign_
        cmpd    #0d2048         ;angle < 180. degrees ?
        bmi     2$              ;yes - skip
        subd    #0d2048         ;angle >= 180. degrees
        com     22,x            ;sines are negative
2$:     cmpd    #0d1025         ;angle > 90. degrees ?
        bmi     3$              ;no - skip
        pshs    a,b             ;else compute (180. - a)
        ldd     #0d2048
        subd    ,s++
3$:     tfr     d,u             ;this is offset into sintbl
        ldd     sintbl,u        ;get sine
        std     14,x            ;save value in val_+2
        bne     4$              ;zero ?
        clr     22,x            ;then positive
4$:     rts                     ;and finished

        .page
        .sbttl  arc position calculation

        ;       sine(a) must be in valy+2 and its sign in signy
        ;       cosine(a) must be in valx+2 and its sign in signx

        ;       arc x and y positions are left in
        ;                       xstep / ystep

        ;       scaled x and y positions are left in
        ;                       arcvlx / arcvly

arcalc: ldy     #radius
        ldx     #xorg           ;compute radius*cosine(a) first
        bsr     1$

        ldx     #yorg           ;compute radius*sine(a) second
```

```
1$:     bsr     a$mult
        std     2,x             ;save as _step
        ldd     35,x            ;get afrac_
        beq     3$              ;=1.0
2$:     std     14,x            ;store multiplier at val_+2
        leay    30,x            ;and multiplicand is arcvl_ = |_step|
        bsr     a$mult          ;then afrac_*radius*[sine(a)/cosine(a)]
        bra     4$
3$:     ldd     2,x             ;get _step
4$:     std     30,x            ;save fractional in arcvl_
        rts                     ;and exit


a$mult: ldd     #0              ;init product
        std     16,x

        lda     15,x            ;low order of val_+2
        ldb     1,y             ;low order
        mul                     ;first product
        addd    ,y              ;offset of 1 lsb
        std     20,x            ;save low order of product
        bcc     1$              ;skip if no carry
        inc     17,x

1$:     lda     14,x            ;high order of val_+2
        ldb     1,y             ;low order
        mul
        addd    17,x            ;add previous partial
        std     17,x            ;save partial product
        bcc     2$              ;skip if no carry
        inc     16,x

2$:     lda     15,x            ;low order of val_+2
        ldb     ,y              ;high order
        mul
        addd    17,x            ;add previous partial
        std     17,x            ;save partial product
        bcc     3$              ;skip if no carry
        inc     16,x

3$:     lda     14,x            ;high order of val_+2
        ldb     ,y              ;high order
        mul
        addd    16,x            ;add previous partial
        tst     20,x            ;round up ?
        bpl     4$              ;no - skip
        addd    #1
4$:     std     30,x            ;save magnitude in arcvl_
        tst     22,x            ;negative ?
        beq     5$              ;no - skip
        coma
        comb
        addd    #1
5$:     rts                     ;and exit


        .page
        .sbttl  arc length in xy steps calculation

        ;       angle is in angle
        ;       xstep contains r*cos(a)
        ;       ystep contains r*sin(a)

        ;       stcntr is updated with counts

arclen: ldd     *angle          ;get angle
        anda    #17             ;mod 4096.
        tfr     d,x
        ldd     *stcntr         ;current counter value
        cmpx    #0d1024         ;first quadrant ?
        bpl     1$              ;no - skip
        subd    *xstep
        addd    *ystep
        bra     6$
1$:     cmpx    #0d2048         ;second quadrant ?
        bpl     2$              ;no - skip
        subd    *ystep
        subd    *xstep
```

```
        bra     5$
2$:     cmpx    #0d3072         ;third quadrant ?
        bpl     3$
        addd    *xstep
        subd    *ystep
        bra     4$
3$:     addd    *xstep          ;fourth quadrant
        addd    *ystep

        addd    *radius         ;update with radii
        addd    *radius
4$:     addd    *radius
        addd    *radius
5$:     addd    *radius
        addd    *radius
6$:     addd    *radius
        bpl     7$              ;length > 0 - skip
        addd    *radius         ;else add 8*radius
        addd    *radius
        addd    *radius
        addd    *radius
        addd    *radius
        addd    *radius
        addd    *radius
        addd    *radius
7$:     std     *stcntr
        rts

        .page
        .sbttl  right scaling

rscale: asra
        rorb
        asra
        rorb
        asra
        rorb
1$:     rts

        .sbttl  left scaling

lscale: aslb
        rola
        aslb
        rola
        aslb
        rola
1$:     rts

        .sbttl  rounding routine after rscale

round:  tsta                    ;negative # ?
        bmi     1$              ;yes - skip
        bcc     2$              ;no round - skip
        addd    #1
        bra     2$
1$:     bcs     2$              ;no round - skip
        subd    #1
2$:     rts                     ;and exit

        .sbttl  absolute value of number

absnum: ldd     *number         ;take abs(number)
        bpl     1$
        coma
        comb
        addd    #1
        std     *number
1$:     rts

        .page
        .sbttl  arc drawing algorithm

        ;  this arc drawing algorithm was taken from work
        ;  by won l. chung
        ;   computer graphics and image processing.  vol 6, 196-198 (1977)

arcdraw:
```

```
        ldd     *fxyarc         ;do test computation #1
        tst     *yvarc
        bpl     1$
        addd    *xvarc
        addd    *xvarc
        bra     2$
1$:     subd    *xvarc
        subd    *xvarc
2$:     addd    #1
        std     *f1arc
        bpl     3$
        coma
        comb
        addd    #1
3$:     std     *af1arc

        ldd     *fxyarc         ;do test computation #2
        tst     *xvarc
        bpl     4$
        subd    *yvarc
        subd    *yvarc
        bra     5$
4$:     addd    *yvarc
        addd    *yvarc
5$:     addd    #1
        std     *f2arc
        bpl     6$
        coma
        comb
        addd    #1
6$:     std     af2arc

        ldd     *f1arc          ;do test computation #3
        addd    *f2arc
        subd    *fxyarc
        bpl     7$
        coma
        comb
        addd    #1
7$:     std     af3arc

        ;  now compute logical variables

        clr     *avarc
        clr     *bvarc
        clr     *cvarc
        ldd     *af1arc
        subd    *af2arc
        ble     8$
        inc     *avarc
8$:     ldd     *af1arc
        subd    *af3arc
        ble     9$
        inc     *bvarc
9$:     ldd     *af2arc
        subd    *af3arc
        ble     10$
        inc     *cvarc

        ;  now update x,y as computed

10$:    ldd     #0              ;init updates
        std     *f1arc
        std     *f2arc
        lda     *bvarc          ;(b&c)
        anda    *cvarc
        bne     11$
        lda     *avarc          ;(/a&/b)
        ora     *bvarc
        beq     12$
        inc     *f2arc+1        ;(a&/c)
        bra     13$
11$:    inc     *f2arc+1
12$:    inc     *f1arc+1
13$:    ldd     *fxyarc
        ldx     #0
        ldy     #0
        ldu     *upcntr
```

```
        tst     *f1arc+1
        beq     16$
        leau    1,u
        tst     *yvarc
        bmi     14$
        neg     *f1arc+1
        com     *f1arc
        subd    *xvarc
        subd    *xvarc
        leax    -0d8,x
        bra     15$
14$:    addd    *xvarc
        addd    *xvarc
        leax    0d8,x
15$:    addd    #1

16$:    tst     *f2arc+1
        beq     19$
        leau    1,u
        tst     *xvarc
        bpl     17$
        neg     *f2arc+1
        com     *f2arc
        subd    *yvarc
        subd    *yvarc
        leay    -0d8,y
        bra     18$
17$:    addd    *yvarc
        addd    *yvarc
        leay    0d8,y
18$:    addd    #1
19$:    std     *fxyarc         ;new value
        clr     ,-s             ;update flag
        stx     *xstep
        beq     22$
        bmi     20$
        ldd     *valx+2         ;update x-position (+)
        addd    *afracx+2
        std     *valx+2
        ldd     *valx
        stb     ,-s
        adcb    *afracx+1
        adca    *afracx
        bra     21$
20$:    ldd     *valx+2         ;update x-position (-)
        subd    *afracx+2
        std     *valx+2
        ldd     *valx
        stb     ,-s
        sbcb    *afracx+1
        sbca    *afracx
21$:    std     *valx
        eorb    ,s+
        andb    #370            ;real update ?
        beq     22$             ;no - skip
        inc     ,s              ;set flag

22$:    sty     *ystep
        beq     25$
        bmi     23$
        ldd     *valy+2         ;update y-position (+)
        addd    *afracy+2
        std     *valy+2
        ldd     *valy
        stb     ,-s
        adcb    *afracy+1
        adca    *afracy
        bra     24$
23$:    ldd     *valy+2         ;update y-position (-)
        subd    *afracy+2
        std     *valy+2
        ldd     *valy
        stb     ,-s
        sbcb    *afracy+1
        sbca    *afracy
24$:    std     *valy
        eorb    ,s+
```

```
        andb    #370            ;real update ?
        beq     25$             ;no - skip
        inc     ,s              ;set flag

25$:    stu     *upcntr
        ldd     *xvarc          ;update
        addd    *f1arc
        std     *xvarc
        ldd     *yvarc
        addd    *f2arc
        std     *yvarc

        ;  now plot this point

        tst     ,s+             ;update ?
        beq     27$             ;no - skip
        jsr     video

        .if     plotter
        tst     *pltron ;plotter on ?
        beq     26$             ;no - skip
        jsr     val.x           ;computations for plotter
        jsr     val.y
        jsr     vector
        jsr     vect.1
        jsr     plotvp          ;plot vector
        ldx     *fxval          ;update all variables
        stx     *xval
        stx     *valx
        ldx     *fyval
        stx     *yval
        stx     *valy
26$:
        .endif

27$:    rts                     ;and exit

        .page
        .sbttl  calculate 32-bit product of d*d

        ;  calculate 32-bit product of d*d

square: ldu     ,s++            ;save return
        clr     ,-s             ;square of argument
        clr     ,-s             ;left here
        clr     ,-s
        clr     ,-s
        tsta                    ;negative ?
        bpl     1$              ;no - skip
        coma
        comb
        addd    #1
1$:     std     ,--s            ;stack argument
        tfr     b,a
        mul                     ;l*l
        std     4,s
        ldd     ,s
        mul                     ;h*l and l*h
        std     ,--s            ;stack
        addd    ,s++            ;add
        bcc     2$
        inc     2,s
2$:     addd    3,s
        bcc     3$
        inc     2,s
3$:     std     3,s
        ldd     ,s++
        tfr     a,b
        mul                     ;h*h
        addd    ,s
        std     ,s              ;final product
        jmp     ,u              ;and return

        .page
        .sbttl  arc plotting routine

        ;  save certain parameters
```

```
arcplot:
        ldd     *xval
        subd    *xorg
        std     *svxstp
        ldd     *yval
        subd    *yorg
        std     *svystp

        ;  mode is always vector in this command

        lda     *rstat1
        sta     *svstat         ;save original status
        anda    #367            ;vector drawing
        sta     *rstat1

        ;  compute update count

        ldd     #0              ;init stcntr
        std     *stcntr
        std     *upcntr         ;and running counter

        ldd     *arcbgn         ;arc beginning
        jsr     sincos          ;compute lengths
        jsr     arcalc

        ;  calculation of fxyarc initial value

        ldd     *arcvly
        jsr     rscale
        jsr     round
        jsr     lscale
        std     *f2arc
        ldd     *ystep
        jsr     rscale
        jsr     round
        std     *yvarc
        jsr     square          ;y^2

        ldd     *arcvlx
        jsr     rscale
        jsr     round
        jsr     lscale
        std     *f1arc
        ldd     *xstep
        jsr     rscale
        jsr     round
        std     *xvarc
        jsr     square          ;x^2

        ldd     *radius
        jsr     rscale
        jsr     square          ;r^2

        tfr     s,y             ;pointer to y^2
        leay    0d8,y
        tfr     s,x             ;pointer to x^2
        leax    4,x

        lda     #3              ;running offset
        clc
1$:     ldb     a,y
        adcb    a,x             ;x^2 + y^2
        stb     a,x
        deca
        bge     1$

        lda     #3              ;running offset
        clc
2$:     ldb     a,x
        sbcb    a,s             ;(x^2 + y^2) - r^2
        stb     a,s
        deca
        bge     2$

        ldd     2,s
        std     *fxyarc         ;arc limit
        leas    0d12,s          ;pop arguments
```

```
        ;  arclength calculations

        ldd     *arcend
        cmpd    *arcbgn         ;same ?
        bne     3$              ;no - skip
        ldd     *radius         ;stcntr=8*radius updates
        aslb
        rola
        aslb
        rola
        aslb
        rola
        bra     4$
3$:     jsr     arclen
        coma                    ;negate length
        comb
        addd    #1
        std     *stcntr
        ldd     *arcend         ;arc ending
        jsr     sincos          ;compute lengths
        jsr     arcalc
        jsr     arclen
4$:     jsr     rscale          ;scale for circle
        cmpd    #0d8
        ble     5$              ;skip if small
        subd    #2              ;offset
5$:     std     *stcntr

        ;  set up for proc

        ldd     *svxstp
        addd    *f1arc
        std     *xstep
        ldd     *svystp
        addd    *f2arc
        std     *ystep
        tst     *piflag         ;pie chart ?
        beq     6$              ;no - skip
        jsr     .$d             ;yes - drop pen
        bra     7$
6$:     jsr     .$u             ;no - lift pen
7$:     jsr     proc            ;and go
        jsr     .$d             ;now pen is down

        ;  arc plotting loop

        ldd     #0              ;initialize val_ fractions
        std     *valx+2
        std     *valy+2
        ldx     #afracx         ;scale fractional update
        jsr     mul8
        ldx     #afracy
        jsr     mul8
8$:     ldd     *upcntr         ;check for finished
        cmpd    *stcntr
        bpl     9$              ;yes - exit
        jsr     arcdraw         ;do another point on surface
        bra     8$

        ;  complete arc to arcend

9$:     ldd     *valx
        std     *xval
        ldd     *valy
        std     *yval
        ldd     *arcvlx
        addd    *svxstp
        std     *xstep
        ldd     *arcvly
        addd    *svystp
        std     *ystep

        jsr     proc

        ;  finishing touches

        ldd     *svxstp         ;prepare for completion
        std     *xstep
```

```
        ldd     *svystp
        std     *ystep

        tst     *piflag         ;pie chart ?
        bne     10$             ;yes - skip
        jsr     .$u             ;raise pen
10$:    jsr     proc            ;back to original position
        lda     *svstat         ;restore status
        sta     *rstat1
        jmp     .$u             ;raise pen at end always
                                ;finished

        .page
        .sbttl  area fill variable definitions

        cxpos   =arcbgn
        cypos   =arcend

        cleft   =angle
        cright  =radius

        pypos   =cypos
        pleft   =stcntr
        pright  =upcntr

        f$color =avarc
        m$color =bvarc
        fm$tst  =cvarc

        $ansfr  =piflag

        af$top  =af1arc
        af$blk  =af2arc
        af$cbk  =af3arc

        $xymax  =0d8*0d511
        pixel1  =0d8
        pixel2  =0d16

        .page
        .sbttl  area fill sequence

$afill: jsr     nxtchr          ;get character

        jsr     $dispatch       ;dispatch accordingly
        .byte   'B
        .word   1$              ;to any boundary
        .byte   'C
        .word   2$              ;to specified color boundary
        .byte   'R
        .word   $afrec          ;a rectangular region
        .byte   -1
        .word   errgos

1$:     ldb     #20             ;non matchable color
        stb     *m$color
        ldb     #-1
        stb     *fm$tst         ;any color boundary
        bra     3$

2$:     sec                     ;character used
        jsr     nxtgos

        ldb     *number+1       ;get boundary match color
        andb    *$dswrt         ;these bits available
        stb     *m$color
        ldb     #0              ;must match boundary
        stb     *fm$tst

3$:     ldd     *xval           ;scale x-position
        anda    #17
        andb    #370
        std     *cxpos
        std     *crxpos
        std     vidx

        ldd     *yval           ;scale y-position
        coma
```

```
        comb
        anda    #17
        andb    #370
        std     *cypos
        std     *crypos
        std     vidy

        lda     vidplt          ;read data
        anda    *$dswrt         ;these bits available
        sta     *f$color
        cmpa    *m$color        ;match search color ?
        beq     7$              ;yes - exit

        jsr     l.scnm          ;find left boundary
        std     *cleft
        jsr     r.scnm          ;find right boundary
        std     *cright
        jsr     dwline          ;draw line segment

        ;  push 'upscan' block

        ldu     #af$bgn         ;buffer pointer
        ldd     *cypos
        subd    #pixel1
        blt     4$
        std     ,u++
        ldd     *cleft
        std     ,u++
        ldd     *cright
        std     ,u++
        ldd     #177400
        std     ,u++

        ;  push 'dnscan block'

4$:     ldd     *cypos
        addd    #pixel1
        cmpd    #$xymax
        bgt     5$
        std     ,u++
        ldd     *cleft
        std     ,u++
        ldd     *cright
        std     ,u++
        ldd     #177777
        std     ,u++
5$:     stu     *af$top

        ;  do filling

6$:     jsr     $apull          ;get a scan block
        bcc     7$              ;exit if none
        jsr     rsscan          ;do rsscan
        bra     6$
7$:     sec
        rts                     ;area filled

        .page
        .sbttl  pull scan block from buffer

        ;  retrieve a scan block
        ;  load vidy register

$apull: ldu     *af$top         ;top of area pointer
        bra     2$
1$:     leau    -6,u            ;to next scan block
2$:     cmpu    #af$bgn         ;bottom of area
        bls     3$              ;yes - exit
        stu     *af$top         ;save pointer
        ldd     ,--u            ;get block type
        bpl     1$              ;deleted - loop
        ldd     ,--u
        std     *pright
        ldd     ,--u
        std     *pleft
        ldd     ,--u
        std     *pypos          ;(=cypos)
        std     *crypos
```

```
        std     vidy            ;load y-line
        stu     *af$cbk         ;current block pointer
        sec
        rts
3$:     stu     *af$top         ;save pointer
        clc
        rts

        .page
        .sbttl  push scan block to buffer

        ;  check buffer for space
        ;       return block pointer in y
        ;       else remove deleted scan blocks and squeeze buffer

        ;       'c'=1 if buffer space available
        ;       'c'=0 if no space in buffer

$apush: ldy     *af$top
        cmpy    #af$end         ;at top of buffer ?
        blo     5$              ;no - space ready
        ldx     #af$bgn-0d8     ;do buffer squeeze
1$:     leax    0d8,x           ;update buffer pointer
2$:     cmpx    *af$top         ;end of buffer ?
        bhs     4$              ;yes - skip
        tst     6,x             ;deleted block ?
        bne     1$              ;no - skip this block
        leay    -0d8,y          ;new top of buffer
        sty     *af$top         ;and buffer pointer
        tst     6,y             ;deleted block ?
        beq     2$              ;yes - deleted !
        cmpy    *af$cbk         ;currently checking this block ?
        bne     3$              ;no - skip
        stx     *af$cbk
        tfr     x,u             ;new pointers
3$:     ldd     ,y
        std     ,x++            ;replace
        ldd     2,y
        std     ,x++
        ldd     4,y
        std     ,x++
        ldd     6,y
        std     ,x++
        bra     1$
4$:     cmpy    #af$end         ;any room ?
        bcs     5$
        pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~40
        sta     *outpl          ;buffer error
        sta     plout
        puls    cc
5$:     rts                     ;c=1(yes), c=0(no)

        .page
        .sbttl  search scan block buffer

        ;  search the buffer for scan blocks
        ;at the current line position and within the scan
        ;boundaries of cleft,cright.  return the block pointer
        ;with the minimum bleft.

        ;  during scan, block buffer is squeezed

        ;  enter with y loaded
        ;  y=177400     search for undeleted 'upscan blocks'
        ;  y=177777     search for undeleted 'dnscan blocks'

findbk: ldu     #0              ;0 = none found
        ldx     #af$bgn-0d8     ;beginning of scan block buffer
        ldd     *cright         ;highest boundary
        std     *af$blk         ;save
1$:     ldd     *cypos          ;current scanning line
2$:     leax    0d8,x           ;update block pointer
3$:     cmpx    *af$top         ;at end of buffer ?
        bhs     6$              ;yes - exit
        tst     6,x             ;a deleted block ?
```

```
        bne     5$              ;no - skip
        sty     ,--s            ;save mode
        ldy     *af$top         ;top of buffer pointer
        leay    -0d8,y
        sty     *af$top         ;save updated pointer
        tst     6,y             ;deleted block
        beq     4$              ;yes - skip
        ldd     ,y              ;replace deleted block
        std     ,x
        ldd     2,y
        std     2,x
        ldd     4,y
        std     4,x
        ldd     6,y
        std     6,x
        ldd     *cypos          ;restore used registers
4$:     ldy     ,s++
        bra     3$
5$:     cmpd    ,x              ;line match ?
        bne     2$              ;no - loop
        cmpy    6,x             ;match block type ?
        bne     2$              ;no - loop
        ldd     *cright         ;check boundaries
        addd    #pixel1
        subd    2,x             ;cright+1 < bleft
        blt     1$
        ldd     *cleft
        subd    #pixel1
        subd    4,x             ;cleft-1 > bright
        bgt     1$
        ldd     2,x             ;bleft
        cmpd    *af$blk         ;check for minimum
        bgt     1$
        std     *af$blk         ;save minimum
        tfr     x,u             ;and pointer to this block
        bra     1$              ;and loop
6$:     stu     *af$blk         ;save pointer
        beq     7$              ;skip if none found
        sec
        rts
7$:     clc
        rts

        .page
        .sbttl  scan left until 'color match'

        ;  returns      'c'=0 if first point matches
        ;                     position is <cxpos>
        ;               'c'=1 for left boundary found
        ;                     match/boundary is <cxpos>-1

        ;                    cxpos left in <d>

l.scnm: ldd     *cxpos          ;position
        std     *crxpos
        std     vidx
        lda     vidplt          ;read data
        anda    *$dswrt         ;these bits available
        tst     *fm$tst         ;match/any ?
        bne     1$              ;any - skip
        cmpa    *m$color        ;color match ?
        beq     6$              ;yes - exit
        bra     2$
1$:     cmpa    *f$color        ;first color ?
        bne     6$              ;no - exit
2$:     ldd     *cxpos          ;position
        beq     5$              ;display boundary
        subd    #pixel1         ;left shift 1 bit
        std     *cxpos          ;new position
        std     *crxpos
        std     vidx
        lda     vidplt          ;read data
        anda    *$dswrt         ;these bits available
        tst     *fm$tst         ;match/any ?
        bne     3$              ;any - skip
        cmpa    *m$color        ;color match ?
        bne     2$              ;no - loop
        bra     4$
```

```
3$:     cmpa    *f$color        ;first color ?
        beq     2$              ;yes - loop
4$:     ldd     *cxpos          ;backoff from boundary
        addd    #pixel1
        std     *cxpos
5$:     sec
        rts
6$:     ldd     *cxpos
        clc
        rts

        .page
        .sbttl  scan right until 'color match'

        ;  returns      'c'=0 if first point matches
        ;                     position is <cxpos>
        ;               'c'=1 for right boundary found
        ;                     match/boundary is <cxpos>+1

        ;                     cxpos left in <d>

r.scnm: ldd     *cxpos          ;position
        std     *crxpos
        std     vidx
        lda     vidplt          ;read data
        anda    *$dswrt         ;these bits available
        tst     *fm$tst         ;match/any ?
        bne     1$              ;any - skip
        cmpa    *m$color        ;color match ?
        beq     6$              ;yes - exit
        bra     2$
1$:     cmpa    *f$color        ;first color ?
        bne     6$              ;no - exit
2$:     ldd     *cxpos          ;position
        cmpd    #$xymax         ;at display boundary ?
        beq     5$              ;yes - exit
        addd    #pixel1         ;right shift 1 bit
        std     *cxpos          ;new position
        std     *crxpos
        std     vidx
        lda     vidplt          ;read data
        anda    *$dswrt         ;these bits available
        tst     *fm$tst         ;match/any ?
        bne     3$              ;any - skip
        cmpa    *m$color        ;color match ?
        bne     2$              ;no - loop
        bra     4$
3$:     cmpa    *f$color        ;first color ?
        beq     2$              ;yes - loop
4$:     ldd     *cxpos          ;backoff from boundary
        subd    #pixel1
        std     *cxpos
5$:     sec
        rts
6$:     ldd     *cxpos
        clc
        rts

        .page
        .sbttl  scan right until 'no color match'

        ;  returns      'c'=0 scanned right to previous line boundary
        ;                     position is <cxpos>
        ;               'c'=1 for left boundary of 'no color match' found
        ;                     match/boundary is <cxpos>-1

        ;                     cxpos left in <d>

r.scnn: ldd     *cxpos          ;position
        cmpd    *pright         ;previous segment limit ?
        bge     4$              ;yes - exit
        addd    #pixel1         ;right shift 1 bit
        std     *cxpos          ;new position
        std     *crxpos
        std     vidx
        lda     vidplt          ;read data
        anda    *$dswrt         ;these bits available
        tst     *fm$tst         ;match/any ?
```

```
        bne     2$              ;any - skip
        cmpa    *m$color        ;color match ?
        beq     r.scnn          ;yes - loop
        bra     3$
2$:     cmpa    *f$color        ;first color ?
        bne     r.scnn          ;no - loop
3$:     ldd     *cxpos
        sec
        rts
4$:     clc
        rts

        .page
        .sbttl  area fill draw line

dwline: ldy     #.$ptrn+2       ;area fill pattern pointer
        ldx     *cleft          ;current segment left boundary
        stx     *crxpos
        stx     vidx

        ldb     *cypos+1        ;current line
        andb    #170
        asrb
        asrb
        ldd     b,y             ;get pattern for this line
        std     ,--s            ;save pattern
        beq     2$
        cmpd    #-1             ;all one's ?
        beq     2$
        ldb     *cleft+1        ;current writing position
        andb    #170            ;bits to rotate
        beq     2$              ;at boundary - skip
        asrb
        asrb
        asrb
1$:     lda     1,s             ;lsb
        rora
        ror     ,s
        ror     1,s
        decb
        bgt     1$              ;rotate for <b> bits

2$:     lda     1,s             ;rotate pattern
        rora
        ror     ,s
        ror     1,s             ;draw this point ?
        bcc     5$              ;no - skip

        clrb                    ;init write color
        lda     *rstat1
        bita    #1              ;replace/complement mode ?
        beq     3$              ;no - skip

        ldb     vidplt          ;get color at point
        lda     *rstat1
        bita    #4              ;replace or complement ?
        beq     3$              ;replace - skip
        eorb    *gcolor         ;complement selected planes
        bra     4$

3$:     orb     *gcolor         ;set selected planes
4$:     stb     *$dsdat         ;current state
        stb     bp$dat
        stb     vidplt          ;write pixel

5$:     leax    pixel1,x
        stx     *crxpos
        stx     vidx
        cmpx    *cright         ;finished ?
        bls     2$              ;no - loop
        leas    2,s             ;pop pattern
        rts                     ;finished

        .page
        .sbttl  rsscan routine
;
;  loop         scan left to boundary
;               scan right to boundary
```

```
;
;               no area - segment finished
;
;               resolve multiple 'upscans' this line segment
;
;               resolve multiple 'dnscans' this line segment
;
;               scan for segments this line
;                       found - loop
;                       else  - finished
;
;       at entry:
;               cypos contains the current y line position
;
;       find first fill segment
;
rsscan: ldd     *pleft          ;set up pointer
        std     *cxpos
1$:     jsr     l.scnm          ;scan left for a match
        bcs     2$              ;'c'=1 if left boundary found
        jsr     r.scnn          ;'c'=0 if on boundary, scan until off boundary
        bcc     3$              ;'c'=0 if no area found, finished
2$:     std     *cleft          ;save left boundary
        jsr     r.scnm          ;scan right until a boundary is found
        std     *cright         ;save right boundary

        jsr     dwline          ;fill current segment

        ldd     *cleft          ;save min/max boundary
        std     ,--s
        jsr     dnrslv          ;resolve 'dnscan blocks'

        ldd     ,s++
        std     *cleft
        jsr     uprslv          ;resolve 'upscan blocks'
        bra     4$

3$:     ldx     *af$cbk         ;delete this block
        clr     6,x

4$:     ldx     #af$bgn-0d8     ;buffer pointer
        ldy     *pypos
5$:     leax    0d8,x
        cmpx    *af$top         ;at top of buffer ?
        bhs     6$              ;yes - skip
        cmpy    ,x              ;this line ?
        bne     5$              ;no - loop
        tst     6,x             ;deleted ?
        beq     5$              ;yes - loop
        stx     *af$cbk         ;save pointer
        ldd     2,x
        std     *pleft
        ldd     4,x
        std     *pright
        bra     rsscan          ;process this segment
6$:     rts                     ;finished

        .page
        .sbttl  resolve down scan blocks

dnrslv: ldd     *cright         ;check segment
        cmpd    *cleft
        blt     1$              ;finished - skip
        ldy     #177777         ;undeleted down scan search
        jsr     findbk
        bcs     3$              ;found a block - skip

        ;  else build upscan block

        ldy     *af$top         ;check for any space
        cmpy    #af$end
        bhs     2$
        ldd     *cypos
        subd    #pixel1
        blt     1$
        std     ,y++
        ldd     *cleft
        std     ,y++
```

```
        ldd     *cright
        std     ,y++
        ldd     #177400
        std     ,y++
        sty     *af$top
1$:     rts                     ;finished
2$:     pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~40
        sta     *outpl          ;buffer error
        sta     plout
        puls    cc
        rts                     ;finished

        ;  if   cleft - 1pixel > bleft

3$:     ldd     *cleft
        subd    #pixel1
        subd    2,u
        ble     4$

        ;  push a 'dnscan block'

        jsr     $apush          ;get a scan block pointer
        bcc     5$              ;no room - skip
        ldd     ,u
        std     ,y++
        ldd     2,u
        std     ,y++
        ldd     *cleft
        subd    #pixel2
        std     ,y++
        ldd     #177777
        std     ,y++
        sty     *af$top
        bra     5$

        ;  if   cleft + 1pixel < bleft

4$:     addd    #pixel2
        bge     5$

        ;  push an 'upscan block'

        jsr     $apush          ;get scan block pointer
        bcc     5$              ;no room - skip
        ldd     *cypos
        subd    #pixel1
        blt     5$
        std     ,y++
        ldd     *cleft
        std     ,y++
        ldd     2,u
        subd    #pixel2
        std     ,y++
        ldd     #177400
        std     ,y++
        sty     *af$top

        ;  if   cright + 1pixel >= bright

5$:     ldd     *cright
        addd    #pixel1
        subd    4,u
        blt     6$

        ;  delete scan block

        clr     6,u
        ldd     4,u             ;update left boundary
        addd    #pixel2
        std     *cleft
        jmp     dnrslv

        ;  else update 'dnscan block'

6$:     ldd     *cright
```

```
        addd    #pixel2
        std     2,u
        rts

        .page
        .sbttl  resolve up scan blocks

uprslv: ldd     *cright         ;check segment
        cmpd    *cleft
        blt     1$              ;finished - skip
        ldy     #177400         ;undeleted up scan search
        jsr     findbk
        bcs     3$              ;found a block - skip

        ;  else build dnscan block

        ldy     *af$top         ;check for space
        cmpy    #af$end
        bhs     2$
        ldd     *cypos
        addd    #pixel1
        cmpd    #$xymax
        bgt     1$
        std     ,y++
        ldd     *cleft
        std     ,y++
        ldd     *cright
        std     ,y++
        ldd     #177777
        std     ,y++
        sty     *af$top
1$:     rts                     ;finished
2$:     pshs    cc
        orcc    #120            ;hold interrupts
        lda     *outpl
        anda    #~40
        sta     *outpl          ;buffer error
        sta     plout
        puls    cc
        rts                     ;finished

        ;  if   cleft - 1pixel > bleft

3$:     ldd     *cleft
        subd    #pixel1
        subd    2,u
        ble     4$

        ;  push an 'upscan block'

        jsr     $apush          ;get a scan block pointer
        bcc     5$              ;no room - skip
        ldd     ,u
        std     ,y++
        ldd     2,u
        std     ,y++
        ldd     *cleft
        subd    #pixel2
        std     ,y++
        ldd     #177400
        std     ,y++
        sty     *af$top
        bra     5$

        ;  if   cleft + 1pixel < bleft

4$:     addd    #pixel2
        bge     5$

        ;  push a 'dnscan block'

        jsr     $apush          ;get scan block pointer
        bcc     5$              ;no room - skip
        ldd     *cypos
        addd    #pixel1
        cmpd    #$xymax
        bgt     5$
        std     ,y++
```

```
        ldd     *cleft
        std     ,y++
        ldd     2,u
        subd    #pixel2
        std     ,y++
        ldd     #177777
        std     ,y++
        sty     *af$top

        ;  if   cright + 1pixel >= bright

5$:     ldd     *cright
        addd    #pixel1
        subd    4,u
        blt     6$

        ;  delete scan block

        clr     6,u
        ldd     4,u             ;update left boundary
        addd    #pixel2
        std     *cleft
        jmp     uprslv

        ;  else update 'upscan block'

6$:     ldd     *cright
        addd    #pixel2
        std     2,u
        rts

        .page
        .sbttl  rectangular region area fill

$afrec: sec
        jsr     nxtgos
        ldd     *number         ;compute screen position
        addd    *xorg           ;x0
        anda    #17
        std     *cleft

        jsr     nxtgos
        ldd     *number         ;y0
        addd    *yorg
        coma
        comb
        anda    #17
        std     *cypos

        jsr     nxtgos
        ldd     *number         ;x1
        addd    *xorg
        anda    #17
        std     *cright
        cmpd    *cleft          ;order x0 and x1
        bhs     1$              ;x1 > x0
        ldx     *cleft
        std     *cleft
        stx     *cright         ;now - x1 > x0

1$:     jsr     nxtgos
        ldd     *number         ;y1
        addd    *yorg
        coma
        comb
        anda    #17
        std     *cxpos
        cmpd    *cypos          ;order y0 and y1
        bhs     2$              ;y1 > y0
        ldx     *cypos
        std     *cypos
        stx     *cxpos          ;now - y1 > y0

2$:     ldy     *cypos          ;load coordinants

3$:     sty     *cypos
        sty     *crypos
        sty     vidy
```

```
        jsr     dwline          ;draw the line
        ldy     *cypos
        leay    0d8,y           ;update y
        cmpy    *cxpos          ;finished ?
        ble     3$              ;no - loop
        bra     $afrec          ;loop for another region

        .page
        .sbttl  get color routine

        ;       cxpos contains the x-position in the line
        ;       cypos contains the y-position in the display

$acolr: ldd     *cypos
        std     *crypos
        std     vidy
        ldd     *cxpos
        std     *crxpos
        std     vidx
        lda     vidplt
        anda    *$dswrt         ;these bits available
        rts

        .page
        .sbttl  load data into display routine

$aload: ldd     *xval           ;set up position
        anda    #17
        andb    #370
        std     *cxpos
        ldd     *yval
        coma
        comb
        anda    #17
        andb    #370
        std     *cypos

1$:     sec                     ;character used
        jsr     nxtgos          ;wait for count

2$:     ldd     *number
        bne     3$              ;skip if points
        rts                     ;exit if no points

3$:     lda     *char           ;check character mode
        cmpa    #'+             ;'+' ?
        beq     4$              ;yes - skip
        lda     *lstat
        ora     #200
        sta     *lstat          ;character mode
        clc                     ;character not used
        jsr     nxtgos          ;wait for character
        bra     3$

4$:     lda     *lstat
        ora     #220
        sta     *lstat          ;special character mode
        sec                     ;character used
        jsr     nxtgos          ;wait for character

        lda     *char           ;get color
        anda    #0xF0
        lsra
        lsra
        lsra
        lsra
        sta     ,-s             ;save second pixel
        lda     *char           ;get color
        anda    #0x0F
        sta     *char           ;save first pixel
        jsr     6$              ;process pixel 1
        lda     ,s+             ;next pixel
        sta     *char

        ldd     *number         ;update count
        subd    #1
        std     *number         ;finished ?
        lbeq    1$              ;yes - loop
```

```
        jsr     6$              ;process pixel 2

        ldd     *number         ;update count
        subd    #1
        std     *number         ;finished ?
        lbeq    1$              ;yes - loop
        lda     *lstat
        ora     #220
        sta     *lstat          ;special character mode
        sec                     ;character used
        rts

6$:     jsr     $acolr          ;read color
        sta     ,-s             ;save
        clrb                    ;init write color

        lda     *rstat1
        bita    #1              ;replace/complement mode ?
        beq     7$              ;no - skip

        ldb     ,s              ;get color

        lda     *rstat1
        bita    #4              ;replace or complement ?
        beq     7$              ;replace - skip
        eorb    *char           ;complement selected planes
        bra     8$

7$:     orb     *char           ;set selected planes
8$:     stb     *$dsdat         ;current state
        stb     bp$dat
        stb     vidplt          ;write pixel
        leas    1,s             ;pop old color

        ldd     *cxpos
        addd    #pixel1
        std     *cxpos
        cmpd    #$xymax
        ble     10$
        ldd     #0
        std     *cxpos
        ldd     *cypos
        subd    #pixel1
        cmpd    #$xymax
        ble     9$
        ldd     #0
9$:     std     *cypos
10$:    rts

        .page
        .sbttl  read back data routine

$aread: ldd     *xval           ;set up position
        anda    #17
        andb    #370
        std     *cxpos
        ldd     *yval
        coma
        comb
        anda    #17
        andb    #370
        std     *cypos

        sec                     ;character used
        jsr     nxtgos          ;wait for pixel count

        ldd     *number
        beq     3$              ;exit for no points to transfer

1$:     jsr     $acolr          ;color at this point
        sta     ,-s             ;save color
        jsr     4$              ;update pixel position
        ldd     *number         ;update pixels left
        subd    #1
        beq     2$              ;exit
        std     *number
```

```
        jsr     $acolr          ;color at this point
        lsla
        lsla
        lsla
        lsla
        ora     ,s              ;combine pixels
        sta     ,s              ;save
        jsr     4$              ;update pixel position

2$:     ldb     ,s+             ;get pixel(s)
        jsr     plcbuf          ;put into buffer

        ldd     *number         ;update pixels left
        subd    #1
        std     *number
        bne     1$              ;loop for more pixel(s)
3$:     clc                     ;character not used
        rts                     ;finished

4$:     ldd     *cxpos          ;update position
        addd    #pixel1
        std     *cxpos
        cmpd    #$xymax
        ble     6$
        ldd     #0              ;else reset position
        std     *cxpos
        ldd     *cypos
        subd    #pixel1
        cmpd    #$xymax
        ble     5$
        ldd     #0
5$:     std     *cypos
6$:     rts

        .page
        .sbttl  display system buffer

        .area   BUFSAV

oldpnl: .blkb   0d32            ;old panel data
                                ;(previous clock tick)

        .area   DPVSAV

hs$sts: .byte   0,0     ;histo enable bits
                        ;bit 0  - $hsto0 ...
                        ;bit 15 - $hstof

xc$sts: .byte   0,0     ;horizontal-cursor enable bits
                        ;bit 0  - xcurs0 ...
                        ;bit 15 - xcursf

yc$sts: .byte   0,0     ;vertical-cursor enable bits
                        ;bit 0  - ycurs0 ...
                        ;bit 15 - ycursf

oc$sts: .byte   0,0     ;'oc' buffer control
                        ;bit 0  - character rom select
                        ;bit 1  - 'oc' buffer select
                        ;bit 7  - 'oc' on

pr$sts: .byte   0,0     ;printer dump start/stop control
                        ;bit 7  - start/stop printer dump

        .byte   0,0     ;****

pl$sts: .byte   0,0     ;control panel status
                        ;bit 7  - (1) panel is attached

pl$nbl: .byte   0,0     ;panel enable flag
                        ;bit 7  - (1) enable scan of panel
                        ;bit 6  - (1) enable track ball updating

hs$clr: .byte   0,0     ;(red) histo & cursor color
        .byte   0,0     ;(green)
        .byte   0,0     ;(blue
        .byte   0,0     ;****
```

```
oc$clr: .byte   0,0     ;(red) 'oc' color
        .byte   0,0     ;(green)
        .byte   0,0     ;(blue)
        .byte   0,0     ;****

        .byte   0,0     ;****
        .byte   0,0     ;****
        .byte   0,0     ;****
        .byte   0,0     ;****
        .byte   0,0     ;****
        .byte   0,0     ;****
        .byte   0,0     ;****
        .byte   0,0     ;****

        .page
        .sbttl  trackball variables

                        ;x-axis trackball rotation
                        ;x+ --> $panel+34
                        ;x- --> $panel+35
        .byte   0,0     ;x+
        .byte   0,0     ;x-
        .byte   0,0     ;summation <x+> - <x->
        .byte   0,0     ;summation <x+> - <x->


                        ;y-axis trackball rotation
                        ;y+ --> $panel+36
                        ;y- --> $panel+37
        .byte   0,0     ;y+
        .byte   0,0     ;y-
        .byte   0,0     ;summation <y+> - <y->
        .byte   0,0     ;summation <y+> - <y->

        .page
        .sbttl  display system panel switches

                        ;thumbwheel #1
                        ;$panel
        .byte   0,0     ;     1's digit
        .byte   0,0     ;    10's digit
        .byte   0,0     ;   100's digit
        .byte   0,0     ;  1000's digit
        .byte   0,0     ; 10000's digit
        .byte   0,0     ;100000's digit

                        ;thumbwheel #2
                        ;$panel+6
        .byte   0,0     ;     1's digit
        .byte   0,0     ;    10's digit
        .byte   0,0     ;   100's digit
        .byte   0,0     ;  1000's digit
        .byte   0,0     ; 10000's digit
        .byte   0,0     ;100000's digit

                        ;$panel+14
        .byte   0,0     ;rotary switch #1
        .byte   0,0     ;rotary switch #2
        .byte   0,0     ;rotary switch #3
        .byte   0,0     ;rotary switch #4


                        ;$panel+20
        .byte   0,0     ;toggle switches
        .byte   0,0     ;pushbutton switches
        .byte   0,0     ;cursor select switches
        .byte   0,0
        .byte   0,0
        .byte   0,0
        .byte   0,0
        .byte   0,0

                        ;$panel+30
        .byte   0,0     ;toggle leds
        .byte   0,0     ;pushbutton leds
        .byte   0,0     ;function leds
        .byte   0,0
        .byte   0,0
```

```
        .byte   0,0
        .byte   0,0
        .byte   0,0


        ; the display of each of the histogram regions is
        ;controlled by a table containing the following
        ;information:
        ;       histo data location     2-bytes
        ;       starting x position     2-bytes
        ;       step size for x         2-bytes
        ;       number of points        2-bytes

dsphst: .blkb   0d128           ;histo display parameters


        ; the display of each of the display x-cursors is
        ;controlled by a table containing the following
        ;information:
        ;       x-left position         2-bytes
        ;       y-position              2-bytes
        ;       x-length                2-bytes
        ;       cursor pattern          2-bytes

xcursr: .blkb   0d128


        ; the display of each of the display y-cursors is
        ;controlled by a table containing the following
        ;information:
        ;       x-position              2-bytes
        ;       y-bottom position       2-bytes
        ;       y-length                2-bytes
        ;       cursor pattern          2-bytes

ycursr: .blkb   0d128

        .page
        .sbttl  host system init service request

        .area   EXTSAV

$nmi:   lda     #>direct
        tfr     a,dp            ;set direct page

        ldx     #0              ;fast but long
        ldy     #0
        ldu     #dpstat+0d32    ;clear 16. internal variables
        pshu    x,y
        pshu    x,y
        pshu    x,y
        pshu    x,y
        pshu    x,y
        pshu    x,y
        pshu    x,y
        pshu    x,y
        ldx     *spjump         ;a diagnostic process ?
        beq     1$              ;no - exit
        jsr     ,x              ;else do process
1$:     rti

        .page
        .sbttl  trackball service 'plus x' rotation

tbxpls: tst     $panel+34       ;clear interrupt
        ldu     #dpstat+0d48    ;x-area
        bra     1$

        .sbttl  trackball service 'plus y' rotation

tbypls=.
        tst     $panel+36       ;clear interrupt
        ldu     #dpstat+0d56    ;y-area
1$:     lda     pl$nbl+1        ;get panel control
        bita    #100            ;trackball enabled ?
        beq     2$              ;no - skip update
        ldd     ,u              ;update terms
        addd    #1
```

```
        std     ,u
        ldd     4,u
        addd    #1
        std     4,u
        ldd     6,u
        addd    #1
        std     6,u
2$:     rti

        .page
        .sbttl  trackball service 'minus x' rotation

tbxmns: tst     $panel+35       ;clear interrupt
        ldu     #dpstat+0d48    ;x-area
        bra     1$

        .sbttl  trackball service 'minus y' rotation

tbymns=.
        tst     $panel+37       ;clear interrupt
        ldu     #dpstat+0d56    ;y-area
1$:     lda     pl$nbl+1        ;get panel control
        bita    #100            ;trackball enabled ?
        beq     2$              ;no - skip update
        ldd     2,u             ;update terms
        subd    #1
        std     2,u
        ldd     4,u
        subd    #1
        std     4,u
        ldd     6,u
        subd    #1
        std     6,u
2$:     rti

        .page
        .sbttl  display system servicing

        ; panel update

$dsrvc: ldu     #$panel ;address of panel
        lda     ,u              ;panel here ?
        anda    #160
        beq     1$              ;yes - skip
        clr     pl$sts+1        ;panel not here
        bra     10$             ;skip update

1$:     lda     #200            ;set status
        sta     pl$sts+1        ;panel here
        tst     pl$nbl+1        ;check panel ?
        bpl     10$             ;no - skip update

        ldy     #dpstat+0d65    ;status area
        ldx     #oldpnl         ;last clock tick data
        ldb     #0d16           ;total encoded switch count

2$:     lda     ,u+             ;get new panel data
        bmi     3$              ;negative - switch open
        cmpa    ,x+             ;same as last time ?
        bne     4$              ;no - skip
        sta     ,y++            ;else place new data
        bra     5$
3$:     leax    1,x             ;update position
4$:     sta     -1,x            ;replace old with current
        leay    2,y             ;skip output
5$:     decb                    ;loop for all switches
        bgt     2$

        ldb     #3              ;total toggle/push button switch count

6$:     lda     ,u+             ;get new panel data
        cmpa    ,x+             ;same as last time ?
        bne     7$              ;no - skip
        sta     ,y++            ;else place new data
        bra     8$
7$:     sta     -1,x            ;replace old with current
        leay    2,y             ;skip output
8$:     decb                    ;loop for all switches
```

```
        bgt     6$

        ldu     #$panel+30      ;panel address
        ldy     #dpstat+0d113   ;status area
        ldb     #3              ;3 output registers

9$:     lda     ,y++            ;get data
        coma
        sta     ,u+
        decb                    ;loop for all leds
        bgt     9$

        ; check printer dump/stop

10$:    lda     pr$sts+1        ;check printer dump start/stop flag
        bpl     11$             ;none - skip
        anda    #177            ;clear bit
        sta     pr$sts+1
        lda     *rstat2
        anda    #~2
        sta     *rstat2         ;indicate dump button pushed !

        ; .1 second updates (high priority)

11$:    dec     *ds$tim         ;update time ?
        bgt     12$             ;no - skip
        lda     #6              ;.1 second updates
        sta     *ds$tim

        lda     *$dssts
        bita    #1              ;'oc' being updated ?
        bne     12$             ;yes - wait until next time
        lda     *$dssts
        ora     #1
        sta     *$dssts         ;updating 'oc'
        jsr     oc$upd          ;do 'oc' update
        lda     *$dssts
        anda    #~1
        sta     *$dssts         ;finished updating

        ; .1 second updates (low priority)

        lda     *$dssts
        bita    #4              ;histo/cursor being updating ?
        bne     12$             ;yes - wait until next time
        lda     *$dssts
        ora     #4
        sta     *$dssts         ;updating histo/cursor
        jsr     hstcur
        lda     *$dssts
        anda    #~4
        sta     *$dssts         ;finished updating

        ; check 'busy' status to reduce update rate

        lda     *outpl
        bita    #100            ;'busy' ?
        bne     12$             ;no - skip
        lda     #0d15           ;wait .25 seconds
        sta     *ds$tim         ;before next update

12$:    rti

        .page
        .sbttl  oc display checks

oc$upd: lda     oc$sts+1        ;'oc' on ?
        bpl     5$              ;no - skip

        lda     *$dssts
        bita    #2              ;just turned on ?
        bne     1$              ;no - skip

        jsr     setmap          ;setup mapping
        lda     *$dssts
        ora     #2
        sta     *$dssts         ;'oc' on
```

```
1$:     lda     oc$sts+1
        bita    #2              ;second buffer ?
        bne     2$              ;yes - skip
        ldu     #$ocbf0         ;address of first buffer
        bra     3$
2$:     ldu     #$ocbf1         ;address of second buffer
3$:     anda    #3              ;mask selects
        sta     ,-s
        lda     *outpl
        anda    #374
        ora     ,s+
        sta     *outpl          ;and save
        sta     plout
        andcc   #257            ;enable interrupts
        lda     #0d32           ;32. x 64. bytes to transfer
        sta     *occntr         ;counter

4$:     pulu    a,b,x,y         ;quick 64. byte transfer
        pulu    a,b,x,y
        pulu    a,b,x,y
        pulu    a,b,x,y
        pulu    a,b,x,y
        pulu    a,b,x,y
        pulu    a,b,x,y
        pulu    a,b,x,y
        pulu    a,b,x,y
        pulu    a,b,x,y
        pulu    x,y
        dec     *occntr
        bgt     4$

        orcc    #120            ;disable interrupts
        lda     *$blank
        ora     #20
        sta     *$blank         ;enable 'oc' display
        sta     bp$dsp
        lda     *$blank+1
        ora     #20
        sta     *$blank+1
        lda     *$blank+2
        ora     #20
        sta     *$blank+2
        rts

5$:     lda     *$blank
        anda    #~20
        sta     *$blank         ;disable 'oc' display
        sta     bp$dsp
        lda     *$blank+1
        anda    #~20
        sta     *$blank+1
        lda     *$blank+2
        anda    #~20
        sta     *$blank+2

        lda     *$dssts
        bita    #2              ;just turned off ?
        beq     6$              ;no - skip
        anda    #374
        sta     *$dssts         ;'oc' off
        jsr     rstmap          ;reset color mapping
6$:     rts

        .page
        .sbttl  histo and cursor processor

hstcur: ldx     #dpstat ;check if anything on
        lda     ,x+
        ora     ,x+
        ora     ,x+
        ora     ,x+
        ora     ,x+
        ora     ,x+
        beq     2$              ;no - do termination of display

        lda     *$dssts
        bita    #10             ;display just turned on ?
        bne     1$              ;no - skip
```

```
        jsr     setmap          ;go build proper color table
                                ;for display of histos and cursors
        jsr     cpydrw          ;copy histo draw routine into bufsav
        lda     *$dssts
        ora     #10
        sta     *$dssts         ;histo/cursor 'on'

1$:     jsr     setmod          ;go set up bit-planes
        andcc   #257            ;enable interrupts
        jsr     x$hsto          ;these may take awhile
        jsr     x$curs
        jsr     y$curs
        orcc    #120            ;disable interrupts
        jsr     rstmod          ;go reset bit-planes
        rts

2$:     lda     *$dssts
        bita    #10             ;just turned off ?
        beq     3$              ;no - skip
        lda     #2              ;start sequence
        bra     4$
3$:     lda     *dstate         ;get termination state
        bne     4$
        rts                     ;termination complete

4$:     deca
        sta     *dstate         ;and save
        bgt     9$

        ; restore graphics controller mode

        lda     *$blank
        anda    #~16
        sta     *$blank         ;blank histo / cursor planes
        sta     bp$dsp

        lda     *$dsclr+1
        ora     #17
        sta     *$dsclr+1       ;all planes
        lda     *$dsclr+2
        anda    #~17
        sta     *$dsclr+2
        lda     *$dsclr
        bita    #1              ;clearing now ?
        beq     5$              ;no - skip
        ora     #17             ;yes
        bra     6$
5$:     anda    #360            ;no
6$:     sta     $dsclr
        sta     bp$clr

        lda     *$blank+1
        anda    #~17
        sta     *$blank+1
        lda     *$blank+2
        ora     #17
        sta     *$blank+2
        lda     *$blank
        bita    #1              ;blanked now ?
        beq     7$              ;yes - skip
        ora     #17             ;no
        bra     8$
7$:     anda    #360            ;yes
8$:     sta     *$blank
        sta     bp$dsp

        lda     #17             ;write to all planes
        sta     *$dswrt
        sta     bp$wrt
        rts

        ; clear bit planes before termination

9$:     lda     *$blank
        anda    #~16
        sta     *$blank         ;blank planes
        sta     bp$dsp
```

```
        lda     *$blank+1
        anda    #~16
        sta     *$blank+1
        lda     *$blank+2
        anda    #~16
        sta     *$blank+2
        lda     *$dsclr
        ora     #16
        sta     *$dsclr         ;clear planes 1, 2 & 3
        sta     bp$clr
        lda     *$dsclr+1
        ora     #16
        sta     *$dsclr+1
        lda     *$dsclr+2
        ora     #16
        sta     *$dsclr+2

        lda     *$dssts
        anda    #~14
        sta     *$dssts         ;display off
        jsr     rstmap          ;reset color mapping
        rts

        .page
        .sbttl  display system bit plane setup

setmod: lda     #16             ;data for planes 1,2 & 3
        sta     bp$dat

        lda     *dstate         ;check display state
        deca
        bge     1$
        lda     #2              ;sequence is     2   1   0   ...
1$:     sta     *dstate         ;display plane  3/2 2/1 1/3  ...
                                ;clearing plane  1   3   2   ...
                                ;writing plane   2   1   3   ...
        deca
        bge     2$

        ; state 0

        lda     *$blank
        anda    #361
        ora     #2
        sta     *$blank         ;display plane 1
        sta     bp$dsp
        lda     *$dsclr
        anda    #361
        ora     #4
        sta     *$dsclr         ;while clearing plane 2
        sta     bp$clr
        lda     *$dsclr+1
        anda    #361
        ora     #4
        sta     *$dsclr+1
        lda     *$dsclr+2
        anda    #361
        ora     #4
        sta     *$dsclr+2
        lda     #10             ;and writing to plane 3
        sta     bp$wrt
        lda     *$blank
        anda    #361
        ora     #12
        sta     *$blank         ;and display plane 3
        sta     bp$dsp
        lda     *$blank+1
        anda    #361
        ora     #12
        sta     *$blank+1
        lda     *$blank+2
        anda    #361
        ora     #12
        sta     *$blank+2
        rts

2$:     deca
        bge     3$
```

```
        ; state 1

        lda     *$blank
        anda    #361
        ora     #4
        sta     *$blank         ;display plane 2
        sta     bp$dsp
        lda     *$dsclr
        anda    #361
        ora     #10
        sta     *$dsclr         ;while clearing plane 3
        sta     bp$clr
        lda     *$dsclr+1
        anda    #361
        ora     #10
        sta     *$dsclr+1
        lda     *$dsclr+2
        anda    #361
        ora     #10
        sta     *$dsclr+2
        lda     #2              ;and writing to plane 1
        sta     bp$wrt
        lda     *$blank
        anda    #361
        ora     #6
        sta     *$blank         ;and display plane 1
        sta     bp$dsp
        lda     *$blank+1
        anda    #361
        ora     #6
        sta     *$blank+1
        lda     *$blank+2
        anda    #361
        ora     #6
        sta     *$blank+2
        rts

        ; state 2

3$:     lda     *$blank
        anda    #361
        ora     #10
        sta     *$blank         ;display plane 3
        sta     bp$dsp
        lda     *$dsclr
        anda    #361
        ora     #2
        sta     *$dsclr         ;while clearing plane 1
        sta     bp$clr
        lda     *$dsclr+1
        anda    #361
        ora     #2
        sta     *$dsclr+1
        lda     *$dsclr+2
        anda    #361
        ora     #2
        sta     *$dsclr+2
        lda     #4              ;and writing to plane 2
        sta     bp$wrt
        lda     *$blank
        anda    #361
        ora     #14
        sta     *$blank         ;and display plane 2
        sta     bp$dsp
        lda     *$blank+1
        anda    #361
        ora     #14
        sta     *$blank+1
        lda     *$blank+2
        anda    #361
        ora     #14
        sta     *$blank+2
        rts

        .page
        .sbttl  display system bit plane reset
```

```
rstmod: lda     #1              ;only plane 0 to write to
        sta     *$dswrt
        sta     bp$wrt
        lda     *$dsdat         ;old data
        sta     bp$dat
        ldx     *crxpos         ;restore vidx
        stx     vidx
        ldx     *crypos         ;restore vidy
        stx     vidy
        rts

        .page
        .sbttl  draw x cursors

x$curs: ldd     xc$sts          ;any cursors to draw ?
        beq     3$              ;no - exit
        ldy     #xcursr         ;address of x-cursor blocks
        ldu     #vidx           ;drawing in x direction
        lda     #0d16           ;16. cursors
        sta     *hscntr         ;save count
        ldd     xc$sts          ;get cursor status
1$:     lsra                    ;check each cursor
        rorb
        bcc     2$              ;not this one
        pshs    a,b             ;push xc$sts
        ldd     2,y             ;y-position
        coma
        comb
        std     vidy
        ldd     ,y              ;get final value
        addd    4,y
        anda    #17
        std     *dstmp1
        ldd     ,y              ;get initial value
        anda    #17
        tfr     d,x
        jsr     curplt          ;plot cursor
        puls    a,b             ;pull xc$sts
2$:     leay    0d8,y           ;next cursor block
        dec     *hscntr         ;more to check ?
        bgt     1$              ;yes - loop
3$:     rts                     ;else - finished

        .page
        .sbttl  draw y cursors

y$curs: ldd     yc$sts          ;any cursors to draw ?
        beq     3$              ;no - exit
        ldy     #ycursr         ;address of y-cursor blocks
        ldu     #vidy           ;drawing in y direction
        lda     #0d16           ;16. cursors
        sta     *hscntr         ;save count
        ldd     yc$sts          ;get cursor status
1$:     lsra                    ;check each cursor
        rorb
        bcc     2$              ;not this one
        pshs    a,b             ;push yc$sts
        ldd     ,y              ;x-position
        std     vidx
        ldd     2,y             ;build final value
        coma
        comb
        anda    #17
        std     *dstmp1
        ldd     2,y             ;build initial value
        addd    4,y
        coma
        comb
        anda    #17
        tfr     d,x
        jsr     curplt          ;plot cursor
        puls    a,b             ;pull xc$sts
2$:     leay    0d8,y           ;next cursor block
        dec     *hscntr         ;more to check ?
        bgt     1$              ;yes - loop
3$:     rts                     ;else - finished

        .page
```

```
        .sbttl  cursor drawing routine

        ;  enter with:
        ;       y =     address of 8-byte cursor definition block
        ;       x =     starting cursor position
        ;       u =     vidx or vidy address
        ;       dstmp1= ending cursor position
        ;       d =     (will be loaded with pattern)

curplt: ldd     6,y             ;get bit pattern
        beq     10$             ;if none - exit

1$:     exg     a,b             ;swap pattern bytes
        bita    #1              ;write this bit ?
        beq     2$              ;no - skip to next
        stx     ,u              ;load scanning register
        sta     vidplt          ;write data
2$:     leax    0d8,x           ;update position
        cmpx    *dstmp1         ;end of cursor ?
        bgt     10$             ;yes - exit

        bita    #2              ;check remaining bits
        beq     3$
        stx     ,u
        sta     vidplt
3$:     leax    0d8,x
        cmpx    *dstmp1
        bgt     10$

        bita    #4
        beq     4$
        stx     ,u
        sta     vidplt
4$:     leax    0d8,x
        cmpx    *dstmp1
        bgt     10$

        bita    #10
        beq     5$
        stx     ,u
        sta     vidplt
5$:     leax    0d8,x
        cmpx    *dstmp1
        bgt     10$

        bita    #20
        beq     6$
        stx     ,u
        sta     vidplt
6$:     leax    0d8,x
        cmpx    *dstmp1
        bgt     10$

        bita    #40
        beq     7$
        stx     ,u
        sta     vidplt
7$:     leax    0d8,x
        cmpx    *dstmp1
        bgt     10$

        bita    #100
        beq     8$
        stx     ,u
        sta     vidplt
8$:     leax    0d8,x
        cmpx    *dstmp1
        bgt     10$

        bita    #200
        beq     9$
        stx     ,u
        sta     vidplt
9$:     leax    0d8,x
        cmpx    *dstmp1
        ble     1$
10$:    rts                     ;exit
```

```
        .page
        .sbttl  process histo drawing

x$hsto: ldd     hs$sts          ;any to draw ?
        beq     3$              ;no - exit

        ldy     #dsphst         ;address of histo blocks
        lda     #0d16           ;16. histos
        sta     *hscntr
        ldd     hs$sts          ;get enable bits
1$:     lsra                    ;histo enabled ?
        rorb
        bcc     2$              ;no - skip
        ldx     6,y             ;get count
        beq     2$              ;'0' - skip
        pshs    a,b,y           ;save these
        ldu     4,y             ;get x-step size
        stu     x$stps
        ldu     2,y             ;get initial x-position
        ldd     ,y              ;data pointer
        lslb
        rola
        tfr     d,y             ;internal data address
        jsr     drwhst          ;go draw histo
        puls    a,b,y           ;retrieve these
2$:     leay    0d8,y           ;next histo block
        dec     *hscntr         ;any more ?
        bgt     1$              ;loop for all histos
3$:     rts

        .page
        .sbttl  copy histo draw routine into bufsav

cpydrw: ldx     #c.pgm          ;source
        ldu     #drwhst         ;destination
        lda     #c.pgme-c.pgm   ;length
1$:     ldb     ,x+
        stb     ,u+
        deca
        bgt     1$
        rts

        .sbttl  draw histo routine

        ; x - number of points to draw
        ; y - address of histo buffer
        ; u - initial x-position

c.pgm   =.
        stu     vidx            ;load x-position
c.pgms  =.+2
        leau    400,u           ;'400' will be replaced with step size
        ldd     ,y++            ;load y-position
        coma
        comb
        std     vidy
        sta     vidplt          ;draw point
        leax    -1,x
        bne     c.pgm           ;loop for all points
        rts
c.pgme  =.

        .area   BUFSAV
                                ;allocate space for routine
drwhst: .blkb   c.pgms-c.pgm    ;address of drwhst routine
x$stps: .blkb   c.pgme-c.pgms
```

```
        .page
        .sbttl  mc6845 crt controller (crtc)

        .module mc6845
        .area   MC6845

;       port handler naming convention
;
;       _       is device unit letter
;       nnnn    is device code
;       ****    is any alpha/numeric sequence
;
;
;       _$nnnn  - unit defined
;       _.nnnn  - device address
;
;
;       $_nnnn  - unit definition macro's or
;                 non-inline code
;
;       _nnnn$  - common subroutine entry points requiring
;                 x,y,u, or d to be specified
;                 before entry
;
;       _nnnni  - common interrupt entry points requiring
;                 x,y,u, or d to be specified
;                 before entry
;
;       .innnn  - in-line initialization macro calls requiring
;                 x,y,u, or d to be specified
;                 before entry
;
;       _.****  - older handlers used this format for internal
;                 labels and entry points. this form should
;                 not be used for new handlers.
;



        ;
        ;  the mc6845 crtc is programmed through 2 register addresses
        ;
        ;       address a.6845+0 = register select register
        ;       address a.6845+1 = specified control register
        ;
        ;       r0  -   horizontal total register
        ;               <7:0> - horizontal sync (hs) frequency is
        ;               determined by the character times per line
        ;               programmed as displayed + non-displayed - 1
        ;       r1  -   horizontal displayed register
        ;               <7:0> - # of displayed characters per line
        ;       r2  -   horizontal sync position
        ;               <7:0> - position of hs pulse relative
        ;               to the first displayed character
        ;       r3  -   sync width register
        ;               <3:0> - hs pulse width in character times
        ;               vertical sync is 16 scan-line times
        ;
        ;       note that r1+r2+r3 < r0
        ;
        ;
        ;       r4  -   vertical total register
        ;       r5  -   vertical total adjust register
        ;               r4<6:0> - integer number of character
        ;               line times - 1
        ;               r5<4:0> - fraction character line times
        ;               vertical sync period is r5+(r4+1)*(r9+1) lines
        ;
        ;       r6  -   vertical display register
        ;               <6:0> - # of displayed character rows
        ;       r7  -   vertical sync (vs) position
        ;               <6:0> - character line times - 1
        ;
        ;       r8  -   interlace mode register
        ;               <1:0> - interlace mode
        ;                0 0  - normal sync mode (non-interlac)
        ;                1 0  -   "      "    "         "
        ;                0 1  - interlace sync mode
        ;                1 1  - interlace sync and video mode
```

```
;               restrictions for interlace operation
;                       r0 must be odd
;               additional restrictions for sync and video interlace
;                       r6 must be even
;                               and must be 1/2 the number required
;                       r7 must be 1/2 normal value
;                       r9 must be odd
;                       r10 and r11 must both be even/odd
;
.page
;       r9  -   maximum scan line address register
;               <4:0> - # of scan lines per character -1
;
;       r10 -   cursor start address
;       r11 -   cursor end address
;               r10<4:0> - first cursor row displayed
;               r11<4:0> - last cursor row displayed
;               r10<6:5> - cursor blink control
;                   0 0  - non blink
;                   0 1  - cursor non-display
;                   1 0  - blink, 1/16 field rate
;                   1 1  - blink, 1/32 field rate
;
;       r12 -   start address register
;       r13 -   start address register
;               <r12:r13>/<5:0>:<7:0> - 14 bit address register
;               for the beginning of the character data
;
;       r14 -   cursor register
;       r15 -   corsor register
;               <r14:r15>/<5:0>:<7:0> - 14 bit address register
;               for the cursor position
;
;       r16 -   light pen register
;       r17 -   light pen register
;               <r16:r17>/<5:0>:<7:0> - 14 bit light pen
;               hit register
;
;
;       mc6845 routines are entered via a jsr xxx
;
;               with x = port address
;               and  y = init table address
;       jsr     i6845$  ;initialize crtc paramters
;
;               with x = port address
;               and  d = read/write data
;       jsr     s6845$  ;set beginning of display
;       jsr     r6845$  ;read cursor position
;       jsr     w6845$  ;write new cursor position
;       jsr     l6845$  ;read light pen register
;

.page
.sbttl  mc6845 setup routine
;
;         the mc6845 crt controller settup is specified by an
;       external data table configured as follows:
;
;       byte 0  -       horizontal total
;       byte 1  -       horizontal displayed
;       byte 2  -       horizontal sync position
;       byte 3  -       horizontal sync width
;
;       byte 4  -       vertical total
;       byte 5  -       vertical total adjust
;       byte 6  -       vertical displayed
;       byte 7  -       vertical sync position
;
;       byte 8  -       interlace mode
;       byte 9  -       maximum scan line
;       byte 10 -       cursor start
;       byte 11 -       cursor end
;
;       bytes 12 & 13 - start address
;
;       bytes 14 & 15 - cursor address
;
```

```
;       enter setup routine with
;               x =     port address
;               y =     data table address

i6845$::
        clra                    ;set counter
1$:     ldb     ,y+             ;get data
        sta     ,x              ;set access register
        stb     1,x             ;load setup data
        inca                    ;update register #
        cmpa    #0d16           ;finished ?
        bne     1$              ;loop until all loaded
        rts                     ;finished


        .page
        .sbttl  special crtc register access

;               x =     port address
;               d =     data or result
;

;       write crtc start address
s6845$::
        pshs    b               ;save b
        ldb     #0d12           ;start address
        stb     ,x              ;set access
        sta     1,x             ;store start address
        incb
        stb     ,x              ;set access
        puls    b               ;get data
        stb     1,x             ;store start address
        rts

;       read cursor value
r6845$::
        ldb     #0d14           ;cursor
        stb     ,x              ;set access
        lda     1,x             ;get cursor
        incb
        stb     ,x              ;set access
        ldb     1,x             ;get cursor
        rts

;       write cursor value
w6845$::
        pshs    b               ;save b
        ldb     #0d14           ;cursor
        stb     ,x              ;set access
        sta     1,x             ;store cursor
        incb
        stb     ,x              ;set access
        puls    b               ;get data
        stb     1,x             ;store cursor
        rts

;       read light pen register
l6845$::
        ldb     #0d16           ;light pen
        stb     ,x              ;set access
        lda     1,x             ;get light pen
        incb
        stb     ,x              ;set access
        ldb     1,x             ;get light pen
        rts
```

```
        .sbttl  vector character table

        .module vector
        .area   VECTOR

        ;vector character rom
        ;of r6571a point character rom

table:: .word   ch0,ch1,ch2,ch3,ch4,ch5,ch6,ch7,ch8,ch9
        .word   ch10,ch11,ch12,ch13,ch14,ch15,ch16,ch17,ch18,ch19
        .word   ch20,ch21,ch22,ch23,ch24,ch25,ch26,ch27,ch28,ch29
        .word   ch30,ch31,ch32,ch33,ch34,ch35,ch36,ch37,ch38,ch39
        .word   ch40,ch41,ch42,ch43,ch44,ch45,ch46,ch47,ch48,ch49
        .word   ch50,ch51,ch52,ch53,ch54,ch55,ch56,ch57,ch58,ch59
        .word   ch60,ch61,ch62,ch63,ch64,ch65,ch66,ch67,ch68,ch69
        .word   ch70,ch71,ch72,ch73,ch74,ch75,ch76,ch77,ch78,ch79
        .word   ch80,ch81,ch82,ch83,ch84,ch85,ch86,ch87,ch88,ch89
        .word   ch90,ch91,ch92,ch93,ch94,ch95,ch96,ch97,ch98,ch99
        .word   ch100,ch101,ch102,ch103,ch104,ch105,ch106,ch107,ch108,ch109
        .word   ch110,ch111,ch112,ch113,ch114,ch115,ch116,ch117,ch118,ch119
        .word   ch120,ch121,ch122,ch123,ch124,ch125,ch126,ch127
        .word   gch0,gch1,gch2,gch3,gch4,gch5,gch6,gch7,gch8,gch9
        .word   nsv0            ;special non-slashed vector '0'
        .word   nsp0            ;special non-slashed point '0'

        .page

        .radix  o               ;octal data table

        dwn     ==      10
        up      ==      200
        end     ==      210

gch0::  .byte   4,dwn,254,54,44,244,up,14,end
gch1:   .byte   dwn,3,16,3,260,140,260,up,end
gch2:   .byte   273,dwn,140,6,340,16,up,63,end
gch3:   .byte   3,dwn,273,73,63,263,up,13,end
gch4:   .byte   261,dwn,42,40,52,12,252,240,242,2,up,71,end
gch5:   .byte   dwn,3,220,40,220,16,220,40,220,3,up,end
gch6:   .byte   60,dwn,343,16,143,up,260,end
gch7:   .byte   260,dwn,153,6,353,up,60,end
gch8:   .byte   13,dwn,66,340,76,up,3,end
gch9:   .byte   3,dwn,76,340,66,up,13,end
ch0::   .byte   160,dwn,304,220,231,12,31,20,104,up,34,end
ch1:    .byte   33,dwn,21,7,60,31,231,260,60,31,11,231,260,up,140,end
ch2:    .byte   24,dwn,20,31,16,220,2,125,up,34,end
ch3:    .byte   107,dwn,221,220,231,12,73,11,231,220,221,1,21,20,up,133,end
ch4:    .byte   100,dwn,220,242,1,60,260,1,42,20,up,116,end
ch5:    .byte   60,dwn,20,21,221,240,221,1,104,20,11,240,221,up,137,11,end
ch6:    .byte   23,dwn,21,31,13,3,21,20,31,16,up,43,end
ch7:    .byte   22,dwn,4,42,20,52,14,252,220,242,2,120,up,54,end
ch8:    .byte   25,dwn,14,31,20,21,up,111,end
ch9:    .byte   20,dwn,6,13,20,42,252,73,21,up,51,end
ch10:   .byte   27,1,dwn,52,14,252,42,20,52,up,40,end
ch11:   .byte   33,dwn,7,14,40,21,3,13,31,up,60,end
ch12:   .byte   25,dwn,20,15,104,1,up,55,end
ch13:   .byte   127,dwn,220,1,11,220,231,31,20,220,252
        .byte   31,60,31,231,220,up,100,end
ch14:   .byte   22,dwn,1,42,20,52,11,252,220,242,up,172,end
ch15:   .byte   24,dwn,21,20,15,5,40,15,5,40,up,35,end
ch16:   .byte   33,dwn,6,42,20,52,11,252,220,242,up,172,end
ch17:   .byte   105,dwn,52,11,252,220,242,1,42,100,up,35,end
ch18:   .byte   24,dwn,21,40,15,5,60,up,35,end
ch19:   .byte   25,dwn,20,14,31,20,21,3,21,up,55,end
ch20:   .byte   60,dwn,6,20,31,12,231,240,221,2,21,20,2,up,137,11,end
ch21:   .byte   24,dwn,20,114,20,up,320,dwn,104,up,54,end
ch22:   .byte   24,dwn,31,12,31,40,21,2,21,up
        .byte   261,dwn,17,11,up,103,end
ch23:   .byte   45,dwn,231,13,31,20,21,2,12,31,20,21,3,221,up,55,end
ch24:   .byte   20,dwn,20,2,221,2,42,40,52,12,231,12,20,up,20,end
ch25:   .byte   22,dwn,52,7,1,100,up,37,11,end
ch26:   .byte   24,dwn,140,242,52,252,up,72,end
ch27:   .byte   66,dwn,252,52,242,140,up,34,end
ch28:   .byte   46,dwn,42,52,242,17,11,up,100,end
ch29:   .byte   106,dwn,0,up,272,dwn,140,up,272,dwn,0,up,112,end
ch30:   .byte   160,dwn,340,104,20,220,304,140,up,37,11,end
ch31:   .byte   26,dwn,21,20,52,20,21,up,353,dwn,21,20,52,20,21,up,33,end
ch32:   .byte   160,20,end
```

```
ch33:   .byte   100,dwn,0,up,3,dwn,5,up,117,11,end
ch34:   .byte   66,dwn,2,up,60,dwn,12,up,56,end
ch35:   .byte   60,dwn,7,1,up,40,dwn,17,11,up,305
        .byte   dwn,140,up,352,dwn,140,up,33,end
ch36:   .byte   21,dwn,120,21,1,221,300,221,1,21,120
        .byte   260,1,17,11,up,100,end
ch37:   .byte   27,dwn,21,31,231,221,up,16,dwn,146,up
        .byte   16,dwn,231,221,21,31,up,31,end
ch38:   .byte   163,dwn,273,240,221,1,104,1,221,240,231,11,156,up,20,end
ch39:   .byte   26,dwn,21,1,up,157,11,end
ch40:   .byte   100,dwn,242,4,42,up,117,11,end
ch41:   .byte   100,dwn,42,4,242,up,117,11,end
ch42:   .byte   104,dwn,60,260,63,273,4,14,263,73
        .byte   260,60,273,63,14,4,73,up,31,end
ch43:   .byte   24,dwn,140,up,263,dwn,16,up,111,end
ch44:   .byte   40,dwn,20,12,231,up,143,end
ch45:   .byte   24,dwn,140,up,34,end
ch46:   .byte   60,dwn,0,up,120,end
ch47:   .byte   21,dwn,146,up,37,end
ch48:   .byte   21,dwn,6,21,100,31,356,31,100,21,6,up,37,end
ch49:   .byte   67,dwn,21,17,11,240,100,up,40,end
ch50:   .byte   27,dwn,21,100,31,11,252,220,273,11,140,up,20,end
ch51:   .byte   27,dwn,21,100,31,12,231,260,60,31,12,231,300,221,up,171,end
ch52:   .byte   140,dwn,7,1,335,140,up,33,end
ch53:   .byte   21,dwn,31,100,21,2,221,320,4,140,up,37,11,end
ch54:   .byte   167,dwn,221,300,231,16,31,100,21,2,221,320,up,174,end
ch55:   .byte   27,dwn,1,140,11,314,13,up,120,end
ch56:   .byte   44,dwn,231,12,31,100,21,2,221,300,221
        .byte   2,21,100,31,12,231,up,54,end
ch57:   .byte   21,dwn,31,100,21,7,320,231,12,31,120,up,34,end
ch58:   .byte   46,dwn,0,up,14,dwn,0,up,152,end
ch59:   .byte   45,dwn,0,up,15,dwn,20,12,231,up,143,end
ch60:   .byte   120,dwn,304,104,up,77,11,end
ch61:   .byte   25,dwn,140,up,352,dwn,140,up,33,end
ch62:   .byte   60,dwn,104,304,up,137,11,end
ch63:   .byte   26,dwn,1,21,100,31,11,273,11,up,12,dwn,0,up,100,end
ch64:   .byte   161,dwn,231,300,221,6,21,100,31,13,231,260
        .byte   1,21,20,12,up,73,end
ch65:   .byte   20,dwn,6,42,40,52,12,340,140,14,up,20,end
ch66:   .byte   20,dwn,120,21,2,221,300,100,21,2,221
        .byte   320,20,17,11,up,140,end
ch67:   .byte   167,dwn,221,260,252,14,52,60,21,up,31,end
ch68:   .byte   20,dwn,120,21,6,221,320,20,17,11,up,140,end
ch69:   .byte   104,64,dwn,340,14,60,260,14,140,up,20,end
ch70:   .byte   104,64,dwn,340,14,60,260,14,up,160,end
ch71:   .byte   167,dwn,221,260,252,14,52,60,21,2,240,up,73,end
ch72:   .byte   20,dwn,4,4,14,140,4,14,14,up,20,end
ch73:   .byte   107,dwn,1,240,100,240,17,11,240,100,up,40,end
ch74:   .byte   21,dwn,31,40,21,7,240,100,up,37,11,end
ch75:   .byte   20,dwn,4,40,240,4,up,140,dwn,314,114,up,20,end
ch76:   .byte   24,4,dwn,14,14,140,up,20,end
ch77:   .byte   20,dwn,4,4,73,11,1,63,14,14,up,20,end
ch78:   .byte   20,dwn,4,4,156,6,14,14,up,20,end
ch79:   .byte   22,dwn,4,42,40,52,14,252,240,242,up,172,end
ch80:   .byte   20,dwn,4,4,120,31,12,231,320,up,174,end
ch81:   .byte   141,dwn,21,4,242,240,252,14,52,40,21,221,52,up,20,end
ch82:   .byte   20,dwn,4,120,21,2,221,320,14,40,114,up,20,end
ch83:   .byte   21,dwn,31,100,21,2,221,300,221,2,21,100,31,up,37,end
ch84:   .byte   100,dwn,4,4,260,140,up,34,14,end
ch85:   .byte   24,4,dwn,17,31,100,21,7,up,34,14,end
ch86:   .byte   24,4,dwn,15,73,63,5,up,34,14,end
ch87:   .byte   24,4,dwn,14,14,63,2,12,73,4,4,up,34,14,end
ch88:   .byte   20,dwn,1,146,1,up,340,dwn,11,156,11,up,20,end
ch89:   .byte   100,dwn,4,263,1,up,140,dwn,11,273,up,114,end
ch90:   .byte   24,4,dwn,140,11,356,11,140,up,20,end
ch91:   .byte   144,4,dwn,260,14,14,60,up,40,end
ch92:   .byte   27,dwn,156,up,31,end
ch93:   .byte   44,4,dwn,60,17,11,260,up,140,end
ch94:   .byte   27,dwn,21,100,31,up,37,end
ch95:   .byte   20,dwn,140,up,20,end
ch96:   .byte   144,4,dwn,11,31,up,36,end
ch97:   .byte   142,dwn,252,240,221,2,42,60,15,up,40,end
ch98:   .byte   24,4,dwn,16,52,40,21,3,221,240,252,13,up,160,end
ch99:   .byte   144,dwn,221,260,231,13,31,60,21,up,51,end
ch100:  .byte   144,4,dwn,16,252,240,221,3,21,40,52,13,up,40,end
ch101:  .byte   22,dwn,120,2,221,260,231,13,31,100,up,40,end
ch102:  .byte   60,dwn,4,240,100,240,3,21,20,31,up,57,end
ch103:  .byte   32,dwn,31,60,21,5,242,240,231,13,31,40,42,3,up,55,end
```

```
ch104:  .byte   24,4,dwn,15,13,3,42,20,52,13,up,40,end
ch105:  .byte   107,dwn,0,up,232,dwn,20,15,up,100,end
ch106:  .byte   32,dwn,31,60,21,7,up,55,end
ch107:  .byte   20,dwn,3,40,240,5,up,113,dwn,252,73,up,40,end
ch108:  .byte   64,4,dwn,14,14,up,120,end
ch109:  .byte   20,dwn,5,40,31,14,4,21,20,31,14,up,20,end
ch110:  .byte   25,dwn,15,3,42,40,31,14,up,40,end
ch111:  .byte   21,dwn,3,21,60,31,13,231,260,221,up,171,end
ch112:  .byte   25,dwn,13,52,40,21,3,221,240,252,16,up,163,end
ch113:  .byte   145,dwn,13,252,240,221,3,21,40,52,16,up,43,end
ch114:  .byte   25,dwn,15,3,42,40,31,up,54,end
ch115:  .byte   21,dwn,31,60,21,221,220,221,220,221,21,60,31,up,54,end
ch116:  .byte   67,dwn,12,240,100,240,14,31,20,21,up,51,end
ch117:  .byte   25,dwn,14,31,60,21,4,up,55,end
ch118:  .byte   25,dwn,13,52,42,3,up,75,end
ch119:  .byte   25,dwn,14,31,20,21,3,13,31,20,21,4,up,35,end
ch120:  .byte   20,dwn,125,up,320,dwn,135,up,40,end
ch121:  .byte   25,dwn,14,31,40,42,3,17,231,260,221,up,162,end
ch122:  .byte   25,dwn,120,335,120,up,40,end
ch123:  .byte   144,4,dwn,240,231,12,231,31,12,31,40,up,40,end
ch124:  .byte   103,dwn,12,up,7,dwn,12,up,116,end
ch125:  .byte   44,4,dwn,40,31,12,31,231,12,231,240,up,140,end
ch126:  .byte   27,dwn,21,20,52,20,21,up,37,end
ch127:  .byte   24,4,dwn,140,up,351,dwn,140,up,351,dwn,140,up
        .byte   351,dwn,140,up,351,dwn,140,up,351,dwn,140,up
        .byte   351,dwn,140,up,351,dwn,140,up,351,dwn,140,up,20,end
nsv0::  .byte   21,dwn,6,21,100,31,16,231,300,221,up,171,end
nsp0::  .byte   076,101,101,101,101,101,101,101
        .byte   076,000,000,000,000,000,000,000
```

```
        .sbttl  6571a rom image

        .module r6571a
        .area   R6571A

        .radix  o

r6571a::
        .byte   000, 000, 000, 000, 061, 112, 104, 112
        .byte   061, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 074, 042, 074, 042, 042
        .byte   074, 040, 040, 100, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 141, 022, 024, 030
        .byte   020, 060, 060, 060, 000, 000, 000, 000
        .byte   060, 110, 100, 100, 040, 060, 110, 110
        .byte   060, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 030, 040, 100, 170, 100, 040
        .byte   030, 000, 000, 000, 000, 000, 000, 000
        .byte   026, 016, 020, 040, 100, 100, 070, 004
        .byte   030, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 054, 122, 022, 022
        .byte   022, 002, 002, 002, 000, 000, 000, 000
        .byte   030, 044, 102, 102, 176, 102, 102, 044
        .byte   030, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 100, 100, 100, 100, 110
        .byte   060, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 100, 110, 120, 140, 120, 112
        .byte   104, 000, 000, 000, 000, 000, 000, 000
        .byte   100, 040, 020, 020, 020, 020, 030, 044
        .byte   102, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 110, 110, 110, 110
        .byte   164, 100, 100, 100, 000, 000, 000, 000
        .byte   000, 000, 000, 142, 042, 044, 050, 060
        .byte   040, 000, 000, 000, 000, 000, 000, 000
        .byte   010, 034, 040, 030, 040, 100, 074, 002
        .byte   014, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 030, 044, 102, 102, 044
        .byte   030, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 077, 124, 024, 024, 024
        .byte   024, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 030, 044, 102, 102, 144
        .byte   130, 100, 100, 100, 000, 000, 000, 000
        .byte   000, 000, 000, 037, 044, 102, 102, 044
        .byte   030, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 077, 110, 010, 010, 010
        .byte   010, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 142, 044, 044, 044, 044
        .byte   030, 000, 000, 000, 000, 000, 000, 000
        .byte   020, 020, 070, 124, 124, 124, 070, 020
        .byte   020, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 142, 024, 010, 024
        .byte   043, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 010, 111, 052, 052, 052
        .byte   034, 010, 010, 010, 000, 000, 000, 000
        .byte   000, 000, 000, 042, 101, 111, 111, 111
        .byte   066, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 034, 042, 101, 101, 101, 042, 042
        .byte   143, 000, 000, 000, 000, 000, 000, 000
        .byte   037, 020, 020, 020, 020, 020, 120, 060
        .byte   020, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 004, 002, 177, 002, 004, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 020, 040, 177, 040, 020, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   010, 034, 052, 010, 010, 010, 010, 010
        .byte   010, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 010, 000, 177, 000, 010, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   177, 040, 020, 010, 006, 010, 020, 040
        .byte   177, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 060, 111, 006, 060, 111, 006, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   010, 010, 010, 010, 010, 010, 000, 000
        .byte   010, 000, 000, 000, 000, 000, 000, 000
        .byte   022, 022, 022, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
```

```
.byte   024, 024, 024, 177, 024, 177, 024, 024
.byte   024, 000, 000, 000, 000, 000, 000, 000
.byte   010, 077, 110, 110, 076, 011, 011, 176
.byte   010, 000, 000, 000, 000, 000, 000, 000
.byte   040, 121, 042, 004, 010, 020, 042, 105
.byte   002, 000, 000, 000, 000, 000, 000, 000
.byte   070, 104, 104, 050, 020, 051, 106, 106
.byte   071, 000, 000, 000, 000, 000, 000, 000
.byte   040, 040, 100, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   010, 020, 040, 040, 040, 040, 040, 020
.byte   010, 000, 000, 000, 000, 000, 000, 000
.byte   010, 004, 002, 002, 002, 002, 002, 004
.byte   010, 000, 000, 000, 000, 000, 000, 000
.byte   010, 111, 052, 034, 177, 034, 052, 111
.byte   010, 000, 000, 000, 000, 000, 000, 000
.byte   000, 010, 010, 010, 177, 010, 010, 010
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   060, 020, 020, 040, 000, 000, 000, 000
.byte   000, 000, 000, 000, 177, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   020, 000, 000, 000, 000, 000, 000, 000
.byte   000, 001, 002, 004, 010, 020, 040, 100
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   076, 101, 103, 105, 111, 121, 141, 101
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   010, 030, 010, 010, 010, 010, 010, 010
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   076, 101, 001, 002, 014, 020, 040, 100
.byte   177, 000, 000, 000, 000, 000, 000, 000
.byte   076, 101, 001, 001, 036, 001, 001, 101
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   002, 006, 012, 022, 042, 177, 002, 002
.byte   002, 000, 000, 000, 000, 000, 000, 000
.byte   177, 100, 100, 100, 176, 001, 001, 101
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   076, 101, 100, 100, 176, 101, 101, 101
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   177, 101, 002, 004, 010, 020, 020, 020
.byte   020, 000, 000, 000, 000, 000, 000, 000
.byte   076, 101, 101, 101, 076, 101, 101, 101
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   076, 101, 101, 101, 077, 001, 001, 101
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 040, 000, 000, 000, 040, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 040, 000, 000, 000, 000
.byte   060, 020, 020, 040, 000, 000, 000, 000
.byte   004, 010, 020, 040, 100, 040, 020, 010
.byte   004, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 177, 000, 177, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   020, 010, 004, 002, 001, 002, 004, 010
.byte   020, 000, 000, 000, 000, 000, 000, 000
.byte   076, 101, 101, 002, 004, 010, 010, 000
.byte   010, 000, 000, 000, 000, 000, 000, 000
.byte   076, 101, 101, 115, 125, 136, 100, 101
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   034, 042, 101, 101, 177, 101, 101, 101
.byte   101, 000, 000, 000, 000, 000, 000, 000
.byte   176, 041, 041, 041, 076, 041, 041, 041
.byte   176, 000, 000, 000, 000, 000, 000, 000
.byte   036, 041, 100, 100, 100, 100, 100, 041
.byte   036, 000, 000, 000, 000, 000, 000, 000
.byte   176, 041, 041, 041, 041, 041, 041, 041
.byte   176, 000, 000, 000, 000, 000, 000, 000
.byte   177, 100, 100, 100, 170, 100, 100, 100
.byte   177, 000, 000, 000, 000, 000, 000, 000
.byte   177, 100, 100, 100, 170, 100, 100, 100
.byte   100, 000, 000, 000, 000, 000, 000, 000
.byte   036, 041, 100, 100, 100, 107, 101, 041
.byte   036, 000, 000, 000, 000, 000, 000, 000
.byte   101, 101, 101, 101, 177, 101, 101, 101
.byte   101, 000, 000, 000, 000, 000, 000, 000
.byte   076, 010, 010, 010, 010, 010, 010, 010
.byte   076, 000, 000, 000, 000, 000, 000, 000
```

```
.byte   037, 004, 004, 004, 004, 004, 004, 104
.byte   070, 000, 000, 000, 000, 000, 000, 000
.byte   101, 102, 104, 110, 160, 110, 104, 102
.byte   101, 000, 000, 000, 000, 000, 000, 000
.byte   100, 100, 100, 100, 100, 100, 100, 100
.byte   177, 000, 000, 000, 000, 000, 000, 000
.byte   101, 143, 125, 111, 111, 101, 101, 101
.byte   101, 000, 000, 000, 000, 000, 000, 000
.byte   101, 141, 121, 111, 105, 103, 101, 101
.byte   101, 000, 000, 000, 000, 000, 000, 000
.byte   034, 042, 101, 101, 101, 101, 101, 042
.byte   034, 000, 000, 000, 000, 000, 000, 000
.byte   176, 101, 101, 101, 176, 100, 100, 100
.byte   100, 000, 000, 000, 000, 000, 000, 000
.byte   034, 042, 101, 101, 101, 101, 105, 042
.byte   035, 000, 000, 000, 000, 000, 000, 000
.byte   176, 101, 101, 101, 176, 110, 104, 102
.byte   101, 000, 000, 000, 000, 000, 000, 000
.byte   076, 101, 100, 100, 076, 001, 001, 101
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   177, 010, 010, 010, 010, 010, 010, 010
.byte   010, 000, 000, 000, 000, 000, 000, 000
.byte   101, 101, 101, 101, 101, 101, 101, 101
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   101, 101, 101, 101, 101, 101, 042, 024
.byte   010, 000, 000, 000, 000, 000, 000, 000
.byte   101, 101, 101, 111, 111, 111, 125, 143
.byte   101, 000, 000, 000, 000, 000, 000, 000
.byte   101, 101, 042, 024, 010, 024, 042, 101
.byte   101, 000, 000, 000, 000, 000, 000, 000
.byte   101, 101, 042, 024, 010, 010, 010, 010
.byte   010, 000, 000, 000, 000, 000, 000, 000
.byte   177, 001, 002, 004, 010, 020, 040, 100
.byte   177, 000, 000, 000, 000, 000, 000, 000
.byte   036, 020, 020, 020, 020, 020, 020, 020
.byte   036, 000, 000, 000, 000, 000, 000, 000
.byte   000, 100, 040, 020, 010, 004, 002, 001
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   074, 004, 004, 004, 004, 004, 004, 004
.byte   074, 000, 000, 000, 000, 000, 000, 000
.byte   076, 101, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   177, 000, 000, 000, 000, 000, 000, 000
.byte   002, 002, 001, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 036, 042, 102, 102, 106
.byte   072, 000, 000, 000, 000, 000, 000, 000
.byte   100, 100, 100, 134, 142, 102, 102, 142
.byte   134, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 074, 102, 100, 100, 102
.byte   074, 000, 000, 000, 000, 000, 000, 000
.byte   002, 002, 002, 072, 106, 102, 102, 106
.byte   072, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 074, 102, 102, 176, 100
.byte   076, 000, 000, 000, 000, 000, 000, 000
.byte   014, 022, 020, 020, 174, 020, 020, 020
.byte   020, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 072, 106, 102, 102, 106
.byte   072, 002, 102, 074, 000, 000, 000, 000
.byte   100, 100, 100, 130, 144, 102, 102, 102
.byte   102, 000, 000, 000, 000, 000, 000, 000
.byte   000, 010, 000, 030, 010, 010, 010, 010
.byte   010, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 002, 002, 002, 002, 002
.byte   002, 002, 102, 074, 000, 000, 000, 000
.byte   100, 100, 100, 104, 110, 160, 110, 104
.byte   102, 000, 000, 000, 000, 000, 000, 000
.byte   020, 020, 020, 020, 020, 020, 020, 020
.byte   020, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 166, 111, 111, 111, 111
.byte   111, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 134, 142, 102, 102, 102
.byte   102, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 074, 102, 102, 102, 102
.byte   074, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 134, 142, 102, 102, 142
.byte   134, 100, 100, 100, 000, 000, 000, 000
```

```
.byte   000, 000, 000, 072, 106, 102, 102, 106
.byte   072, 002, 002, 002, 000, 000, 000, 000
.byte   000, 000, 000, 134, 142, 100, 100, 100
.byte   100, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 074, 102, 060, 014, 102
.byte   074, 000, 000, 000, 000, 000, 000, 000
.byte   000, 020, 020, 174, 020, 020, 020, 022
.byte   014, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 102, 102, 102, 102, 102
.byte   074, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 104, 104, 104, 104, 050
.byte   020, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 101, 111, 111, 111, 111
.byte   066, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 102, 044, 030, 030, 044
.byte   102, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 102, 102, 102, 102, 106
.byte   072, 002, 102, 074, 000, 000, 000, 000
.byte   000, 000, 000, 176, 004, 010, 020, 040
.byte   176, 000, 000, 000, 000, 000, 000, 000
.byte   016, 020, 020, 020, 040, 020, 020, 020
.byte   016, 000, 000, 000, 000, 000, 000, 000
.byte   010, 010, 010, 000, 000, 010, 010, 010
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   070, 004, 004, 004, 002, 004, 004, 004
.byte   070, 000, 000, 000, 000, 000, 000, 000
.byte   060, 111, 006, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   177, 177, 177, 177, 177, 177, 177, 177
.byte   177, 000, 000, 000, 000, 000, 000, 000
```

```
        .sbttl   figrom character rom

        .module figrom
        .area   FIGROM

        .radix  o

figrom::
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
```

```
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
```

```
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
        .byte   377, 377, 377, 377, 377, 377, 377, 377
```

```
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
.byte   377, 377, 377, 377, 377, 377, 377, 377
```

```
        .sbttl   vt1xx character rom

        .module vt1xx
        .area   VT1XX

        .radix  o

vt1xx::
        .byte   000, 010, 034, 034, 076, 076, 034, 034
        .byte   010, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 076, 024, 076, 024, 076, 024, 076
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 050, 050, 070, 050, 050, 034, 010
        .byte   010, 010, 000, 000, 000, 000, 000, 000
        .byte   000, 070, 040, 060, 040, 074, 020, 030
        .byte   020, 020, 000, 000, 000, 000, 000, 000
        .byte   000, 030, 040, 040, 030, 014, 022, 034
        .byte   024, 022, 000, 000, 000, 000, 000, 000
        .byte   000, 040, 040, 040, 070, 034, 020, 030
        .byte   020, 020, 000, 000, 000, 000, 000, 000
        .byte   000, 030, 044, 044, 030, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 010, 010, 076, 010, 010, 000, 076
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 044, 064, 054, 044, 020, 020, 020
        .byte   020, 034, 000, 000, 000, 000, 000, 000
        .byte   000, 044, 044, 030, 030, 034, 010, 010
        .byte   010, 010, 000, 000, 000, 000, 000, 000
        .byte   020, 020, 020, 020, 360, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 360, 020, 020, 020
        .byte   020, 020, 020, 020, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 037, 020, 020, 020
        .byte   020, 020, 020, 020, 000, 000, 000, 000
        .byte   020, 020, 020, 020, 037, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   020, 020, 020, 020, 377, 020, 020, 020
        .byte   020, 020, 020, 020, 000, 000, 000, 000
        .byte   377, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 377, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 377, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 377, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   377, 000, 000, 000, 000, 000, 000, 000
        .byte   020, 020, 020, 020, 037, 020, 020, 020
        .byte   020, 020, 020, 020, 000, 000, 000, 000
        .byte   020, 020, 020, 020, 360, 020, 020, 020
        .byte   020, 020, 020, 020, 000, 000, 000, 000
        .byte   020, 020, 020, 020, 377, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 377, 020, 020, 020
        .byte   020, 020, 020, 020, 000, 000, 000, 000
        .byte   020, 020, 020, 020, 020, 020, 020, 020
        .byte   020, 020, 020, 020, 000, 000, 000, 000
        .byte   004, 010, 020, 040, 020, 010, 004, 074
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   040, 020, 010, 004, 010, 020, 040, 074
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 076, 024, 024, 024, 024, 044
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 002, 004, 076, 010, 076, 020, 040
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 014, 022, 020, 070, 020, 030, 024
        .byte   060, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 030, 030, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 060, 160, 050, 060, 034, 020, 030
        .byte   020, 034, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 010, 010, 010, 010, 010, 000, 010
        .byte   000, 000, 000, 000, 000, 000, 000, 000
        .byte   000, 024, 024, 024, 000, 000, 000, 000
        .byte   000, 000, 000, 000, 000, 000, 000, 000
```

```
.byte   000, 024, 024, 076, 024, 076, 024, 024
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 010, 036, 050, 034, 012, 074, 010
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 060, 062, 004, 010, 020, 046, 006
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 010, 024, 024, 030, 052, 044, 032
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 010, 010, 020, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 004, 010, 020, 020, 020, 010, 004
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 020, 010, 004, 004, 004, 010, 020
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 010, 052, 034, 052, 010, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 010, 010, 076, 010, 010, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 030, 030
.byte   010, 020, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 076, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 030, 030
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 001, 002, 004, 010, 020, 040, 100
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 042, 046, 052, 062, 042, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 010, 030, 010, 010, 010, 010, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 042, 002, 004, 010, 020, 076
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 076, 004, 010, 004, 002, 042, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 004, 014, 024, 044, 076, 004, 004
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 076, 040, 074, 002, 002, 042, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 014, 020, 040, 074, 042, 042, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 076, 002, 004, 010, 020, 020, 020
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 042, 042, 034, 042, 042, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 042, 042, 036, 002, 004, 030
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 030, 030, 000, 030, 030, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 030, 030, 000, 030, 030, 010
.byte   020, 000, 000, 000, 000, 000, 000, 000
.byte   000, 002, 004, 010, 020, 010, 004, 002
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 076, 000, 076, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 040, 020, 010, 004, 010, 020, 040
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 042, 002, 004, 010, 000, 010
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 014, 022, 056, 052, 056, 040, 036
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 042, 042, 076, 042, 042, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 074, 042, 042, 074, 042, 042, 074
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 042, 040, 040, 040, 042, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 070, 044, 042, 042, 042, 044, 070
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 076, 040, 040, 074, 040, 040, 076
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 076, 040, 040, 074, 040, 040, 040
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 036, 040, 040, 046, 042, 042, 036
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 042, 042, 042, 076, 042, 042, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 010, 010, 010, 010, 010, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
```

```
.byte   000, 016, 004, 004, 004, 004, 044, 030
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 042, 044, 050, 060, 050, 044, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 040, 040, 040, 040, 040, 040, 076
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 042, 066, 052, 052, 042, 042, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 042, 042, 062, 052, 046, 042, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 042, 042, 042, 042, 042, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 074, 042, 042, 074, 040, 040, 040
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 042, 042, 042, 052, 044, 032
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 074, 042, 042, 074, 050, 044, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 034, 042, 040, 034, 002, 042, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 076, 010, 010, 010, 010, 010, 010
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 042, 042, 042, 042, 042, 042, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 042, 042, 042, 024, 024, 010, 010
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 042, 042, 042, 052, 052, 052, 024
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 042, 042, 024, 010, 024, 042, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 042, 042, 024, 010, 010, 010, 010
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 076, 002, 004, 010, 020, 040, 076
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 016, 010, 010, 010, 010, 010, 016
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 100, 040, 020, 010, 004, 002, 001
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 070, 010, 010, 010, 010, 010, 070
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 010, 024, 042, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 076
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 010, 010, 004, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 034, 002, 036, 042, 036
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 040, 040, 074, 042, 042, 042, 074
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 034, 042, 040, 040, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 002, 002, 036, 042, 042, 042, 036
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 034, 042, 076, 040, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 014, 022, 070, 020, 020, 020, 020
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 032, 046, 042, 046, 032
.byte   002, 034, 000, 000, 000, 000, 000, 000
.byte   000, 040, 040, 074, 042, 042, 042, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 010, 000, 030, 010, 010, 010, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 002, 000, 002, 002, 002, 002, 002
.byte   042, 034, 000, 000, 000, 000, 000, 000
.byte   000, 040, 040, 044, 050, 064, 042, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 030, 010, 010, 010, 010, 010, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 064, 052, 052, 052, 052
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 074, 042, 042, 042, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 034, 042, 042, 042, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 054, 062, 042, 062, 054
.byte   040, 040, 000, 000, 000, 000, 000, 000
```

```
.byte   000, 000, 000, 032, 046, 042, 046, 032
.byte   002, 002, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 054, 062, 040, 040, 040
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 034, 040, 034, 002, 034
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 020, 020, 070, 020, 020, 022, 014
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 042, 042, 042, 046, 032
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 042, 042, 042, 024, 010
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 042, 042, 052, 052, 024
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 042, 024, 010, 024, 042
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 042, 042, 042, 046, 032
.byte   002, 034, 000, 000, 000, 000, 000, 000
.byte   000, 000, 000, 076, 004, 010, 020, 076
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 014, 020, 020, 040, 020, 020, 014
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 010, 010, 010, 000, 010, 010, 010
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 030, 004, 004, 002, 004, 004, 030
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 020, 052, 004, 000, 000, 000, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
.byte   000, 000, 034, 076, 076, 076, 034, 000
.byte   000, 000, 000, 000, 000, 000, 000, 000
```

```
-xms
dspcgc
mc6845
vector
r6571a
figrom
vt1xx
-b RAM = 0
-b VARSAV = varsav
-b DPVSAV = dpvsav
-b BUFSAV = bufsav
-b VECTOR = chrom1
-b R6571A = chrom2
-b VT1XX  = chrom3
-b FIGROM = chrom4
-b PGMSAV = pgmsav
-b EXTSAV = extsav
-b IRQINT = irqint
-e
```

```
.title  Option File for Monitor - 09

.module monopt

;  The following labels and data allocations
;  are defined in DSPCGC.

;  This file should be the first file to be linked
;  to insure that the dsiplatch table is cleared.

; $xtrn0:       .blkb   4       ; 4096 bytes allocated
; $xtrn1:       .blkb   4       ; for optional functions
; $xtrn2:       .blkb   4
; $xtrn3:       .blkb   4
; $xtrn4:       .blkb   4
; $xtrn5:       .blkb   4
; $xtrn6:       .blkb   4
; $xtrn7:       .blkb   4
; $xtrn8:       .blkb   4
; $xtrn9:       .blkb   4

; $xtrnl:       .blkb   4       ; loadable printer driver $t_ entry
; $xscrn:       .blkb   4       ; loadable printer driver screen entry

; $xtend                        ; end of user area


;  System Vector Table as defined in DSPCGC.

; extbypls:     .blkb   2       ; loadable system vectors
; extbymns:     .blkb   2
; extbxpls:     .blkb   2
; extbxmns:     .blkb   2
; exundfnd:     .blkb   2
; exdltint:     .blkb   2
; exdlrint:     .blkb   2
; exclocki:     .blkb   2

; exrsrv:       .blkb   2
; exswi3:       .blkb   2
; exswi2:       .blkb   2
; exfirq:       .blkb   2
; exirq:        .blkb   2
; exswi:        .blkb   2
; exnmi:        .blkb   2

.page
.sbttl  Option Dispatcher

.area   OPTFUN  (ABS,OVR)

.radix  o

.blkb   4               ; $0
.blkb   4               ; $1
.blkb   4               ; $2
.blkb   4               ; $3
.blkb   4               ; $4
.blkb   4               ; $5
.word   .$6             ; $6 Command Entry
.word   .$6-0x5AA5
.word   .$7             ; $7
.word   .$7-0x5AA5
.word   .$8             ; $8
.word   .$8-0x5AA5
.blkb   4               ; $9

.blkb   4               ; loadable printer driver $t_ entry
                        ; check code

.blkb   4               ; loadable printer driver screen entry
                        ; check code


.page
.sbttl  Monitor - 09 Option

.area   MONOPT  (ABS,OVR)
```

```
mon$id: .byte   15,12
        .ascii  *DSPCGC   V02.02*
        .byte   15,12
        .ascii  *MONDEB - 09   6809 Monitor - V01.00*
        .byte   15,12
        monl$id =       .-mon$id        ; length of version
        .ascii  *Copyright 1988*
        .byte   15,12
        .ascii  *Otselic Specialties*
        .byte   15,12
        .ascii  *721 Berkeley*
        .byte   15,12
        .ascii  *Kent, Ohio  44240*
        .byte   15,12
        monl$cr =       .-mon$id        ; length of notice


        .page
        .sbttl  $6 command

.$6:    jsr     nxtchr          ; get character

        jsr     $dispatch
        .byte   'V
        .word   2$              ; version select
        .byte   'C
        .word   3$              ; copyright select
        .byte   0

2$:     lda     #monl$id        ; mon$id character version #
        bra     4$

3$:     lda     #monl$cr        ; mon$cr copyright version #

4$:     sta     number
        ldx     #mon$id         ; version string
5$:     ldb     ,x+             ; get character
        stx     qt              ; save pointer
        jsr     plcbuf          ; send character
        ldx     qt              ; get pointer
        dec     number          ; more ?
        bne     5$              ; yes - loop
        bra     .$6

        .page
        .sbttl  MONDEB - 09 Startup Entry Points

.$7:    ldd     #typswi         ; check if settup
        cmpd    exswi
        bne     .$8
        swi                     ; trap into MOND09
        rts

.$8:    ldd     #typswi
        std     exswi           ; swi interrupt vector
        ldd     #dspidta
        std     conin
        std     altin
        ldd     #dspodta
        std     conout
        std     altout
        jsr     mond09          ; start up MONDEB - 09
        rts

        .page
        .sbttl  DSPGGC I/O Drivers

        ;*******************************************************
        ;*      default DSPCGC i/o drivers
        ;*******************************************************

        ;* dspidta - return console input character
        ;* output: c=0 if no data ready, c=1 a=character
        ;*
        ;* DSPCGC panel leds are sequenced here

dspidta:
```

```
        tst     hrcsr           ; a character ?
        bmi     1$              ; yes - skip

        ;* LED scanner

        pshs    d
        ldd     workpg          ; led display update
        addd    #4
        std     workpg
        coma
        anda    #0o160          ; mask bits
        pshs    a
        lda     outpl           ; or with other control bits
        anda    #~0o160
        ora     ,s+
        sta     plout           ; load output register
        puls    d

        clc                     ; no character
        rts                     ; return to caller

1$:     lda     hrdata          ; get character
        cmpa    #0o141          ; translate lower to upper
        bcs     2$
        cmpa    #0o173
        bcc     2$
        suba    #0o40
2$:     sta     hrdata          ; character taken
        sec
        rts

        ;* dspodta - output character to console device
        ;* input: a=character to send
        ;* all registers transparent

dspodta:
        tst     htcsr           ; transmitter ready ?
        bpl     dspodta         ; no - loop until ready
        sta     htdata          ; send character
        rts
```

```
        .title  MONDEB - 09

        ;       ******************************************************
        ;       *                                                    *
        ;       *       MONDEB - 09  -  a monitor/debugger           *
        ;       *               for the 6809  microprocessor         *
        ;       *                                                    *
        ;       ******************************************************
        ;
        ; author:       Don Peters      (6800 version)
        ; date:         April 1977
        ;
        ; modified for the 6809
        ; by:           Alan R. Baldwin
        ; date:         Nov 1988
        ;
        ;         This 6809 monitor/debugger does not use the
        ;       direct page addressing mode.  Thus the user
        ;       program has complete control of the page
        ;       assignment and may use the monitor routines
        ;       without concern for variable locations.
        ;
        ;         To add user functions to MONDEB - 09
        ;
        ;       1) add your commands to the end of command list #1
        ;       2) add the command entry points to the jump table
        ;       3) append any local command lists after MONDEB's
        ;       4) provide an external initialization routine to
        ;               set up the console (and/or alternate)
        ;               ports and any other start up processing.
        ;          MONDEB will call 'userinit' if inituser = 1.
        ;

        .module mond09

        standalone = 0  ; standalone flag indicating the
                        ; vectors and associated code are
                        ; to be included during assembly

        inituser = 0    ; call 'userinit' flag indicating
                        ; a user startup routine will be
                        ; called during intialization

        .page
        .sbttl  MONDEB - 09 working variables

        .radix  d

        .if     standalone

        .area   WORKPG  (REL,CON)

        .else

        .area   WORKPG  (ABS,OVR)

        .endif

        .setdp  0

workpg: .blkb   256-(endpg-mstack)
                        ; main stack storage
mstack: .blkb   12      ; stack storage for rti instruction

ttybuf: .blkb   72      ; start of input line buffer
ttyend: .blkb   1       ; end of input line buffer
bufbeg: .blkb   2       ; input line start of buffer
bufend: .blkb   2       ; input line end of buffer

comadr: .blkb   2       ; address of beginning of command lists
synptr: .blkb   2       ; input line character pointer for good syntax
linptr: .blkb   2       ; input line character pointer (content =>
                        ; content of synptr)
bolflg: .blkb   1       ; "beginning of line flg"
delim:  .blkb   1       ; characters permitted as valid command/modifier
                        ; delimiter
ibcode: .blkb   1       ; input base (1=hex, 2=dec, 3=oct)
dbcode: .blkb   1       ; display base (1=hex, 2=dec, 3=oct, 4=bin)
```

```
dbnbr:  .blkb   1       ; display base number (e.g., 16,10,8, or 2)
nbrhi:  .blkb   1       ; most significant byte of scanned number
nbrlo:  .blkb   1       ; least significant byte of scanned number
ranglo: .blkb   2       ; range lower limit picked up by gtrang
ranghi: .blkb   2       ; range upper limit picked up by gtrang
verfrm: .blkb   2       ; beginning address of range to verify
verto:  .blkb   2       ; ending address of range to checksum verify
chksum: .blkb   1       ; checksum of range given in the verify command
brkadr: .blkb   2       ; address of inserted breakpoint
brkins: .blkb   1       ; instruction which should be there normally
inpflg: .blkb   1       ; alternate input flag
outflg: .blkb   1       ; alternate output flag
outadr: .blkb   2       ; alternate address that the output chars go to
hdxflg: .blkb   1       ; half-duplex terminal flag (if non-zero, no echo)
cplcnt: .blkb   1       ; "characters per line" count
cplmax: .blkb   1       ; "characters per line" maximum

        ; temporary (locally used) variables

temp1:  .blkb   2       ; in: main
temp2:  .blkb   2       ; in: main
temp3:  .blkb   2       ; in: main
temp4:  .blkb   2       ; in: main
temp5:  .blkb   2       ; in: main
temp6:  .blkb   2       ; in: main

nummat: .blkb   1       ; used in command
lisnum: .blkb   1       ; used in command
comnum: .blkb   1       ; used in command
lisptr: .blkb   2       ; used in command
decdig: .blkb   1       ; decimal digit being built
numbhi: .blkb   1       ; used by outnum
numblo: .blkb   1       ; used by outnum
nbr2x:  .blkb   2       ; used by number
timcon: .blkb   2       ; delay time constant
bytect: .blkb   1       ; record byte count used in load command
cksm:   .blkb   1       ; record checksum used in load command

conin:  .blkb   2       ; address of console input routine
conout: .blkb   2       ; address of console output routine
altin:  .blkb   2       ; address of alternate input routine
altout: .blkb   2       ; address of alternate output routine

.rsrvd: .blkb   2       ; reserved vector
.swi3:  .blkb   2       ; swi3 vector
.swi2:  .blkb   2       ; swi2 vector
.firq:  .blkb   2       ; firq vector
.irq:   .blkb   2       ; irq vector
.swi:   .blkb   2       ; swi vector
.nmi:   .blkb   2       ; nmi vector
spsave: .blkb   2       ; saved stack pointer
endpg:

        ; convenient equivalences for local variables

memadr  =       temp1   ; display, set, search, test
strnum  =       temp2   ; fndstr
eoschr  =       temp2+1 ; fndstr

        ; for "search" command

bytptr  =       temp2
nbytes  =       temp3
nbrmat  =       temp3+1
bytstr  =       temp4

        ; other constants

cr      =       13      ; carriage return
lf      =       10      ; line feed


        .page
        .sbttl  MOND09 Startup

        .if     standalone

        .area   MONDEB  (REL,CON)
```

```
reset:  lds     #mstack+12      ; load stack pointer
        bsr     mond09          ; stack pc and start mondeb
        bra     reset
        .else

        .area   MONDEB  (ABS,OVR)

        .endif

mond09: pshs    u,y,x,dp,b,a,cc ; pseudo swi
        sts     spsave          ; save the pointer
        orcc    #0b01010000     ; hold irq and firq interrupts
        lds     #mstack         ; initialize the stack pointer
        jsr     inital          ; initialize variables
        jsr     docrlf          ; advance to a clean line
        ldx     #msghed         ; get address of header
        jsr     outstr          ; type it
        ldx     #ttybuf-1       ; get address of terminal input buffer
        stx     bufbeg          ; save it
        ldx     #ttyend         ; define end of input buffer
        stx     bufend          ; 72 char capacity, incl cr
        lda     #3              ; delimiter class definition
        sta     delim           ; space or comma (code 3)
        bra     promp1

        ; prepare to get a new command

prompt: jsr     docrlf  ; type cr-lf
        inc     bolflg  ; set 'beginning of line' flag
        ldx     synptr  ; point to current character
        lda     ,x      ; get it
        cmpa    #';     ; semicolon?
        beq     getcmd  ; continue scan if it is,
                        ; skipping the prompt
promp1: ldx     #msgprm ; type prompt
        jsr     outstr
        jsr     getlin  ; get line of input
        cmpb    #3      ; abort line on a control-c
        beq     prompt

        ldx     bufbeg  ; set syntax scanning pointer
        stx     synptr  ; to beginning of buffer/line
        lda     1,x     ; get first char
        jsr     tsteol  ; reprompt on an empty line
        beq     prompt  ; (first char = cr,lf,or ;)

        .page
        .sbttl  Get Command

getcmd: lda     #1      ; use list 1 when matching
        jsr     comand  ; now go for a match
        beq     prompt  ; reprompt if just a cr was typed
        bgt     jmpcmd  ; good command if positive

badsyn: ldx     bufbeg  ; get start of line

1$:     cpx     linptr  ; space over to the error in syntax
        beq     2$      ; at error ?
        jsr     outsp   ; output a space
        inx             ; no, move on
        bra     1$

        ; the 'extra' char '1' is compensated for by the prompt
        ; char on the preceeding line

2$:     lda     #'^     ; at error - get an up-arrow
        jsr     outchr  ; print it
        jsr     docrlf
        bra     promp1  ; ignore any succeeding packed
                        ; commands

        ; *******
        ; *there should be no more characters on the input line
        ;       (except delimiters)

nomore: jsr     skpdlm
        bcs     prompt  ; if carry bit set, end of line
```

```
                        ; (normal)
        bra     badsyn  ; there is something there but shouldn't be

        .page
        .sbttl  Command Dispatcher

        ; ******
        ; execute a computed 'goto' to the proper command

jmpcmd: asla            ; multiply command by 2
        ldx     #1$-2
        jmp     [a,x]   ; jump to designated command

1$:     .word   break
        .word   contin
        .word   compar
        .word   copy
        .word   displa
        .word   dbase
        .word   delay
        .word   dump
        .word   goto
        .word   help
        .word   ibase
        .word   load
        .word   reg
        .word   set
        .word   search
        .word   test
        .word   verify
        .word   cli
        .word   clf
        .word   sei
        .word   sef
        .word   xirq
        .word   xfirq
        .word   xnmi
        .word   xrsrvd
        .word   xswi
        .word   xswi2
        .word   xswi3

        .page
        .sbttl  REG - display registers

reg:                    ; print stack stored swi data
disreg: ldx     spsave  ; get saved stack pointer
        clr     comnum  ; start at beginning of the
                        ; register name list
        bsr     1$      ; type condition codes
        bsr     1$      ; type acca
        bsr     1$      ; type accb
        bsr     1$      ; type dp
        jsr     docrlf  ; advance to a clean line
        bsr     2$      ; type index reg x
        bsr     2$      ; type index reg y
        bsr     2$      ; type index reg u
        jsr     docrlf  ; advance to a clean line
        leax    -9,x    ; back to d
        bsr     2$      ; type d
        leax    7,x     ; back to pc
        bsr     2$      ; type program counter

        ; type the stack pointer location

        bsr     3$      ; type stack pointer id
        ldx     #spsave
        jsr     out2by  ; type the value
        jmp     nomore

        ; output content of a 1 byte register

1$:     bsr     3$
        jsr     out1by
        inx
        rts

        ; output content of a 2 byte register
```

```
2$:     bsr     3$
        jsr     out2by
        leax    2,x     ; skip to next word in stack
        rts

        ; misc setup for register display

3$:     jsr     outsp   ; output a space
        inc     comnum  ; skip to next register name
        lda     #5      ; register name is in list 5
        jsr     typcmd  ; type it
        jsr     outeq   ; type an '='
        rts

        .page
        .sbttl  Software Interrupt Entry Point

typswi: sts     spsave
        ldx     #msgswi
        jsr     outstr

        ; decrement pc so it points to 'swi' instruction

        ldx     spsave
        ldd     10,x
        subd    #1
        cmpd    brkadr  ; a break point ?
        bne     disreg  ; no - allow to pass
        std     10,x    ; backup PC
        bra     disreg  ; go display registers

        .sbttl  GOTO - Go To Memory Address

goto:   jsr     mnumber ; get destination
        beq     contin  ; if none, just continue
        ldd     nbrhi
        lds     spsave  ; place new PC
        std     10,s    ; fall through to Continue

        .sbttl  Continue - Continue From a 'SWI'

contin: lds     spsave  ; in case sp was modified via set
        rti             ; command

        .sbttl  SEI - Set Interrupt Mask

sei:    orcc    #0b00010000
        jmp     nomore

        .sbttl  CLI - Clear Interrupt Mask

cli:    andcc   #~0b00010000
        jmp     nomore

        .sbttl  SEF - Set Fast Interrupt Mask

sef:    orcc    #0b01000000
        jmp     nomore

        .sbttl  CLF - Clear Fast Interrupt Mask

clf:    andcc   #~0b01000000
        jmp     nomore

        .page
        .sbttl  COPY - Copy From One Location To Another

copy:   jsr     gtrang  ; get source range into ranglo &
                        ; ranghi
        lble    badsyn  ; error if no source
        jsr     mnumber ; get destination
        lble    badsyn  ; error if no destination
        ldx     ranglo  ; get source address pointer
        ldu     nbrhi   ; get destination address pointer
        lda     ,x      ; get byte from source
1$:     sta     ,u+     ; save byte in destination
        cpx     ranghi  ; compare to end of input range
```

```
        lbeq    nomore  ; done if equal
        lda     ,x+     ; get byte from source
        bra     1$      ; loop for next byte

        .page
        .sbttl  BREAK - Set Breakpoint at Specified Address

break:  jsr     mnumber ; get breakpoint location
        bmi     3$      ; if not numeric, look for '?'
        beq     2$      ; if no modifier, remove old breakpoint
        ldx     brkadr  ; get current break address
        lda     ,x      ; and the char there
        cmpa    #0x3f   ; compare to 'swi'
        bne     1$      ; equal?
        lda     brkins  ; yes, restore the old instruction
        sta     ,x      ; restore it

1$:     ldx     nbrhi   ; get new breakpoint
        stx     brkadr  ; save it
        lda     ,x      ; get instruction stored there
        sta     brkins  ; save it
        lda     #0x3f   ; get code for software interrupt
        sta     ,x      ; put it at breakpoint
        bra     5$      ; all done

2$:     ldx     brkadr  ; get address of break
        lda     ,x      ; get inst. there
        cmpa    #0x3f   ; swi?
        bne     5$      ; if not, return & prompt
        lda     brkins  ; was a swi - get previous inst.
        sta     ,x      ; & restore it
        bra     5$

3$:     lda     #4
        jsr     comand  ; scan for it
        lble    badsyn  ; bad syntax if not '?'
        ldx     brkadr  ; it is, get break address
        lda     ,x      ; get instruction there
        cmpa    #0x3f   ; is it a 'swi'?
        beq     4$      ; if yes, say so
        ldx     #msgnbr ; get that message
        jsr     outstr  ; say it
        bra     5$

4$:     ldx     #msgbat ; get that message
        jsr     outstr  ; say it
        ldx     #brkadr ; get break address
        jsr     out2by  ; type it
5$:     jmp     nomore

        .page
        .sbttl  IBASE - Set Input Base

ibase:  lda     #3      ; look for hex,dec,or oct in list #3
        jsr     comand
        bmi     2$      ; unrecognizable base, try'?'
        bgt     1$
        lda     #1      ; no base given - default to hex
1$:     sta     ibcode  ; save base code
        jmp     nomore

2$:     lda     ibcode  ; get ib code in case its needed
        pshs    a       ; save it on stack temporarily
        bra     ibdbq

        .sbttl  DBASE - Set Display Base

dbase:  lda     #3      ; look for hex,dec,oct, or bin in list #3
        jsr     comand
        bmi     3$      ; unrecognizable base, try '?'
        bgt     1$
        lda     #1      ; no base given - default to hex
1$:     sta     dbcode
        ldx     #2$-1   ; get numeric base from table
        lda     a,x
        sta     dbnbr   ; save it
        jmp     nomore  ; done
```

```
2$:     .byte   16      ; display base table
        .byte   10
        .byte   8
        .byte   2

3$:     lda     dbcode  ; get db code in case its needed
        pshs a          ; save it on stack temporarily

ibdbq:  lda     #4      ; look for '?' in list #4
        jsr     comand
        puls b          ; retrieve input base/display base code
        lble    badsyn  ; error if the 'something' was not a '?'
        lda     #3      ; base code is in list 3
        stb     comnum  ; store base code
        jsr     typcmd  ; type out base
        jmp     nomore

        .page
        .sbttl  Display - Display Memory Data

displa: jsr     gtrang  ; get memory display range
        lble    badsyn  ; address is required
        ldx     ranglo  ; initialize address pointer
        stx     memadr

        lda     #6      ; search list 6 for
        jsr     comand  ; display modifiers 'data' or 'used'
        lbmi    badsyn  ; any other modifier is illegal
        deca            ; adj display modifier code so that:
        sta     comnum  ; -1=addr & data, 0=data, 1=used
        clrb            ; init 'data values per line' counter
        incb
1$:     ldx     #memadr
        tst     comnum  ; which display option?
        bmi     6$      ; if 'address & data', go there
        decb            ; count data values per line
        bne     2$      ; if count not up, skip address output
        jsr     docrlf  ; get to line beginning
        jsr     out2by  ; output address
        jsr     outsp   ; and a space
        ldb     dbnbr   ; reset line counter

2$:     ldx     memadr  ; point to data at that address
        tst     comnum  ; want 'data' option?
        bgt     3$      ; if not, go to 'used' code

        jsr     outsp   ; output preceedng space
        bra     7$

3$:     lda     ,x      ; get the data
        bne     4$
        lda     #'.     ; its zero, get a '.'
        bra     5$

4$:     lda     #'+     ; its non-zero, get a '+'
5$:     jsr     outchr  ; output the '.' or '+'
        bra     8$

6$:     jsr     outsp   ; output a preceeding space
        jsr     out2by  ; type address
        jsr     outeq   ; type  '='
        ldx     ,x      ; get content
7$:     jsr     out1by  ; type it
8$:     cpx     ranghi  ; are we done
        lbeq    nomore  ; if yes, back to prompt
        inx             ; no, inc memory address
        stx     memadr  ; save it
        bra     1$

        .page
        .sbttl  SET - Set Memory Locations

set:    jsr     gtrang  ; get memory location/range
        bmi     5$      ; if not an address, look for a
                        ; register name
        lbeq    badsyn  ; an address modifier is required

        ; range of address specified?
```

```
        ldx     ranglo
        cpx     ranghi
        beq     2$      ; if single address, set up
                        ; addresses individually

        ; set a range of addresses to a single value

        jsr     mnumber ; get that value
        lble    badsyn  ; its   required
        lda     nbrlo   ; put it in acca
1$:     sta     ,x      ; store it in destination
        cpx     ranghi  ; end of range hit?
        lbeq    nomore  ; if yes, all done
        inx             ; no, on to next address in range
        bra     1$      ; loop to see it

        ; set addresses up individually

2$:     stx     memadr  ; save memory loc
3$:     jsr     mnumber ; get data to put there
        beq     4$      ; end of line?
        lblt    badsyn  ; abort if bad syntax
        lda     nbrlo   ; load data byte
        ldx     memadr  ; load address
        sta     ,x+     ; store data
        bra     2$

4$:     ldx     synptr  ; point to end of line
        lda     ,x      ; get char there
        cmpa    #lf     ; line feed?
        lbne    nomore  ; if not,back to prompt
        ldx     #memadr ; yes, get next address to be set
        jsr     out2by  ; type it
        jsr     outsp   ; and a space
        jsr     getlin  ; get a new line
        ldx     bufbeg  ; get buffer beginning
        stx     synptr  ; equate it to syntax scan pointer
        bra     3$      ; go pick up data

        ; look for (register name, register value) pairs

5$:     lda     #5
        jsr     comand  ; pick up a register name
        lbmi    badsyn  ; error if unrecognizable
        lbeq    nomore  ; done if end of line
        pshs a          ; save register name(number)
        jsr     mnumber ; get new register value
        puls a          ; restore register name(number)
        lble    badsyn  ; got good register value?
        deca
        ldu     #6$
        lda     a,u
        leau    a,u
        ldx     spsave  ; yes, point to top of stack
        ldd     nbrhi   ; get register value
        jsr     ,u      ; go to function
        bra     5$      ; and loop

6$:     .byte   7$-6$
        .byte   8$-6$
        .byte   9$-6$
        .byte   10$-6$
        .byte   11$-6$
        .byte   12$-6$
        .byte   13$-6$
        .byte   14$-6$
        .byte   15$-6$
        .byte   16$-6$

7$:     stb     ,x      ; condition codes
        rts

8$:     stb     1,x     ; acca
        rts

9$:     stb     2,x     ; accb
        rts
```

```
10$:    stb     3,x     ; dp
        rts

11$:    std     4,x     ; ix
        rts

12$:    std     6,x     ; iy
        rts

13$:    std     8,x     ; iu
        rts

14$:    std     1,x     ; d
        rts

15$:    std     10,x    ; pc
        rts

16$:    std     spsave  ; sp
        rts

        .page
        .sbttl  Checksum Verify a Block of Memory

verify: jsr     gtrang  ; get a number range
        beq     1$      ; no modifier means check what we have
        lbmi    badsyn  ; anything else is illegal

        ldx     ranglo  ; good range given,
        stx     verfrm  ; transfer it to checksum addresses
        ldx     ranghi
        stx     verto

        bsr     cksum   ; compute checksum
        sta     chksum  ; save it
        ldx     #chksum ; type the checksum
        jsr     out1by
        bra     3$

1$:     bsr     cksum   ; compute checksum
        cmpa    chksum  ; same as stored checksum?
        bne     2$

        ldx     #msgver ; they verify - say so
        jsr     outstr
        bra     3$

2$:     ldx     #msgnve ; they don't - say so
        jsr     outstr
3$:     jmp     nomore

        ; compute the checksum from addresses verfrm to verto
        ; return the checksum in acca

cksum:  clra            ; init checksum to zero
        ldx     verfrm  ; get first address
        dex             ; init to one less
1$:     inx             ; start of checksum loop
        adda    ,x      ; update checksum in acca with
                        ; byte pointed to
        cpx     verto   ; hit end of range?
        bne     1$      ; if not, loop back
        coma            ; complement the sum
        rts             ; return with it


        .page
        .sbttl  Search Memory for Byte String

        ; global variables used
        ;  linptr - input line character pointer
        ;  lisptr - command list character pointer
        ;  ranglo - 'search from' address
        ;  ranghi - 'search to' address
        ;
        ; local variables used
        ;  memadr - starting memory address where a match
```

```
        ;               occurred
        ;  bytptr - address pointer used to fill bytstr and
        ;               substr buffers
        ;  nbytes - number of bytes in byte string
        ;  nbrmat - number of chars that match so far in the
        ;               matching process
        ;  bytstr - starting address of 6 character byte sring
        ;               buffer
        ;
        ; the search string occupies temp4, temp5, & temp6
        ;               (6 bytes max)

search: jsr     gtrang  ; get search range
        lble    badsyn  ; abort if no pair
        ldx     #bytstr ; get start of byte string
                        ; to search for
        stx     bytptr  ; set pointer to it
        clr     nbytes  ; zero # of bytes in byte string

1$:     jsr     mnumber ; get a byte string
        beq     2$      ; begin search if eol
        lblt    badsyn
                        ; good byte, add it to string
        inc     nbytes  ; count this  byte
        lda     nbytes  ; don't accept over 6 bytes
        cmpa    #6
        lbgt    badsyn
        lda     nbrlo   ; get (low order) byte
        ldx     bytptr  ; get byte pointer
        sta     ,x+     ; save byte, bump pointer
        stx     bytptr  ; save it
        bra     1$

2$:     tst     nbytes  ; is # of bytes to look for >0
        lbeq    badsyn  ; if not,bad syntax
        ldx     ranglo  ; initialize memory pointer
        dex
        stx     linptr

3$:     ldx  #bytstr-1  ; initialize byte pointer
        stx     lisptr
        clr     nbrmat  ; set 'number of bytes that
                        ; matched' to zero
        jsr     getlst  ; get byte from byte string

4$:     jsr     mgetchr ; get byte from memory range
        cba             ; compare memory & byte string
                        ; characters
        beq     5$      ; if no match, test for range end
        cpx     ranghi  ; have we reached the range
                        ; search upper limit?
        lbeq    nomore  ; yes, go prompt for next command
        bra     4$

5$:     stx     memadr  ; match achieved - save address of match
6$:     inc     nbrmat  ; bump number matched
        lda     nbrmat
        cmpa    nbytes  ; have all characters matched?
        beq     8$      ; if so, match achieved

        jsr     getlst  ; haven't matched all yet, go get next pair
        jsr     mgetchr ; even if past 'search to' address
        cba
        beq     6$

7$:     ldx     memadr  ; mismatch on some byte past the first one

        cpx     ranghi  ; this test handles special case
        lbeq    nomore  ; of a match on range end
        stx     linptr
        bra     3$      ; go reset the byte string pointer


8$:     ldx     #memadr ; match on byte string achieved,
        jsr     out2by  ; type out memory address
        jsr     outsp   ; and a space
        bra     7$
```

```
        .page
        .sbttl  Test Ram

test:   jsr     gtrang  ; get an address range
        lble    badsyn  ; abort if no pair
        ldu     ranglo  ; ranglo holds starting address of range
                        ; ranghi holds ending address of range
1$:     lda     ,u      ; get byte stored at test location
        pshs    a       ; save it
        clr     ,u      ; zero the location
        tst     ,u      ; test it
        beq     2$      ; ok if = zero
        ldx     #msgccl ; can't clear location
        bra     4$

2$:     dec     ,u      ; set location to $ff
        lda     #0xff
        cmpa    ,u      ; did it get set to $ff?
        beq     3$
        ldx     #msgcso ; can't set location to one's
        bra     4$

3$:     puls    a
        sta     ,u      ; restore previous content
        cmpu    ranghi  ; hit end of test range?
        lbeq    nomore  ; yes, all done
        leau    1,u     ; no, move to test next location
        bra     1$

4$:     stx     temp3   ; save error message temporarily
        ldx     #temp4
        stu     ,x
        jsr     out2by  ; type out bad address
        jsr     outeq   ; and equal sign
        ldx     temp4
        jsr     out1by  ; its content
        jsr     outsp   ; a space
        ldx     temp3
        jsr     outstr  ; and the type of error
        jsr     docrlf
        bra     3$

        .page
        .sbttl  vector settup commands

        ; rsrvd - set up rsvd pointer

xrsrvd: jsr     numinx  ; get pointer in ix
        stx     .rsrvd  ; save it
        jmp     nomore

        ; swi3 - set up swi3 pointer

xswi3:  jsr     numinx  ; get pointer in ix
        stx     .swi3   ; save it
        jmp     nomore

        ; swi2 - set up swi2 pointer

xswi2:  jsr     numinx  ; get pointer in ix
        stx     .swi2   ; save it
        jmp     nomore

        ; firq - set up interrupt pointer

xfirq:  jsr     numinx  ; get pointer in ix
        stx     .firq   ; save it
        jmp     nomore

        ; irq - set up interrupt pointer

xirq:   jsr     numinx  ; get pointer in ix
        stx     .irq    ; save it
        jmp     nomore

        ; swi - set up swi pointer

xswi:   jsr     numinx  ; get pointer in ix
```

```
        stx     .swi    ; save it
        jmp     nomore

        ; nmi - set up non-maskable interrupt pointer

xnmi:   jsr     numinx  ; get ponter in ix
        stx     .nmi    ; save it
        jmp     nomore

        .page
        .sbttl  compare numbers

        ; compare - output sum & difference of two input numbers

compar: jsr     numinx  ; get first number
        stx     ranglo  ; put it in ranglo
        jsr     numinx  ; get second number
        stx     nbrhi   ; save it in nbrhi

        ; compute and output the sum

        jsr     sumnum  ; compute sum
        ldx     #msgsis ;get its title
        bsr     1$      ; output title & sum

        jsr     difnum  ; compute difference
        ldx     #msgdis ; get its title
        bsr     1$      ; output title & diff

        jmp     nomore

        ; compute and output the result

1$:     jsr     outstr  ; output it
        ldx     #ranghi ; get result
        jsr     out2by  ; display result
        rts

        .page
        .sbttl  dump memory in S1-S9 format

        ; *****
        ; dump - dump portion of memory in S1-S9 format
        ;
        ; get address range: start in ranglo (2 bytes), end in
        ;               ranghi (2bytes)
        ; if no address range is given, use whatever is n
        ;               ranlo & ranghi

dump:   jsr     gtrang
        clr     temp5   ; initialize to dump to terminal

1$:     lda     #2      ; look for a 'TO' modifier
        jsr     comand
        beq     2$
        lble    badsyn  ; error if bad syntax
        cmpa    #1      ; TO ?
        bne     1$      ; go look for another modifier

        jsr     numinx  ; get 'TO' address
        stx     outadr  ; save it
        inc     temp5   ; remember this
        bra     1$      ; go look for another modifier

2$:     tst     temp5
        beq     3$
        inc     outflg  ; set flag for proper output device

3$:     ldd     ranghi  ; compute # of bytes to output
        subd    ranglo  ; subtract lo bytes
        cmpd    #16     ; diff 0-15.
        bcs     5$
4$:     ldb     #15

        ; to get frame count, add 1
        ; (diff of 0 implies 1 output) + # of data bytes
        ; + 2 addr bytes + 1 checksum byte
```

```
5$:     addb    #4
        stb     temp3   ; temp3 is the frame count
        subb    #3
        stb     temp4   ; temp4 is the record byte count

        ldx     #msgs1  ; output a 'S1' header data record
        jsr     outstr
        clrb            ; zero checksum

        ldx     #temp3  ; punch frame count
        bsr     7$

        ldx     #ranglo ; punch address
        bsr     7$
        bsr     7$

        ldx     ranglo  ; load memory pointer
6$:     bsr     7$      ; output data byte
        dec     temp4   ; dec byte count
        bne     6$
        stx     ranglo  ; save memory pointer

        comb            ; complement checksum
        pshs    b       ; put it on stack
        tfr     s,x     ; let ix point to it
        bsr     7$      ; output checksum
        puls    b       ; pull it off stack
        ldx     ranglo  ; restore memory pointer
        dex
        cpx     ranghi  ; hit end of range?
        bne     3$

        ldx     #msgs9  ; yes, output an 'S9' record
        jsr     outstr
        clr     outflg  ; set to terminal output
        jmp     nomore

7$:     jsr     out1by  ; output a byte pointed to by ix as
        addb    ,x+     ; 2 hex characters, update checksum
        rts


        .page
        .sbttl  Load S1-S9 format Data

load:   jsr     inpchr  ; get a char
        cmpa    #'S     ; is it an S ?
        bne     load

        jsr     inpchr  ; got an 'S', examine next character
        cmpa    #'9     ; done if its a '9'
        beq     3$

        cmpa    #'1     ; is it a '1'?
        bne     load    ; if not, look for next 'S'

        clr     cksm    ; clear checksum

        jsr     5$      ; read record byte count
        suba    #2
        sta     bytect  ; save count minus 2 address bytes

        bsr     4$      ; build address

1$:     bsr     5$      ; read a data byte into acca
        dec     bytect  ; count it
        beq     2$      ; if done with record, check checksum
        sta     ,x+     ; not done, store byte in memory
        bra     1$      ; on to next memory address

2$:     inc     cksm    ; test checksum by adding 1
        beq     load    ; if ok, result should be zero

        ldx     #msgnve ; record checksum error
        jsr     outstr
        ldx     #temp1  ; get record address of it
        jsr     out2by  ; type it too
3$:     clr     inpflg  ; reset flag to normal terminal input
```

```
        jmp     nomore

4$:     bsr     5$      ; build address
        sta     temp1
        bsr     5$
        sta     temp1+1
        ldx     temp1
        rts

5$:     bsr     6$      ; get left hex digit
        asla            ; move to hi 4 bits
        asla
        asla
        asla
        tab             ; save it in acca
        bsr     6$      ; get right hex digit
        aba             ; combine then in acca
        tab             ; update the checksum
        addb    cksm
        stb     cksm
        rts

6$:     jsr     inpchr  ; input a hex char & convert to internal form
        suba    #'0
        bmi     8$      ; not hex if below ascii '1'
        cmpa    #'9-'0
        ble     7$      ; ok if ascii '9' or less
        cmpa    #'A-'0  ; below ascii 'A'?
        bmi     8$      ; error if it is
        cmpa    #'F-'0  ; over ascii 'F'?
        bgt     8$      ; error if it is
        suba    #7      ; conv ascii A-F to hex A-F
7$:     rts

8$:     ldx     #msgcnh ; error - char not hex, say so
        jsr     outstr
        rts

        .page
        .sbttl  Delay Function

        ; *****
        ; delay - delay specified # of milliseconds

delay:  jsr     numinx  ; get delay time
        bsr     timdel
        jmp     nomore

        ; *****
        ; time delay subroutine
        ; ix is input as the # of milliseconds to delay
        ; adj timcon so (7*timcon*cycle time=1 ms)

timdel: pshs    d
1$:     ldd     timcon

2$:     subd    #1      ; a 7 cycle loop
        bne     2$

        dex             ; decrement millisecond counter
        bne     1$
        puls    d,pc

        .page
        .sbttl  Help List

help:   jsr     docrlf  ; next line
        ldx     #comlst ; command list

1$:     ldb     #4      ; commands per line
        stb     temp1

2$:     ldb     #12     ; positions per command
                        ; must be larger than longest command
3$:     lda     ,x+     ; get character
        cmpa    #cr     ; <cr> is end of command
        beq     4$
        jsr     outchr  ; print command character
```

```
        decb
        bne     3$

4$:     lda     ,x      ; get character
        cmpa    #lf     ; <lf> is end of list
        beq     6$      ; finished
        dec     temp1   ; per line done ?
        bne     5$      ; no - skip

        jsr     docrlf  ; next line
        bra     1$

5$:     lda     #'      ; space
        jsr     outchr
        decb
        bne     5$
        bra     2$

6$:     jsr     docrlf  ; next line
        jmp     nomore

        .page
        .sbttl  Command Lists

        ;===============================================
        ;
        ; c o m m a n d   l i s t   s c a n n i n g
        ;    r o u t i n e
        ;
        ; this routine seeks a match of the characters pointed
        ; at by the input line scanning pointer to one of the
        ; commands in a list specified by acca.
        ; the result of the scan for a match is returned in
        ; acca as follows:
        ;
        ;       acca=-1: the match was unsuccessful. the syntax
        ;                pointer (synptr) was not updated
        ;                       (advanced).
        ;
        ;       acca= 0: the match was unsuccessful since there
        ;                       were
        ;                 no more characters, i.e., the end of
        ;                       the
        ;               line was reached.
        ;
        ;       acca=+n: successful match. the syntax pointer
        ;                       was updated
        ;                to the first character following the
        ;                 command
        ;                 delimiter.  acca holds the number of
        ;                     the
        ;                 command matched.
        ; global variables for external communication
        ; synptr - good syntax input line char pointer
        ; linptr - input line character pointer
        ; delim - class of permissible command delimiters
        ;
        ; temporary 2 byte internal variables
        ; lisptr - command list character pointer
        ;
        ; temporary 1 byte internal variables
        ; nummat - number of characters that successfully match
        ; lisnum - # of list within which a match will be sought
        ; comnum - command number matched
        ;
        ; constants used
        ; cr - carriage return
        ; lf - line feed
        ;
        ; a, b & ix are not preserved

comand: sta     lisnum  ; save list # to match within
        jsr     skpdlm  ; test if we are at the end of the line
        bcc     1$
        clra
        rts

        ; initialize the command list pointer to one less than
```

```
        ;               the beginning of the command lists

1$:     ldx     comadr  ; entry point

        ; move to the beginning of the desired command list

        lda     lisnum  ; search for 'string' # lisnum
        ldb     #lf     ; use lf as a 'string' terminator
        bsr     fndstr
        stx     lisptr

        ; the list pointer, lisptr, now points to one less than
        ; the first character
        ; of the first command in the desired list

        clr     comnum

        ; reset input line pointer to: 1) beginning of line, or
        ; to 2) point where last successful scan terminated

2$:     inc     comnum  ; initialize the command # to 1
        ldx     synptr
        stx     linptr
        clr     nummat  ; clear number of characters
                        ; matched
3$:     jsr     mgetchr ; get input line char in accb
        jsr     tstdlm  ; test for a delimiter
        bne     4$      ; success (found delimiter)
        jsr     getlst  ; get command list char in acca
        cmpa    #lf     ; has end of command list been reached ?
        beq     5$      ; if so, potential match failure
        cmpa    #cr     ; has end of command been reached ?
        beq     5$      ; if so, potential match failure
        cba             ; compare the two characters
        bne     6$      ; match not possible on this command
        inc     nummat  ; they match, compare the succeeding characters
        bra     3$      ; inc number of characters matched

4$:     ldx     linptr  ; successful match
        stx     synptr  ; update good syntax pointer
        lda     comnum  ; return command number
        rts

        ; no match

5$:     tst     nummat  ; did at least one match?
        beq     6$      ; to next command if none matched
        jsr     tstdlm  ; at least one matched - test for delimiter
        bne     4$      ; if a delimiter, match has been achieved
        lda     ,x      ; retrieve last character

        ; illegal delimiter
        ; move to next command within list

6$:     cmpa    #lf     ; end of this list?
        beq     7$      ; if so, nothing on list matched
        cmpa    #cr     ; is it a cr?
        beq     2$      ; yes, next command
        jsr     getlst  ; get next command list character
        bra     6$      ; no, get to end of command

7$:     clra    ; match failure
        deca    ; no match possible within this list
        rts

        .page
        .sbttl  Typeout Command

        ;==================================
        ; this routine types out command number "comnum"
        ; the list is specified in acca
        ; accb & ix are preserved

typcmd: pshs    b,x
        ldx  #comlst-1  ; move to head of command lists
        ldb     #lf     ; and list terminator
        bsr     fndstr  ; go to head of desired list
        lda     comnum  ; get command number
```

```
        ldb     #cr     ; get command terminator
        bsr     fndstr  ; go to head of desired command

1$:     inx             ; move to next character
        lda     ,x      ; get a command character
        cmpa    #cr     ; is it a command terminator?
        beq     2$      ; if so, return
        jsr     outchr  ; no, type it
        bra     1$

2$:     puls    b,x,pc

        .page
        .sbttl  Find string

        ;====================================
        ; move to beginning of desired string number (in acca)
        ; each string is terminated by an end of string
        ;   character (in accb)
        ; the index register is assumed initialized pointing to
        ; one less than the first character of the first string
        ; acca, accb & ix are not preserved
        ; local variables
        ; strnum - string # to find
        ; eoschr - "end of string" character

fndstr: sta     strnum  ; save string number
        stb     eoschr  ; save terminator
        clrb
1$:     incb            ; string 1 is the first string
        cmpb    strnum  ; is this the right string?
        beq     3$      ; if so, done

        ; no, swallow up characters until an end of string char is hit

2$:     inx             ; bump pointer to next one
        lda     ,x      ; get char pointed at
        cmpa    eoschr  ; end of string hit?
        beq     1$      ; if it is, bump the string counter
        bra     2$      ; no, move on to next char
3$:     rts             ; ix set properly, return

        .page
        .sbttl  Skip Leading Delimiters

        ;================================================
        ; skip leading delimiters
        ; this routine should be called prior to scanning for
        ; any information
        ; on the input line
        ; the current character is ignored if the scannng
        ; pointer is at the beginning of a line. if not, the
        ; scanning pointer skips over spaces and commas
        ; until an end of line or non-delimiter is found.
        ; the carry bit is set if and end of line is encountered.

        ; acca, accb, & ix are not preserved

skpdlm: clc
        tst     bolflg  ; at beginning of line?
        bgt     2$

        ; look at current input character

1$:     ldx     synptr  ; get pointer to it
        lda     ,x      ; get char
        bsr     tsteol  ; test for end of line
        bne     2$
        sec             ; yes, end hit, set carry
        rts

        ; "peek" at next char in line

2$:     ldb     1,x     ; get it
        bsr     tstdlm  ; see if its a delimiter
        bne     3$
        rts             ; if not, return
```

```
        ; next char is a delimiter

3$:     jsr     mgetchr ; move to next char in input line
        stx     synptr  ; update syntax pointer
        bra     1$      ; go test for end of line


        .page
        .sbttl  Test for End-of-Line Character

        ;=========================================
        ; test for end-of-line character
        ; z bit of cc reg set if char in acca is a terminator
        ; acca, accb, & ix are preserved

tsteol: cmpa    #cr     ; carriage return?
        beq     1$
        cmpa    #lf     ; line feed? (continued lines)
        beq     1$
        cmpa    #';     ; for several commands on one line
1$:     rts

        .sbttl  Test for Delimeter

        ;=============================================
        ; check the character n accb aganst the delimiter(s)
        ; specified by variable delim
        ; accb & ix are preserved
        ; acca is set to 0 if accb is not a delimiter, to 1
        ; if it is
        ;  if delim=1, space s delimiter
        ;  if delim=2, comma is delimiter
        ;  if delim=3, space or comma is delimiter
        ;  if delim=4, any non-alphanumeric is a delimiter
        ; test for end-of-line (logical or physical)

tstdlm: pshs b
        tba
        bsr     tsteol
        puls b
        beq     5$

        lda     delim
        cmpa    #1
        bne     1$
        cmpb    #'      ; want a space - is it?
        bne     6$
        bra     5$

1$:     cmpa    #2
        bne     3$
2$:     cmpb    #',     ; want a comma - is it?
        bne     6$
        bra     5$
3$:     cmpa    #3
        bne     4$
        cmpb    #'      ; want either, is it a space?
        beq     5$
        bra     2$      ; or a comma?

4$:     cmpa    #4
        bne     7$      ; error if delm not 1-4
        cmpb    #'0     ; test if char is 0-9 inclusive
        blt     5$
        cmpb    #'9
        ble     6$

        cmpb    #'A     ; test if char is A to Z inclusive
        blt     5$
        cmpb    #'Z
        ble     6$      ; over Z - its a delimiter

5$:     lda     #1      ; char in accb is a delimiter
        rts

6$:     clra            ; char in accb is not a delimiter
        rts
                        ; error in specifying delimiter class
```

```
7$:     swi             ; have monitor type out pertinent
                        ; statistics

        .page
        .sbttl  Sum Two Numbers

        ;================================================
        ; add the 2 byte number stored in (ranglo,ranglo+1) to
        ; the number stored in (nbrhi,nbrlo) and put the result
        ; in (ranghi,ranghi+1)

sumnum: pshs    d       ; add lo order bytes
        ldd     ranglo
        addd    nbrhi
        std     ranghi
        puls    d,pc

        .sbttl  Subtract Two Numbers

        ;================================================
        ; subtract the 2 byte number stored in (nbrhi,nbrlo)
        ; from the two byte number stored in (ranglo,ranglo+1)
        ; and put the result in (ranghi,ranghi+1)
        ; accb & ix are preserved
        ; acca is altered

difnum: pshs    d       ; subtract lo order bytes
        ldd     ranglo
        subd    nbrhi
        std     ranghi
        puls    d,pc

        .page
        .sbttl  Get Range

        ;================================================
        ; this routine scans the input line for a pair of numbers
        ; representing an address range. a colon separating the
        ; pair implies "thru", while an "!" implies "thru the
        ; following"
        ; e.g., 100:105 is equivalent to 100!5
        ; a single number implies a range of 1
        ; on return (ranglo,ranglo+1) holds the range start, and
        ; (ranghi,ranghi+1) holds the range end
        ; acca, accb, & ix are not preserved

gtrang: bsr     mnumber ; pick up first number
        bgt     1$
        blt     2$
        rts             ; nothing more on input line


1$:     ldx     nbrhi   ; good single number
        stx     ranglo  ; transfer it to ranglo
        bra     3$      ; and to ranghi

        ; bad number, but is it bad due to a ":" or "!" delimiter?
        ; get the terminator for the first number

2$:     ldx     linptr
        lda     ,x
        cmpa    #':     ; was it a colon?
        bne     4$      ; if not, go test for "!"
        bsr     8$      ; was ":", process first number &
                        ; get next one
        ble     5$      ; illegal if end of line or non-numeric

3$:     ldx     nbrhi   ; transfer second number to ranghi
        stx     ranghi
        bra     7$

4$:     cmpa    #'!     ; was delimiter a "!"?
        beq     6$      ; if yes, get 2nd number
        clra            ; illegal delimiter, return
        deca
5$:     rts

6$:     bsr     8$      ; was "!", process first number &
```

```
                        ; get next one
        ble     5$
        bsr     sumnum  ; compute range end, put into ranghi

7$:     lda     #1      ; successful exit
        rts

        ; update syntax pointer, move first number to ranglo,
        ; & get 2nd number

8$:     stx     synptr  ; update syntax pointer
        ldx     nbrhi   ; get first number of the pair
        stx     ranglo  ; save it in "low range" value
        bsr     mnumber ; pick up the second number of the pair
        rts

        .page
        .sbttl  Get number in IX

        ;================================================
        ; get a 2 byte number & return it in the index register

numinx: bsr     mnumber
        bgt     1$
        jmp     badsyn
1$:     ldx     nbrhi
        rts

        .page
        .sbttl  Scan For a Number

        ;================================================
        ; scan for a number
        ; return the most significant byte in nbrhi
        ; and the least significant byte in nbrlo
        ; the result of the scan for a number is returned in
        ; acca as follows:
        ;
        ;       acca=-1: the match was unsuccessful. the syntax
        ;                pointer (synptr) was not updated.
        ;
        ;       acca= 0: the scan was unsuccessful since there
        ;                were no more characters. (i.e., the end
        ;                of the line was encountered.)
        ;
        ;       acca=+1: the scan was successful. the syntax
        ;                pointer was updated to the first character
        ;                following the command.
        ;
        ; ix is preserved
        ; global variables for external communication
        ; nbrhi - number hi byte
        ; nbrlo - number lo byte
        ; ibcode - input base code
        ; dbcode - display base code
        ;
        ; local variables
        ; nbr2x - used in decimal conversion
        ; initialize both bytes to zero

mnumber:pshs    x       ; save ix
        clr     nbrhi
        clr     nbrlo
        ldx     synptr  ; initialize the line scanning pointer
        stx     linptr
        jsr     skpdlm  ; are we at end of line?
        bcc     1$
        clra            ; yes, zero acca
        puls    x,pc

1$:     jsr     mgetchr ; get a character from the input
                        ; line into accb
        jsr     tstdlm  ; test for a delimiter
        bne     6$      ; good delimiter if acca is non-zero
        subb    #'0     ; subtract ascii 0
        bmi     8$      ; error if less

        lda     ibcode  ; determine input base & go to right routine
```

```
        cmpa    #1      ; hex code ?
        beq     2$

        cmpa    #2      ; decimal code ?
        beq     4$

        cmpa    #3      ; octal code ?
        beq     5$

        ; hex input processing

2$:     cmpb    #'9-'0  ; default an illegal input base to hex
        ble     3$      ; if 9 or less
        cmpb    #'A-'0
        bmi     8$      ; not hex if < A
        cmpb    #'F-'0
        bgt     8$      ; not hex if > F
        subb    #7      ; move A-F above 0-9

3$:     bsr     9$      ; shift lo & hi bytes left 4 bits
        bsr     9$
        orb     nbrlo
        stb     nbrlo
        bra     1$

        ; decimal input

4$:     cmpb    #9
        bgt     8$      ; not decimal if > 9

        ; multiply saved value by 10 & add in new digit

        bsr     10$     ; multiply current number by 2 to get 2x value
        ldx     nbrhi   ; save this *2 number temporarily
        stx     nbr2x
        bsr     9$      ; multiply this # by 4 to get 8x value
        clra            ; new digit <a,b>
        addd    nbr2x   ; note that 10x=2x+8x
        bcs     8$      ; carry out of ms byte is an error
        addd    nbrhi
        bcs     8$      ; carry out of ms byte is an error
        std     nbrhi   ; save result
        bra     1$

        ; octal input

5$:     cmpb    #7
        bgt     8$      ; not octal if > 7
        bsr     9$      ; shift hi & lo bytes 3 places left
        bsr     10$     ; carry out of hibyte is illegal
        orb     nbrlo   ; add in new digit
        stb     nbrlo
        bra     1$

6$:     ldx     linptr  ; good number - scan was successful
        stx     synptr  ; update good syntax line pointer
        lda     #1      ; set "good scan" flag
        puls    x,pc

7$:     leas    2,s
8$:     clra            ; conversion error - scan was unsuccessful
        deca
        puls    x,pc

9$:     asl     nbrlo   ; shift a two byte number left one position
        rol     nbrhi
        bcs     7$
10$:    asl     nbrlo   ; shift a two byte number left one position
        rol     nbrhi
        bcs     7$
        rts

        .page
        .sbttl  General Output Routines

        ;=========================================
        ; output a space
```

```
outsp:  lda     #'
        jsr     outchr
        rts

        ;=========================================
        ; output an "=" sign

outeq:  lda     #'=
        jsr     outchr
        rts

        ;=========================================
        ; output a 1 byte number

out1by: pshs    d
        ldb     #1
        bsr     outnum
        puls    d,pc

        ;=========================================
        ; output a 2 byte number

out2by: pshs    d
        ldb     #2
        bsr     outnum
        puls    d,pc

        ;=========================================
        ; display the number pointed at by the address in the
        ; index register and output it according to the base
        ; specified in "dbcode"
        ; leading zeros are included
        ; acca & ix are preserved
        ; accb is input as the number of bytes comprising the
        ; number.
        ; global variables for external communication
        ; ibcode - input base code
        ; dbcode - display base code
        ;
        ; local variables
        ; decdig - decimal digit being built
        ; numbhi - hi byte of number being output
        ; numblo - lo byte of number being output

outnum: pshs    d,x     ; save these
        ldx     ,x      ; get the two bytes at that address
        stx     numbhi  ; put them in a scratch area for processing
        lda     dbcode  ; get display base

        cmpa    #1
        beq     1$
        cmpa    #2
        beq     4$
        cmpa    #3
        beq     11$
        cmpa    #4
        beq     14$

        ; output a hex number

1$:     aslb            ; 1 byte=2 chars, 2 bytes=4 chars
2$:     bsr     16$     ; get next 4 bits
        bsr     16$

        anda    #0x0f   ; extract 4 bits
        cmpa    #9
        ble     3$
        adda    #7      ; convert 10:15 to A-F

3$:     bsr     18$
        decb
        bne     2$
        puls    d,x,pc  ; restore registers & return

        ; output a decimal number

4$:     decb            ; test # of bytes to output
        beq     5$
```

```
        ldx     #9$     ; initialize for output of a 2 byte number
        ldd     numbhi
        bra     6$

5$:     ldx     #10$    ; initialize for output of a 1 byte number
        clra
        ldb     numbhi
6$:     clr     decdig  ; clear the digit to output
7$:     subd    ,x      ; subtract the power of 10 conversion constant
        bcs     8$      ; test for borrow (carry)
        inc     decdig  ; no borrow yet - nc digit being built
        bra     7$      ; repeat loop

        ; building of digit to output is complete - print it

8$:     pshs    a       ; save lo byte of number being output
        lda     decdig  ; get digit
        bsr     18$     ; print it
        puls    a       ; restore lo byte

        addd    ,x++    ; borrow generated - cancel last subtraction
        cpx     #9$+10  ; are we thru with units conversion?
        bne     6$      ; if not, back to get next digit
        puls    d,x,pc  ; if yes, restore registers & return

        ; decimal output conversion constants

9$:     .word   10000
        .word   1000
10$:    .word   100
        .word   10
        .word   1

        ; output an octal number

11$:    aslb            ; first approximation of # of
        clra            ; digits to output
        cmpb    #2
        bgt     12$
        bsr     16$     ; 1 byte - get first 2 bits
        bsr     18$
        bra     13$     ; go output last 2 digits

12$:    bsr     17$     ; two byte # - output hi order bit/digit
        bsr     18$
        incb            ; 5 more digits to go

13$:    bsr     16$     ; get next 3 bits
        bsr     17$
        anda    #7      ; extract 3 bits
        bsr     18$
        decb            ; count this digit
        bne     13$     ; are we done?
        puls    d,x,pc  ; if yes, restore registers & return

14$:    aslb    ; output a binary number
        aslb
        aslb
15$:    bsr     17$     ; get next bit
        anda    #1      ; extract the bit
        bsr     18$     ; output it
        decb            ; count it
        bne     15$     ; are we done?
        puls    d,x,pc  ; if yes, restore registers & return

16$:    bsr     17$     ; left shift 2 bits

17$:    asl     numblo  ; left shift the 3 byte number 1 bit
        rol     numbhi
        rola
        rts

18$:    adda    #'0     ; convert to a numeric ascii digit & output it
        jsr     outchr
        rts

        .page
        .sbttl  Get Character Routines
```

```
        ;================================================
        ; this routine gets the next character from the input
        ; line buffer
        ; acca is preserved
        ; accb is loaded with the character
        ; ix is incremented and left pointing to the character
        ; returned

mgetchr:
        ldx     linptr
        inx
        ldb     ,x
        stx     linptr
        clr     bolflg  ; set flag to not at "beginning of line"
        rts

        ;================================================
        ; this routine gets the next character in the command
        ; lists
        ; acca is the character retrieved
        ; accb is preserved
        ; ix is incremented & left pointing to the character returned

getlst: ldx     lisptr  ; get current list pointer
        inx             ; move pointer to next character
        lda     ,x      ; get character pointed at
        stx     lisptr  ; save pointer
        rts             ; and return

        .page
        .sbttl  Command Lists

        ;================================================
        ; command lists
        ;  a carriage return signifies end-of-command
        ;  a line feed signifies end-of-command-list
        ; list 1 - major commands

comlst: .ascii  /BREAK/         ; set breakpoint (swi code)
        .byte   cr
        .ascii  /CONTINUE/      ; continue from "swi"
        .byte   cr
        .ascii  /COMPARE/       ; print sum & difference of 2 numbers
        .byte   cr
        .ascii  /COPY/          ; copy from one location to another
        .byte   cr
        .ascii  /DISPLAY/       ; display memory data
        .byte   cr
        .ascii  /DBASE/         ; set display base
        .byte   cr
        .ascii  /DELAY/         ; delay specified # of bytes
        .byte   cr
        .ascii  /DUMP/          ; dump memory in mikbug or image format
        .byte   cr
        .ascii  /GOTO/          ; go to memory address
        .byte   cr
        .ascii  /HELP/          ; help listing
        .byte   cr
        .ascii  /IBASE/         ; set input base
        .byte   cr
        .ascii  /LOAD/          ; load mikbug tape
        .byte   cr
        .ascii  /REG/           ; display registers
        .byte   cr
        .ascii  /SET/           ; set memory data
        .byte   cr
        .ascii  /SEARCH/        ; search memory for a byte string
        .byte   cr
        .ascii  /TEST/          ; test a range of memory
        .byte   cr
        .ascii  /VERIFY/        ; verify that memory content is unchanged
        .byte   cr
        .ascii  /CLI/           ; clear interrupt mask
        .byte   cr
        .ascii  /CLF/           ; clear fast interrupt mask
        .byte   cr
        .ascii  /SEI/           ; set interrupt mask
```

```
.byte   cr
.ascii  /SEF/           ; set fast interrupt mask
.byte   cr
.ascii  /IRQ/           ; set interrupt pointer
.byte   cr
.ascii  /FIRQ/          ; set fast interrupt pointer
.byte   cr
.ascii  /NMI/           ; set non-maskable interrupt pointer
.byte   cr
.ascii  /RSRVD/         ; set reserved interrupt pointer
.byte   cr
.ascii  /SWI/           ; set software interrupt pointer
.byte   cr
.ascii  /SWI2/          ; set swi2 interrupt pointer
.byte   cr
.ascii  /SWI3/          ; set swi3 interrupt pointer
.byte   cr
.byte   lf              ; end of list 1

; list 2 - modifier to dump

.ascii  /TO/            ; destination
.byte   cr
.byte   lf              ; end of list 2

; list 3 - number base specifiers

.ascii  /HEX/           ; base 16
.byte   cr
.ascii  /DEC/           ; base 10
.byte   cr
.ascii  /OCT/           ; base 8
.byte   cr
.ascii  /BIN/           ; base 2
.byte   cr
.byte   lf              ; end of list 3

; list 4 - information request

.ascii  /?/
.byte   cr
.byte   lf              ; end of list 4

; list 5 - register names

.ascii  /.CC/
.byte   cr
.ascii  /.A/
.byte   cr
.ascii  /.B/
.byte   cr
.ascii  /.DP/
.byte   cr
.ascii  /.IX/
.byte   cr
.ascii  /.IY/
.byte   cr
.ascii  /.IU/
.byte   cr
.ascii  /.AB/
.byte   cr
.ascii  /.PC/
.byte   cr
.ascii  /.SP/
.byte   cr
.byte   lf              ; end of list 5

; list 6 - modifiers to "display"

.ascii  /DATA/
.byte   cr
.ascii  /USED/
.byte   cr
.byte   lf              ; end of list 6

; list 7 - modifier to "load"

.ascii  /FROM/          ; source
```

```
        .byte   cr
        .byte   lf              ; end of list 7


        .page
        .sbttl  Get a line

        ;=====================================================
        ;
        ; this routine constructs a line of input by getting all
        ; input characters up to and including a carriage return
        ; (which then designates "end of line").
        ; typing rubout will delete the previous character
        ; typing control-c will abort the line
        ; typing control-z will use the previous line
        ; the input line is stored beginning at the address
        ; stored in bufbeg and ending at the address stored
        ; in bufend
        ; acca, accb, & ix are not preserved
        ;
        ; global variables
        ; bufbeg - input line start of buffer
        ; bufend - input line end of buffer
        ;
        ; local constants

getlin: ldx     bufbeg  ; set pointer to one less than
                        ; the beginning of the line buffer
        clrb            ; accb holds last input char
1$:     cpx     bufend  ; check current line end against buffer end
        bne     2$

        ; line too long - abort it as if a control-c had been typed

        ldx     #msgltl ; get message
        jsr     outstr  ; output it
        ldb     #3      ; put ctl-c in accb
        rts

2$:     jsr     inpchr  ; get a character (returned in acca)
        anda    #0x7f   ; drop parity bit

        ; control-z copies from present position to previous end of line

        cmpa    #26     ; is char a control-z?
        bne     3$
        jsr     docrlf  ; yes, type cr-lf
        rts

3$:     cmpa    #13     ; is char a cr?
        beq     4$
        cmpa    #10     ; or a lf?
        bne     6$
4$:     inx
        sta     ,x      ; yes, store the terminator
        tst     hdxflg  ; test for half-duplex terminal
        bne     5$
        jsr     docrlf  ; type cr-lf
5$:     rts             ; now return

6$:     cmpa    #3      ; is char a control-c?
        bne     7$      ; no
        tab             ; return ctl-c in accb
        lda     #'^     ; echo an up-arrow
        jsr     outchr
        rts

7$:     cmpa    #127    ; no, is it delete?
        beq     10$     ; yes - delete character
        inx             ; not a delete, so advance to next char
        sta     ,x      ; store it in inplin
        tab             ; last character in b
        tst     hdxflg  ; check half duplex
        bne     9$      ; yes - skip
        jsr     outchr  ; echo character
9$:     bra     1$      ; get another

        ; current character is a delete
```

```
        ; test line length - if its zero, ignore this delete
        ; since we can't delete prior to first char in input line

10$:    cpx     bufbeg
        beq     1$
        pshs    x
        ldx     #11$    ; delete character on screen
        bsr     outstr
        puls    x
        ldb     ,-x     ; last character
        bra     1$

11$:    .byte  8,32,8,4 ; BS, SPACE, BS, EOT


        ;=================================================
        ; initialization routine

inital: lda     #1
        sta     ibcode  ; set input base to hex
        sta     dbcode  ; set display base to hex

        ; set up display base number

        lda     #16
        sta     dbnbr

        ; max # of characters per line

        lda     #72
        sta     cplmax
        clr     inpflg  ; default input from terminal
        clr     outflg  ; default output to terminal
        clr     hdxflg  ; clear half-duplex flag

        ; set up swi interrupt address pointer

        ldx     #typswi ; type "swi" & do "reg" command
        stx     .swi

        ; clear breakpoint address

        ldd     #0xfffe ; normal systems have rom here
        std     brkadr

        ; initialize to mondeb's command lists

        ldx     #comlst-1
        stx     comadr

        ; time constant for a .5 microsecond clock

        ldd     #285
        std     timcon

        .if     inituser
        jmp     userinit        ; user returns via rts
        .else
        rts
        .endif

        ;=================================================
        ; output a character string which begins at the address
        ; in the index register
        ; acca & accb are preserved
        ; ix is left pointing to the string terminator

outstr: pshs    a
1$:     lda     ,x      ; get char pointed to
        cmpa    #4      ; is it a string terminator?
        beq     2$      ; done if it is
        bsr     outchr  ; isn't, output it
        inx             ; on to next char
        bra     1$
2$:     puls    a,pc

        ;=================================================
        ; input a character
```

```
inpchr: tst     inpflg  ; console ?
        bne     2$

1$:     jsr     [conin] ; tst for a character
        bcc     1$      ; none - loop until there is
        rts

2$:     jsr     [altin] ; tst for a character
        bcc     2$      ; none - loop until there is
        rts

        ;=====================================================
        ; output the character in acca to the desired output
        ; device/location
        ; if outflg = 0, output is to terminal
        ; if outflg # 0, output is to address in outadr
        ;    & this address is then incremented

outchr: pshs    d,x     ; save d and x
        ldb     outflg  ; test output destination flag
        decb
        ble     1$      ; skip this code if terminal output

        ; output to something other than terminal

        ldx     outadr  ; get output char destination addr
        sta     ,x+     ; save char in memory
        stx     outadr  ; update output address
        puls    d,x,pc

1$:     cmpa    #lf     ; ignore line feeds
        bne     2$
        puls    d,x,pc

2$:     cmpa    #cr     ; test for carriage return
        bne     3$
        bsr     docrlf
        puls    d,x,pc

3$:     ldb     cplcnt  ; get "chars per line" count
        cmpb    cplmax
        bge     4$      ; send cr-lf if greater

        ; less than max, but also send cr-lf if 10 from end and
        ; printing a space

        addb    #10
        cmpb    cplmax
        blt     5$
        cmpa    #'      ; near end, test if about to print a space
        bne     5$

        ; terminal line full or nearly full - interject a cr-lf

4$:     bsr     docrlf
5$:     inc     cplcnt  ; bump counter
        bsr     chrout  ; send it to acia1
        puls    d,x,pc

        ;=====================================================
        ; send a carriage return-line feed to the terminal

docrlf: pshs    d
        lda     #cr
        bsr     chrout
        lda     #lf
        bsr     chrout
        clr     cplcnt  ; zero "chars/line" count
        puls    d,pc

        ;=====================================================
        ; send char in acca

chrout: tst     outflg  ; check destination
        bne     1$
        jmp  [conout]   ; output a character
1$:     jmp  [altout]
```

```
        ;=====================================================
        ; misc text

msghed: .ascii  /MONDEB-09 1.00/
        .byte   cr,4
msgprm: .byte   '*,4
msgswi: .byte   cr,lf
        .ascii  /swi:/
        .byte   cr,lf,4
msgltl: .ascii  /too long/
        .byte   4
msgnbr: .ascii  /not set/
        .byte   4
msgbat: .ascii  /set @ /
        .byte   4
msgver: .ascii  /ok/
        .byte   4
msgnve: .ascii  /checksum error /
        .byte   4
msgccl: .ascii  /can't clear/
        .byte   4
msgcso: .ascii  /can't set to ones/
        .byte   4
msgsis: .ascii  /sum is /
        .byte   4
msgdis: .ascii  /, dif is /
        .byte   4
msgs1:  .byte   cr,lf,0,0,'S,'1,4
msgs9:  .byte   cr,lf,0
        .ascii  /S9030000FC/
        .byte   cr,lf,4
msgcnh: .ascii  /char not hex/
        .byte   cr,4


        ;*************************************************
        ;*            default interrupt transfers        *
        ;*************************************************

        .if     standalone
rsrvd:  jmp     [.rsrvd]        ; reserved vector
swi3:   jmp     [.swi3]         ; swi3 vector
swi2:   jmp     [.swi2]         ; swi2 vector
firq:   jmp     [.firq]         ; firq vector
irq:    jmp     [.irq]          ; irq vector
swi:    jmp     [.swi]          ; swi vector
nmi:    jmp     [.nmi]          ; nmi vector
        .endif

        .page
        .sbttl  dispatch table

        .if     standalone

        .area   DISPAT  (REL,CON)

        .word   timdel  ; time delay for # of ms specified by ix
        .word   cksum   ; return checksum of an address range in acca
        .word   mgetchr ; return (in accb) char pointed to by linptr
        .word   getlst  ; return (in acca) char pointed to by lisptr
        .word   gtrang  ; pick up an address range in ranglo & ranghi
        .word   mnumber ; pick up a number & return it in nbrhi & nbrlo
        .word   skpdlm  ; skip over input line delimiters
        .word   tstdlm  ; test char in accb for a delimiter
        .word   tsteol  ; test char in acca for end-of-line
        .word   comand  ; search specified command list for a command
        .word   typcmd  ; types out command number "comnum" in list acca
        .word   out1by  ; display the 1 byte number pointed at by ix
        .word   out2by  ; display the 2 byte number pointed at by ix
        .word   getlin  ; get a line of input into the tty buffer
        .word   outstr  ; output char string ix points to
        .word   docrlf  ; send cr-lf with delay & zero line count
        .word   outchr  ; like chrout, but with folding, cr delay, & lf
        .word   chrout  ; send acca to console
        .word   inpchr  ; get a char, return it in acca
        .word   prompt  ; to prompt for new command
        .word   mond09  ; start of mondeb
```

```
.endif

.page
.sbttl  Hardware Interrupt Tables

;*******************************************************
;*             MONDEB-09 hardware vector table
;*
;*  this table is used if the MONDEB-09 rom addresses
;*  the mc6809 hardware vectors.
;*******************************************************

.if     standalone

.area   INTVEC  (ABS,OVR)

. = 0xFFF0              ; vector position

.word   rsrvd           ; reserved slot
.word   swi3            ; software interrupt 3
.word   swi2            ; software interrupt 2
.word   firq            ; fast interrupt request
.word   irq             ; interrupt request
.word   swi             ; software interrupt
.word   nmi             ; non-maskable interrupt
.word   reset           ; restart
.endif
```

```
-xms
monopt
mond09
rel
-b RAM = 0
-b VARSAV = varsav
-b DPVSAV = dpvsav
-b BUFSAV = bufsav
-b VECTOR = chrom1
-b R6571A = chrom2
-b VT1XX  = chrom3
-b FIGROM = chrom4
-b PGMSAV = pgmsav
-b EXTSAV = extsav
-b IRQINT = irqint
-b OPTFUN = optfun
-b WORKPG = 0o17400
-b MONDEB = 0o20000
-b MONOPT = 0o30000
-e
```

```
        .title  Option File for Assist09

        .module astopt

        ;  The following labels and data allocations
        ;  are defined in DSPCGC.

        ;  This file should be the first file to be linked
        ;  to insure that the dsiplatch table is cleared.

        ; $xtrn0:       .blkb   4       ; 4096 bytes allocated
        ; $xtrn1:       .blkb   4       ; for optional functions
        ; $xtrn2:       .blkb   4
        ; $xtrn3:       .blkb   4
        ; $xtrn4:       .blkb   4
        ; $xtrn5:       .blkb   4
        ; $xtrn6:       .blkb   4
        ; $xtrn7:       .blkb   4
        ; $xtrn8:       .blkb   4
        ; $xtrn9:       .blkb   4

        ; $xtrnl:       .blkb   4       ; loadable printer driver $t_ entry
        ; $xscrn:       .blkb   4       ; loadable printer driver screen entry

        ; $xtend                        ; end of user area


        ;  System Vector Table as defined in DSPCGC.

        ; extbypls:     .blkb   2       ; loadable system vectors
        ; extbymns:     .blkb   2
        ; extbxpls:     .blkb   2
        ; extbxmns:     .blkb   2
        ; exundfnd:     .blkb   2
        ; exdltint:     .blkb   2
        ; exdlrint:     .blkb   2
        ; exclocki:     .blkb   2

        ; exrsrv:       .blkb   2
        ; exswi3:       .blkb   2
        ; exswi2:       .blkb   2
        ; exfirq:       .blkb   2
        ; exirq:        .blkb   2
        ; exswi:        .blkb   2
        ; exnmi:        .blkb   2

        .page
        .sbttl  Assist09 SWI Functions

        ;*******************************************************
        ;* assist09 monitor swi functions
        ;*
        ;*  the following equates define functions provided
        ;*  by the assist09 monitor via the swi instruction.
        ;*******************************************************

inchnp  =       0               ; input char in a reg - no parity
outch   =       1               ; output char from a reg
pdata1  =       2               ; output string
pdata   =       3               ; output cr/lf then string
out2hs  =       4               ; output two hex and space
out4hs  =       5               ; output four hex and space
pcrlf   =       6               ; output cr/lf
space   =       7               ; output a space
monitr  =       8               ; enter assist09 monitor
vctrsw  =       9               ; vector examine/switch
brkpt   =       10              ; user program breakpoint
pause   =       11              ; task pause function
numfun  =       11              ; number of available functions

        ;* sub-codes for accessing the vector table.
        ;* they are equivalent to offsets in the table.
        ;* relative positioning must be maintained.

.avtbl  =       0               ; address of vector table
.cmdl1  =       2               ; first command list
.rsvd   =       4               ; reserved hardware vector
.swi3   =       6               ; swi3 routine
```

```
.swi2   =       8               ; swi2 routine
.firq   =       10              ; firq routine
.irq    =       12              ; irq routine
.swi    =       14              ; swi routine
.nmi    =       16              ; nmi routine
.reset  =       18              ; reset routine
.cion   =       20              ; console on
.cidta  =       22              ; console input data
.cioff  =       24              ; console input off
.coon   =       26              ; console output on
.codta  =       28              ; console output data
.cooff  =       30              ; console output off
.hsdta  =       32              ; high speed printdata
.bson   =       34              ; punch/load on
.bsdta  =       36              ; punch/load data
.bsoff  =       38              ; punch/load off
.pause  =       40              ; task pause routine
.expan  =       42              ; expression analyzer
.cmdl2  =       44              ; second command list
.pad    =       46              ; character pad and new line pad
.echo   =       48              ; echo/load and null bkpt flag

        .page
        .sbttl  Option Dispatcher

        .area   OPTFUN  (ABS,OVR)

        .radix  o

        .blkb   4               ; $0
        .blkb   4               ; $1
        .blkb   4               ; $2
        .blkb   4               ; $3
        .blkb   4               ; $4
        .blkb   4               ; $5
        .word   .$6             ; $6 Command Entry
        .word   .$6-0x5AA5
        .word   .$7             ; $7
        .word   .$7-0x5AA5
        .word   .$8             ; $8
        .word   .$8-0x5AA5
        .blkb   4               ; $9

        .blkb   4               ; loadable printer driver $t_ entry
                                ; check code

        .blkb   4               ; loadable printer driver screen entry
                                ; check code


        .page
        .sbttl  Assist09 Option

        .area   ASTOPT  (ABS,OVR)

ast$id: .byte   15,12
        .ascii  *DSPCGC   V02.02*
        .byte   15,12
        .ascii  *Assist09 6809 Monitor - V01.00*
        .byte   15,12
        astl$id =       .-ast$id        ; length of version
        .ascii  *Copyright 1988*
        .byte   15,12
        .ascii  *Otselic Specialties*
        .byte   15,12
        .ascii  *721 Berkeley*
        .byte   15,12
        .ascii  *Kent, Ohio  44240*
        .byte   15,12
        astl$cr =       .-ast$id        ; length of notice


        .page
        .sbttl  $6 command

.$6:    jsr     nxtchr          ; get character

        jsr     $dispatch
```

```
        .byte   'V
        .word   2$              ; version select
        .byte   'C
        .word   3$              ; copyright select
        .byte   0

2$:     lda     #astl$id        ; ast$id character version #
        bra     4$

3$:     lda     #astl$cr        ; ast$cr copyright version #

4$:     sta     number
        ldx     #ast$id         ; version string
5$:     ldb     ,x+             ; get character
        stx     qt              ; save pointer
        jsr     plcbuf          ; send character
        ldx     qt              ; get pointer
        dec     number          ; more ?
        bne     5$              ; yes - loop
        bra     .$6

        .page
        .sbttl  Assist09 Startup Entry Points

.$7:    ldd     exswi           ; verify settup
        cmpd    #swi
        bne     .$8
        lda     #1
        swi                     ; reenter monitor
        .byte   monitr
        pshs    cc
        orcc    #120            ; inhibit interrupts
        lda     outpl           ; reset panel led's
        sta     plout
        puls    cc,pc           ; return to DSPCGC

.$8:    ldd     #swi            ; load swi vector
        std     exswi
        leas    astack,pcr      ; stack pointer
        jsr     bldvtr          ; build vector table
        ldd     #dspidta        ; console input
        std     .cidta,u        ; load vector table
        ldd     #dspodta        ; console output
        std     .codta,u        ; load vector table
        clra                    ; announce
        swi                     ; monitor fireup
        .byte   monitr          ; to enter command processing
        pshs    cc
        orcc    #120            ; inhibit interrupts
        lda     outpl           ; reset panel led's
        ora     #0o160          ; led's off
        sta     outpl
        sta     plout
        puls    cc
        jmp     main            ; restart DSPCGC

        .page
        .sbttl  DSPGGC I/O Drivers

        ;*******************************************************
        ;*      default DSPCGC i/o drivers
        ;*******************************************************

        ;* dspidta - return console input character
        ;* output: c=0 if no data ready, c=1 a=character
        ;*
        ;* DSPCGC panel leds are sequenced here

dspidta:
        tst     hrcsr           ; a character ?
        bmi     1$              ; yes - skip

        ;* LED scanner

        pshs    d
        ldd     workpg          ; led display update
        addd    #4
        std     workpg
```

```
        coma
        anda    #0o160          ; mask bits
        pshs    a
        lda     outpl           ; or with other control bits
        anda    #~0o160
        ora     ,s+
        sta     plout           ; load output register
        puls    d

        clc                     ; no character
        rts                     ; return to caller

1$:     lda     hrdata          ; get character
        cmpa    #0o141          ; translate lower to upper
        bcs     2$
        cmpa    #0o173
        bcc     2$
        suba    #0o40
2$:     sta     hrdata          ; character taken
        sec
        rts

        ;* dspodta - output character to console device
        ;* input: a=character to send
        ;* all registers transparent

dspodta:
        tst     htcsr           ; transmitter ready ?
        bpl     dspodta         ; no - loop until ready
        sta     htdata          ; send character
        rts
```

```
        .title  assist09 - mc6809 monitor

        .module assist09

        .radix  d

        ;*  Modification date:  November 23, 1988

        ;******************************************************
        ;*  miscelaneous        equates
        ;******************************************************

dftchp  =       0               ; default character pad count
dftnlp  =       0               ; default new line pad count
prompt  =       '>              ; prompt character
numbkp  =       8               ; number of breakpoints

eot     =       0x04            ; end of transmission
bell    =       0x07            ; bell character
lf      =       0x0a            ; line feed
cr      =       0x0d            ; carriage return
can     =       0x18            ; cancel (ctl-x)

        .page
        .sbttl  SWI Functions

        ;******************************************************
        ;* assist09 monitor swi functions
        ;*
        ;*  the following equates define functions provided
        ;*  by the assist09 monitor via the swi instruction.
        ;******************************************************

inchnp  =       0               ; input char in a reg - no parity
outch   =       1               ; output char from a reg
pdata1  =       2               ; output string
pdata   =       3               ; output cr/lf then string
out2hs  =       4               ; output two hex and space
out4hs  =       5               ; output four hex and space
pcrlf   =       6               ; output cr/lf
space   =       7               ; output a space
monitr  =       8               ; enter assist09 monitor
vctrsw  =       9               ; vector examine/switch
brkpt   =       10              ; user program breakpoint
pause   =       11              ; task pause function
numfun  =       11              ; number of available functions

        ;* sub-codes for accessing the vector table.
        ;* they are equivalent to offsets in the table.
        ;* relative positioning must be maintained.

.avtbl  =       0               ; address of vector table
.cmdl1  =       2               ; first command list
.rsvd   =       4               ; reserved hardware vector
.swi3   =       6               ; swi3 routine
.swi2   =       8               ; swi2 routine
.firq   =       10              ; firq routine
.irq    =       12              ; irq routine
.swi    =       14              ; swi routine
.nmi    =       16              ; nmi routine
.reset  =       18              ; reset routine
.cion   =       20              ; console on
.cidta  =       22              ; console input data
.cioff  =       24              ; console input off
.coon   =       26              ; console output on
.codta  =       28              ; console output data
.cooff  =       30              ; console output off
.hsdta  =       32              ; high speed printdata
.bson   =       34              ; punch/load on
.bsdta  =       36              ; punch/load data
.bsoff  =       38              ; punch/load off
.pause  =       40              ; task pause routine
.expan  =       42              ; expression analyzer
.cmdl2  =       44              ; second command list
.pad    =       46              ; character pad and new line pad
.echo   =       48              ; echo/load and null bkpt flag

numvtr  =       48/2+1          ; number of vectors
```

```
hivtr   =       48              ; highest vector offset

        .page
        .sbttl  Work Area

        ;*******************************************************
        ;* work area
        ;*
        ;*  The direct page register during most routine
        ;*  operations will point to this work area.  the Stack
        ;*  initially starts under the reserved work areas as
        ;*  defined herein.
        ;*******************************************************

        .area   WORKPG  (ABS,OVR)
        .setdp  0

workpg:                         ; beginning of work aera

.blkb   0d256-(endpg-astack)    ; stack space

astack:                         ; top of assist09 stack
tstack: .blkb   0d21            ; temporary stack hold
delim:  .blkb   1               ; expression delimiter/work byte
misflg: .blkb   1               ; load cmd/thru breakpoint flag
swicnt: .blkb   1               ; trace "swi" nest level count
pcnter: .blkb   2               ; last program counter
pstack: .blkb   2               ; command recovery stack
rstack: .blkb   2               ; reset stack pointer
anumber:.blkb   2               ; binary build area
basepg: .blkb   1               ; base page value
addr:   .blkb   2               ; address pointer value
window: .blkb   2               ; window
bkptop: .blkb   0x10            ; breakpoint opcode table
bkptbl: .blkb   0x10            ; breakpoint table
vectab: .blkb   0x32            ; vector table
bkptct: .blkb   1               ; breakpoint count
swibfl: .blkb   1               ; bypass swi as breakpoint flag
pauser: .blkb   4               ; pause routine
endpg:

        .page
        .sbttl  Assist09 Code

        .area   ASSIST09 (ABS,OVR)

        ;*******************************************************
        ;* bldvtr - build assist09 vector table
        ;*
        ;*  hardware reset calls this subroutine to build the
        ;*  assist09 vector table.
        ;*
        ;*  input: s->valid stack ram
        ;*  output: u->vector table address
        ;*          dpr->assist09 work area page
        ;*          the vector table and defaults are initialized
        ;*
        ;*  all registers volatile
        ;*******************************************************

bldvtr: leax    vectab,pcr      ; address vector table
        tfr     x,d             ; obtain base page address
        tfr     a,dp            ; setup dpr
        sta     *basepg         ; store for quick reference
        leau    ,x              ; return table to caller
        stu     ,x++            ; and init vector table address
        ldb     #numvtr-3       ; number relocatable vectors
        pshs    b               ; store index on stack
        leay    initvt,pcr      ; load from addr
1$:     tfr     y,d             ; prepare address resolve
        addd    ,y++            ; to absolute address
        std     ,x++            ; into vector table
        dec     ,s              ; count down
        bne     1$              ; branch if more to insert
        ldb     #intve-intvs    ; static value init length
2$:     lda     ,y+             ; load next byte
        sta     ,x+             ; store into position
        decb                    ; count down
```

```
        bne     2$              ; loop until done
        puls    pc,b            ; return to initializer

        ;*******************************************************
        ;* reset entry point
        ;*
        ;*  hardware reset enters here if assist09 is enabled
        ;*  to receive the mc6809 hardware vectors.  we call
        ;*  the bldvtr subroutine to initialize the vector
        ;*  table, stack, and then fireup the monitor via swi
        ;*  call.
        ;*******************************************************

reset:  leas    astack,pcr      ; setup initial stack
        bsr     bldvtr          ; build vector table
1$:     clra                    ; issue startup message
        tfr     a,dp            ; default to page zero
        swi                     ; perform monitor fireup
        .byte   monitr          ; to enter command processing
        bra     1$              ; reenter monitor if 'continue'

        .page
        .sbttl  Vector Table

        ;*******************************************************
        ;* initvt - initialize vector table
        ;*
        ;*  this table is relocated to ram and represents the
        ;*  initial state of the vector table. all addresses
        ;*  are converted to absolute form.  this table starts
        ;*  with the second entry, ends with static constant
        ;*  initialization data which carries beyond the table.
        ;*******************************************************

initvt: .word   cmdtb1-.        ; default first command table
        .word   rsrvdr-.        ; default undefined hardware vector
        .word   swi3r-.         ; default swi3
        .word   swi2r-.         ; default swi2
        .word   firqr-.         ; default firq
        .word   irqr-.          ; default irq routine
        .word   swir-.          ; default swi routine
        .word   nmir-.          ; default nmi routine
        .word   reset-.         ; restart vector
        .word   cion-.          ; default cion
        .word   cidta-.         ; default cidta
        .word   cioff-.         ; default cioff
        .word   coon-.          ; default coon
        .word   codta-.         ; default codta
        .word   cooff-.         ; default cooff
        .word   hsdta-.         ; default hsdta
        .word   bson-.          ; default bson
        .word   bsdta-.         ; default bsdta
        .word   bsoff-.         ; default bsoff
        .word   cpause-.        ; default pause routine
        .word   exp1-.          ; default expression analyzer
        .word   cmdtb2-.        ; default second command table

        ;* constants
        ;*
intvs:  .byte   dftchp,dftnlp   ; default null padds
        .word   0               ; default echo
        .byte   0               ; initial breakpoint count
        .byte   0               ; swi breakpoint level
        rts                     ; default pause routine
intve   =       .

        .page
        .sbttl  SWI Handler

        ;*******************************************************
        ;* assist09 swi handler
        ;*
        ;*  the swi handler provides all interfacing necessary
        ;*  for a user program.  a function byte is assumed to
        ;*  follow the swi instruction.  it is bound checked
        ;*  and the proper routine is given control.  this
        ;*  invocation may also be a breakpoint interrupt.
        ;*  if so, the breakpoint handler is entered.
```

```
        ;*
        ;* input: machine state defined for swi
        ;* output: varies according to function called. pc on
        ;*
        ;*     callers stack incremented by one if valid call.
        ;* volatile registers: see functions called
        ;*
        ;* state: runs disabled unless function clears i flag.
        ;*******************************************************

        ;* swi function vector table

swivtb: .word    zinch-swivtb   ; inchnp
        .word   zotch1-swivtb   ; outch
        .word   zpdta1-swivtb   ; pdata1
        .word   zpdata-swivtb   ; pdata
        .word   zot2hs-swivtb   ; out2hs
        .word   zot4hs-swivtb   ; out4hs
        .word   zpcrlf-swivtb   ; pcrlf
        .word   zspace-swivtb   ; space
        .word   zmontr-swivtb   ; monitr
        .word   zvswth-swivtb   ; vctrsw
        .word   zbkpnt-swivtb   ; breakpoint
        .word   zpause-swivtb   ; task pause

swir:   dec     swicnt,pcr      ; up "swi" level for trace
        lbsr    lddp            ; setup page and verify stack

        ;* check for breakpoint trap

        ldu     10,s            ; load program counter
        leau    -1,u            ; back to swi address
        tst     *swibfl         ; this "swi" breakpoint ?
        bne     2$              ; no - branch to let through
        lbsr    cbkldr          ; obtain breakpoint pointers
        negb                    ; obtain positive count
1$:     decb                    ; count down
        bmi     2$              ; branch when done
        cmpu    ,y++            ; ?  was this a breakpoint
        bne     1$              ; branch if not
        stu     10,s            ; set program counter back
        lbra    zbkpnt          ; go do breakpoint

2$:     clr     *swibfl         ; clear in case set
        pulu    d               ; obtain function byte, up pc
        cmpb    #numfun         ; ? too high
        lbhi    error           ; yes, do breakpoint
        stu     10,s            ; bump program counter past swi
        aslb                    ; function code times two
        leau    swivtb,pcr      ; obtain vector branch address
        ldd     b,u             ; load offset
        jmp     d,u             ; jump to routine

        .page
        .sbttl  Monitor Entry

        ;*******************************************************
        ;* registers to function routines:
        ;*  dp-> work area page
        ;*  d,y,u=unreliable           x=as called from user
        ;*  s=as from swi interrupt
        ;*******************************************************

        ;*******************************************************
        ;* [swi function 8]
        ;*  monitor entry
        ;*
        ;*  fireup the assist09 monitor.
        ;*  the stack with its values for the direct page
        ;*  register and condition code flags are used as is.
        ;*   1) initialize console i/o
        ;*   2) optionally print signon
        ;*   3) enter command processor
        ;*
        ;* input: a=0 init console and print startup message
        ;*        a#0 omit console init and startup message
        ;*******************************************************
```

```
signon: .ascii  /Assist09 -- 6809 Monitor/      ; signon eye-catcher
        .byte   eot

zmontr: sts     *rstack         ; save for bad stack recovery
        tst     1,s             ; ? init console and send msg
        bne     1$              ; branch if not
        jsr [vectab+.cion,pcr]  ; ready console input
        jsr [vectab+.coon,pcr]  ; ready console output
        leax    signon,pcr      ; ready signon eye-catcher
        swi                     ; perform
        .byte   pdata           ; print string
1$:                             ; fall through to cmd

        .page
        .sbttl  Command Processor

        ;*******************************************************
        ;* command handler
        ;*
        ;*  breakpoints are removed at this time.
        ;*  prompt for a command, and store all characters
        ;*  until a separator on the stack.
        ;*  search for first matching command subset,
        ;*  call it or give '?' response.
        ;*
        ;*  during command search:
        ;*      b=offset to next entry on x
        ;*      u=saved s
        ;*      u-1=entry size+2
        ;*      u-2=valid number flag (>=0 valid)/compare cnt
        ;*      u-3=carriage return flag (0=cr has been done)
        ;*      u-4=start of command store
        ;*      s+0=end of command store
        ;*******************************************************

        ;*******************************************************
        ;* commands are entered as a subroutine with:
        ;*    dpr->assist09 direct page work area
        ;*    z=1 carriage return entered
        ;*    z=0 non carriage return delimiter
        ;*    s=normal return address
        ;*
        ;*  the label "cmdbad" may be entered to issue an
        ;*  an error flag (?).
        ;*******************************************************

cmd:    swi                     ; to new line
        .byte   pcrlf           ; function

        ;* disarm the breakpoints

cmdnep: lbsr    cbkldr          ; obtain breakpoint pointers
        bpl     2$              ; branch if not armed or none
        negb                    ; make positive
        stb     *bkptct         ; flag as disarmed
1$:     decb                    ; ?  finished
        bmi     2$              ; branch if so
        lda     -numbkp*2,y     ; load opcode stored
        sta     [,y++]          ; store back over "swi"
        bra     1$              ; loop until done
2$:     ldx     10,s            ; load users program counter
        stx     *pcnter         ; save for expression analyzer
        lda     #prompt         ; load prompt character
        swi                     ; send to output handler
        .byte   outch           ; function
        leau    ,s              ; remember stack restore address
        stu     *pstack         ; remember stack for error use
        clra                    ; prepare zero
        clrb                    ; prepare zero
        std     *anumber        ; clear number build area
        std     *misflg         ; clear miscel. and swicnt flags
        ldb     #2              ; set d to two
        pshs    d,cc            ; place defaults onto stack

        ;* check for "quick" commands.

        lbsr    read            ; obtain first character
        leax    cmpadp+2,pcr    ; ready memory entry point
```

```
        cmpa    #'/             ; open last used memory ?
        beq     11$             ; yes - doit

        ;* process next character

3$:     cmpa    #'              ; ? blank or delimiter
        bls     5$              ; branch yes, we have it
        pshs    a               ; build onto stack
        inc     -1,u            ; count this character
        cmpa    #'/             ; ? memory command
        beq     12$             ; branch if so
        lbsr    bldhxc          ; treat as hex value
        beq     4$              ; branch if still valid number
        dec     -2,u            ; flag as invalid number
4$:     lbsr    read            ; obtain next character
        bra     3$              ; test next character

        ;* got command, now search tables

5$:     suba    #cr             ; set zero if carriage return
        sta     -3,u            ; setup flag
        ldx     *vectab+.cmdl1  ; start with first cmd list
6$:     ldb     ,x+             ; load entry length
        bpl     7$              ; branch if not list end
        ldx     *vectab+.cmdl2  ; now to second cmd list
        incb                    ; ? to continue to default list
        beq     6$              ; branch if so
cmdbad=.
        lds     *pstack         ; restore stack
        leax    errmsg,pcr      ; point to error string
        swi                     ; send out
        .byte   pdata1          ; to console
        bra     cmd             ; and try again

        ;* search next entry

7$:     decb                    ; take account of length byte
        cmpb    -1,u            ; ? entered longer than entry
        bhs     9$              ; branch if not too long
8$:     abx                     ; skip to next entry
        bra     6$              ; and try next
9$:     leay    -3,u            ; prepare to compare
        lda     -1,u            ; load size+2
        suba    #2              ; to actual size entered
        sta     -2,u            ; save size for countdown
10$:    decb                    ; down one byte
        lda     ,x+             ; next command character
        cmpa    ,-y             ; ? same as that entered
        bne     8$              ; branch to flush  if not
        dec     -2,u            ; count down length of entry
        bne     10$             ; branch if more to test
        abx                     ; to next entry
        ldd     -2,x            ; load offset
        leax    d,x             ; compute routine address+2
11$:    tst     -3,u            ; set cc for carriage return test
        leas    ,u              ; delete stack work area
        jsr     -2,x            ; call command
        lbra    2$              ; go get next command
12$:    tst     -2,u            ; ? valid hex number entered
        bmi     cmdbad          ; branch error if not
        leax    cmemn-cmpadp,x  ; to different entry
        ldd     *anumber        ; load number entered
        bra     11$             ; and enter memory command

        .page
        .sbttl  assist09 Command Tables

        ;******************************************************
        ;* assist09 command tables
        ;*
        ;*  these are the default command tables.  external
        ;*  tables of the same format may extend/replace
        ;*  these by using the vector swap function.
        ;*
        ;* entry format:
        ;*     +0...total size of entry (including this byte)
        ;*     +1...command string
        ;*     +n...two byte offset to command (entryaddr-.)
```

```
        ;*
        ;*  the tables terminate with a one byte -1 or -2.
        ;*  the -1 continues the command search with the
        ;*         second command table.
        ;*  the -2 terminates command searches.
        ;*****************************************************

        ;* this is the default list for the second command
        ;* list entry.

cmdtb2: .byte   -2              ; stop command searches

        ;* this is the default list for the first command
        ;* list entry.

cmdtb1:                         ; monitor command table
        .byte   4
        .ascii  /B/             ; 'breakpoint' command
        .word   cbkpt-.
        .byte   4
        .ascii  /C/             ; 'call' command
        .word   ccall-.
        .byte   4
        .ascii  /D/             ; 'display' command
        .word   cdi-.
        .byte   4
        .ascii  /E/             ; 'encode' command
        .word   cencde-.
        .byte   4
        .ascii  /G/             ; 'go' command
        .word   cgo-.
        .byte   4
        .ascii  /L/             ; 'load' command
        .word   cload-.
        .byte   4
        .ascii  /M/             ; 'memory' command
        .word   cmem-.
        .byte   4
        .ascii  /N/             ; 'nulls' command
        .word   cnulls-.
        .byte   4
        .ascii  /O/             ; 'offset' command
        .word   coffs-.
        .byte   4
        .ascii  /P/             ; 'punch' command
        .word   cpunch-.
        .byte   4
        .ascii  /R/             ; 'registers' command
        .word   creg-.
        .byte   4
        .ascii  /V/             ; 'verify' command
        .word   cver-.
        .byte   4
        .ascii  /W/             ; 'window' command
        .word   cwindo-.
        .byte   -1              ; end, continue with the second

        .page
        .sbttl  SWI Functions

        ;*****************************************************
        ;* [swi functions 4 and 5]
        ;*
        ;*  4 - out2hs - decode byte to hex and add space
        ;*  5 - out4hs - decode word to hex and add space
        ;*
        ;*  input: x->byte or word to decode
        ;*  output: characters sent to output handler
        ;*         x->next byte or word
        ;*****************************************************

zout2h: lda     ,x+             ; load next byte
        pshs    d               ; save - do not reread
        ldb     #16             ; shift by 4 bits
        mul                     ; with multiply
        bsr     zouthx          ; send out as hex
        puls    d               ; restore bytes
        anda    #0x0f           ; isolate right hex
```

```
zouthx: adda    #0x90           ; prepare a-f adjust
        daa                     ; adjust
        adca    #0x40           ; prepare character bits
        daa                     ; adjust
send:   jmp [vectab+.codta,pcr] ; send to out handler

zot4hs: bsr     zout2h          ; convert first byte
zot2hs: bsr     zout2h          ; convert byte to hex
        stx     4,s             ; update users x register

        ;* fall into space routine

        ;*******************************************************
        ;* [swi function 7]
        ;*  ace - send blank to output handler
        ;*
        ;*  input: none
        ;*  output: blank send to console handler
        ;*******************************************************

zspace: lda     #'              ; load blank
        bra     zotch2          ; send and return

        ;*******************************************************
        ;* [swi function 9]
        ;*  swap vector table entry
        ;*
        ;*  input:  a=vector table code (offset)
        ;*          x=0 or replacement value
        ;*  output: x=previous value
        ;*******************************************************

zvswth: lda     1,s             ; load requesters a
        cmpa    #hivtr          ; ? sub-code too high
        bhi     zotch3          ; ignore call if so
        ldy     *vectab+.avtbl  ; load vector table address
        ldu     a,y             ; u=old entry
        stu     4,s             ; return old value to callers x
        stx     -2,s            ; ? x=0
        beq     zotch3          ; yes, do not change entry
        stx     a,y             ; replace entry
        bra     zotch3          ; return from swi

        .page

        ;*******************************************************
        ;* [swi function 0]
        ;*  inchnp - obtain input char in a (no parity)
        ;*
        ;*  nulls and rubouts are ignored.
        ;*  automatic line feed is sent upon recieving a
        ;*      carriage return.
        ;*  unless we are loading from tape.
        ;*******************************************************

zinchp: bsr     xqpaus          ; release processor
zinch:  bsr     xqcidt          ; call input data appendage
        bcc     zinchp          ; loop if none available
        tsta                    ; test for null
        beq     zinch           ; ignore null
        cmpa    #0x7f           ; ? rubout
        beq     zinch           ; branch yes to ignore
        sta     1,s             ; store into callers a
        tst     *misflg         ; ? load in progress
        bne     zotch3          ; branch if so to not echo
        cmpa    #cr             ; ? carriage return
        bne     1$              ; no, test echo byte
        lda     #lf             ; load line feed
        bsr     send            ; always echo line feed
1$:     tst     *vectab+.echo   ; ? echo desired
        bne     zotch3          ; no, return

        ;* fall through to outch

        ;*******************************************************
        ;* [swi function 1]
        ;*  outch - output character from a
        ;*
```

```
        ;*  input:  none
        ;*  output: if linefeed is the output character then
        ;*           c=0 no ctl-x recieved, c=1 ctl-x recieved
        ;*******************************************************

zotch1: lda     1,s             ; load character to send
        leax    zpcrls,pcr      ; default for line feed
        cmpa    #lf             ; ? line feed
        beq     zpdtlp          ; branch to check pause if so
zotch2: bsr     send            ; send to output routine
zotch3: inc     *swicnt         ; bump up "swi" trace nest level
        rti                     ; return from "swi" function

        ;*******************************************************
        ;* [swi function 6]
        ;*  pcrlf - send cr/lf to console handler
        ;*
        ;*  input: none
        ;*  output: cr and lf sent to handler
        ;*          c=0 no ctl-x, c=1 ctl-x recieved
        ;*******************************************************

zpcrls: .byte   eot             ; null string

zpcrlf: leax    zpcrls,pcr      ; ready cr,lf string

        ;* fall into cr/lf code

        ;*******************************************************
        ;* [swi function 3]
        ;*  pdata - output cr/lf and string
        ;*
        ;*  input: x->string
        ;*  output: cr/lf and string sent to output console
        ;*         handler.
        ;*     c=0 no ctl-x, c=1 ctl-x recieved
        ;*
        ;*  note: line feed must follow carriage return for
        ;*       proper punch data.
        ;*******************************************************

zpdata: lda     #cr             ; load carriage return
        bsr     send            ; send it
        lda     #lf             ; load line feed

        ;* fall into    pdata1

        ;*******************************************************
        ;* [swi function 2]
        ;*  pdata1 - output string till eot (0x04)
        ;*
        ;*  this routine pauses if an input byte becomes
        ;*  available during output transmission until a
        ;*  second is recieved.
        ;*
        ;*  input: x->string
        ;*  output: string sent to output console driver
        ;*          c=0 no ctl-x, c=1 ctl-x recieved
        ;*******************************************************

zpdtlp: bsr     send            ; send character to driver
zpdta1: lda     ,x+             ; load next character
        cmpa    #eot            ; ? eot
        bne     zpdtlp          ; loop if not

        ;* fall into pause check function

        ;*******************************************************
        ;* [swi function 12]
        ;*  pause - return to task dispatching and check
        ;*
        ;*  for freeze condition or ctl-x break
        ;*  this function enters the task pause handler so
        ;*  optionally other 6809 processes may gain control.
        ;*  upon return, check for a 'freeze' condition
        ;*  with a resulting wait loop, or condition code
        ;*  return if a control-x is entered from the input
        ;*  handler.
```

```
        ;*
        ;*  output: c=1 if ctl-x has entered, c=0 otherwise
        ;*****************************************************

zpause: bsr     xqpaus          ; release control at every line
        bsr     chkabt          ; check for freeze or abort
        tfr     cc,b            ; prepare to replace cc
        stb     ,s              ; overlay old one on stack
        bra     zotch3          ; return from "swi"

        ;* chkabt - scan for input pause/abort during output
        ;* output: c=0 ok, c=1 abort (ctl-x issued)
        ;* volatile: u,x,d

chkabt: bsr     xqcidt          ; attempt input
        bcc     2$              ; branch no to return
        cmpa    #can            ; ? ctl-x for abort
        bne     3$              ; branch no to pause
1$:     comb                    ; set carry
2$:     rts                     ; return to caller with cc set

3$:     bsr     xqpaus          ; pause for a moment
        bsr     xqcidt          ; ? key for start
        bcc     3$              ; loop until recieved
        cmpa    #can            ; ? abort signaled from wait
        beq     1$              ; branch yes
        clra                    ; set c=0 for no abort
        rts                     ; and return

        ;* save memory with jumps

xqpaus: jmp [vectab+.pause,pcr] ; to pause routine
xqcidt: jsr [vectab+.cidta,pcr] ; to input routine
        anda    #0x7f           ; strip parity
        rts                     ; return to caller


        ;* lddp - setup direct page register, verify stack.
        ;* an invalid stack causes a return to the command
        ;* handler.
        ;* input: fully stacked registers from an interrupt
        ;* output: dpr loaded to work page

errmsg: .byte   '?,bell,0x20,eot ; error response

ldrtn:  rts
lddp:   ldb     basepg,pcr      ; load direct page high byte
        tfr     b,dp            ; setup direct page register
        cmpa    3,s             ; ? is stack valid
        beq     ldrtn           ; yes, return
        lds     *rstack         ; reset to initial stack pointer
error:  leax    errmsg,pcr      ; load error report
        swi                     ; send out before registers
        .byte   pdata           ; on next line

        ;* fall into breakpoint handler

        ;*****************************************************
        ;* [swi function 10]
        ;*  breakpoint program function
        ;*
        ;*  print registers and go to command handler
        ;*****************************************************

zbkpnt: bsr     zbkstk          ; stack an extra word
zbkcmd: lbra    cmdnep          ; now enter command handler
zbkstk: lbsr    regprt          ; print out registers
        rts

        ;*****************************************************
        ;*    irq, reserved, swi2 and swi3 interrupt handlers
        ;*  the default handling is to cause a breakpoint.
        ;*****************************************************

swi2r:                          ; swi2 entry
swi3r:                          ; swi3 entry
irqr:                           ; irq entry
nmir:                           ; nmi entry
```

```
rsrvdr: bsr     lddp            ; set base page, validate stack
        bra     zbkpnt          ; force a breakpoint

        ;*******************************************************
        ;*        firq handler
        ;*  just return for the firq interrupt
        ;*******************************************************

firqr:          rti             ; immediate return

        .page
        .sbttl  Read / Verify / Punch Routines

        ;* bson - turn on read/verify/punch mechanism

bson:   inc     *misflg         ; set load in progress flag
        rts                     ; return to caller

        ;* bsoff - turn off read/verify/punch mechanism
        ;* a,x volatile

bsoff:  dec     *misflg         ; clear load in progress flag
        rts                     ; return to caller

        ;* bsdta - read/verify/punch handler
        ;* input: s+6=code byte, verify(-1),punch(0),load(1)
        ;*        s+4=start address
        ;*        s+2=stop address
        ;*        s+0=return address
        ;* output: z=1 normal completion, z=0 invalid load/ver
        ;* registers are volatile

bsdta:  ldu     2,s             ; u=to address or offset
        tst     6,s             ; ? punch
        beq     10$             ; branch yes

        ;* during read/verify: s+2=msb address save byte
        ;*                     s+1=byte counter
        ;*                     s+0=checksum
        ;*                     u holds offset

        leas    -3,s            ; room for work/counter/checksum
1$:     swi                     ; get next character
        .byte   inchnp          ; function
2$:     cmpa    #'S             ; ? start of s1/s9
        bne     1$              ; branch not
        swi                     ; get next character
        .byte   inchnp          ; function
        cmpa    #'9             ; ? have s9
        beq     5$              ; yes, return good code
        cmpa    #'1             ; ? have new record
        bne     2$              ; branch if not
        clr     ,s              ; clear checksum
        bsr     9$              ; obtain byte count
        stb     1,s             ; save for decrement

        ;* read address

        bsr     9$              ; obtain high value
        stb     2,s             ; save it
        bsr     9$              ; obtain low value
        lda     2,s             ; make d=value
        leay    d,u             ; y=address+offset

        ;* store text

3$:     bsr     9$              ; next byte
        beq     6$              ; branch if checksum
        tst     9,s             ; ? verify only
        bmi     4$              ; yes, only compare
        stb     ,y              ; store into memory
4$:     cmpb    ,y+             ; ? valid ram
        beq     3$              ; yes, continue reading
5$:     puls    pc,x,a          ; return with z set proper

6$:     inca                    ; ? valid checksum
        beq     1$              ; branch yes
        bra     5$              ; return z=0 invalid
```

```
        ;* byte builds 8 bit value from two hex digits in

7$:     bsr     9$              ; obtain first hex
        ldb     #16             ; prepare shift
        mul                     ; over to a
        bsr     9$              ; obtain second hex
        pshs    b               ; save high hex
        adda    ,s+             ; combine both sides
        tfr     a,b             ; send back in b
        adda    2,s             ; compute new checksum
        sta     2,s             ; store back
        dec     3,s             ; decrement byte count
8$:     rts                     ; return to caller

9$:     swi                     ; get next hex
        .byte   inchnp          ; character
        lbsr    cnvhex          ; convert to hex
        beq     8$              ; return if valid hex
        puls    pc,u,y,x,a      ; return to caller with z=0

        ;* punch stack use: s+8=to address
        ;*                  s+6=return address
        ;*                  s+4=saved padding values
        ;*                  s+2 from address
        ;*                  s+1=frame count/checksum
        ;*                  s+0=byte count

10$:    ldu     *vectab+.pad    ; load padding values
        ldx     4,s             ; x=from address
        pshs    u,x,d           ; create stack work area
        ldd     #24             ; set a=0, b=24
        stb     *vectab+.pad    ; setup 24 character pads
        swi                     ; send nulls out
        .byte   outch           ; function
        ldb     #4              ; setup new line pad to 4
        std     *vectab+.pad    ; setup punch padding

        ;* calculate size

11$:    ldd     8,s             ; load to
        subd    2,s             ; minus from=length
        cmpd    #24             ; ? more than 23
        blo     12$             ; no, ok
        ldb     #23             ; force to 23 max
12$:    incb                    ; prepare counter
        stb     ,s              ; store byte count
        addb    #3              ; adjust to frame count
        stb     1,s             ; save

        ;*punch cr,lf,nuls,s,1

        leax    16$,pcr         ; load start record header
        swi                     ; send out
        .byte   pdata           ; function

        ;* send frame count

        clrb                    ; initialize checksum
        leax    1,s             ; point to frame count and addr
        bsr     14$             ; send frame count

        ;*data address

        bsr     14$             ; send address hi
        bsr     14$             ; send address low

        ;*punch data

        ldx     2,s             ; load start data address
13$:    bsr     14$             ; send out next byte
        dec     ,s              ; ? final byte
        bne     13$             ; loop if not done
        stx     2,s             ; update from address value

        ;*punch checksum

        comb                    ; complement
```

```
        stb     1,s             ; store for sendout
        leax    1,s             ; point to it
        bsr     15$             ; send out as hex
        ldx     8,s             ; load top address
        cmpx    2,s             ; ? done
        bhs     11$             ; branch not
        leax    17$,pcr         ; prepare end of file
        swi                     ; send out string
        .byte   pdata           ; function
        ldd     4,s             ; recover pad counts
        std     *vectab+.pad    ; restore
        clra                    ; set z=1 for ok return
        puls    pc,u,x,d        ; return with ok code

14$:    addb    ,x              ; add to checksum
15$:    lbra    zout2h          ; send out as hex and return

16$:    .byte   'S,'1,eot       ; cr,lf,nulls,S,1
17$:    .ascii  /S9030000FC/    ; eof string
        .byte   cr,lf,eot

        ;* hsdta - high  speed print memory
        ;* input: s+4=start address
        ;*        s+2=stop address
        ;*        s+0=return address
        ;* x,d volatile
        ;*  send title

hsdta:  swi                     ; send new line
        .byte   pcrlf           ; function
        ldb     #6              ; prepare 6 spaces
1$:     swi                     ; send blank
        .byte   space           ; function
        decb                    ; count down
        bne     1$              ; loop if more
        clrb                    ; setup byte count
2$:     tfr     b,a             ; prepare for convert
        lbsr    zouthx          ; convert to a hex digit
        swi                     ; send blank
        .byte   space           ; function
        swi                     ; send another
        .byte   space           ; blank
        incb                    ; up another
        cmpb    #0x10           ; ? past 'f'
        blo     2$              ; loop until so
3$:     swi                     ; to next line
        .byte   pcrlf           ; function
        bcs     8$              ; return if user entered ctl-x
        leax    4,s             ; point at address to convert
        swi                     ; print out address
        .byte   out4hs          ; function
        ldx     4,s             ; load address proper
        ldb     #16             ; next sixteen
4$:     swi                     ; convert byte to hex and send
        .byte   out2hs          ; function
        decb                    ; count down
        bne     4$              ; loop if not sixteenth
        swi                     ; send blank
        .byte   space           ; function
        ldx     4,s             ; reload from address
        ldb     #16             ; count
5$:     lda     ,x+             ; next byte
        bmi     6$              ; too large, to a dot
        cmpa    #'              ; ? lower than a blank
        bhs     7$              ; no, branch ok
6$:     lda     #'.             ; convert invalid to a blank
7$:     swi                     ; send character
        .byte   outch           ; function
        decb                    ; ? done
        bne     5$              ; branch no
        cpx     2,s             ; ? past last address
        bhs     8$              ; quit if so
        stx     4,s             ; update from address
        lda     5,s             ; load low byte address
        asla                    ; ? to section boundry
        bne     3$              ; branch if not
        bra     hsdta           ; branch if so
8$:     swi                     ; send new line
```

```
        .byte   pcrlf           ; function
        rts                     ; return to caller

        ;*******************************************************
        ;*     a s s i s t 0 9    c o m m a n d s
        ;*******************************************************

        ;**********   registers - display and change registers

creg:   bsr     regprt          ; print registers
        inca                    ; set for change function
        bsr     regchg          ; go change, display registers
        rts                     ; return to command processor

        ;*******************************************************
        ;*      regprt - print/change registers subroutine
        ;*  will abort to 'cmdbad' if overflow detected during
        ;*  a change operation.  change displays registers when
        ;*  done.
        ;*
        ;* register mask list consists of:
        ;*  a) characters denoting register
        ;*  b) zero for one byte, -1 for two
        ;*  c) offset on stack to register position
        ;*
        ;* input:    +4=stacked registers
        ;*        a=0 print, a#0 print and change
        ;* output: (only for register display)
        ;*         c=1 control-x entered, c=0 otherwise
        ;*
        ;* volatile: d,x (change)
        ;*           b,x (display)
        ;*******************************************************

regmsk: .byte   'P,'C,-1,19     ; pc reg
        .byte   'A,0,10         ; a reg
        .byte   'B,0,11         ; b reg
        .byte   'X,-1,13        ; x reg
        .byte   'Y,-1,15        ; y reg
        .byte   'U,-1,17        ; u reg
        .byte   'S,-1,1         ; s reg
        .byte   'C,'c,0,9       ; cc reg
        .byte   'D,'p,0,12      ; dp reg
        .byte   0               ; end of list

regprt: clra                    ; setup print only flag
regchg: leax    4+12,s          ; ready stack value
        pshs    y,x,a           ; save on stack with option
        leay    regmsk,pcr      ; load register mask
1$:     ldd     ,y+             ; load next char or <=0
        tsta                    ; ? end of characters
        ble     2$              ; branch not character
        swi                     ; send to console
        .byte   outch           ; function byte
        bra     1$              ; check next
2$:     lda     #'-             ;  ready '-'
        swi                     ; send out
        .byte   outch           ; with outch
        leax    b,s             ; x->register to print
        tst     ,s              ; ? change option
        bne     5$              ; branch yes
        tst     -1,y            ; ? one or two bytes
        beq     3$              ; branch zero means one
        swi                     ; perform word hex
        .byte   out4hs          ; function
        bra     4$

3$:     swi                     ; perform byte hex
        .byte   out2hs          ; function
4$:     ldd     ,y+             ; to front of next entry
        tstb                    ; ? end of entries
        bne     1$              ; loop if more
        swi                     ; force new line
        .byte   pcrlf           ; function
        puls    pc,y,x,a        ; restore stack and return

5$:     bsr     bldnnb          ; input binary number
        beq     7$              ; if change then jump
```

```
        cmpa    #cr             ; ? no more desired
        beq     9$              ; branch nope
        ldb     -1,y            ; load size flag
        decb                    ; minus one
        negb                    ; make positive
        aslb                    ; times two (=2 or =4)
6$:     swi                     ; perform spaces
        .byte   space           ; function
        decb
        bne     6$              ; loop if more
        bra     4$              ; continue with next register
7$:     sta     ,s              ; save delimiter in option

        ;*                     (always > 0)

        ldd     *anumber        ; obtain binary result
        tst     -1,y            ; ? two bytes worth
        bne     8$              ; branch yes
        lda     ,-x             ; setup for two
8$:     std     ,x              ; store in new value
        lda     ,s              ; recover delimiter
        cmpa    #cr             ; ? end of changes
        bne     4$              ; no, keep on truck'n

        ;* move stacked data to new stack in case stack
        ;* pointer has changed

9$:     leax    tstack,pcr      ; load temp area
        ldb     #21             ; load count
10$:    puls    a               ; next byte
        sta     ,x+             ; store into temp
        decb                    ; count down
        bne     10$             ; loop if more
        lds     -20,x           ; load new stack pointer
        ldb     #21             ; load count again
11$:    lda     ,-x             ; next to store
        pshs    a               ; back onto new stack
        decb                    ; count down
        bne     11$             ; loop if more
        puls    pc,y,x,a        ; restore stack and return

        ;*******************************************************
        ;*  bldnum - builds binary value from input hex
        ;*  the active expression handler is used.
        ;*
        ;* input: s=return address
        ;* output: a=delimiter which terminated value
        ;*                            (if delm not zero)
        ;*         "number"=word binary result
        ;*         z=1 if input recieved, z=0 if no hex recieved
        ;*
        ;*  registers are transparent
        ;*******************************************************

        ;* execute single or extended rom expression handler
        ;*
        ;* the flag "delim" is used as follows:
        ;*   delim=0  no leading blanks, no forced terminator
        ;*   delim=chr  accept leading 'chr's, forced terminator

bldnnb: clra                    ; no dynamic delimiter
        sta     *delim          ; store as delimiter
        jmp [vectab+.expan,pcr] ; to exp analyzer

        ;* build with leading blanks

bldnum: lda     #'              ; allow leading blanks
        sta     *delim          ; store as delimiter
        jmp [vectab+.expan,pcr] ; to exp analyzer

        ;* this is the default single rom analyzer. we accept:
        ;*    1) hex input
        ;*    2) 'M' for last memory examine address
        ;*    3) 'P' for program counter address
        ;*    4) 'W' for window value
        ;*    5) '@' for indirect value

exp1:   pshs    x,b             ; save registers
```

```
1$:     bsr     bldhxi          ; clear number, check first char
        beq     3$              ; if hex digit continue building

        ;* skip blanks if desired

        cmpa    *delim          ; ? correct delimiter
        beq     1$              ; yes, ignore it

        ;* test for m or p

        ldx     *addr           ; default for 'm'
        cmpa    #'M             ; ? memory examine addr wanted
        beq     5$              ; branch if so
        ldx     *pcnter         ; default for 'p'
        cmpa    #'P             ; ? last program counter wanted
        beq     5$              ; branch if so
        ldx     *window         ; default to window
        cmpa    #'W             ; ? window wanted
        beq     5$
2$:     puls    pc,x,b          ; return and restore registers

        ;* got hex, now continue building

3$:     bsr     bldhex          ; compute next digit
        beq     3$              ; continue if more
        bra     6$              ; search for +/-

        ;* store value and check if need delimiter

4$:     ldx     ,x              ; indirection desired
5$:     stx     *anumber        ; store result
        tst     *delim          ; ? to force a delimiter
        beq     2$              ; return if not with value
        bsr     read            ; obtain next character

        ;* test for + or -

6$:     ldx     *anumber        ; load last value
        cmpa    #'+             ; ? add operator
        bne     8$              ; branch not
        bsr     10$             ; compute next term
        pshs    a               ; save delimiter
        ldd     *anumber        ; load new term
7$:     leax    d,x             ; add to x
        stx     *anumber        ; store as new result
        puls    a               ; restore delimiter
        bra     6$              ; now test it
8$:     cmpa    #'-             ; ? subtract operator
        beq     9$              ; branch if so
        cmpa    #'@             ; ? indirection desired
        beq     4$              ; branch if so
        clrb                    ; set delimiter return
        bra     2$              ; and return to caller
9$:     bsr     10$             ; obtain next term
        pshs    a               ; save delimiter
        ldd     *anumber        ; load up next term
        nega                    ; negate a
        negb                    ; negate b
        sbca    #0              ; correct for a
        bra     7$              ; go add to expresion

        ;* compute next expression term
        ;* output: x=old value
        ;*         'number'=next term

10$:    bsr     bldnum          ; obtain next value
        lbne    cmdbad          ; abort command if invalid
        rts                     ; return if valid number

        ;*******************************************************
        ;*  build binary value using input characters.
        ;*
        ;* input: a=ascii hex value or delimiter
        ;*           +0=return address
        ;*           +2=16 bit result area
        ;* output: z=1 a=binary value
        ;*         z=0 if invalid hex character (a unchanged)
        ;*
```

```
        ;* volatile: d
        ;*****************************************************

bldhxi: clr     *anumber        ; clear number
        clr     *anumber+1      ; clear number
bldhex: bsr     read            ; get input character
bldhxc: bsr     cnvhex          ; convert and test character
        bne     cnvrts          ; return if not a number
        ldb     #16             ; prepare shift
        mul                     ; by four places
        lda     #4              ; rotate binary into value
1$:     aslb                    ; obtain next bit
        rol     *anumber+1      ; into low byte
        rol     *anumber        ; into hi byte
        deca                    ; count down
        bne     1$              ; branch if more to do
        bra     cnvok           ; set good return code

        ;*****************************************************
        ;* convert ascii character to binary byte
        ;*
        ;* input: a=ascii
        ;* output: z=1 a=binary value
        ;*         z=0 if invalid
        ;*
        ;* all registers transparent
        ;* (a unaltered if invalid hex)
        ;*****************************************************

cnvhex: cmpa    #'0             ; ? lower tigh hex
        blo     cnvrts          ; branch not value
        cmpa    #'9             ; ? possible a-f
        ble     cnvgot          ; branch no to accept
        cmpa    #'A             ; ? less than ten
        blo     cnvrts          ; return if minus (invalid)
        cmpa    #'F             ; ? not too large
        bhi     cnvrts          ; no, return too large
        suba    #7              ; down to binary
cnvgot: anda    #0x0f           ; clear high byte
cnvok:  orcc    #4              ; force zero on for valid hex
cnvrts: rts                     ; return to caller

        ;* get input char, abort command if control-x (cancel)

read:   swi                     ; get next character
        .byte   inchnp          ; function
        cmpa    #can            ; ? abort command
        lbeq    cmdbad          ; branch to abort if so
        rts                     ; return to caller


        ;************   console - dumby routines

cidta:  clc                     ; never a character
codta:                          ; dumby character out
cion:                           ; input console initialization
coon:                           ; output console initialization
cioff:                          ; console input off
cooff:                          ; console output off
cirtn:  rts

        ;************   pause - process pause routine

cpause: jmp     pauser          ; go to default pause routine


        ;************   go - start program execution

cgo:    bsr     goaddr          ; build address if needed
        rti                     ; start executing

        ;* find optional new program counter. also arm the
        ;* breakpoints.

goaddr: puls    y,x             ; pop return addresses from cmd and cgo
        pshs    x               ; restore return from cgo
        beq     1$              ; <cr> ? yes - use current pc
```

```
        ;* obtain new program counter

        lbsr    cdnum           ; obtain new program counter
        std     12,s            ; store into stack

1$:     ldx     12,s            ; load program counter
        lbsr    cbkldr          ; obtain table
        neg     *bkptct         ; complement to show armed
2$:     decb                    ; ? done
        bmi     5$              ; return when done
        lda     [,y]            ; load opcode
        sta     -numbkp*2,y     ; store into opcode table
        lda     #0x3f           ; ready "swi" opcode
        cmpx    ,y              ; starting at a breakpoint ?
        bne     4$              ; no - go set breakpoint

        cmpa    [,y++]          ; ? swi breakpointed
        bne     2$              ; no, skip setting of flag
        sta     *swibfl         ; show upcomming swi not brkpnt
        bra     2$              ; check others

4$:     sta     [,y++]          ; store and move up table
        bra     2$              ; and continue

5$:     rts

        ;************   call - call address as subroutine

ccall:  bsr     goaddr          ; fetch address if needed
        puls    u,y,x,dp,d,cc   ; restore users registers
        jsr     [,s++]          ; call user subroutine
1$:     swi                     ; perform breakpoint
        .byte   brkpt           ; function
        bra     1$              ; loop until user changes pc

        ;************   memory - display/change memory
        ;*      cmem and cmpadp are direct entry points from
        ;*      the command handler for quick commands

cmem:   lbsr    cdnum           ; obtain address
cmemn:  std     *addr           ; store default
1$:     ldx     *addr           ; load pointer
        lbsr    zout2h          ; send out hex value of byte
        lda     #'-             ; load delimiter
        swi                     ; send out
        .byte   outch           ; function
2$:     lbsr    bldnnb          ; obtain new byte value
        beq     3$              ; branch if number

        ;* coma - skip byte

        cmpa    #',             ; ? comma
        bne     4$              ; branch not
        stx     *addr           ; update pointer
        leax    1,x             ; to next byte
        bra     2$              ; and input it
3$:     ldb     *anumber+1      ; load low byte value
        bsr     13$             ; go overlay memory byte
        cmpa    #',             ; ? continue with no display
        beq     2$              ; branch yes

        ;* quoted string

4$:     cmpa    #''             ; ? quoted string
        bne     6$              ; branch no
5$:     bsr     read            ; obtain next character
        cmpa    #''             ; ? end of quoted string
        beq     7$              ; yes, quit string mode
        tfr     a,b             ; to b for subroutine
        bsr     13$             ; go update byte
        bra     5$              ; get next character

        ;* blank - next byte

6$:     cmpa    #0x20           ; ? blank for next byte
        bne     8$              ; branch not
        stx     *addr           ; update pointer
7$:     swi                     ; give space
```

```
        .byte   space           ; function
        bra     1$

        ;* dot - next byte with address

8$:     cmpa    #'.             ; ? dot for next byte
        bne     9$              ; branch no
        swi                     ; force new line
        .byte   pcrlf           ; function
        stx     *addr           ; store next address
        bra     cmpadp          ; branch to show

        ;* up arrow - previous byte and address

9$:     cmpa    #'^             ; ? up arrow for previous byte
        bne     11$             ; branch not
        leax    -2,x            ; down to previous byte
        stx     *addr           ; store new pointer
10$:    swi                     ; force new line
        .byte   pcrlf           ; function
cmpadp=.
        bsr     12$             ; go print its value
        bra     1$              ; then prompt for input

        ;* slash - for current byte with address

11$:    cmpa    #'/             ; ? slash for current display
        beq     10$             ; yes, send address
        rts                     ; return from command

        ;* print current address

12$:    ldx     *addr           ; load pointer value
        pshs    x               ; save x on stack
        leax    ,s              ; point to it for display
        swi                     ; display pointer in hex
        .byte   out4hs          ; function
        puls    pc,x            ; recover pointer and return

        ;* update byte

13$:    ldx     *addr           ; load next byte pointer
        stb     ,x+             ; store and increment x
        cmpb    -1,x            ; ?  successfull store
        bne     14$             ; branch for '?'  if not
        stx     *addr           ; store new pointer value
        rts                     ; back to caller

14$:    pshs    a               ; save a register
        lda     #'?             ; show invalid
        swi                     ; send out
        .byte   outch           ; function
        puls    pc,a            ; return to caller

        ;************   window  -  set window value

cwindo: bsr     cdnum           ; obtain window value
        std     *window         ; store it in
        rts                     ; end command

        ;************   display - high speed display memory

cdi:    bsr     cdnum           ; fetch address
        andb    #0xf0           ; force to 16 boundry
        tfr     d,y             ; save in y
        leax    15,y            ; default length
        bcs     1$              ; branch if end of input
        bsr     cdnum           ; obtain count
        leax    d,y             ; assume count, compute end addr
1$:     pshs    y,x             ; setup parameters for hsdata
        cmpd    2,s             ; ? was it count
        bls     2$              ; branch yes
        std     ,s              ; store high address
2$:     jsr [vectab+.hsdta,pcr] ; call print routine
        puls    pc,u,y          ; clean stack and end command

        ;* obtain number - abort if none
        ;* only delimiters of cr, blank, or '/' are accepted
```

```
        ;* output: d=value, c=1 if carriage return delmiter,
        ;*                                 else c=0
cdnum:  lbsr    bldnum          ; obtain number
        lbne    cmdbad          ; branch if invalid
        cmpa    #'/             ; ? valid delimiter
        lbhi    cmdbad          ; branch if not for error
        cmpa    #cr+1           ; leave compare for carriage ret
        ldd     *anumber        ; load number
        rts

        ;************   punch - punch memory in s1-s9 format

cpunch: bsr     cdnum           ; obtain start address
        tfr     d,y             ; save in y
        bsr     cdnum           ; obtain end address
        clr     ,-s             ; setup punch function code
        pshs    y,d             ; store values on stack
ccalbs: jsr [vectab+.bson,pcr]  ; initialize handler
        jsr [vectab+.bsdta,pcr] ; perform function
        pshs    cc              ; save return code
        jsr [vectab+.bsoff,pcr] ; turn off handler
        puls    cc              ; obtain condition code saved
        lbne    cmdbad          ; branch if error
        puls    pc,y,x,a        ; return from command

        ;************   load - load memory from s1-s9 format

cload:  bsr     clvofs          ; call setup and pass code
        .byte   1               ; load function code for packet

clvofs: leau    [,s++]          ; load code in high byte of u
        leau    [,u]            ; not changing cc and restore s
        beq     1$              ; branch if carriage return next
        bsr     cdnum           ; obtain offset
        bra     2$

1$:     clra                    ; create zero offset
        clrb                    ; as default
2$:     pshs    u,dp,d          ; setup code, null word, offset
        bra     ccalbs          ; enter call to bs routines

        ;************   verify - compare memory with files

cver:   bsr     clvofs          ; compute offset if any
        .byte   -1              ; verify fnctn code for packet

        ;************   nulls  -  set new line and char padding

cnulls: bsr     cdnum           ; obtain new line pad
        std     *vectab+.pad    ; reset values
        rts                     ; end command

        ;************   offset - compute short and long
        ;*                              branch offsets

coffs:  bsr     cdnum           ; obtain instruction address
        tfr     d,x             ; use as from address
        bsr     cdnum           ; obtain to address

        ;* d=to instruction, x=from instruction offset byte(s)

        leax    1,x             ; adjust for *+2 short branch
        pshs    y,x             ; store work word and value on s
        subd    ,s              ; find offset
        std     ,s              ; save over stack
        leax    1,s             ; point for one byte display
        sex                     ; sign extend low byte
        cmpa    ,s              ; ? valid one byte offset
        bne     1$              ; branch if not
        swi                     ; show one byte offset
        .byte   out2hs          ; function
1$:     ldu     ,s              ; reload offset
        leau    -1,u            ; convert to long branch offset
        stu     ,x              ; store back where x points now
        swi                     ; show two byte offset
        .byte   out4hs          ; function
        swi                     ; force new line
        .byte   pcrlf           ; function
```

```
        puls    pc,x,d          ; restore stack and end command

        ;************   breakpoint - display/enter/delete/clear
        ;*                              breakpoints

cbkpt:  beq     5$              ; branch display of just 'b'
        lbsr    bldnum          ; attempt value entry
        beq     7$              ; branch to add if so
        cmpa    #'-             ; ? correct delimiter
        bne     9$              ; no, branch for error
        lbsr    bldnum          ; attempt delete value
        beq     2$              ; got one, go delete it
        clr     *bkptct         ; was 'b -', so zero count
1$:     rts                     ; end command

        ;* delete the entry

2$:     bsr     11$             ; setup registers and value
3$:     decb                    ; ? any entries in table
        bmi     9$              ; branch no, error
        cmpx    ,y++            ; ? is this the entry
        bne     3$              ; no, try next

        ;* found, now move others up in its place

4$:     ldx     ,y++            ; load next one up
        stx     -4,y            ; move down by one
        decb                    ; ? done
        bpl     4$              ; no, continue move
        dec     *bkptct         ; decrement breakpoint count
5$:     bsr     11$             ; setup registers and load value
        beq     1$              ; return if none to delete
6$:     leax    ,y++            ; point to next entry
        swi                     ; display in hex
        .byte   out4hs          ; function
        decb                    ; count down
        bne     6$              ; loop if more to do
        swi                     ; skip to new line
        .byte   pcrlf           ; function
        rts                     ; return to end command

        ;* add new entry

7$:     bsr     11$             ; setup registers
        cmpb    #numbkp         ; ? already full
        beq     9$              ; branch error if so
        lda     ,x              ; load byte to trap
        stb     ,x              ; try to change
        cmpb    ,x              ; ? changable ram
        bne     9$              ; branch error if not
        sta     ,x              ; restore byte
8$:     decb                    ; count down
        bmi     10$             ; branch if done to add it
        cmpx    ,y++            ; ? entry already here
        bne     8$              ; loop if not
9$:     lbra    cmdbad          ; exit with error

10$:    stx     ,y              ; add this entry
        clr     -numbkp*2+1,y   ; clear optional byte
        inc     *bkptct         ; add one to count
        bra     5$              ; and now display all of 'em

        ;* setup registers for scan

11$:    ldx     *anumber        ; load value desired
cbkldr: leay    bkptbl,pcr      ; load start of table
        ldb     *bkptct         ; load entry count
        rts                     ; return

        ;************   encode  -  encode a postbyte

cencde: clr     ,-s             ; default to not indirect
        clrb                    ; zero postbyte value
        leax    conv1,pcr       ; start table search
        swi                     ; obtain first character
        .byte   inchnp          ; function
        cmpa    #'[             ; ? indirect here
        bne     2$              ; branch if not
```

```
        lda     #0x10           ; set indirect bit on
        sta     ,s              ; save for later
1$:     swi                     ; obtain next character
        .byte   inchnp          ; function
2$:     cmpa    #cr             ; ? end of entry
        beq     4$              ; branch yes
3$:     tst     ,x              ; ? end of table
        lbmi    cmdbad          ; exit with error
        cmpa    ,x++            ; ? this the character
        bne     3$              ; branch if not
        addb    -1,x            ; add this value
        bra     1$              ; get next input
4$:     leax    conv2,pcr       ; point at table 2
        tfr     b,a             ; save copy in a
        anda    #0x60           ; isolate register mask
        ora     ,s              ; add in indirection bit
        sta     ,s              ; save back as postbyte skeleton
        andb    #0x9f           ; clear register bits
5$:     tst     ,x              ; ? end of table
        lbeq    cmdbad          ; exit with error
        cmpb    ,x++            ; ? same value
        bne     5$              ; loop if not
        ldb     -1,x            ; load result value
        orb     ,s              ; add to base skeleton
        stb     ,s              ; save postbyte on stack
        leax    ,s              ; point to it
        swi                     ; send out as hex
        .byte   out2hs          ; function
        swi                     ; to next line
        .byte   pcrlf           ; function
        puls    pc,b            ; end of command

        ;* table one defines valid input in sequence
conv1:  .byte   'A,0x04,'B,0x05,'D,0x06,'H,0x01
        .byte   'H,0x01,'H,0x01,'H,0x00,',,0x00
        .byte   '-,0x09,'-,0x01,'S,0x70,'Y,0x30
        .byte   'U,0x50,'X,0x10,'+,0x07,'+,0x01
        .byte   'P,0x80,'C,0x00,'R,0x00,'],0x00
        .byte   0xff            ; end of table

        ;*conv2 uses above conversion to set postbyte
        ;*                              bit skeleton.

conv2:  .word   0x1084,0x1100   ; R,         H,R
        .word   0x1288,0x1389   ; HH,R       HHHH,R
        .word   0x1486,0x1585   ; A,R        B,R
        .word   0x168b,0x1780   ; D,R        ,R+
        .word   0x1881,0x1982   ; ,R++       ,-R
        .word   0x1a83,0x828c   ; ,--R       HH,pcr
        .word   0x838d,0x039f   ; HHHH,pcr   [HHHH]
        .byte   0               ; end of table

        ;************************************************
        ;*           default interrupt transfers        *
        ;************************************************

rsrvd:  jmp     [vectab+.rsvd,pcr]      ; reserved vector
swi3:   jmp     [vectab+.swi3,pcr]      ; swi3 vector
swi2:   jmp     [vectab+.swi2,pcr]      ; swi2 vector
firq:   jmp     [vectab+.firq,pcr]      ; firq vector
irq:    jmp     [vectab+.irq,pcr]       ; irq vector
swi:    jmp     [vectab+.swi,pcr]       ; swi vector
nmi:    jmp     [vectab+.nmi,pcr]       ; nmi vector

        .page
        .sbttl  Hardware Interrupt Tables

        ;******************************************************
        ;*             assist09 hardware vector table
        ;*
        ;*  this table is used if the assist09 rom addresses
        ;*  the mc6809 hardware vectors.
        ;******************************************************

        .=  bldvtr+0d2048-0d16  ; assume 2K ROM

        .word   rsrvd           ; reserved slot
        .word   swi3            ; software interrupt 3
```

```
.word   swi2            ; software interrupt 2
.word   firq            ; fast interrupt request
.word   irq             ; interrupt request
.word   swi             ; software interrupt
.word   nmi             ; non-maskable interrupt
.word   reset           ; restart
```

```
-xms
astopt
assist
rel
-b RAM = 0
-b VARSAV = varsav
-b DPVSAV = dpvsav
-b BUFSAV = bufsav
-b VECTOR = chrom1
-b R6571A = chrom2
-b VT1XX  = chrom3
-b FIGROM = chrom4
-b PGMSAV = pgmsav
-b EXTSAV = extsav
-b IRQINT = irqint
-b OPTFUN = optfun
-b WORKPG   = 0o17400
-b ASSIST09 = 0o20000
-b ASTOPT =   0o30000
-e
```

```
        .title  Loadable EPSON Printer Driver

        ; overlay the Option Table with entries
        ; for this driver

        .area   OPTFUN  (ABS,OVR)

        .word   .$0             ;$0
        .word   .$0-0x5AA5
        .blkb   4               ;$1
        .blkb   4               ;$2
        .blkb   4               ;$3
        .blkb   4               ;$4
        .blkb   4               ;$5
        .blkb   4               ;$6
        .blkb   4               ;$7
        .blkb   4               ;$8
        .blkb   4               ;$9

        .word   $ld             ;loadable printer driver $t_ entry
        .word   $ld-0x5AA5      ;check code

        .word   ldscrn          ;loadable printer driver screen entry
        .word   ldscrn-0x5AA5   ;check code

        .page
        .sbttl  Driver Identification

        .module epson

        .area   EPSON   (ABS,OVR)

        .radix  o

ld$id:  .byte   15,12
        .ascii  *DSPCGC   V02.02*
        .byte   15,12
        .ascii  *EPSON MX/FX 80/100 PRINTER DRIVER - V01.00*
        .byte   15,12
        eps$id  =       .-ld$id         ;length of version
        .ascii  *COPYRIGHT 1988*
        .byte   15,12
        .ascii  *OTSELIC SPECIALTIES*
        .byte   15,12
        .ascii  *721 BERKELEY*
        .byte   15,12
        .ascii  *KENT, OHIO  44240*
        .byte   15,12
        eps$cr  =       .-ld$id         ;length of notice


        .page
        .sbttl  $1 command

.$0:    jsr     nxtchr          ;get character

        jsr     $dispatch
        .byte   'V
        .word   2$              ;version select
        .byte   'C
        .word   3$              ;copyright select
        .byte   0

2$:     lda     #eps$id         ;eps$id character version #
        bra     4$

3$:     lda     #eps$cr         ;eps$cr copyright version #

4$:     sta     number
        ldx     #ld$id          ;version string
5$:     ldb     ,x+             ;get character
        stx     qt              ;save pointer
        jsr     plcbuf          ;send character
        ldx     qt              ;get pointer
        dec     number          ;more ?
        bne     5$              ;yes - loop
        bra     .$0
```

```
        .page
        .sbttl  'epson' printer handler

        ; 'epson' printer messages

ldmsg0: .byte   15,12           ; <cr><lf>
ldmsg1: .byte   15,12,200       ; <cr><lf> <end>
ldmsg2: .byte   15,200          ; <cr> <end>
ldmsg4: .byte   33,'K,200       ; 60 dots/inch
ldmsg9: .byte   33,'*,5,200     ; 72 dots/inch
ldmsg5: .byte   33,'L,200       ; 120 dots/inch
ldmsg6: .byte   33,'2,200       ; 1/6" line spacing
ldmsg7: .byte   33,'3,1,200     ; 1/216" line spacing
ldmsg8: .byte   33,'3,27,200    ; 23/216" line spacing
ldmsg3: .byte   33,'3,30,200    ; 24/216" line spacing

        .page
        .sbttl  panel selected epson screen dump

ldscrn: lda     $tdump          ;screen dump mode
        sta     $pdens
        bne     1$              ;not low - skip
        ldd     #0d512          ;lines in display
        std     $line0
        ldd     #0              ;offset
        std     $pofst
        ldd     #0              ;y-top
        std     $cur0
        ldd     #0d128          ;center of x range
        std     $cur0-2
        ldd     #0d480          ;printer width
        bra     3$

1$:     cmpa    #1
        bne     2$              ;not graphic - skip
        ldd     #0d512          ;lines in display
        std     $line0
        ldd     #0d35           ;offset
        std     $pofst
        ldd     #0              ;y-top
        std     $cur0
        ldd     #0              ;x-left
        std     $cur0-2
        ldd     #0d512          ;printer width
        bra     3$

2$:     ldd     #0d512          ;lines in display
        std     $line0
        ldd     #0d200          ;offset
        std     $pofst
        ldd     #0              ;y-top
        std     $cur0
        ldd     #0              ;x-left
        std     $cur0-2
        ldd     #0d512          ;printer width

3$:     std     $dots
        addd    $pofst
        std     $plen           ;total width
        bsr     $ld             ;and print full screen
        ldy     #ldmsg0         ;two <cr><lf>'s
        jsr     printm

lddone: lda     rstat0
        bita    #2              ;button released ?
        beq     lddone          ;loop until released
        lda     rstat2
        ora     #2
        sta     rstat2          ;clear stop flag
        ldy     #ldmsg6         ;reset line spacing
        jmp     printm          ;finished

        .page
        .sbttl  epson color mapping

$ld:    ldx     #curmap         ;current rgb mapping
        ldy     #$trmap         ;$trmap result table
```

```
        lda     ,x+             ;build background flag
        ora     ,x+
        ora     ,x+
        beq     1$
        ora     #377
1$:     sta     $bkgnd          ;save background flag
        sta     ,y+             ;and first color
        lda     #0d15           ;15 more colors to transform
2$:     ldb     ,x+             ;transform 'red'
        orb     ,x+             ;transform 'green'
        orb     ,x+             ;transform 'blue'
        beq     3$
        orb     #377
3$:     tst     $bkgnd          ;background ?
        beq     4$              ;no - skip
        comb
4$:     stb     ,y+             ;save transformed code
        deca
        bgt     2$

        lda     $pdens          ;check density
        beq     ld60            ;plot low density
        bgt     ld72            ;plotter graphics density
        bra     ld120           ;plot high density

        .page
        .sbttl  'epson' low density dump routine

ld60:   ldd     #0d900          ;maximum line length
        jsr     tbound          ;bound parameters
        clr     $pdens          ;low density
        ldy     #ldmsg2         ;go to left margin
        jsr     printm
        ldy     #ldmsg3         ;8/72" line spacing
1$:     jsr     printm
        lda     rstat2
        bita    #2              ;stop dump ?
        lbeq    lddone          ;yes - exit
        ldy     #ldmsg4         ;load dots/inch
        jsr     ldline          ;dump a line to printer
        ldb     #0d8            ;update position
        jsr     tupd0
        lble    lddone          ;exit if end
        ldy     #ldmsg1         ;advance to next line
        bra     1$              ;and loop

        .page
        .sbttl  'epson' plotter graphics dump routine

ld72:   ldd     #0d1080         ;maximum line length
        jsr     tbound          ;bound parameters
        clr     $pdens          ;low density
        ldy     #ldmsg2         ;go to left margin
        jsr     printm
        ldy     #ldmsg3         ;8/72" line spacing
1$:     jsr     printm
        lda     rstat2
        bita    #2              ;stop dump ?
        lbeq    lddone          ;yes - exit
        ldy     #ldmsg9         ;load dots/inch
        jsr     ldline          ;dump a line to printer
        ldb     #0d8            ;update position
        jsr     tupd0
        lble    lddone          ;exit if end
        ldy     #ldmsg1         ;advance to next line
        bra     1$              ;and loop

        .page
        .sbttl  'epson' high density dump routine

ld120:  ldd     #0d1800         ;maximum line length
        jsr     tbound          ;bound parameters
        lda     #-1             ;high density
        sta     $pdens
        ldy     #ldmsg2         ;move to left margin
1$:     jsr     printm
        lda     rstat2
        bita    #2              ;stop dump ?
```

```
        lbeq    lddone          ;yes - exit
        ldy     #ldmsg7         ;1/216" line spacing
        jsr     printm
        ldy     #ldmsg5         ;load dots/inch
        jsr     ldline          ;dump a line
        ldb     #1              ;update position
        jsr     tupd0
        ldy     #ldmsg1         ;<cr><lf>
        jsr     printm
        ldy     #ldmsg8         ;23/216" line spacing
        jsr     printm
        ldy     #ldmsg5         ;load line length
        jsr     ldline          ;dump line
        ldb     #0d15           ;update position
        jsr     tupd0
        lble    lddone          ;exit if finished
        ldy     #ldmsg1         ;<cr><lf>
        bra     1$              ;loop for next line

        .page
        .sbttl  'epson' dump line to printer

ldline:
        jsr     printm
        ldb     $plen+1
        jsr     printb
        ldb     $plen
        jsr     printb

        ldd     $dots           ;init dot counter
        coma
        comb
        std     $rdots
        lda     $pzoom          ;and x-zoom
        sta     $rzoom

        inc     $rdots+1        ;update dots left
        bne     1$
        inc     $rdots
        lbpl    17$             ;exit if done

1$:     ldy     $pofst          ;now dump offset nulls
        beq     3$              ;if none - skip
        clrb                    ;null
2$:     jsr     printb
        leay    -1,y
        bne     2$

3$:     jsr     tmv01           ;load variable set 1
        jsr     tmv12           ;load variable set 2

4$:     lda     #0d8            ;8. rows in swath
        sta     $dotctr
        ldx     $cur2-2         ;load x-address
        stx     crxpos
        stx     vidx
        ldx     $cur2           ;load y-address
        bra     10$             ;entry point

5$:     ldx     $line2          ;running line count
        ldb     $zoom2          ;zoom factor
        tst     $pdens          ;low - skip
        beq     6$
        leax    -1,x            ;one less line
        dec     $zoom2
6$:     leax    -1,x            ;one less line
        stx     $line2          ;save count
        ble     12$             ;past last line
        dec     $zoom2
        bge     13$             ;not time - skip
        lda     $pzoom          ;restore zoom value
        ldx     $cur2           ;buffer pointer
        tst     $pdens          ;low - skip
        beq     8$
        tsta                    ;high density zoomed ?
        beq     7$              ;no zoom - skip
        decb                    ;update zoom factor
        bge     8$
```

```
        tfr     a,b             ;reset zoom factor
        bra     8$
7$:     leax    10,x            ;next row
8$:     leax    10,x            ;next row
        decb                    ;update zoom factor
        bge     9$
        tfr     a,b             ;reset zoom factor
9$:     stb     $zoom2          ;and save
10$:    stx     $cur2           ;updated buffer pointer

        ;       $cur2-2 contains the x-address of the data
        ;       $cur2   contains the y-address of the data

        ;       'transformed color' left in $color

        stx     crypos
        stx     vidy            ;y position
        lda     vidplt          ;read data
        anda    $blank          ;use selected bits
        bita    #20             ;oc buffer ?
        beq     11$             ;no - skip
        ora     #17             ;color 17 for oc
11$:    anda    #17
        lda     a,u             ;$trmap transformed color
        sta     $color          ;save color
        bra     14$
12$:    lda     $bkgnd          ;else background
        bra     14$
13$:    lda     $color          ;get color
14$:    rora                    ;shift bit into 'c'
        rol     char            ;and into char
        dec     $dotctr         ;loop for all 8. rows
        bgt     5$
15$:    jsr     printc          ;then print the character

        inc     $rdots+1        ;update dots left
        bne     16$
        inc     $rdots
        bpl     17$             ;exit if done

16$:    dec     $rzoom          ;update x-zoom
        bge     15$             ;loop if more x-zoom

        lda     $pzoom          ;reset zoom
        sta     $rzoom
        ldd     $cur1-2         ;update column position
        addd    #0d8
        std     $cur1-2
        jsr     tmv12           ;reload variable set 2
        lbra    4$

17$:    rts                     ;and finished
```

```
-xms
epson
rel
-b RAM = 0
-b VARSAV = varsav
-b DPVSAV = dpvsav
-b BUFSAV = bufsav
-b VECTOR = chrom1
-b R6571A = chrom2
-b VT1XX  = chrom3
-b FIGROM = chrom4
-b PGMSAV = pgmsav
-b EXTSAV = extsav
-b IRQINT = irqint
-b OPTFUN = optfun
-b EPSON  = optfun+0d256
-e
```

```
.title  Option File for DSPCGC

.module dspopt

;  Include this file in Down Loadable Code
;  destined for the DSPCGC.

;  The following labels and data allocations
;  are defined in DSPCGC.

;  This file should be the first file to be linked
;  to insure that the dsiplatch table is cleared.

; $xtrn0:       .blkb   4       ; 4096 bytes allocated
; $xtrn1:       .blkb   4       ; for optional functions
; $xtrn2:       .blkb   4
; $xtrn3:       .blkb   4
; $xtrn4:       .blkb   4
; $xtrn5:       .blkb   4
; $xtrn6:       .blkb   4
; $xtrn7:       .blkb   4
; $xtrn8:       .blkb   4
; $xtrn9:       .blkb   4

; $xtrnl:       .blkb   4       ; loadable printer driver $t_ entry
; $xscrn:       .blkb   4       ; loadable printer driver screen entry

; $xtend                        ; end of user area


;  System Vector Table as defined in DSPCGC.

; extbypls:     .blkb   2       ; loadable system vectors
; extbymns:     .blkb   2
; extbxpls:     .blkb   2
; extbxmns:     .blkb   2
; exundfnd:     .blkb   2
; exdltint:     .blkb   2
; exdlrint:     .blkb   2
; exclocki:     .blkb   2

; exrsrv:       .blkb   2
; exswi3:       .blkb   2
; exswi2:       .blkb   2
; exfirq:       .blkb   2
; exirq:        .blkb   2
; exswi:        .blkb   2
; exnmi:        .blkb   2

.page
.sbttl  Option Dispatcher

.area   OPTFUN  (ABS,OVR)

.radix  o

.word   .$0HDR          ; $0 header entry
.word   .$0HDR-0x5AA5
.word   0               ; $1
.word   0
.word   0               ; $2
.word   0
.word   0               ; $3
.word   0
.word   0               ; $4
.word   0
.word   0               ; $5
.word   0
.word   0               ; $6 Command Entry
.word   0
.word   0               ; $7
.word   0
.word   0               ; $8
.word   0
.word   .$9             ; $9
.word   .$9-0x5AA5

.word   0               ; loadable printer driver $t_ entry
```

```
        .word   0               ; check code

        .word   0               ; loadable printer driver screen entry
        .word   0               ; check code

        .page
        .sbttl  DSPCGC Header

opt$id: .byte   15,12
        .ascii  *DSPCGC   V02.02*
        .byte   15,12
        .ascii  *DSPCGC Optional Function Header - V01.00*
        .byte   15,12
        optl$id =       .-opt$id        ; length of version
        .ascii  *Copyright 1988*
        .byte   15,12
        .ascii  *Otselic Specialties*
        .byte   15,12
        .ascii  *721 Berkeley*
        .byte   15,12
        .ascii  *Kent, Ohio  44240*
        .byte   15,12
        optl$cr =       .-opt$id        ; length of notice


        .page
        .sbttl  $0 command

.$0HDR: jsr     nxtchr          ; get character

        jsr     $dispatch
        .byte   'V
        .word   2$              ; version select
        .byte   'C
        .word   3$              ; copyright select
        .byte   0

2$:     lda     #optl$id        ; optl$id character version #
        bra     4$

3$:     lda     #optl$cr        ; optl$cr copyright version #

4$:     sta     number
        ldx     #opt$id         ; version string
5$:     ldb     ,x+             ; get character
        stx     qt              ; save pointer
        jsr     plcbuf          ; send character
        ldx     qt              ; get pointer
        dec     number          ; more ?
        bne     5$              ; yes - loop
        bra     .$0HDR

        .page
        .sbttl  .$9 -- Reset DSPCGC Interrupt Vectors

        ;  The default Vector Table is loaded from the
        ;  following table defined in DSPCGC.

        ; irqtbl:       .word   tbypls, extbypls
        ;               .word   tbymns, extbymns
        ;               .word   tbxpls, extbxpls
        ;               .word   tbxmns, extbxmns
        ;               .word   undfnd, exundfnd
        ;               .word   dlrint, exdltint
        ;               .word   dlrint, exdlrint
        ;               .word   clocki, exclocki

        ;               .word   undfnd, exrsrv
        ;               .word   undfnd, exswi3
        ;               .word   undfnd, exswi2
        ;               .word   undfnd, exfirq
        ;               .word   undfnd, exirq
        ;               .word   undfnd, exswi
        ;               .word   $nmi,   exnmi
        ;               .word   0,      0               ; end of table

        ;  Reset Interrupt Table using this routine
```

```
.$9:    ldx     #irqtbl         ; table pointer
1$:     ldd     ,x++            ; get vector entry
        beq     2$
        std     [,x++]          ; place in vector table
        bra     1$              ; loop

2$:     rts                     ; finished
```

```
-xms
dspopt
rel
-b RAM = 0
-b VARSAV = varsav
-b DPVSAV = dpvsav
-b BUFSAV = bufsav
-b VECTOR = chrom1
-b R6571A = chrom2
-b VT1XX  = chrom3
-b FIGROM = chrom4
-b PGMSAV = pgmsav
-b EXTSAV = extsav
-b IRQINT = irqint
-b OPTFUN = optfun
-e
```

DISPLAY SYSTEM

RESET

BUSY

ERROR

HOLD

CONT

PRINTER

ON

OFF

DUMP

PLOTTER

ON

OFF

CAL

VIDEO

ON

OFF

CLEAR

PANEL

ARB — 85

FTGL(1)

ON

OFF

LIST

ON

OFF

TAPE

IN

OUT

GATE ADC

ON

OFF

DISPLAY

ON

OFF

OVERLAY

ON

OFF

EXPAND

ON

OFF

STRIP

ON

OFF

1 START

2 STOP

3 EXECUTE FRSW(2)

4 ZERO DATA

FRSW(1)

1 2 3 4 5 6 7 8 9 10 11 12

FRSW(2)

DUMP HISTO

READ HISTO

OPEN

CLOSE

COM

CONT

MODE

SET REAL

SET ORG

SET LIVE

LENGTH

1K 2K 4K 8K 16K 32K 256 512

SCALE

1K 2K 4K 8K 16K 32K 64K 128 256 512 1MEG 256K

FPSH()

5

6

7

8

ADC ON

(2)

MT ERROR

CH OFLO

1 2

CURSOR CONTROL

(5) (6) (7) (8)

FLAMP()

FTMB(2)

ADC NO.

C C C C O O

BOUND

FTMB(1)

ORIGIN

TIME (0.1SEC)

0 0 0 0 0 1

ARB - 85

DISPLAY SYSTEM

RESET

BUSY

ERROR

HOLD

CONT

PRINTER

ON

OFF

DUMP

PLOTTER

ON

OFF

CAL

VIDEO

ON

OFF

CLEAR

ARB – 85

akit terminal

DISPLAY SYSTEM

RESET

BUSY ERROR HOLD CONT

PRINTER

ON OFF DUMP

PLOTTER

ON OFF CAL

VIDEO

ON OFF CLEAR

PANEL

ARB — 85

VIDEO

PLOTTER

PRINTER

1953

1 AMP

FUSE

ON

OFF

Heathkit terminal

DISPLAY SYSTEM

RESET

BUSY

ERROR

HOLD

PRINTER

ON

OFF

PLOTTER

ON

OFF

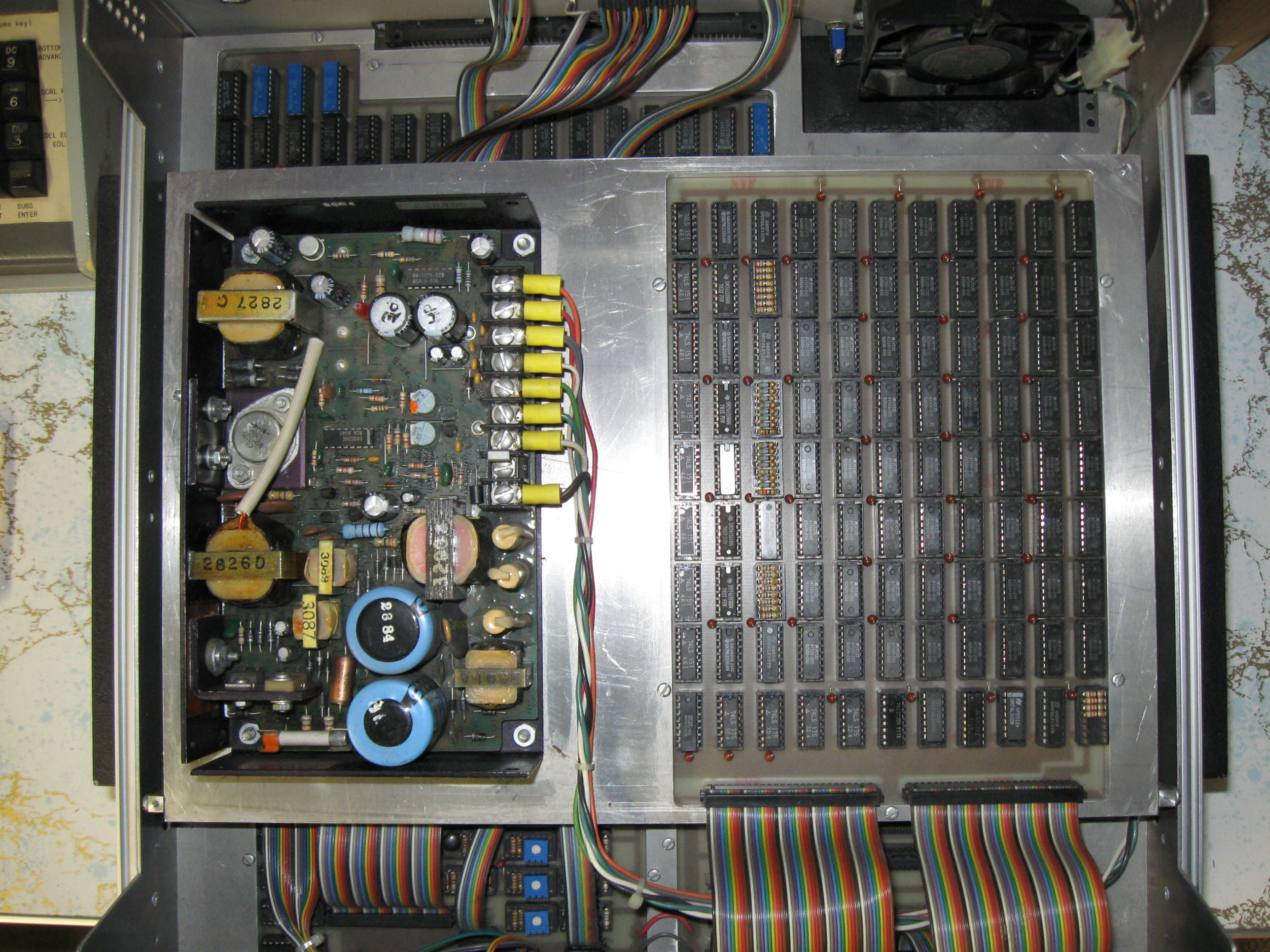
VIDEO

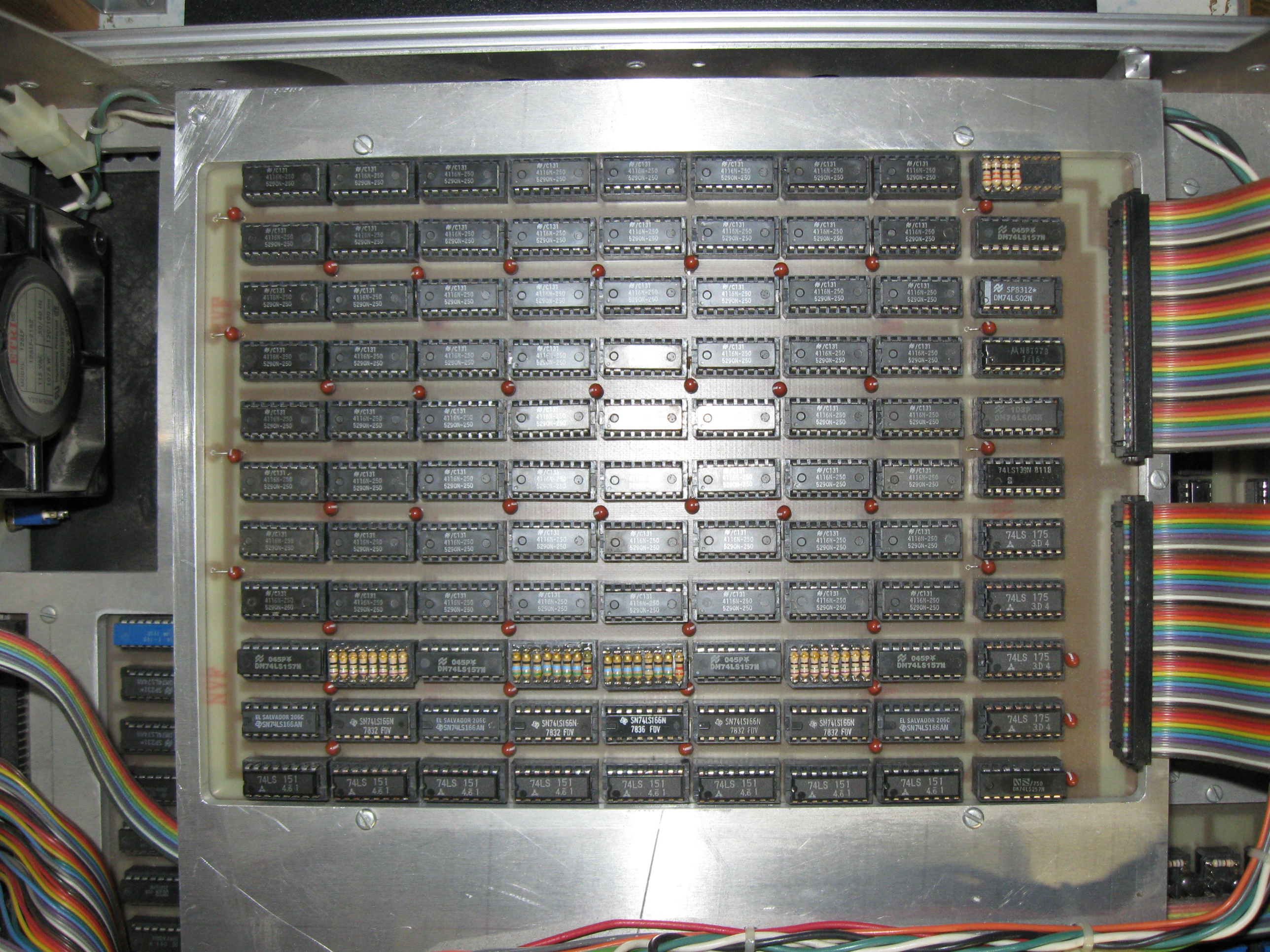
ON

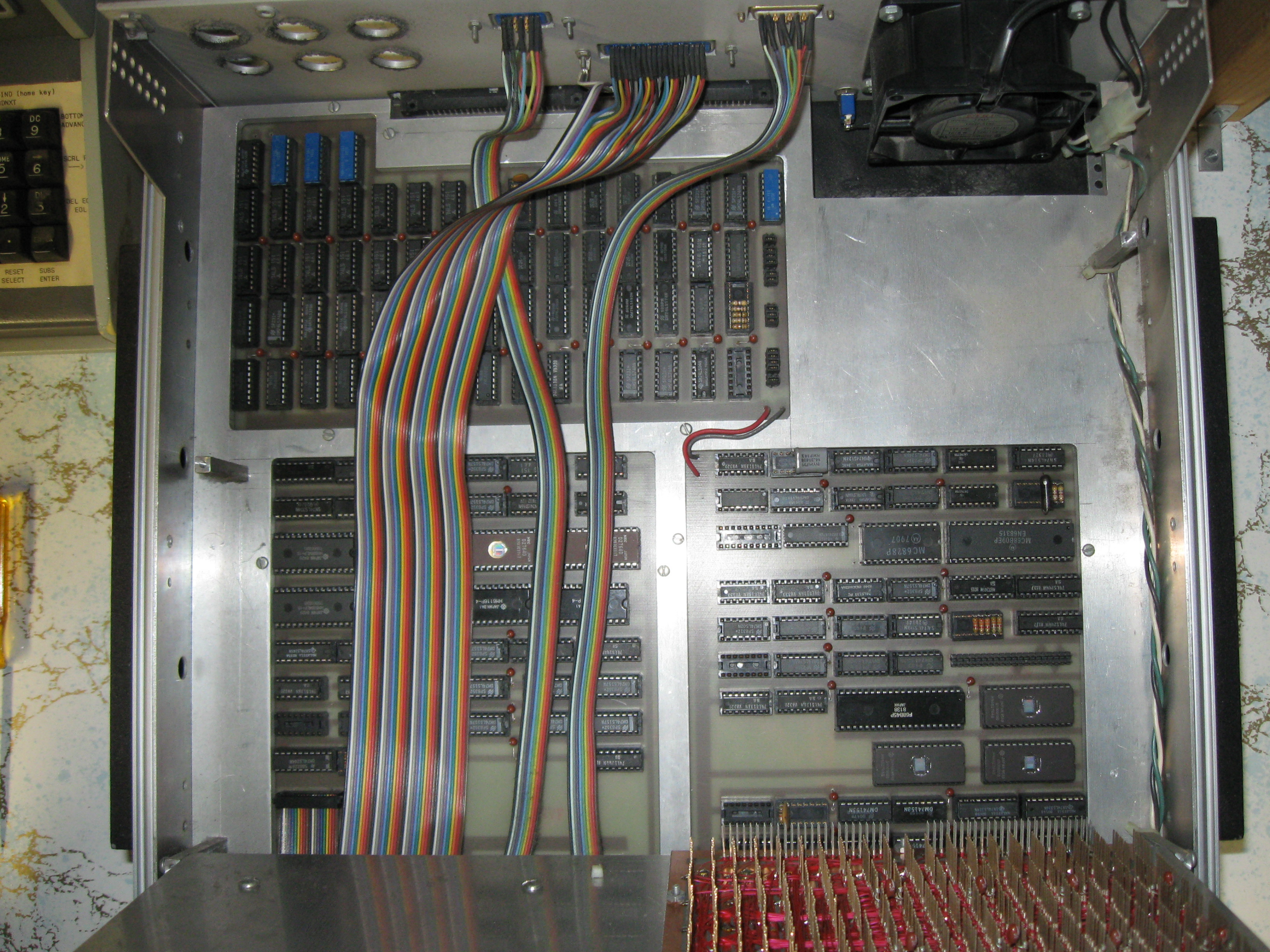
OFF

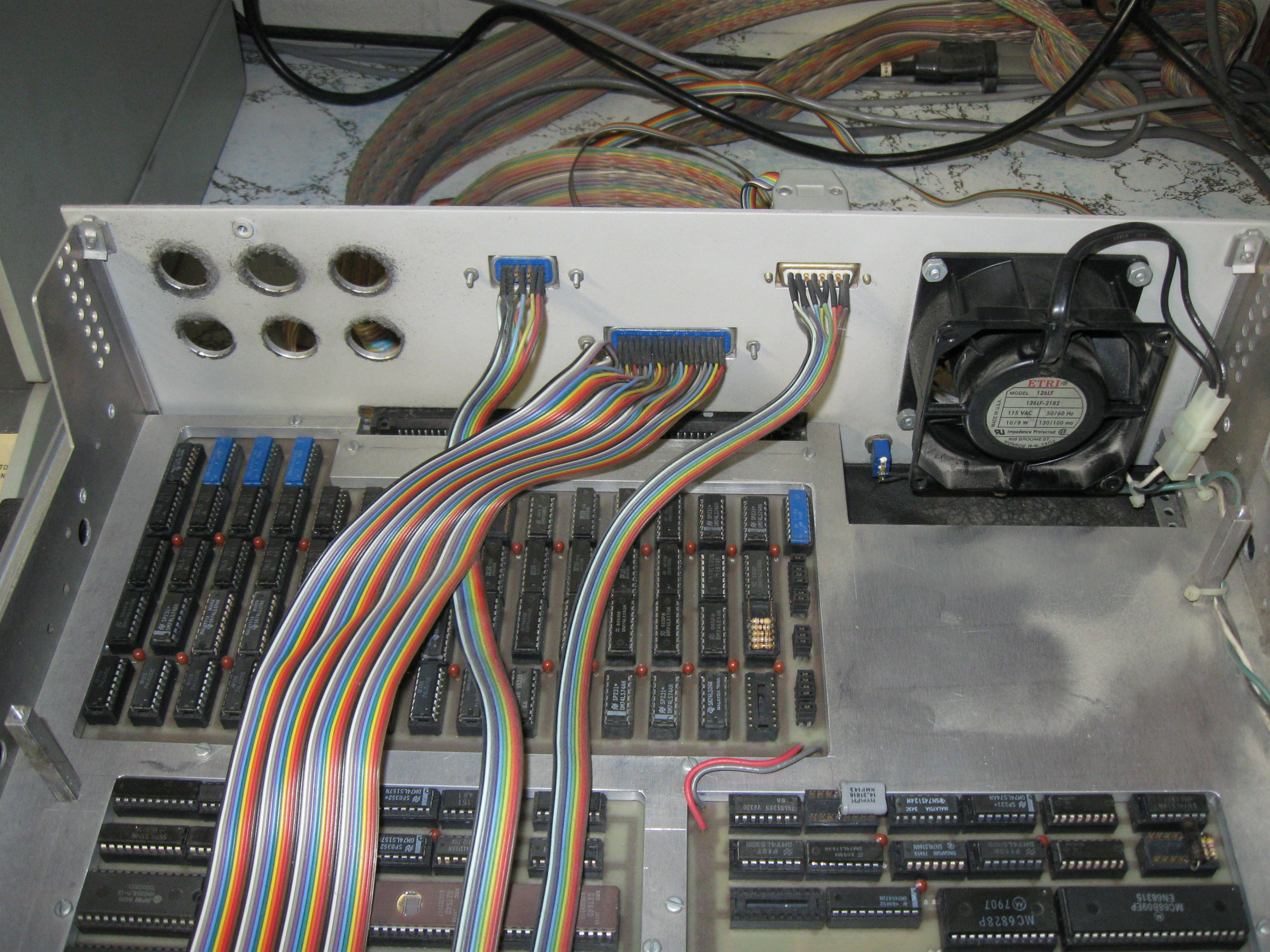
PANEL

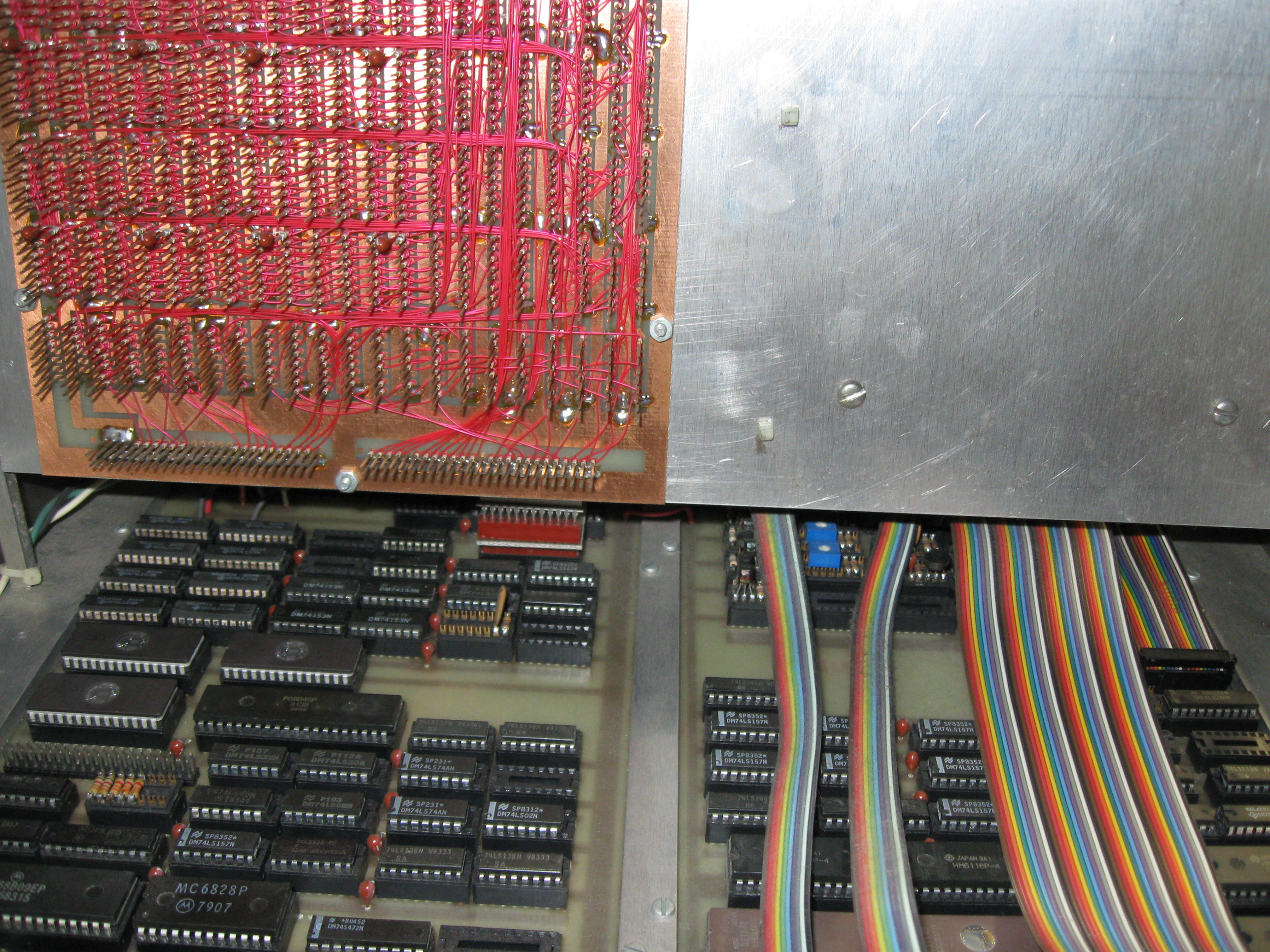
ARB — 85


2827 C
2826D
3087
28 84


M/C131 4116N-250 5290N-250
045PX DM74LS157N
SP8312* DM74LS02N
MN8T97B 7816
74LS139N 8118
74LS 175 3.D 4
74LS 151 4.6 1
EL SALVADOR 206C SN74LS166AN
SN74LS166N 7832 FDV
SN74LS166N 7836 FDV




ETRI
MODEL 126LF
126LF-2182
115 VAC 50/60 Hz
10/8 W 130/100 ma
Impedance Protected
608 BROOME ST.
MONROE N.C.
MADE IN U.S.A.
MC6828P
7907
MC68B09EP
EN6831S


MC6828P
7907
SP8352*
DM74LS157N
SP231*
DM74LS74AN
SP8312*
DM74LS02N
74LS138N V8333
SA
DM74153N
DM74LS04N
DM74LS08N
DM74S472N
HM6116P-4
JAPAN 3A1