AN EVALUATION OF THE NSC800 8-BIT MICROPROCESSOR FOR DIGITAL SIGNAL PROCESSING APPLICATIONS

by

MAC A. CODY

B. S. Kansas State University, 1979

A MASTER'S THESIS

submitted in partial fulfillment for the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1981

Approved by:

M S P Lucas

Major Professor

AN EVALUATION OF THE NSC800 8-BIT MICROPROCESSOR FOR DIGITAL SIGNAL PROCESSING APPLICATIONS

A Master's Thesis

by

MAC A. CODY

Department of Electrical Engineering
Kansas State University
Manhattan, Kansas
1981

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERENTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.

CHAPTER I

INTRODUCTION

Within the past few years, several investigations have been made into the implementation of adaptive digital prediction algorithms on microprocessor systems. The common goal of these projects has been to determine whether or not a dedicated microcomputer can adequately perform the task of real time signal processing of intrusion detection algorithms in the field. Using algorithms developed by Ahmed [1,2], this has been accomplished with some degree of success by Nickel [3] and Hass [4].

Although these microprocessor implementations have been to a large extent successful, they have not been placed into actual field use due to practical limitations of one type or another. The Texas Instruments SBP9900 16-bit I^2L microprocessor [4], though fast and low in power consumption, suffered from production problems at Texas Instruments and has since been replaced by the improved SBP9989. The RCA ATMAC [3] was also a very fast microprocessor and consumed very little power due to its CMOS SOS construction. Its complex Instruction Set and high cost have precluded its use in the signal processing arena and it has since been withdrawn from the marketplace. The Zilog Z80 [3] proved to be a capable device for signal processing, but its NMOS construction consumed too much power for the requirements of the field application. Also a military specified version of this processor does not exist. What is needed is a microprocessor that is capable of high speed instruction execution, consumes little power, and is available in quantity while meeting military use specifications.

The subject of the work done by the author was the evaluation of a new microprocessor which may satisfy the above requirements. This is the National Semiconductor NSC800 microprocessor. Based on a new process called double-polysilicon CMOS (P^2CMOS), it is hoped that this processor will realize the operational speed of NMOS microprocessor counterparts with the low power consumption inherent in CMOS devices. This increased speed is accomplished by applying proven NMOS processes in new ways. Further details of this process can be found in the National reference [5].

The NSC800, as a microprocessor, is a result of the combination of two popular NMOS devices; the Intel 8085 and the Zilog Z80. The bus structure of the NSC800 is essentially a copy of the 8085's. The lower 8 bits of the address bus are multiplexed with the 8-bit data bus. The upper 8 bits of address are not multiplexed. Several levels of hardware interrupt are provided as well as lines for bus status and control. Similar lines also exist on the 8085. Not of the 8085 heritage are the dynamic memory refresh control and power save lines. Refresh control is also used by the Z80, while the power save feature is unique to the NSC800. A logic low on this pin causes the CPU to go into a low power mode which uses power only for the oscillator and the system clock. The instruction set of the NSC800 is identical to the Z80's. There is an internal write-only interrupt mask register at I/O location BB hex in the NSC800, which is the only difference between these two processors in their internal makeup. Once again, more information on the architecture of the NSC800 can be found in the National reference.

Along with the NSC800, National Semiconductor has also released a line of small and medium scale integration support devices. They are based on the popular 74 series TTL/CMOS and 82 series NMOS products. Since they are also constructed using P^2CMOS, their gate delays are much shorter than standard CMOS devices while consuming the same power. In fact, the gate delays of P^2CMOS devices are nearly identical to standard NMOS devices and even approach that of TTL in some cases. Table 1.1 is a list of the presently available P^2CMOS devices and their NMOS, CMOS, and TTL counterparts.

In Chapter II, we shall cover the design and construction of a single board computer system utilizing the NSC800 CPU and various P^2CMOS devices. Chapter III discusses the evaluation of the hardware of this single board computer, with emphasis on the NSC800 and its support devices. In Chapter IV, the implementation of the Widrow LMS algorithm and the lattice algorithm will be performed. Evaluation of the NSC800 as a signal processor will then be done. Conclusions from these evaluations will be made in Chapter V.

Table 1.1: A List of Currently Available P^2CMOS Logic Devices and Their Pin-Compatible Counterparts in Other Logic Families

Device Description	P^2CMOS Logic Devices	Available Logic Versions		
		TTL	NMOS	Standard CMOS
Quad 2-Input NAND Gates	74PC00	74LS00	*	74C00
Quad 2-Input NOR Gates	74PC02	74LS02	*	74C02
Hex Inverters	74PC04	74LS04	*	74C04
Quad 2-Input AND Gates	74PC08	74LS08	*	74C08
Quad 2-Input OR Gates	74PC32	74LS32	*	74C32
Dual D-Type Flip Flops	74PC74	74LS74	*	74C74
3-to-8 Line Decoder/Demux.	74PC138	74LS138	8205	*
8-Bit Bidirectional Bus Driver	82PC08	INS8208	8208	*
8-Bit Input/Output Port	82PC12	74S412 DP8212	8212	*

*No Part Available

CHAPTER II

SINGLE BOARD COMPUTER DESIGN

The main purpose of this single board computer was the evaluation of the National Semiconductor NSC800 microprocessor and its family of support devices. This evaluation encompassed hardware and software performance in signal processing applications. These applications are of a complex nature, mathematicly speaking, therefore, the computer needed to be capable of performing both data acquisition and rapid numerical operations on that data.

After some thought, a block diagram of the single board computer was developed. This block diagram appears in Figure 2.1. From this block diagram, a set of schematics were generated. These schematics appear in Figures 2.2 through 2.8.

Figure 2.2 shows the NSC800 CPU and the accompanying addressing and demultiplexing electronics. Addressing of the major blocks of the computer memory map is accomplished through the 3-to-8-line decoder, U3, and the NOR gate, U24. Each line of the decoder selects a 1K byte block of memory. This results in partial address decoding for the multiply/divide, analog interface, and parallel I/O circuits. These devices can be addressed at many locations within their respective blocks. Separation of memory addressing from input/output addressing is done through selecting the decoder with the IO/M line. Demultiplexing of the lower 8 bits of the address is done through the 82PC12 8-bit I/O latch, U2. The address latch

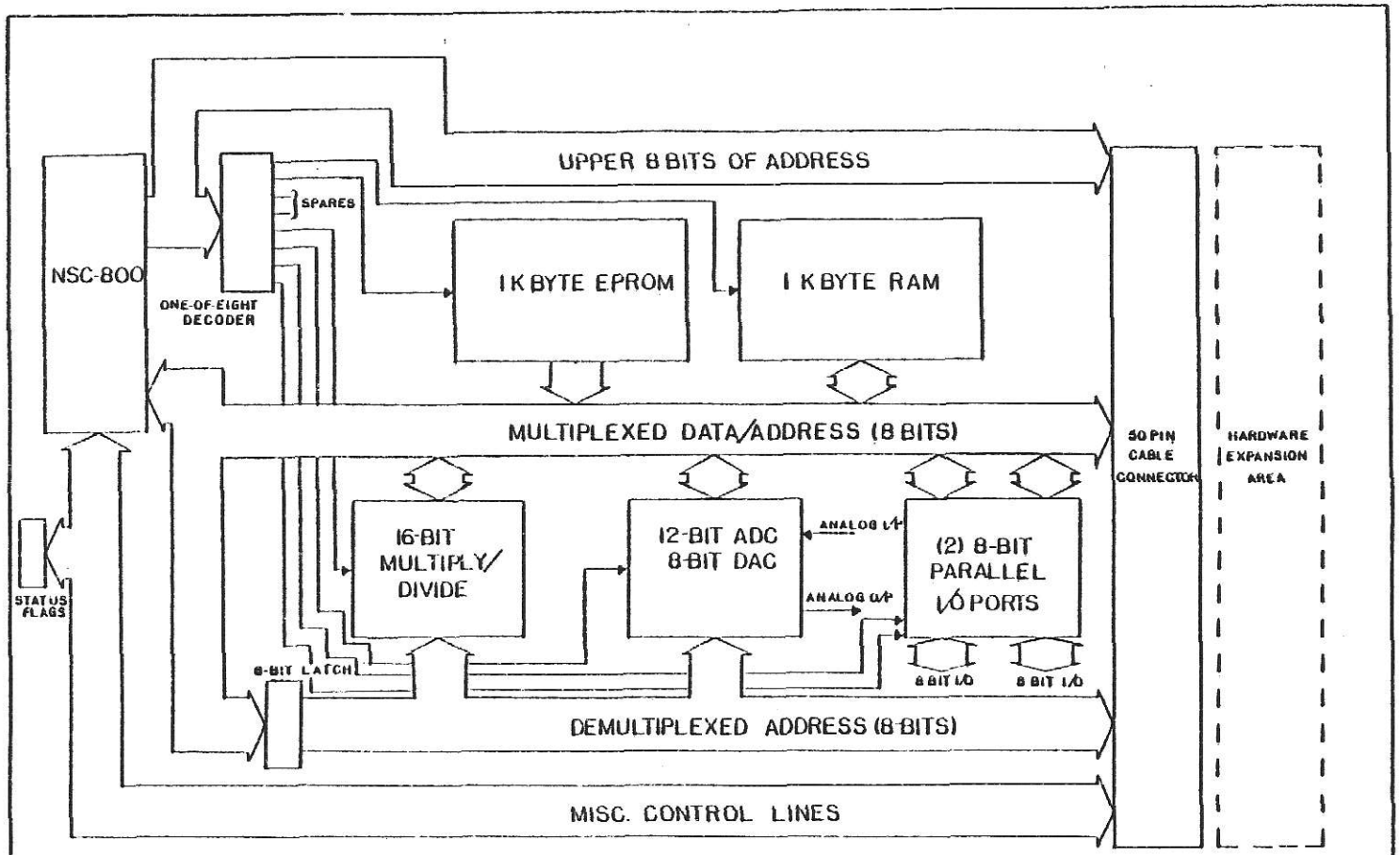

FIGURE 2.1 BLOCK DIAGRAM OF THE SINGLE BOARD COMPUTER

enable (ALE) signal of the CPU performs the strobe operation. Individual device selection is then done through this latched address and appropriate discrete logic. Pull-downs are on address and data lines to reduce noise on the bus. Pull-ups are on some control lines so that proper logic levels are maintained when they are not in use.

Figure 2.3 contains the general control and support circuitry and the bus connector layout. The swichable crystal network (X1, X2, and SW2) allows the user the choice between 5 MHz and 8 MHz internal oscillator frequencies, i.e., 2.5 MHz and 4 MHz system clock frequencies respectively [The user must be forewarned that the access time to the EPROMs is such that the CPU must be running at the 2.5 MHz system clock frequency in order to assure reliable data access from them]. The computer is provided with a 50-pin ribbon cable connector to allow access to the bus and power lines. Table 2.1 outlines the functions of the pins on this connector.

Two control latches, flip flops U25 with light-emitting diodes CR1-4, are provided. One latch alerts the user to the operational state of the CPU, i.e. running (R) or halted (H). The other latch is connected to the bus through the $IO/\overline{M}$ and A7 lines. Upon reset of the system, the latch is reset so that the 'Down Load' LED is on. The setting or resetting of the flag can be controled through an I/O instruction. Any I/O instruction which is addressed to locations 80 to FF hex (hexadecimal) sets the latch. An I/O instruction to locations 00 to 7F hex resets the latch. This latch can be used for general-purpose controlling or signalling. The fourteen-stage ripple counter, U28, provides the clock signal for the A/D converter. Other divided frequencies are provided through the wire wrap

posts of J4. Precision voltage references (+5 and -10 volts) for the A/D and D/A converters are provided through IC's U34 and U35. A manual reset/power-on reset is also provided with manual reset occuring when the user pushes the momentary switch SW1.

Figure 2.4 is the 1K EPROM and device selection circuitry. For the memory, two Intersil IM6653AI 1Kx4 bit EPROMs were chosen. This was done for several reasons. First and foremost, this was the largest CMOS EPROM available at the time. This arrangement allowed for very simple device selection and a reduction in parts count. Secondly, it became apparent that this device was also the fastest CMOS EPROM available on the market (300 nanoseconds access time). The 1Kx8 bit RAM circuit appears in Figure 2.5. It consists of eight Intersil IM6518A-1s 1Kx1 bit CMOS RAMs. As with the EPROMs above, the primary reasons for choosing this particular device was its CMOS construction and its brief access time (95 nsec maximum!). This access time is far faster than that necessay, but it may be needed when faster versions of the NSC800 become available. Figure 2.6 shows the hardware multiply and divide circuit. It is made up of two RCA CDP1855s, U14-15, which are cascaded to allow 16-bitx16-bit unsigned multiplies and 32-bit/16-bit unsigned divides.

The analog interfaces of the NSC800 SBC can be found in Figure 2.7. The analog input consists of a National Semiconductor ADC 1210 12-bit A/D converter, U16. This is a successive approximation A/D converter that has been configured for bipolar signal input with complementary logic on the output. The input range is -2.5 volts to +2.5 volts. The converter is free running with the binary output latched into two 74C374 CMOS eight-bit

latches. Latching of the A/D converter output into the latches is conditional upon the reading of the latches by the CPU. The latches are addressed in the memory map so as to take advantage of the low byte/high byte orientation of the NSC800 instruction set's addressing scheme. Note that the addressing technique used results in a savings in component count at a penalty of possible bus contention of the byte latches. Therefore, one must be careful to guarantee that this will not occur. The analog output electronics consists of an Analog Devices AD7524 8-bit Buffered Multiplying D/A converter. It has been configured for bipolar operation with offset binary code input. The user should note this type of data scheme and adjust his data accordingly. BNC connectors are provided for both analog input and output and are so labeled.

The final schematic, Figure 2.8, consists of two 8-bit parallel I/O data ports. Each port is made up of two 74C373 CMOS 8-bit data latches. Both data latches are accessed by the same address code, but one is for input only and the other is for output only. The port lines are available through connector J2 and J3 for I/O ports 1 and 2 respectively.

From the design of this single board computer, a memory map was generated. This memory map is shown in Table 2.2 . Note that unused address lines for the MDU, A/D and D/A converters, and I/O ports are treated as logic zeros. In fact, these devices can be accessed at many locations in their corresponding decoder segments (refer to the discussion of the 3-to-8 decoder above). In order to avoid confusion, the above-mentioned convention is adopted in this text for programming exercises. Note that there are spare select lines. When these are used

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

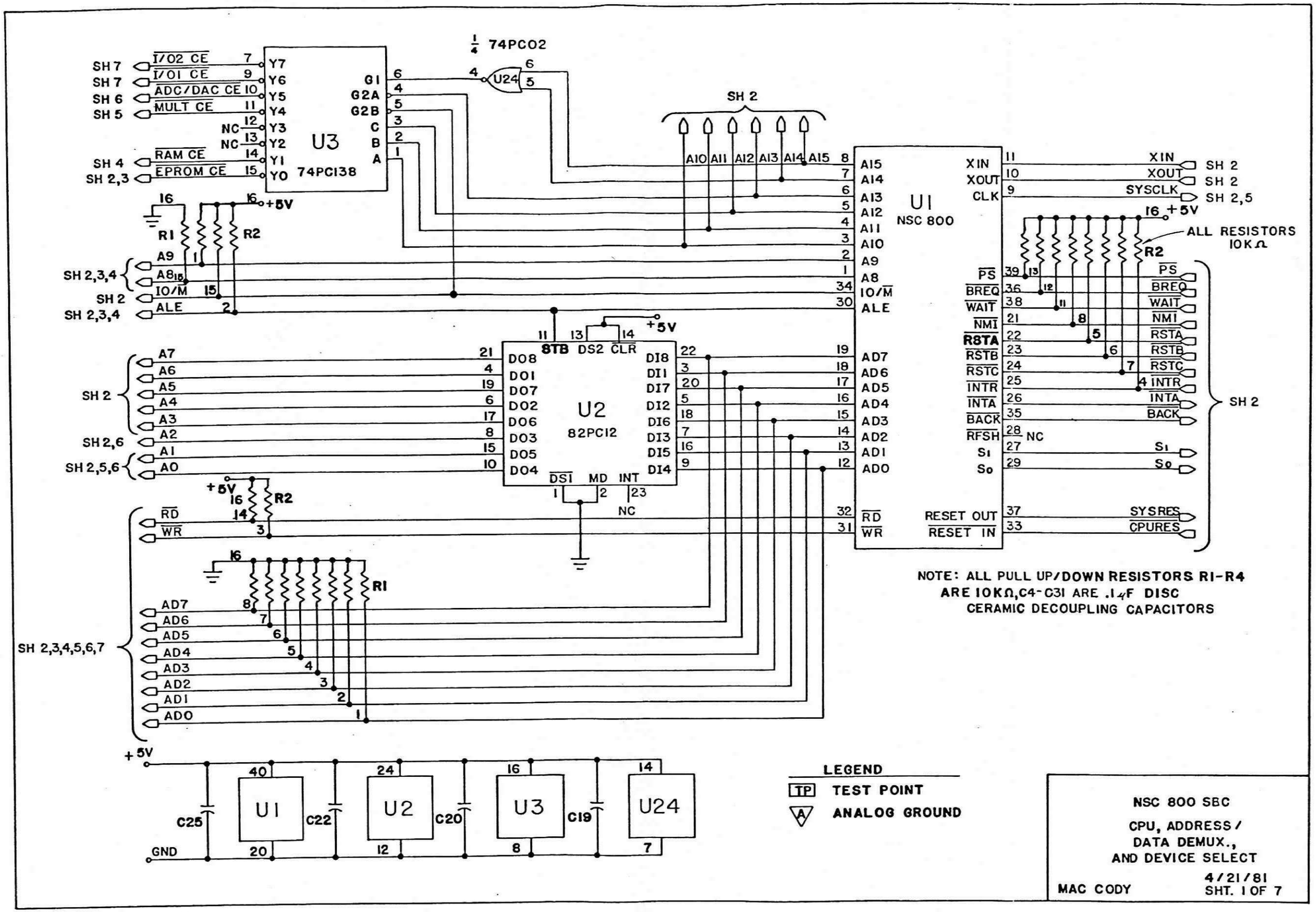
U3
74PC138
U1
NSC 800
U2
82PC12
1/4 74PC02
U24
SH 2
SH 7
SH 6
SH 5
SH 4
SH 2,3
SH 2,3,4
SH 2,5,6
SH 2,6
SH 2,3,4,5,6,7
SH 2,5
I/O2 CE
I/O1 CE
ADC/DAC CE
MULT CE
RAM CE
EPROM CE
+5V
R1
R2
ALL RESISTORS 10KΩ
XIN
XOUT
CLK
SYSCLK
PS
BREQ
WAIT
NMI
RSTA
RSTB
RSTC
INTR
INTA
BACK
RFSH
S1
S0
RESET OUT
RESET IN
SYSRES
CPURES
RD
WR
STB
DS2
CLR
DS1
MD
INT
NC
GND
C25
C22
C20
C19
NOTE: ALL PULL UP/DOWN RESISTORS R1-R4 ARE 10KΩ, C4-C31 ARE .1μF DISC CERAMIC DECOUPLING CAPACITORS
LEGEND
TP TEST POINT
A ANALOG GROUND
NSC 800 SBC
CPU, ADDRESS/ DATA DEMUX., AND DEVICE SELECT
MAC CODY
4/21/81
SHT. 1 OF 7

FIGURE 2.2

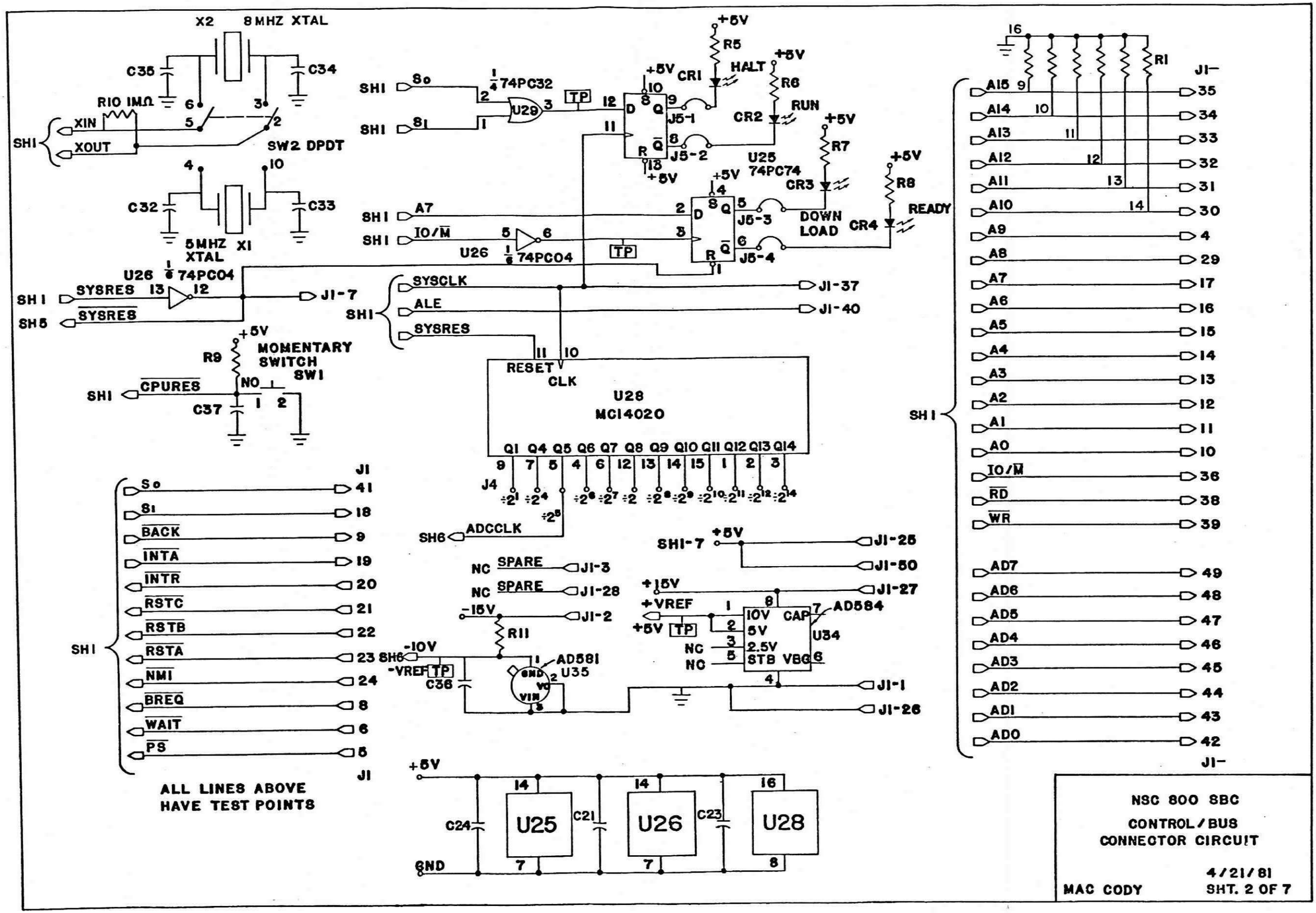
X2 8MHZ XTAL
C35
C34
R10 1MΩ
XIN
XOUT
SW2 DPDT
C32
C33
5MHZ XTAL X1
U26 1/6 74PC04
SH1 SYSRES
SH5 SYSRES
J1-7
R9
MOMENTARY SWITCH SW1
SH1 CPURES
C37
SH1 S0
SH1 S1
1/4 74PC32
U29
TP
+5V
R5
CR1
HALT
R6
CR2
RUN
J5-1
J5-2
U25 74PC74
R7
CR3
DOWN LOAD
R8
CR4
READY
J5-3
J5-4
SH1 A7
SH1 IO/M
U26 1/6 74PC04
SYSCLK
ALE
SYSRES
J1-37
J1-40
RESET
CLK
U28
MC14020
Q1 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14
J4
÷2¹ ÷2⁴ ÷2⁵ ÷2⁶ ÷2⁷ ÷2 ÷2⁸ ÷2⁹ ÷2¹⁰ ÷2¹¹ ÷2¹² ÷2¹⁴
SH6 ADCCLK
SH1-7
J1-25
J1-50
NC SPARE J1-3
NC SPARE J1-28
-15V J1-2
R11
-10V SH6
-VREF
C36
AD581 U35
+15V J1-27
+VREF
+5V
NC
NC
10V CAP
5V
2.5V
STB VBG
AD584 U34
J1-1
J1-26
SH1 S0 41
S1 18
BACK 9
INTA 19
INTR 20
RSTC 21
RSTB 22
RSTA 23
NMI 24
BREQ 8
WAIT 6
PS 5
J1
ALL LINES ABOVE HAVE TEST POINTS
R1
SH1 A15 35
A14 34
A13 33
A12 32
A11 31
A10 30
A9 4
A8 29
A7 17
A6 16
A5 15
A4 14
A3 13
A2 12
A1 11
A0 10
IO/M 36
RD 38
WR 39
AD7 49
AD6 48
AD5 47
AD4 46
AD3 45
AD2 44
AD1 43
AD0 42
J1-
C24
C21
C23
U25
U26
U28
GND
NSC 800 SBC
CONTROL/BUS
CONNECTOR CIRCUIT
MAC CODY
4/21/81
SHT. 2 OF 7

FIGURE 2.3

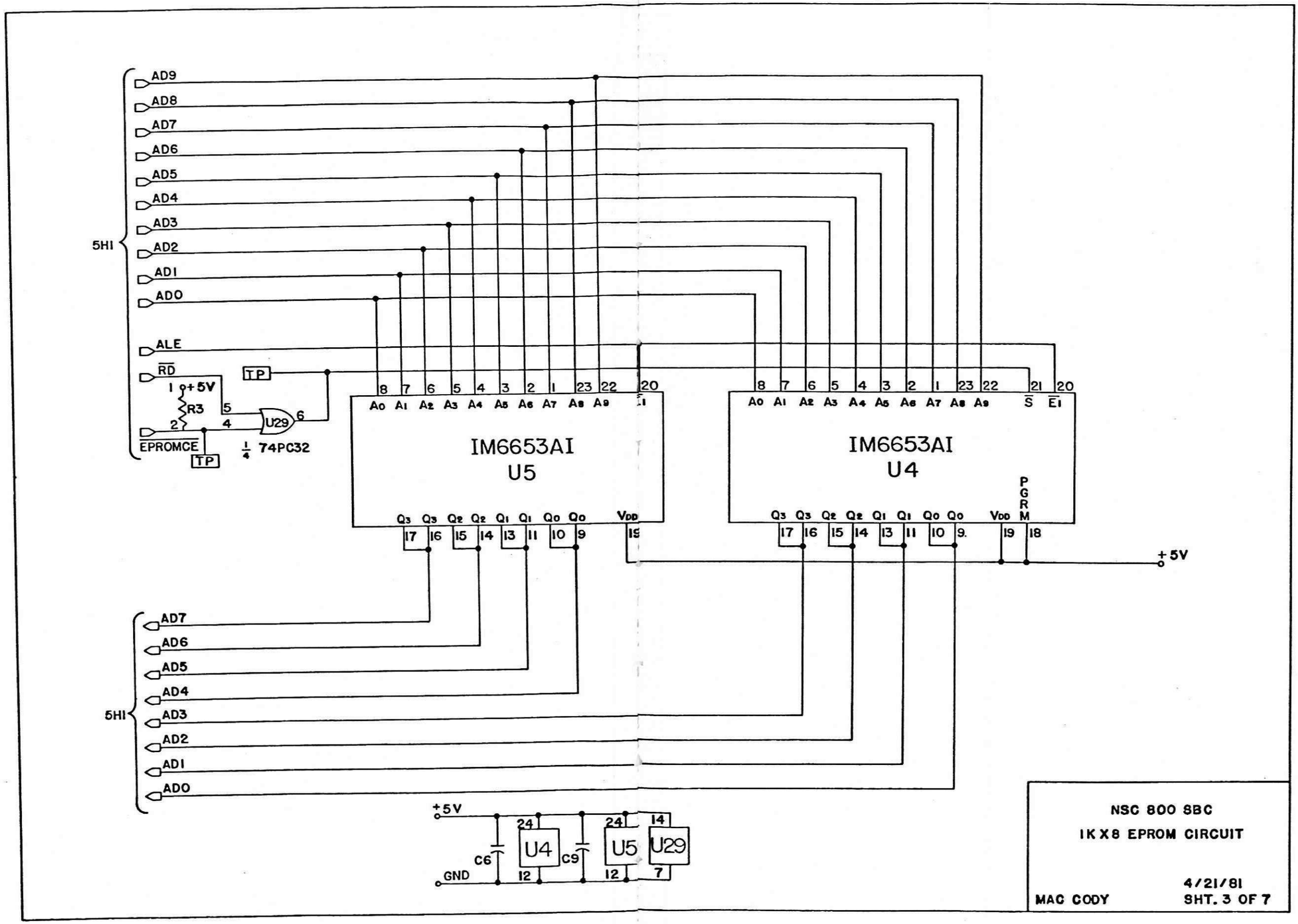
AD9
AD8
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
5HI
ALE
RD
TP
+5V
R3
U29
EPROMCE
TP
1/4 74PC32
IM6653AI
U5
IM6653AI
U4
PGRM
+5V
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
5HI
+5V
GND
C6
C9
U4
U5
U29
NSC 800 SBC
1K X 8 EPROM CIRCUIT
MAC CODY
4/21/81
SHT. 3 OF 7

FIGURE 2.4

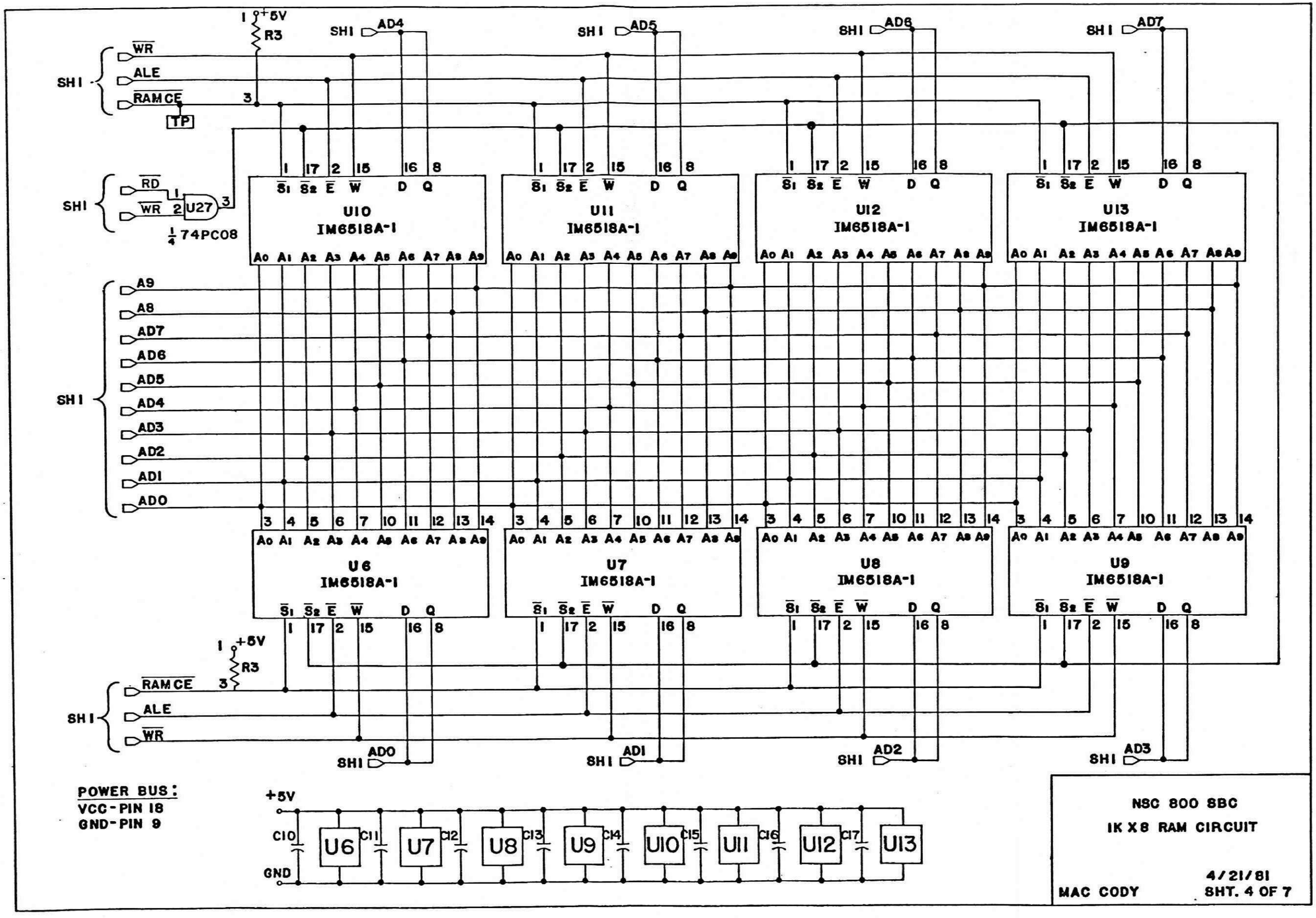
+5V
R3
SH1
WR
ALE
RAMCE
TP
AD4
AD5
AD6
AD7
RD
U27
1/4 74PC08
U10
IM6518A-1
U11
IM6518A-1
U12
IM6518A-1
U13
IM6518A-1
A9
A8
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
U6
IM6518A-1
U7
IM6518A-1
U8
IM6518A-1
U9
IM6518A-1
AD0
AD1
AD2
AD3
POWER BUS:
VCC-PIN 18
GND-PIN 9
+5V
GND
C10
C11
C12
C13
C14
C15
C16
C17
NSC 800 SBC
1K X 8 RAM CIRCUIT
MAC CODY
4/21/81
SHT. 4 OF 7

FIGURE 2.5

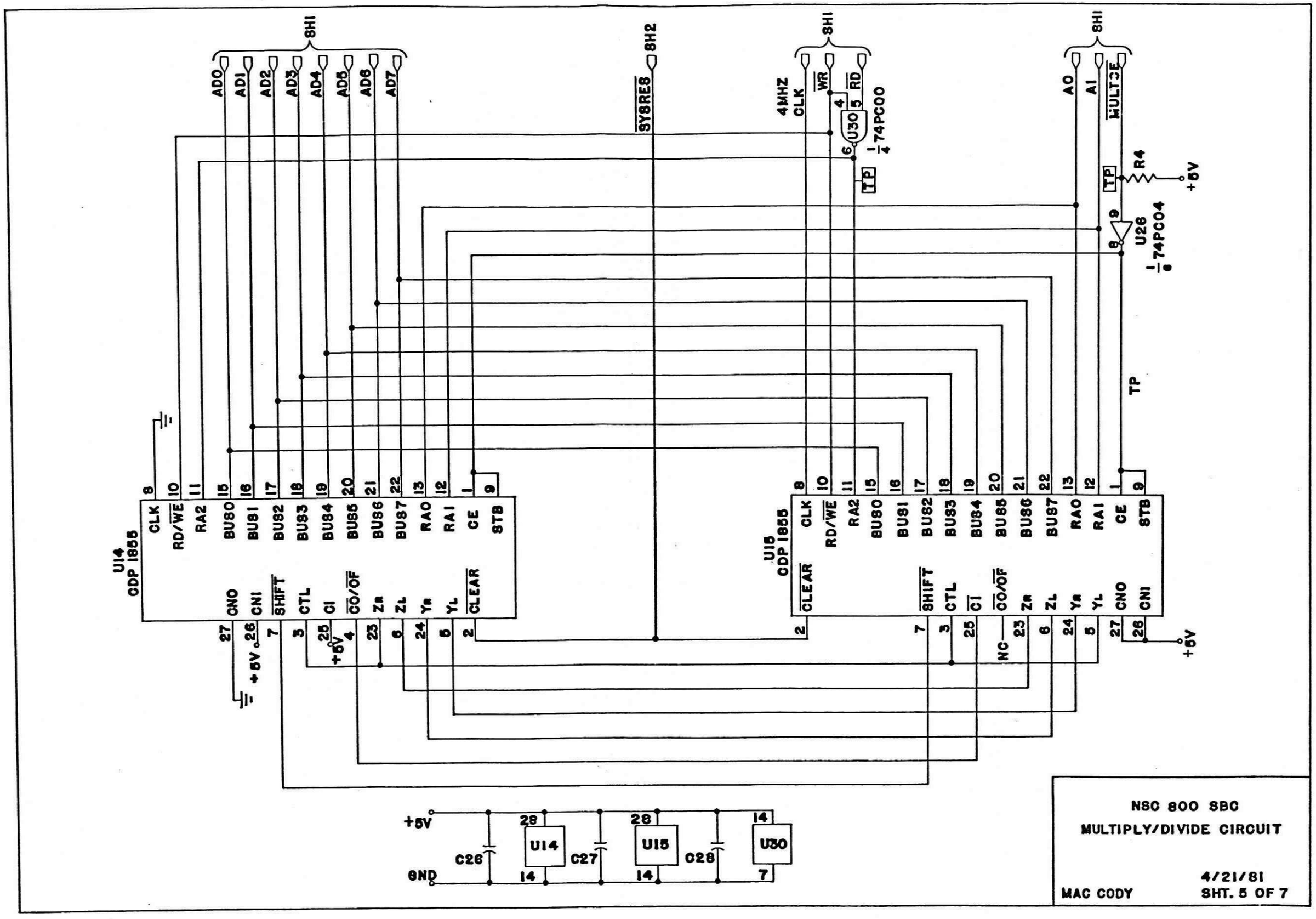
8H1
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
8H2
SYSRES
4MHZ CLK
WR
RD
U30
1/4 74PC00
TP
A0
A1
MULTCE
R4
+5V
U26
1/6 74PC04
U14
CDP 1855
CLK
RD/WE
RA2
BUS0
BUS1
BUS2
BUS3
BUS4
BUS5
BUS6
BUS7
RA0
RA1
CE
STB
CNO
CNI
SHIFT
CTL
CI
CO/OF
ZR
ZL
YR
YL
CLEAR
U15
CDP 1855
NC
GND
C26
C27
C28
U14
U15
U30
NSC 800 SBC
MULTIPLY/DIVIDE CIRCUIT
MAC CODY
4/21/81
SHT. 5 OF 7

FIGURE 2.6

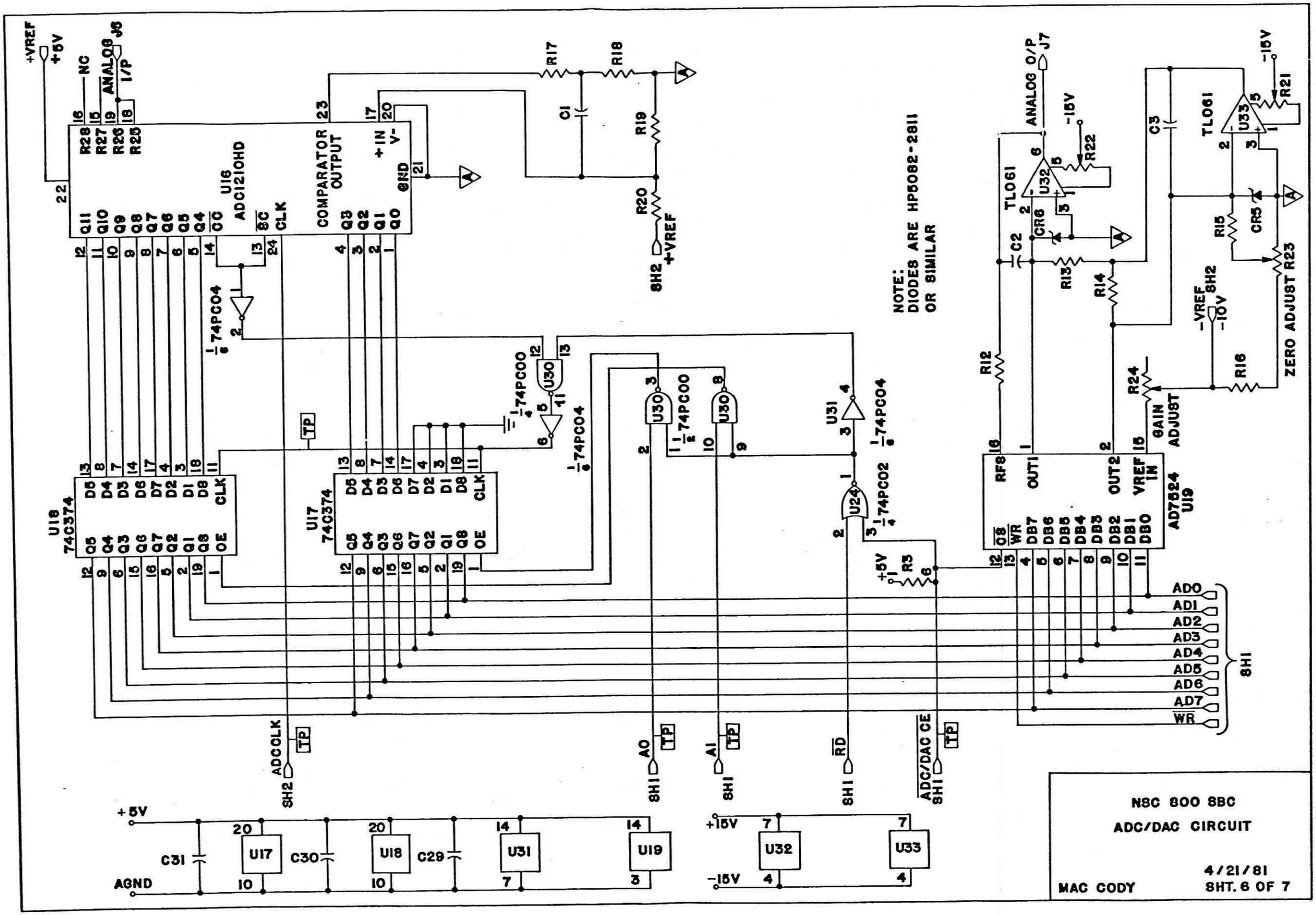
+VREF
+5V
NC
ANALOG I/P
J6
R28
R27
R26
R25
U16
ADC1210HD
COMPARATOR OUTPUT
+IN
V-
GND
CLK
R17
R18
C1
R19
R20
SH2
+VREF
1/6 74PC04
1/4 74PC00
U30
U31
U24
1/4 74PC02
TP
U18
74C374
U17
74C374
D5 D4 D3 D6 D7 D2 D1 D8
CLK
Q5 Q4 Q3 Q6 Q7 Q2 Q1 Q8
OE
ADCCLK
A0
A1
RD
ADC/DAC CE
SH1
NOTE:
DIODES ARE HP5082-2811
OR SIMILAR
ANALOG O/P
J7
-15V
R22
TL061
U32
CR6
C2
R13
R14
R12
C3
U33
R21
R15
CR5
R23
ZERO ADJUST
-VREF
-10V
R16
R24
GAIN ADJUST
RF8
OUT1
OUT2
VREF IN
AD7524
U19
CS
WR
DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
R3
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
WR
+5V
AGND
C31
C30
C29
U17
U18
U31
U19
+15V
-15V
U32
U33
NSC 800 SBC
ADC/DAC CIRCUIT
MAC CODY
4/21/81
SHT. 6 OF 7

FIGURE 2.7

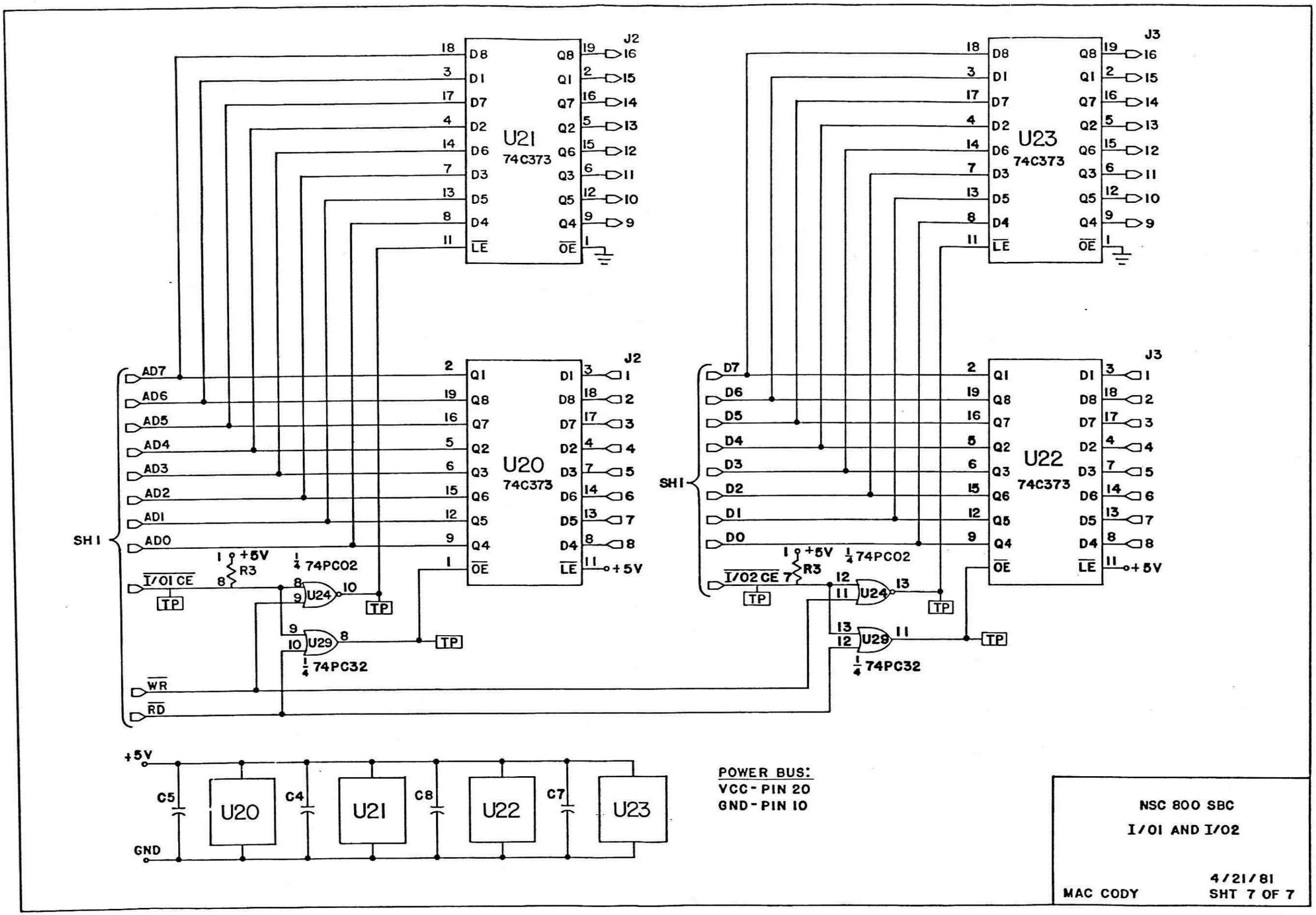
U21
74C373
U20
74C373
U23
74C373
U22
74C373
J2
J3
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
SH1
I/O1 CE
I/O2 CE
WR
RD
+5V
R3
1/4 74PC02
U24
U29
1/4 74PC32
TP
C5
C4
C8
C7
GND
POWER BUS:
VCC - PIN 20
GND - PIN 10
NSC 800 SBC
I/O1 AND I/O2
MAC CODY
4/21/81
SHT 7 OF 7

FIGURE 2.8

Table 2.1 Ribbon Cable Connector Pin Assignment

Pin	Function	Pin	Function
1	Gnd	26	Gnd
2	-15 volts	27	+15 volts
3	Spare pin	28	Spare pin
4	A9	29	A8
5	$\overline{PS}$	30	A10
6	$\overline{WAIT}$	31	A11
7	$\overline{SYSRES}$	32	A12
8	$\overline{BREQ}$	33	A13
9	$\overline{BACK}$	34	A14
10	A0	35	A15
11	A1	36	IO/$\overline{M}$
12	A2	37	SYSCLK
13	A3	38	$\overline{RD}$
14	A4	39	$\overline{WR}$
15	A5	40	ALE
16	A6	41	S0
17	A7	42	AD0
18	S1	43	AD1
19	$\overline{INTA}$	44	AD2
20	$\overline{INTR}$	45	AD3
21	$\overline{RSTC}$	46	AD4
22	$\overline{RSTB}$	47	AD5
23	$\overline{RSTA}$	48	AD6
24	$\overline{NMI}$	49	AD7
25	+5 volts	50	+5 volts

Table 2.2. Memory Map of the NSC800 SBC

START ADDRS	END ADDRS	DEVICE OPERATION READ	WRITE
0000H	03FFH	EPROM	[no operation]
0400H	07FFH	RAM	RAM
0800H	0BFFH	[spare]	[spare]
0C00H	0FFFH	[spare]	[spare]
1000H	-	MDU - X register	MDU - X register
1001H	-	MDU - Z register	MDU - Z register
1002H	-	MDU - Y register	MDU - Y register
1003H	-	MDU - status register	MDU - control register
1400H	-	ADC - no operation	DAC - latch (8-bit)
1401H	-	ADC - MSB (8 bits)	"
1402H	-	ADC - LSB (4 bits)	"
1403H	-	illeagal address	"
1800H	-	Input port 1	Output port 1
1C00H	-	Input port 2	Output port 2

(or if any modification to the memory map is made) it should be promptly noted so that possible bus contentions can be avoided.

Once the single board computer was designed, a printed circuit board was created. First, the components were arranged, over several trials, to obtain a reasonable package layout. This resulted in the board having segregated areas for CPU/control functions, memory, digital I/O, analog I/O, and hardware multiply/divide electronics. The component layout is illustrated in Figure 2.9. After this was accomplished, the master printed circuit artwork was made using Bishop Graphics 2X printed circuit layout materials. The package layout made a central bus structure possible. All the electronics sections could then be easily reached from the bus. A copious amount of test points were also added to ease circuit troubleshooting. The master was composed of three 2x masks. One was composed of all component and through board connections which were common to both sides of the board. The other two consisted of the wiring traces for the circuit side of the board or circuit traces and designator lettering for the component side of the board. To make one complete component side or circuit side mask, it was necessary to photograph the proper side mask with the common mask layered on it. This assured positive alignment between the pads on both sides of the PC board. From the 1x photoreduction, a plated-through PC board was made with satisfactory results. Figures 2.10a and b show the two sides of the finished PC board. A list of the components used in the computer is given in Table 2.3. Directions for assembling the single board computer are given below:

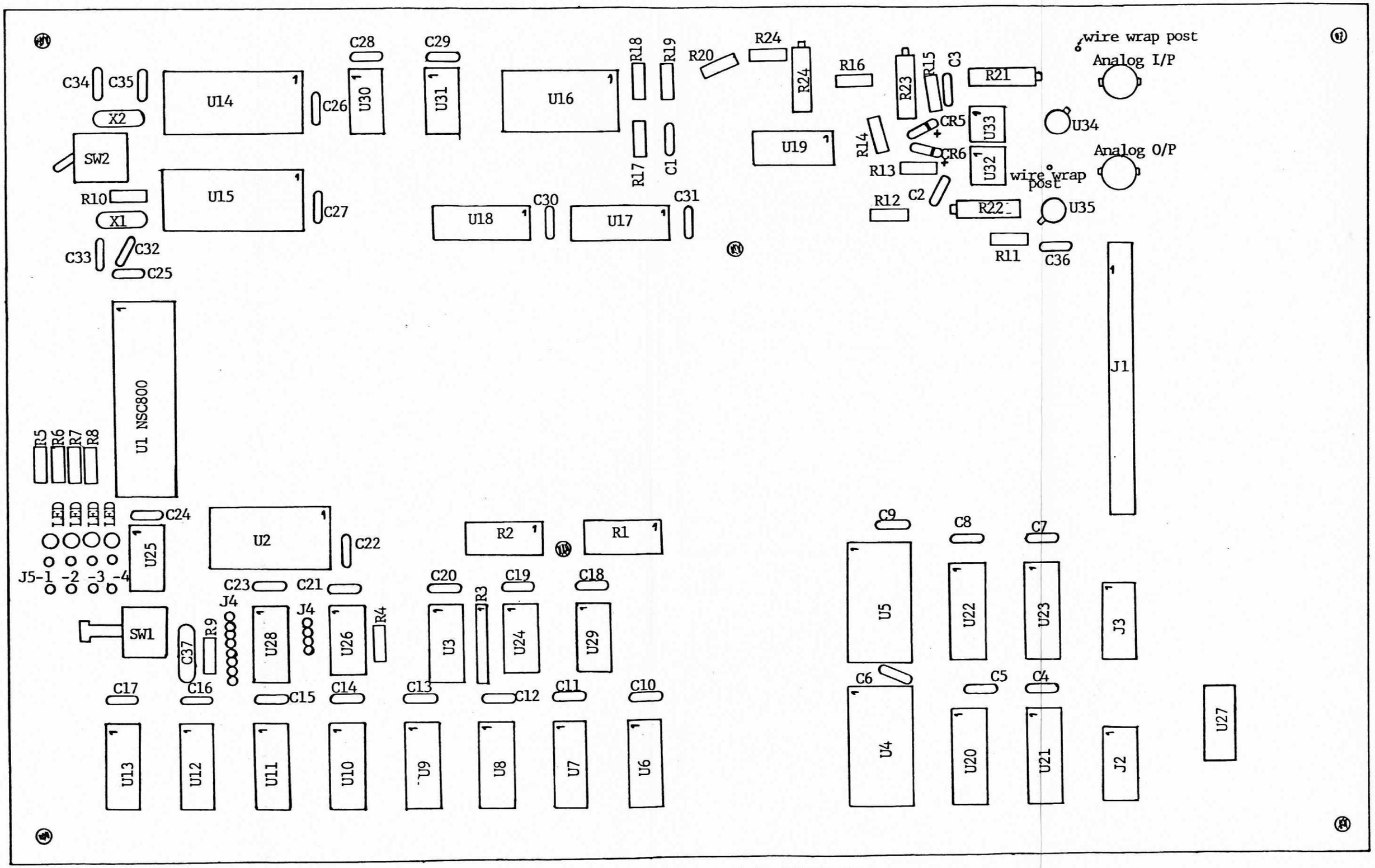

Figure 2.9: Component Layout of the NSC800 SBC

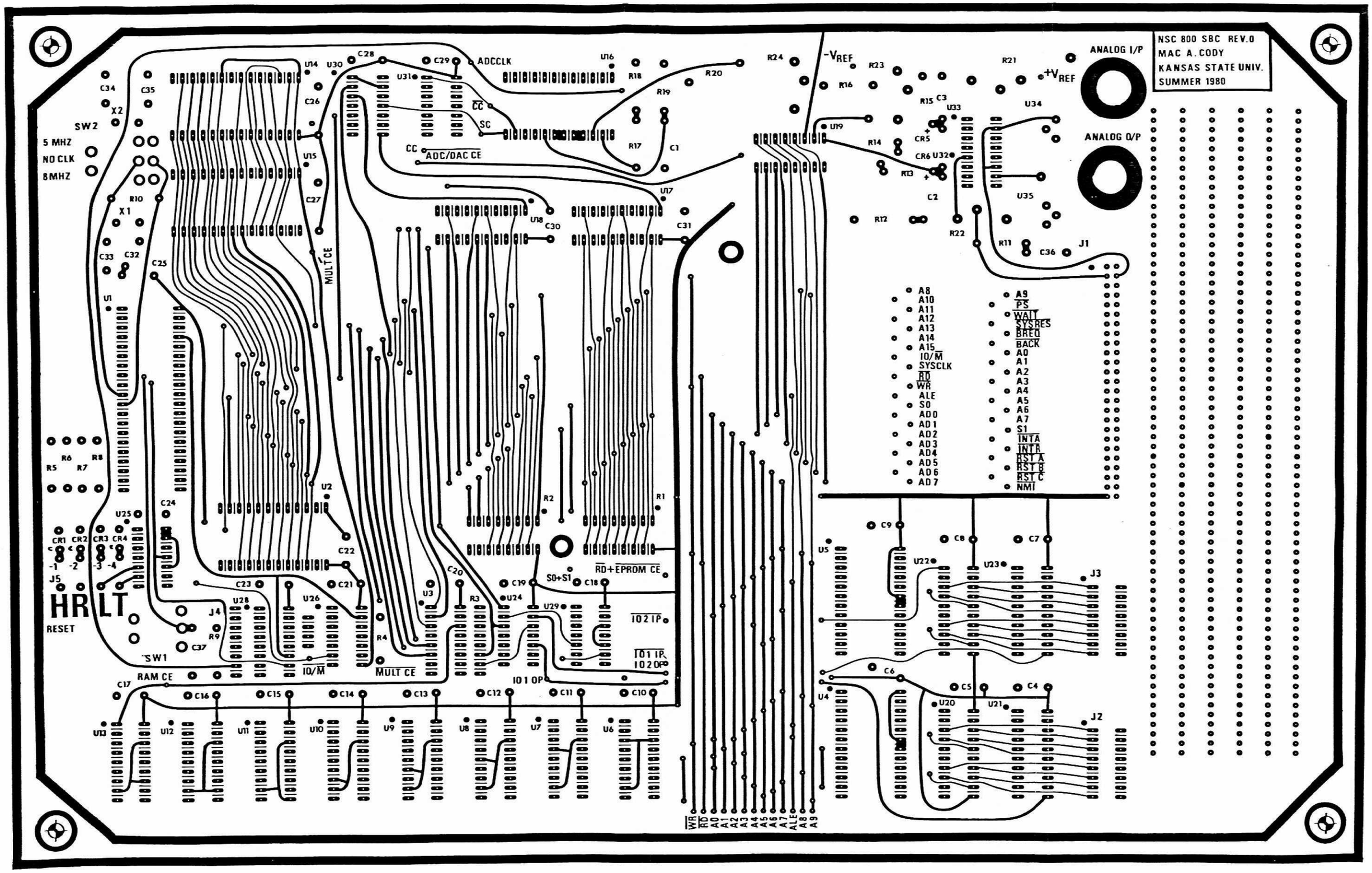

FIGURE 2.10a: NSC800 SBC PRINTED CIRCUIT BOARD (COMPONENT SIDE)

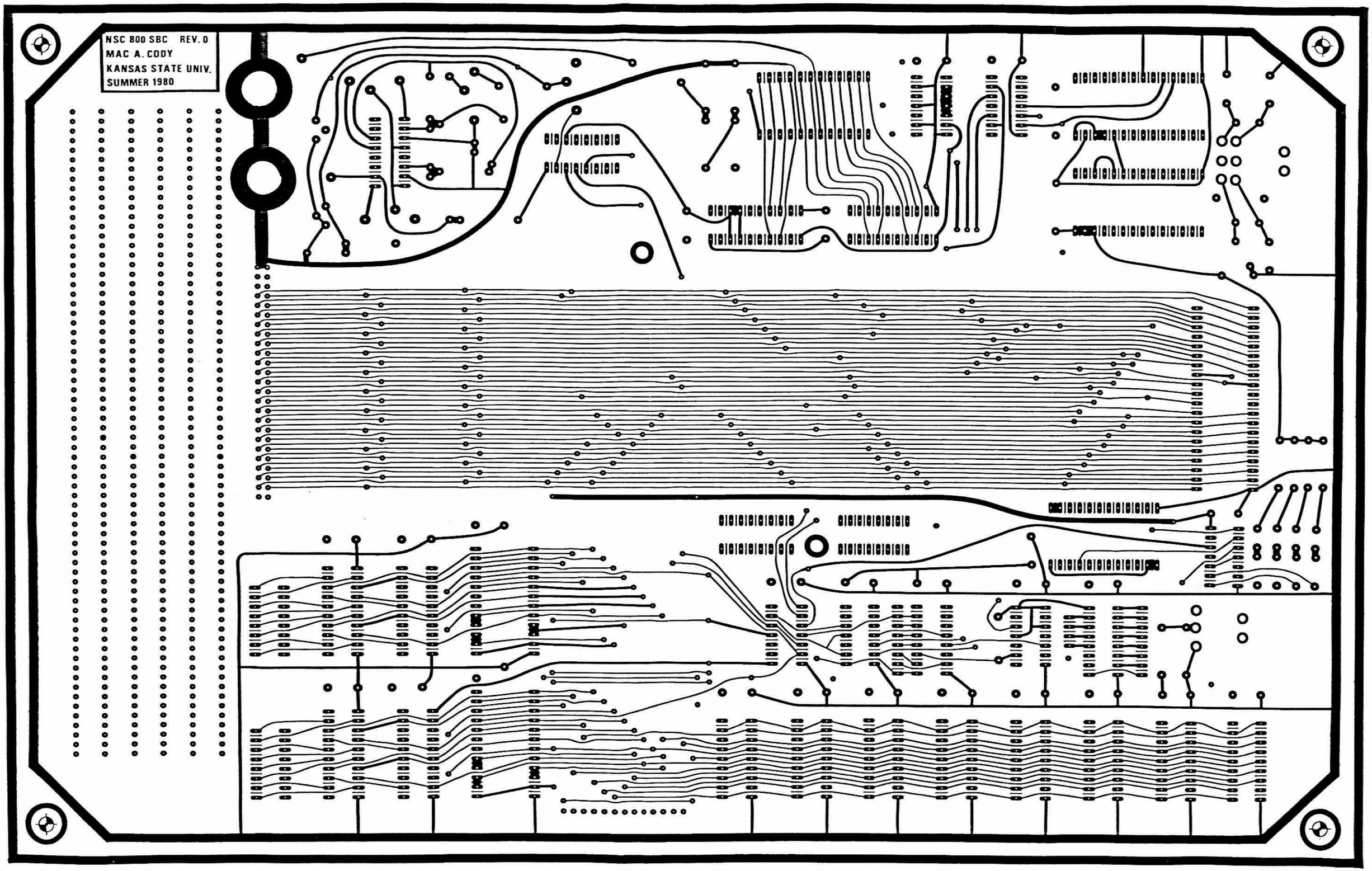

FIGURE 2.10b: NSC800 SBC PRINTED CIRCUIT BOARD (CIRCUIT SIDE)

Table 2.3: Component List for the NSC800 SBC

Integrated Circuits

U1 - NSC800 microprocessor

U2 - 82PC12 8-bit Input/Output Port

U3 - 74PC138 3-Line to 8-Line Decoder/Demultiplexer

U4, U5 - Intersil IM6653AI 1024x4 bit CMOS EPROM

U6 - U13 - Intersil IM6518A-1 1024x1 bit CMOS RAM

U14, U15 - RCA CDP1855 Multiply/Divide Unit

U16 - National Semiconductor AD1210HD 12-Bit CMOS A/D Converter

U17, U18 - 74C374 Octal D-Type Flip-Flop

U19 - Analog Devices AD7524 8-Bit Buffered Multiplying D/A Converter

U20 - U23 -74C373 Octal D-Type Latch

U24 - 74PC02 Quad 2-Input NOR Gate

U25 - 74PC74 Dual D Flip-Flop

U26, U31 - 74PC04 Hex Inverter

U27 - 74PC08 Quad 2-Input AND Gate

U28 - Motorola MC14020 14-Stage Ripple Carry Binary Counter

U29 - 74PC32 Quad 2-Input OR Gate

U30 - 74PC00 Quad 2-Input NAND Gate

U32, U33 - Texas Instruments TL061 Low-Power JFET-Input Operational Amplifier

U34 - Analog Devices AD584 Pin Programmable Precision Voltage Reference

U35 - Analog Devices AD581 High Precision 10 Volt IC Reference

Diodes

CR1 - CR4 - Light Emitting Diodes, Red

CR5, CR6 - Hewlitt Packard HP5082-2811 Schottky Diode

Table 2.3: Components List (continued)

Capacitors

C1 - 100 picofarad disc ceramic

C2, C3, C32 - C35 - 10 picofarad disc ceramic

C4 - C31 - 0.1 microfarad disc ceramic

C36 - 0.01 microfarad disc ceramic

C37 - 0.22 microfarad

Resistors

R1, R2 - 10 Kohm 15-Resistor DIP, common node at pin 16, (Bourns 16-2-103)

R3 - 10 Kohm 7-Resistor SIP, common node at pin 1, (Bourns 08-1-103)

R4, R19 - 10 Kohm, 1/4 watt resistor

R5 - R8 - 300 ohm, 1/4 watt resistor

R9, R10, R15 - 1 Mohm, 1/4 watt resistor

R11 - 2.4 Kohm $\pm$ 5%, 1/4 watt resistor

R12, R18 - 1 Kohm, 1/4 watt resistor

R13, R14 - 5 Kohm, 1/4 watt resistor (must match 0.1% or better)

R16 - 100 Kohm, 1/4 watt resistor

R17 - 200 Kohm, 1/4 watt resistor

R20 - 30 Kohm, 1/4 watt resistor

R21 - R23 - 10 Kohm trimpot (Bourns 3006P-1-103)

R24 - 2 Kohm trimpot (Bourns 3006P-1-202)

Miscellaneous Parts

J1 - 50-pin male ribbon cable connector with wire wrap pins (3M No. 3433)

J2, J3 - 16-pin wire wrap socket

J4 - twelve double-ended wire wrap pins (Vector No. T46)

J5 - eight wire wrap socket pins (Vector No. R32)

J6, J7 - double-ended wire wrap pin (Vector No. T46) and
a Female BNC socket (74868UG-1094/UG)

Table 2.3: Component List (continued)

SW1 - SPST momentary push-button switch (C & K 8121)

SW2 - DPDT three-position toggle switch (C & K 7203)

X1 - 5 MHz crystal

X2 - 8 MHz crystal (alternate, 1 MHz crystal)

One 14-pin wire wrap socket (for U27).

Four wire wrap socket pins for the crystal sockets (Vector No. R32)

One 40-pin solder tab socket

Two 28-pin solder tab sockets

Four 24-pin solder tab sockets

Six 20-pin solder tab sockets

Eight 18-pin solder tab sockets

Four 16-pin solder tab sockets

Five 14-pin solder tab sockets

Two 3-pin solder tab sockets (Augat No.8058-1G23)

NSC800 SBC Construction:

Printed Circuit Board Corrections (part 1):

1) Cut trace between pin 4 of U24 and pin 4 of U3. Cut trace between pin 6 of U3 and A13 of the SBC bus.

2) Cut trace between pin 3 of U34 and R24.

3) Cut trace between pin 16 of R1 and Vcc.

4) Cut trace between pin 3 of U30 and pin 1 of U18. Cut trace between pin 8 of U30 and pin 1 of U17.

5) Cut trace between pin 11 of U30 and the $\overline{SC}$ test point.

6) Cut trace between pin 1 and pin 13 of U30. Cut trace between pin 13 and pin 9 of U30. Cut trace between pin 12 of U30 and A2 of the SBC bus.

7) Cut traces between pins 2, 3, 4, and 5 of U31.

8) Cut the trace between the ADCCLK test point and the seventh jumper pad of J4 (left side of U28).

9) Cut all traces connecting together every pin 1 and pin 17 of U6 - U13.

Component Placement:

1) Solder all sockets and jumper pins into the designated positions on the component side of the printed circut board (the side with the designator lettering). For convenience, place the sockets with their pin 1 designators over the dots on the circuit board. These designate pin 1 also.

2) Solder in discrete components. Note that CR5 and CR6 are placed with the banded ends over the plus signs printed on the circuit board. Note that the LEDs are placed with the cathodes towards the bottom of the board (when the the designators are read normally).

3) Solder wire wrap pins into the twelve holes of J4 (eight on the left and four on the right of U28). Solder wire wrap pins into the holes near the BNC connectors. One goes to pin 6 of U32 while the other goes to pin 19 of U16.

4) Solder in the switches, taking care to also solder the mounting tabs to the board. This insures good mechanical connection of the switches to the board.

5) Solder a 14-pin wire wrap socket into the lower section of the kludge space. This will hold U27.

Printed Circuit Board Corrections (part 2):

(30 gauge Kynar wire works fine here!)

1) Wire pin 24 of U4 to the Vcc pin of C6. Wire pin 14 of U30 to the Vcc pin of C28.

2) Wire pin 4 of U24 to pin 6 of U3. Wire pin 4 of U3 to pin 11 of R1 (A13 of the SBC bus).

3) Wire pin 16 of R1 to ground.

4) Wire pin 2 of U34 to Vref pin of R24.

5) Wire pin 3 of U30 to pin 1 of U17. Wire pin 8 of U30 to pin 1 of U18.

6) Wire pin 13 of U16 to pin 14 of U16.

7) Wire pin 2 of U31 to pin 12 of U30. Wire pin 9 of U30 to pin 3 of U31. Wire pin 4 of U31 to pin 13 of U30. Wire pin 11 of U30 to pin 5 of U31.

8) Wire ADCCLK test point to the fifth jumper pad of J4.

9) Wire together all pin 1's of U6 - U13 to $\overline{\text{RAMCE}}$. Wire together all pin 17's of U6 - U13 to pin 3 of U27.

10) Wire pin 1 of U27 to pin 39 ($\overline{\text{RD}}$) of J1 and pin 2 of U27 to pin 40 ($\overline{\text{WR}}$) of J1. Wire pins 4, 5, 9, 10, 12, and 13 of U27 to pin 7 (ground) of U27.

Optional Correction (allows use of a CDP1852 in place of the 82PC12):

1) Cut trace between pin 23 and pin 24 of U2.

2) Wire pin 23 of U2 to pin 12 of U2 (ground).

Completion of the Assembly:

1) Insert the integrated circuits into their designated sockets carefully. Remember to place the I.C.s with their number one pins placed in the corresponding holes of the sockets. They are designated by the dot on the circuit board.

2) Solder together pin 1 and pin 2 of U34, leaving enough length in pin 2 to insert it into the socket. Insert the I.C. into the socket with pin 4 to pin 1 of the socket (ground), pin 2 to pin 2 of the socket (Vref), and pin 8 to pin 3 of the socket (+Vs). All other pins of U34 are unconnected.

3) Insert the crystals into their proper sockets.

4) Insert jumper wires into the pin sockets of J5 to enable lighting LEDs.

5) Solder lengths of wire wrap wire to the center terminals of the BNC connectors. Wrap the wire connected to the Analog Input BNC to the wire wrap post which is soldered to the trace going to pin 19 of U16. Wrap the wire connected to the Analog Output BNC to the wire wrap post soldered to the trace going to pin 6 of U32.

Test Software for the NSC800 SBC:

In order assure proper operation of the computer, several test programs have been written. These should be run to test the computer before proceeding with any other software development. These test programs are:

1) SYSTST - A general test of the major subsystems of the computer. With

this program, correct operation of the CPU, EPROM, RAM, D/A converter, and I/O ports can be determined. This is a go/no go test and must be passed successfully before any further tests are attempted. The routine initializes a counter and a pointer to RAM. Next, the counter value is stored out to RAM and then read back from the same location. The CPU then writes out to the multiply/divide units (which are not inserted in their sockets) and the D/A converter. To test the parallel I/O ports, the counter value is written out to I/O Port 1. The output lines have been wired to the input lines with corresponding bits paired together. This allows the CPU to read the counter value back in through the input port of I/O Port 1. The same operation is repeated with I/O Port 2. Finally, the counter value and the RAM pointer are incremented, with the pointer conditioned to always point at the RAM space. If the program is executing correctly, then a saw-tooth waveform will appear on the output of the D/A converter.

2) ANATST - This routine tests the A/D converter for proper operation. It also allows for adjusting the A/D and D/A converter circuits for zero offset. During execution, the CPU initializes a delay counter and then enters a delay loop. After the delay is completed, the CPU reads the high byte of the A/D latches and immediately writes the value to the D/A converter. The program continues in a endless loop.

3) MULTST - This routine tests the RCA CDP1855 Multiply/Divide Unit (MDU) for correct multiply operation. The MDU is initialized and a dummy multiplicand (FFFF hexidecimal) loaded into the internal registers of the NSC800. Next, a 16-bit multiplier is read in from the parallel I/O ports. The high byte is from I/O Port 2 and the low byte is from I/O Port 1. Both

operands are loaded into the MDUs and the multiply operation started. After a short wait, a 32-bit product is read from the MDUs into the registers of the CPU. The routine then enters an endless loop which reads the to input ports, checking for all bits 'high' or 'low'. If they are all 'high', the most significant word of the product is written out to the output ports. If the bits are all 'low', the least significant word is output. The multiply test is started by pressing the reset button.

4) DIVTST - This routine tests the divide operation of the MDU. Its execution format is nearly identical to the MULTST routine. Instead, of a 16-bit dummy value, a 32-bit dummy dividend (10000000 hexidecimal) is created. After loading the dividend and the 16-bit divisor (again from the I/O ports), a divide operation is performed. The quotient and remainder are output to the I/O ports when the input port bits are all low or high respectively. The divide test is again started by pressing the reset button.

5) TIMTST - This routine was developed in order to test the access timing of the multiply/divide units. There was some question as to whether they were operating correctly or not. The program initializes a counter and a pointer to the MDUs. The value of the counter is written to one of the bytes of one of the MDU registers and the counter incremented. The MDU sequence counter is reset and the values read from the register with each value also written to the D/A converter. The MDU pointer is then incremented and checked to see if it is still pointing to the X, Y, or Z register of the MDU. If not, the low byte of the pointer is reset to zero. This routine operates in an endless loop and correct operation is recognized if a 'stepped' saw-tooth waveform appears on the output of the

D/A converter.

The assembly language listings of these programs are given below:

```
PASS 1 DONE
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0001
 ADDR   CODE        STMT SOURCE STATEMENT

                    0001 ;SYSTST (SYSTEM TEST) IS A SHORT PROGRAM TO 'EXERCISE'
                    0002 ;ALL THE NSC800 SBC SUBSYSTEMS.  IT IS THE FIRST PROGRAM
                    0003 ;TO BE RUN TO DETERMINE IF THE COMPUTER IS FUNCTIONING
                    0004 ;PROPERLY.
                    0005 ;
 >0000              0006        ORG 0000H
'0000  210004       0007 SYSTST LD HL,0400H ;POINT TO START OF RAM.
'0003  0600         0008        LD B,00H ;INITIALIZE THE COUNTER.
'0005  70           0009 LOOP   LD (HL),B ; STORE REGISTER B IN RAM.
'0006  7E           0010        LD A,(HL) ;GET BACK DATA FROM RAM.
'0007  320010       0011        LD (1000H),A ;STORE OUT TO MDU SPACE.
'000A  320014       0012        LD (1400H),A ;STORE OUT TO THE DAC.
'000D  320018       0013        LD (1800H),A ;STORE OUT TO I/O PORT 1.
'0010  3A0018       0014        LD A,(1800H) ;READ FROM I/O PORT 1.
'0013  32001C       0015        LD (1C00H),A ;STORE OUT TO I/O PORT 2.
'0016  3A001C       0016        LD A,(1C00H) ;READ FROM I/O PORT 2.
'0019  47           0017        LD B,A ;PUT RESULT BACK IN REGISTER B AND
'001A  04           0018        INC B ;INCREMENT.  IF ALL GOES WELL, THIS SHOULD
                    0019 ;              BE THE NEXT SEQUENTIAL NUMBER.  CORRECT
                    0020 ;              OPERATION ALSO HAS A SAWTOOTH WAVEFORM ON
                    0021 ;              ON THE OUTPUT OF THE DAC.
'001B  23           0022        INC HL ;POINT TO NEXT LOCATION IN RAM.
'001C  3E07         0023        LD A,07H ;MASK OFF LOW THREE BITS OF THE
'001E  A4           0024        AND H ;H REGISTER.  HL MUST ALWAYS POINT AT RAM.
'001F  2002         0025        JR NZ PNTROK ;IF RESULT NOT ZERO, POINTER IS OK.
'0021  3E04         0026        LD A,04H ;REINITIALIZE POINTER IF IT IS ZERO.
'0023  67           0027 PNTROK LD H,A ;PLACE RESULT IN H REGISTER.
'0024  C30500'      0028        JP LOOP ;REPEAT THE OPERATION.
                    0029        END

ERRORS=0000
```

```
PASS 1 DONE
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0001
 ADDR  CODE       STMT SOURCE STATEMENT

                  0001 ;THIS IS THE ANALOG HARDWARE TEST ROUTINE (ANATST).  IT
                  0002 ;IS DESIGNED TO DETERMINE IF THE A/D CONVERTER IS WORKING
                  0003 ;PROPERLY.  THIS ROUTINE CAN ALSO BE USED TO BALANCE THE
                  0004 ;A/D AND D/A CONVERTERS BY FEEDING THE OUTPUT OF THE D/A
                  0005 ;INTO THE INPUT OF THE A/D.
                  0006 ;
'0000  0620       0007 ANATST LD B,20H ;INITIALIZE COUNTER.
'0002  05         0008 WAIT   DEC B ;COUNT DOWN
'0003  C20200'    0009        JP NZ WAIT ;DONE WITH COUNT YET?
'0006  3A0214     0010        LD A,(1402H) ;READ HIGH BYTE OF A/D CONVERTER.
'0009  320014     0011        LD (1400H),A ;OUTPUT TO D/A CONVERTER.
'000C  C30000'    0012        JP ANATST ;REPEAT IT AGAIN.
                  0013        END

ERRORS=0000
```

```
PASS 1 DONE
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0001
 ADDR  CODE        STMT SOURCE STATEMENT

                   0001 ;      THIS PROGRAM TESTS THE MULTIPLY OPERATION OF THE
                   0002 ;CDP1855 MDU ON THE MSC800 SBC.  A 16 BIT OPERATOR GIVEN
                   0003 ;BY THE USER AT INPUT PORTS 1 & 2 IS MULTIPLIED BY A
                   0004 ;16 BIT OPERAND PROVIDED IN THE PROGRAM.  THE 32 BIT
                   0005 ;RESULT IS PROVIDED THROUGH THE OUTPUT PORTS 1 & 2
                   0006 ;TO THE USER IN 16 BIT BLOCKS.  THE HIGH WORD IS OUTPUT
                   0007 ;IF ALL THE BITS OF THE INPUT PORTS ARE HIGH (FFFFH).
                   0008 ;THE LOW WORD IS OUTPUT IF ALL THE BITS OF THE INPUT
                   0009 ;PORTS ARE LOW (0000H).  ALL OTHERS ARE IGNORED.
                   0010 ;      THE OPERATOR MUST BE SET BEFORE STARTING THE
                   0011 ;PROGRAM AS THE PROGRAM SUPPLIES NO PROMPTS.  WHEN THE
                   0012 ;MULTIPLICATION IS FINISHED, THE PROGRAM WILL OUTPUT
                   0013 ;A 00H TO THE DAC (IT WAS FIRST INITIALIZED TO FFH).
                   0014 ;THE PROGRAM ENTERS AN INFINITE LOOP WHILE OUTPUTTING
                   0015 ;THE HIGH OR LOW WORD ON REQUEST.
                   0016 ;
 >0000             0017        ORG 00H
'0000  3EFF        0018 MULTST LD A,0FFH ;INITIALIZE THE DAC TO FFH.
'0002  320014      0019        LD (1400H),A
'0005  DD210010    0020        LD IX,1000H ;SET UP POINTER TO MDU X REG.
'0009  DD360368    0021        LD (IX+3H),68H ;INITIALIZE MDU TO TWO UNITS, NO
                   0022 ;                      OPERATION, RESET SEQUENCE COUNTER
                   0023 ;                      AND Y REGISTER.
'000D  11FFFF      0024        LD DE,0FFFFH ;OPERAND IS SUPPLIED BY PROGRAM.
'0010  3A001C      0025        LD A,(1C00H) ;HIGH BYTE IS IN INPUT PORT 2.
'0013  67          0026        LD H,A
'0014  3A0018      0027        LD A,(1800H) LOW BYTE IS IN INPUT PORT 1.
'0017  6F          0028        LD L,A
'0018  DD7200      0029        LD (IX+0H),D ;LOAD OPERATOR INTO X REG.
'001B  DD7300      0030        LD (IX+0H),E
'001E  DD7401      0031        LD (IX+1H),H ;LOAD OPERATOR INTO Z REG.
'0021  DD7501      0032        LD (IX+1H),L
'0024  DD360361    0033        LD (IX+3H),61H ;RESET SEQUENCE COUNTER, START
                   0034 ;                     MULTIPLY OPERATION.
'0028  00          0035        NOP           ;WAIT FOR MULTIPLY TO END.
'0029  00          0036        NOP
'002A  00          0037        NOP
'002B  00          0038        NOP
'002C  00          0039        NOP
'002D  00          0040        NOP
'002E  00          0041        NOP
'002F  00          0042        NOP
'0030  00          0043        NOP
'0031  DD6602      0044        LD H,(IX+2H) ;PLACE MSB OF 32-BIT RESULT IN H REG.
'0034  DD6E02      0045        LD L,(IX+2H) ;MSB-1 IN L REG.
'0037  DD5601      0046        LD D,(IX+1H) ;MSB-2 IN D REG.
'003A  DD5E01      0047        LD E,(IX+1H) ;LSB IN E REG.
'003D  3E00        0048        LD A,00H ;SET DAC TO 00H.
'003F  320014      0049        LD (1400H),A
'0042  3A0018      0050 PORTST LD A,(1800H) ;GET PORT 1 BYTE.
'0045  47          0051        LD B,A ;SAVE IN REGISTER B.
'0046  3A001C      0052        LD A,(1C00H) ;GET PORT 2 BYTE.
'0049  A0          0053        AND B ;AND THE TWO BYTES TOGETHER.
'004A  CA5C00'     0054        JP Z LOWBTE ;OUTPUT LOW WORD TO PORTS 1 & 2.
'004D  3C          0055        INC A ;INCREMENT, IF NOT ZERO ...
'004E  C24200'     0056        JP NZ PORTST ;DON'T OUTPUT DATA.
'0051  7C          0057        LD A,H ;OUTPUT MSB BYTE TO PORT 2.
'0052  32001C      0058        LD (1C00H),A
```

```
                                                SD SYSTEMS Z80 ASSEMBLER PAGE 0002
 ADDR  CODE       STMT SOURCE STATEMENT

'0055  7D         0059        LD A,L ;OUTPUT MSB-1 BYTE TO PORT 1.
'0056  320018     0060        LD (1800H),A
'0059  C34200'    0061        JP PORTST ;CHECK FOR NEW WORD TO OUTPUT.
'005C  7A         0062 LOWBTE LD A,D ;OUTPUT MSB-2 TO PORT 2.
'005D  32001C     0063        LD (1C00H),A
'0060  7B         0064        LD A,E ;OUTPUT LSB TO PORT 1.
'0061  320018     0065        LD (1800H),A
'0064  C34200'    0066        JP PORTST ;CECK FOR NEW WORD TO OUTPUT.
                  0067        END

ERRORS=0000
```

```
PASS 1 DONE
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0001
 ADDR  CODE       STMT SOURCE STATEMENT

                  0001 ;      THIS PROGRAM PERFORMS A DIVISION TEST OF THE
                  0002 ;CDP1855 MDU ON THE MSC800 SBC.  A DIVISOR PROVIDED BY
                  0003 ;THE USER AT PORTS 1 & 2 IS DIVIDED INTO A DIVIDEND
                  0004 ;PROVIDED BY THE PROGRAM.  THE REMAINDER AND QUOTIENT ARE
                  0005 ;AVAILABLE FOR INSPECTION AT THE OUTPUT PORTS 1 & 2.
                  0006 ;THE QUOTIENT IS OUTPUT IF ALL BITS OF THE INPUT PORTS
                  0007 ;ARE HIGH (FFFFH).  THE REMAINDER IS OUTPUT IF ALL
                  0008 ;THE BITS ARE LOW (0000H).  ALL OTHER INPUTS ARE IGNORED.
                  0009 ;      THE DIVISOR MUST BE SET BEFORE RUNNING THE PROGRAM
                  0010 ;AS NO PROMPTS EXIST.  IF THE DIVISION IS SUCCESSFUL, THE
                  0011 ;DAC WILL GO FROM HIGH TO LOW VOLTAGE (FFH TO 00H).  IF
                  0012 ;AN OVERFLOW OCCURS, THE DAC OUTPUT WILL GO TO 80H.
                  0013 ;
'0000  3EFF       0014 DIVTST LD A,0FFH ;INITIALIZE DAC TO FFH.
'0002  320014     0015        LD (1400H),A
'0005  DD210010   0016        LD IX,1000H ;SET POINTER TO MDU REG. X.
'0009  DD360360   0017        LD (IX+3H),60H ;INITIALIZE THE MDU TO TWO UNITS,
                  0018 ;                       RESET SEQUENCE COUNTER, AND NO
                  0019 ;                       OPERATION.
'000D  210010     0020        LD HL,1000H ;LOAD HL WITH HIGH WORD OF DIVIDEND.
'0010  110000     0021        LD DE,0000H ;LOAD DE WITH LOW WORD OF DIVIDEND.
'0013  3A001C     0022        LD A,(1C00H) ;GET HIGH BYTE OF DIVISOR IN B REG.
'0016  47         0023        LD B,A
'0017  3A0018     0024        LD A,(1800H) ;GET LOW BYTE OF DIVISOR IN C REG.
'001A  4F         0025        LD C,A
'001B  DD7402     0026        LD (IX+2H),H ;LOAD MSW OF OPERAND IN Y REG.
'001E  DD7502     0027        LD (IX+2H),L
'0021  DD7201     0028        LD (IX+1H),D ;LOAD LSW OF OPERAND IN Z REG.
'0024  DD7301     0029        LD (IX+1H),E
'0027  DD7000     0030        LD (IX+0H),B ;LOAD 16-BIT OPERATOR IN X REG.
'002A  DD7100     0031        LD (IX+0H),C
'002D  DD360362   0032        LD (IX+3H),62H ;RESET SEQUENCE COUNTER AND START
                  0033 ;                       THE DIVIDE
'0031  00         0034        NOP ;WAIT FOR DIVIDE TO END.
'0032  00         0035        NOP
'0033  00         0036        NOP
'0034  00         0037        NOP
'0035  00         0038        NOP
'0036  00         0039        NOP
'0037  00         0040        NOP
'0038  00         0041        NOP
'0039  00         0042        NOP
'003A  DD6602     0043        LD H,(IX+2H) ;PUT REMAINDER IN H & L REGISTERS.
'003D  DD6E02     0044        LD L,(IX+2H)
'0040  DD5601     0045        LD D,(IX+1H) ;PUT QUOTIENT IN D & E REGISTERS.
'0043  DD5E01     0046        LD E,(IX+1H)
'0046  DD7E03     0047        LD A (IX+03H) ;GET STATUS BYTE
'0049  CB0F       0048        RRC A ;SET DAC TO 00H IF NO OVERFLOW, 80H IF ONE.
'004B  320014     0049        LD (1400H),A
'004E  3A0018     0050 PORTST LD A,(1800H) ;GET PORT 1 BYTE.
'0051  47         0051        LD B,A ;SAVE IN REGISTER B.
'0052  3A001C     0052        LD A,(1C00H) ;GET PORT 2 BYTE.
'0055  A0         0053        AND B ;AND THE TWO BYTES TOGETHER.
'0056  CA6800'    0054        JP Z LOWBTE ;OUTPUT LOW WORD TO PORTS 1 & 2.
'0059  3C         0055        INC A ;INCREMENT, IF NOT ZERO ...
'005A  C24E00'    0056        JP NZ PORTST ;DON'T OUTPUT DATA.
'005D  7C         0057        LD A,H ;OUTPUT MSB BYTE TO PORT 2.
'005E  32001C     0058        LD (1C00H),A
```

```
                                                SD SYSTEMS Z80 ASSEMBLER PAGE 0002
 ADDR  CODE       STMT SOURCE STATEMENT

'0061  7D         0059        LD A,L ;OUTPUT MSB-1 BYTE TO PORT 1.
'0062  320018     0060        LD (1800H),A
'0065  C34E00'    0061        JP PORTST ;CHECK FOR NEW WORD TO OUTPUT.
'0068  7A         0062 LOWBTE LD A,D ;OUTPUT MSB-2 TO PORT 2.
'0069  32001C     0063        LD (1C00H),A
'006C  7B         0064        LD A,E ;OUTPUT LSB TO PORT 1.
'006D  320018     0065        LD (1800H),A
'0070  C34E00'    0066        JP PORTST ;CECK FOR NEW WORD TO OUTPUT.
                  0067        END

ERRORS=0000
```

```
PASS 1 DONE
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0001
 ADDR  CODE        STMT SOURCE STATEMENT

                   0001 ;THIS PROGRAM PERFORMS A TIMING TEST OF THE CDP1855 MDU'S.
                   0002 ;A COUNTER AND MDU POINTER ARE INITIALIZED AND COUNTER
                   0003 ;VALUES ARE FIRST WRITTEN OUT TO AND READ FROM THE MDU
                   0004 ;REGISTERS.  THEN, THE VALUES ARE WRITTEN OUT TO THE D/A
                   0005 ;CONVERTER AND THE POINTER INCREMENTED.  THE PROCESS IS
                   0006 ;REPEATED.  CORRECT OPERATION IS OBSERVED IF A 'STEPPED'
                   0007 ;SAW-TOOTH WAVEFORM IS SEEN ON THE OUTPUT OF THE D/A
                   0008 ;CONVERTER.
                   0009 ;
 >0000             0010        ORG 0000H
'0000  210010      0011 TIMTST LD HL,1000H ;INITIALIZE THE MDU POINTER.
'0003  0600        0012        LD B,00H ;INITIALIZE THE COUNTER.
'0005  3E60        0013 LOOP   LD A,60H ;INITIALIZE THE MDU, RESET THE SEQUENCE
'0007  320310      0014        LD (1003H),A ;COUNTER.
'000A  70          0015        LD (HL),B ;STORE TO HIGH BYTE OF MDU REGISTER.
'000B  04          0016        INC B ;INCREMENT THE COUNTER.
'000C  70          0017        LD (HL),B ;STORE TO NEXT BYTE OF MDU REGISTER.
'000D  04          0018        INC B ;INCREMENT THE COUNTER.
'000E  70          0019        LD (HL),B ;STORE TO THIRD  BYTE OF MDU REGISTER.
'000F  04          0020        INC B ;INCREMENT THE COUNTER.
'0010  70          0021        LD (HL),B ;STORE TO LOW BYTE OF MDU REGISTER.
'0011  04          0022        INC B ;INCREMENT COUNTER FOR NEXT INTERATION.
'0012  320310      0023        LD (1003H),A ;RESET SEQUENCT COUNTER AGAIN.
'0015  7E          0024        LD A,(HL) ;READ HIGH BYTE OF MDU REGISTER.
'0016  320014      0025        LD (1400H),A ;OUTPUT VALUE TO D/A CONVERTER.
'0019  7E          0026        LD A,(HL) ;READ NEXT BYTE OF MDU REGISTER.
'001A  320014      0027        LD (1400H),A ;OUTPUT VALUE TO D/A CONVERTER.
'001D  7E          0028        LD A,(HL) ;READ THIRD BYTE OF MDU REGISTER.
'001E  320014      0029        LD (1400H),A ;OUTPUT VALUE TO D/A COMVERTER.
'0021  7E          0030        LD A,(HL) ;READ LOW BYTE OF MDU REGISTER.
'0022  320014      0031        LD (1400H),A ;OUTPUT VALUE TO D/A CONVERTER.
'0025  2C          0032        INC L ;POINT TO NEXT MDU REGISTER.
'0026  3E03        0033        LD A,03H ;CHECK FOR POINTING TO THE MDU
'0028  BD          0034        CP L      ;STATUS REGISTER.
'0029  C20500'     0035        JP NZ LOOP ;IF POINTER IS NOT, REPEAT LOOP.
'002C  2E00        0036        LD L,00H ;IF IT IS, RESET LOW BYTE OF POINTER.
'002E  C30500'     0037        JP LOOP ;NOW REPEAT THE LOOP.
                   0038        END

ERRORS=0000
```

CHAPTER III

SYSTEM HARDWARE EVALUATION

As the first part of the evaluation of the NSC800, a study was made of its performance as a component of the single board computer. This, in turn, required evaluation of the computer itself; which was divided into three parts: 1) the printed circuit board, 2) the computer circuitry outside of the NSC800, and 3) the NSC800.

The printed circuit board was conceived as a time-saving and cost-effective alternative to wire wrap board construction. Due to the complexity of the computer's design, it was decided that it would be far easier to make and correct a printed circuit board than it would be to hunt through the tangle of wires on a wire wrap board to find a mistake. This proved to be the case as the board, though not perfect, had relatively few errors. Most of these were easy to correct. The use of many test points on the board aided in debugging the finished system and was responsible for uncovering most of the design errors in the computer. In no instance were the corrections on the board due to substandard construction techniques. The artificers at Owens Printed Circuit Inc., Wichita, Kansas, should be complimented for their quality work in the fabrication of this printed circuit board.

As with the printed circuit board, the design of the single board computer was largely successful. The only subsystem which required any change was the A/D converter section. This consisted of changing the

reference voltage to +5 volts, altering the output latches to a high byte/low byte orientation in the memory map, and connecting the A/D converter for free running operation (see Chapter II). The first alteration was due to conditions imposed by the NSC800. Although the NSC800 was designed for an operational power supply voltage range of 3 to 12 volts, the engineering evaluation sample provided to us had an operational range of only 4 to 8 volts. This required the adjustment to 5 volt operation. In addition, the lack of a sample of the 82PC12 required the use of an Intel 8212, which also operates at only 5 volts. The change to the low byte/high byte orientation of the output latches was for programming convenience. This change allowed for reading of the entire 12 bits of the A/D converter with a 16-bit load instruction by the CPU. The switch to free-running conversion was due to the ADC1210 A/D converter. It was discovered that the start conversion signal must be synchronized with the conversion clock to insure proper converter operation. To alter the converter design for this would have required additional logic circiutry. This was not available on the board hence, free-running conversion was used. In conjunction with this, the latching of the conversion value into the output latches was made conditional on the reading of the latches by the CPU. This prevents the possibility that logic level transitions, i.e. bad data, would be presented to the CPU during a read of either output latch.

After the above corrections had been made, the computer passed the first two hardware test routines with no difficulty. This was with the use of a 8 MHz oscillator crystal, i.e., a 4 MHz system clock, which was extraordinary because the engineering sample provided to us was intended to

operate with an oscillator crystal of up to 5 MHz only [It also implied that the system was operating with EPROMs that should have been too slow to access with the bus timing speeds involved with a system clock of 4 MHz.]. However, the computer did not pass the hardware multiply and divide tests. The reason for this was the RCA CDP1855 Multiply/Divide Units (MDUs); with a supply voltage of 5 volts, the access time of the MDU registers was too long for the bus timing of the NSC800. The discovery of this fault was found when attempting to run the multiply test (MULTST) software. After the development of a short MDU timing test routine (TIMTST), it was found that the MDUs appeared to operate correctly with a system clock of 1 MHz. For further tests, the single board computer was operated with a 1 MHz system clock. The adaptive digital predictors which used the MDUs would be limited to this system clock frequency. An alternate solution would be to provide a wait state generator for the MDU enable line ($\overline{\text{MULTCE}}$). This would then guarantee proper timing for access of the MDU registers while allowing the remainder of the system to operate at full speed.

During the evaluation of the NSC800 and the single board computer, several difficulties arose due to the NSC800 itself. As mentioned above, the promised operation range of 3 to 12 volts was not available. According to engineers at National Semiconductor, this specification will probably never be achieved. In all likelihood, the power supply operation range will be limited to 5 volts $\pm$ 10 percent. During the software evaluation, it was found that some of the instructions did not appear to execute correctly with a system clock frequency greater than approximately 500 kHz.

After the five test routines were run and successful computer operation was obtained (see MDU discussion above and the test software overview in Chapter II), the three adaptive digital predictor programs were run (see Chapter IV for a discussion of the three programs). It was found that with a 4.0 MHz system clock frequency, the processor halted completely (the HALT, H, light turned on). With a 2 MHz system clock the programs ran but the predictor outputs were random values, not coherent signals as one would expect. Finally, the programs did work correctly with a 500 kHz system clock frequency. The possible fault was first thought to be in the MDUs again, even though they operated correctly with a 1 MHz system clock. Another fault could have been the A/D converter since there had been problems with it before, though it performed correctly with the analog test routine at 4 MHz. These conclusions were discarded because it was discovered that the Widrow alogrithm which used a software multiply did not operate correctly with a system clock frequency above 500 kHz also. With the A/D converter input strapped to ground, the Widrow algorithm still did not run correctly above 500 kHz.

Further tests showed that the problem was not in the NSC800 itself but in a bus conflict between the NSC800 and the IM6518 RAMs. This conflict was caused by not adequately selecting the RAM during the read and write cycles. The RAM was selected only by a logic ´0´ on the $\overline{\text{RAMCE}}$ line, with $\overline{\text{WR}}$ determining if the operation was a read or a write. At the beginning of a RAM access cycle the $\overline{\text{RAMCE}}$ line would go low as soon as the 74PC138 one-of-eight decoder had propagated the device selection from the address lines. Since the access time of the IM6518A-1 is very short (95 nanoseconds, worst case) and the $\overline{\text{WR}}$ line would still be a logic ´1´ until

after the address data was removed, there would be a bus conflict with the address data and the data coming from the RAM. The bus conflict was solved by ANDing the $\overline{RD}$ and $\overline{WR}$ signals and using this signal to select the RAM along with the $\overline{RAMCE}$ line. The $\overline{WR}$ signal was maintained to select between read and write operations.

With the correction in place, the adaptive digital predictor programs ran correctly with higher system clock frequencies. The Widrow algorithm with software multiply ran with a system clock frequency of 2.5 MHz while the Widrow with hardware multiply and the adaptive lattice predictor ran with 2.0 and 1.0 MHz system clocks respectively. The limitations of the two latter programs was the multiply/divide units. Though the Widrow with hardware multiply did appear to run correctly with a 2.0 MHz clock, it is still recommended to either place a wait-state generator on the MDU select line or run programs using the MDUs at 1.0 MHz. From the prior MDU tests, correct operation of the devices cannot be guaranteed above 1.0 MHz system clock frequency.

The NSC800 does appear to operate correctly at system clock frequencies of 2.5 MHz or lower. Attempts to run the processor at 4.0 MHz were failures; the processor still halting at this frequency (see Chapter IV). If National Semiconductor continues development of the NSC800, they will undoubtedly achieve the goal of 4.0 MHz system clock frequency. It is discouraging to be forced to operate the device only at 5 volts. This defeats many of the advantages of using CMOS electronics. Primary among these is the loss of low-noise operation performance of CMOS, since the electronics cannot be operated at higher voltages than TTL or NMOS

electronics. The lower voltage operation can also degrade overall system speed performance. The use of standard CMOS devices with P2CMOS logic will cause the degradation, an example being the CDP1855 multiply/divide units discussed above. The problem can be solved with wait-state generators, but there would be an increase in system cost and some slowing of operation speed would still be present. It is of interest to note that the small-scale integration (74PC series) devices are specified to operate with a power supply voltage of up to 10 volts [5]. These devices perform quite well, even at 5 volts. The difficulties with the NSC800 itself may be in the very complexity of the device; production yields precluding devices with such a wide operational voltage range. Hopefully with experience, National will be able to produce units with the desired voltage range.

CHAPTER IV

NSC800 SOFTWARE EVALUATION - THE WIDROW AND LATTICE ADAPTIVE DIGITAL PREDICTORS

For the software evaluation of the NSC800, two adaptive digital predictor (ADP) algorithms were implemented. These were the Widrow Least Means Square (LMS) algorithm and the adaptive lattice predictor (ALP) algorithm. They are both used in intrusion-detection applications. A discussion of these two algorithms will be carried out followed by an analysis of the results of their implementation.

The Widrow LMS Algorithm

The Widrow LMS algorithm [7] applies the method of steepest decent in solving the problem of signal prediction. This method comes into play in determining the coefficients of the predictor for each iteration of the algorithm. The response of the filter is adaptive as a result. Nonstationary signals, such as random noise, can then be predicted and removed by the filter leaving signals from other sources, in our case an intruder. A moving average filter (MAF) takes the error output from the last sixteen iterations of the algorithm and averages them. This result is output to the D/A converter. A flowchart of the Widrow LMS algorithm as implemented in Z80 assembly language on the NSC800 appears in Figure 4.1.

The Widrow algorithm was implemented in two forms. The first was done with software multiplies (called WIDADP). It was essentially identical to that developed by Nickel [3] with changes made to make it compatible with

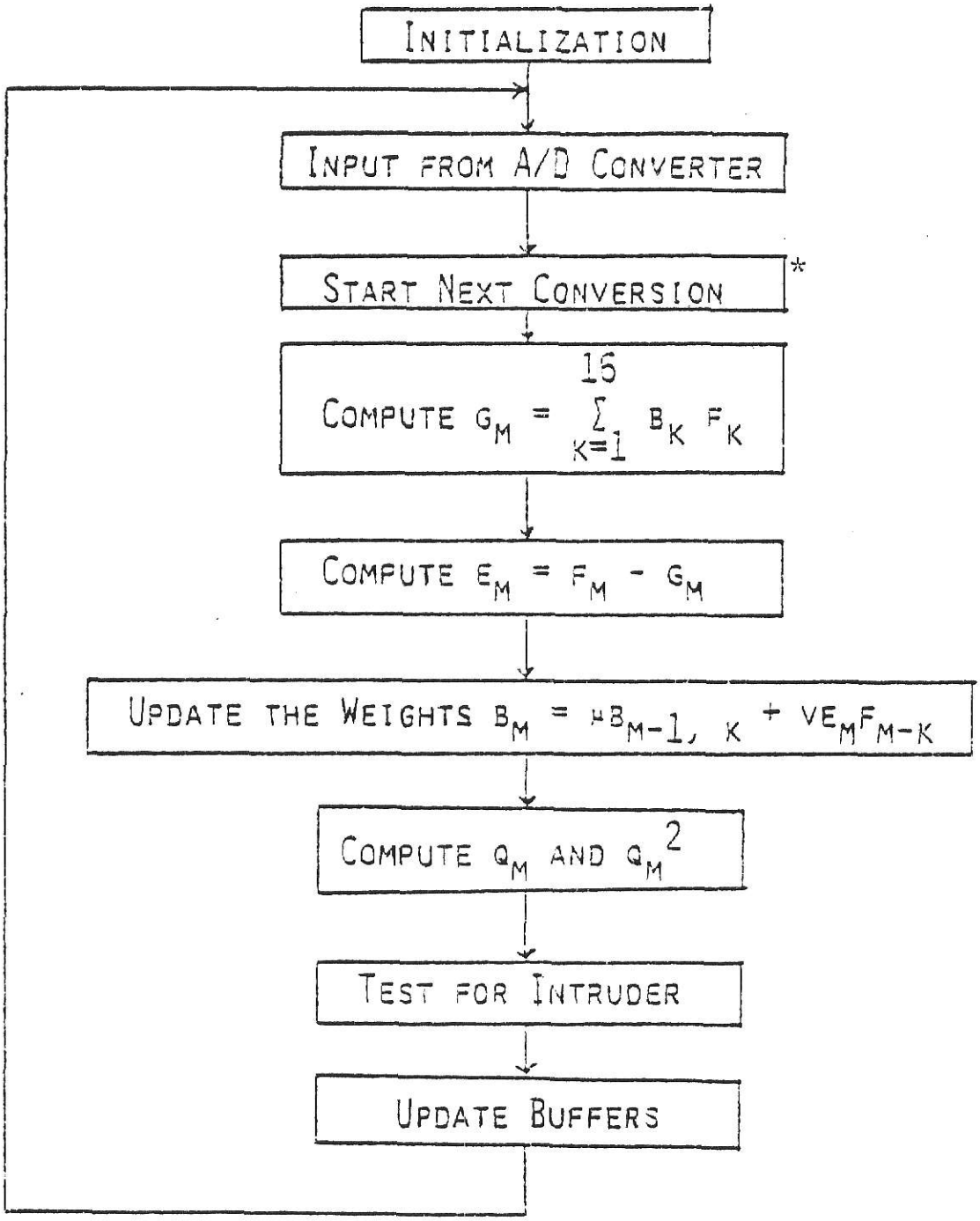

FIGURE 4.1. Z80 PROGRAM FLOWCHART [3]
μ AND V ARE CONSTANTS,
K IS THE DELAY LENGTH,
B_K ARE THE WEIGHTS USED
IN THE LMS ALGORITHM.

*Not implemented on the NSC800 version of the algorithm.

the single board computer memory map. An assembly language listing of this program is given in Appendix 1. The program serves as a benchmark to compare the performance of the NSC800 to the Z80 implementation done by Nickel. The second version (called WADPHM) was executed with hardware multiplies using the RCA CDP1855 Multiply/Divide Units (see Chapter II). This was done in order to show how hardware multiplication increases the execution rate of the ADP. The assembly language listing of this program appears in Appendix 2.

The Adaptive Lattive Predictor Algorithm

The adaptive lattice predictor was developed by Ahmed [2] as an alternative to the Widrow LMS predictor in intrusion detection applications. For rapid convergence to occur in an adaptive Widrow algorithm, the eigenvalue distribution of the input data covariance matrix must be small. Slow convergence results in sluggish predictor performance. If the convergence is too fast, then there is an increase in the steady-state error of the filter output. It is desirable to obtain a filter structure whose convergence parameters are independent of the eigenvalue spread, i.e., the input data. The adaptive lattice predictor is one such filter structure.

The lattice algorithm is derived through the use of forward and backward prediction errors, e(n) and w(n). Its structure is formed of stages, which can be obtained recursively through the use of matrix bordering techniques. Recursion formulas for the forward and backward prediction errors can then be found. The lattice structure can have as many stages as required for the accuracy of prediction desired. The

forward error prediction of the last stage is the output of the lattice predictor. In order to make the lattice algorithm adaptive, the filter coefficients are found through a recursive formula based on the method of steepest decent. Combining these equations results in an algorithm which should converge faster than the Widrow while maintaining a lower steady-state error in the output.

The implementation of the ALP on the NSC800 (called LATZ80) was a straightforward operation. A flowchart for the Z80 program as implemented on the NSC800 is shown in Figure 4.2. Numerical data was handled as sixteen-bit signed, fixed point numbers. When brought into the algorithm, The data has one sign bit with the implied decimal point to the immediate right of it. An arithmetic shift to the right yields two sign bits with the decimal point to their right. This was the form in which all numeric values were handled and stored, with the exception of some filter constants. The multiplies and a divide were performed in hardware with the CDP1855 MDU. Since the MDU performs unsigned multiplies and divides, hardware driver subroutines were developed which allow signed multiplies and divides with this device.

The signed multiply operation was performed using the fudge method. Given two signed numbers, a fudge value was formed by the conditional sum of the two numbers. If the sign of one number is negative, the other number is added to the fudge value. This is not done if the first number is positive. The operation is repeated with the roles of the numbers reversed. The two signed numbers are then multiplied and the fudge value subtracted from the product. The subroutine returns the upper sixteen bits

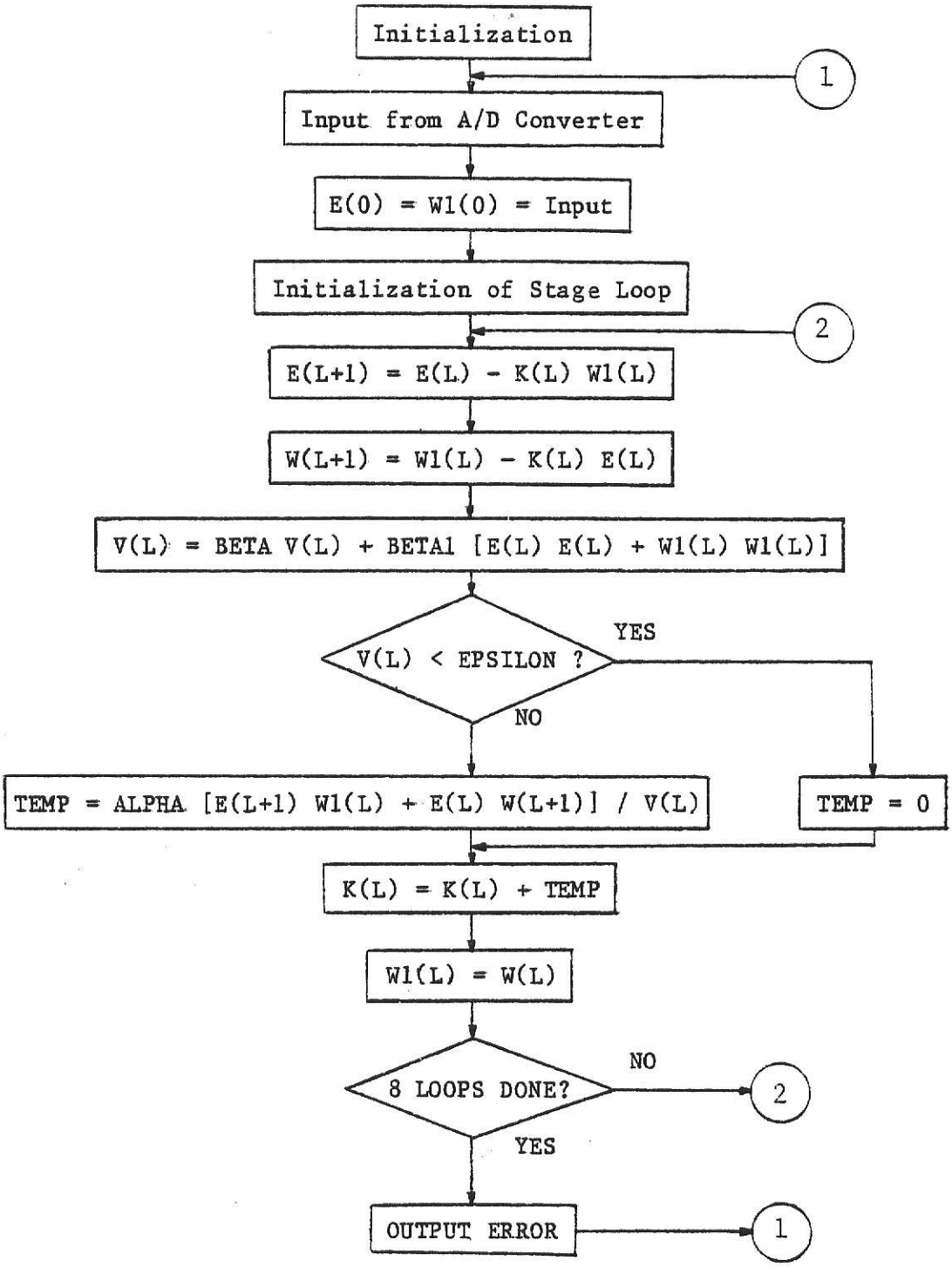

Figure 4.2: Z80 Program Flowchart for the Adaptive Lattice Predictor

of the signed multiply to the main routine.

The execution of the signed divide was done in a more direct fashion. In the general case, the sign of the two operands are found and saved. Next, the operands are converted to positive (unsigned) numbers and loaded into the MDU for division. After division, the quotient is corrected to its proper sign by consulting the original signs of the two operands. This operation was simplified because of the nature of the numbers divided. In the lattice algorithm, the negative gradient of the error is divided by the instantaneous power of the stage of the lattice [2]. Although the negative gradient can be positive or negative, the power term (i.e., the variance) is by definition positive. This allowed some simplification of the subroutine. Only the gradient term was converted to an unsigned value before division. The division operation was executed in this form with a 16-bit divisor and 32-bit dividend (the lower 16-bits are set to zero). The subroutine returns a 16-bit quotient.

The error output of the adaptive lattice predictor was the final result of the algorithm. A MAF was not implemented in the algorithm. The assembly listing of the lattice algorithm appears in Appendix 3.

Results of the Predictor Implementations

The execution rates of the three programs were largely determined by the time necessary to obtain the signed multiplies. Program timing of the three ADP programs are given in Tables 4.1, 4.2, and 4.3. Note that these would be the execution rates if the NSC800 operated with a system clock frequency of 4 MHz. The NSC800s that were made available to us were limited to a system clock frequency of only 2.5 MHz. Limitation of

Table 4.1: Program Timing of the Z80 Widrow ADP with Software Multiply [3]

Routine	Time in microseconds	+	Multiply Times in microseconds	=	Total Time in microseconds
Initialization	707.25	+	0.0	=	707.25
Input from A/D	21.5	+	0.0	=	21.5
Compute g	1107.25	+	16 x 147.25	=	3463.25
Compute e_m	29.75	+	0.0	=	29.75
Update the weights	1265.75	+	16 x 147.25	=	3621.75
Compute q_m	50.75	+	1 x 147.75	=	198.0
Block move buffers	351.0	+	0.0	=	351.0
Total Times (excluding Initialization)	2826.0	+	4859.25	=	7685.25

Program execution rate = 130 Hz

Note: These times were calculated for a Z80 CPU clock rate of 4.0 MHz.

Table 4.2: Program Timing of the Z80 Widrow ADP with Hardware Multiply

Routine	Time in microseconds	+	Multiply Times in microseconds	=	Total Time in microseconds
Initialization	707.25	+	0.0	=	707.25
Input from A/D	21.5	+	0.0	=	21.5
Compute g	1107.25	+	16 x 70.5	=	2235.25
Compute e_m	29.75	+	0.0	=	29.75
Update the weights	1265.75	+	16 x 70.5	=	2393.75
Compute q_m	50.75	+	1 x 70.5	=	121.25
Block move buffers	351.0	+	0.0	=	351.0
Total Times (excluding initialization)	2826.0	+	2326.5	=	5152.5

Program execution rate = 194 Hz

Note: These times were calculated for a Z80 CPU clock rate of 4.0 MHz.

Table 4.3: Program Timing of the Z80 Lattice ADP

Routine	Time in microseconds	+	Multiply Times in microseconds	=	Total Time in microseconds
Initialization	573.0	+	0.0	=	573.0
Input from A/D and Stage Init.	32.25	+	0.0	=	32.25
Times for one stage (eight stages total):					
Compute e(L+1)	47.5	+	1 x 74.5	=	122.0
Compute w(L+1)	44.75	+	1 x 74.5	=	119.25
Compute v(L)	66.5	+	4 x 74.5	=	364.5
Update the weights	106.0	+	3 x 74.5 + 1 x 73.75 (divide)	=	403.25
w1(L) = w(L)	27.5	+	0.0	=	27.5
Output to D/A	10.5	+	0.0	=	10.5
Total Time (excluding Initialization)	2380.75	+	5954.0	=	8334.75

Program execution rate = 121 Hz

Note: These times were calculated for a Z80 CPU clock rate of 4.0 MHz.

execution speed was also caused by the hardware multiply/divide curcuit (see Chapter III). A list of execution times for the three programs a different system clock frequencies is given in Table 4.4. It should be noted that the variance in execution speeds is due to program flow. Some parts of the programs are not always executed, hence the speed variations.

It was rather surprising to find that the lattice algorithm was slower than the Widrow with software multiply until one realized that the eight stages of the lattice required seventy-two signed multiplies and eight signed divides in its execution. This is compared to only thirty-two signed multiplies for the Widrow. Although the hardware multiply executes twice as fast as the software multiply, this is not enough of a difference to allow the lattice to run faster. The difficulty is that while the unsigned multiply/divide takes only 4 microseconds to occur (with a 4.0 MHz system clock), the hardware driver subroutine to convert the signed numbers to unsigned numbers and back takes about 70 microseconds to run. The execution rate of this algorithm would be improved greatly if a device that performed signed hardware multiplies and divides was available or if the NSC800 did operate with a clock frequency of 4 MHz.

A test of the two algorithms was preformed by using line sensor data files made available from Sandia Laboratiries. These files were stored in digital floating-point form in a Data General NOVA/4X minicomputer and had been originally sampled at eight samples per second. Previous studies had shown that sufficient information about the input signal was present at this sample rate to allow detection of an intruder. At higher sample rates, more signal power due to noise was received than signal power due to

Table 4.4 Execution Times of the Digital Predictor Programs

System Clock Frequency	Program		
	WIDADP	WADPHM	LATZ80
500 kHz	17.6 Hz	26.0 Hz	15.0 - 20.0 Hz
1.0 MHz	35.1 Hz	57.8 Hz	30.6 - 40.1 Hz
2.0 MHz	70.2 Hz	103.7 Hz	I.O.* 60.0 Hz
2.5 MHz	87.8 Hz	I.O.* 129 Hz	I.O.* 74.6 Hz
4.0 MHz	N.O.+	N.O.+	N.O.+

*Incorrect Operation; MDU Malfunction

+No Operation; CPU Halts

Note: These are approximate execution times.

the intruder, resulting in more false alarms. The intruder signal power is concentrated at very low frequencies (below 4 Hz) [1].

The data files were scaled to integer values and sent out from a Data General DG4288 D/A converter module to the A/D converter of the single board computer. The DG4288 was strapped for two's complement number representation with full-scale output of ±2.5 volts. After processing of the data by the NSC800 SBC, the result is output through the D/A converter to the NOVA computer. The analog signal is input into the NOVA through a Data General DG4281 analog multiplexer module that was strapped for unity gain. The actual conversion was performed by a Data General DG4280 A/D converter module which was set for ±10 volt full-scale signal input. A FORTRAN program was written to move file data to and from disk files. It also called Data General's Sensor Access Manager (SAM) FORTRAN subroutine library which controled the interface hardware.

The line sensor files consisted of creeper (intruder) signals mixed with normal backgroud noise and wind gust signals. This would be a good test for the predictor algorithms since the creeper would be hard to detect with wind gusts producing interference. Plots of an input signal and the output signals of the three programs are shown in Figure 4.3. As can be seen, the two Widrow programs had nearly identical responses. The differences are due to quantization errors and loss of multiplication accuracy in the software multiply operation. The software multiply only multiplies the high bytes of the operands, losing the cross-term components of the low bytes of the operands. The hardware multiply does not do this. Both Widrow routines were successful in removing the normal background

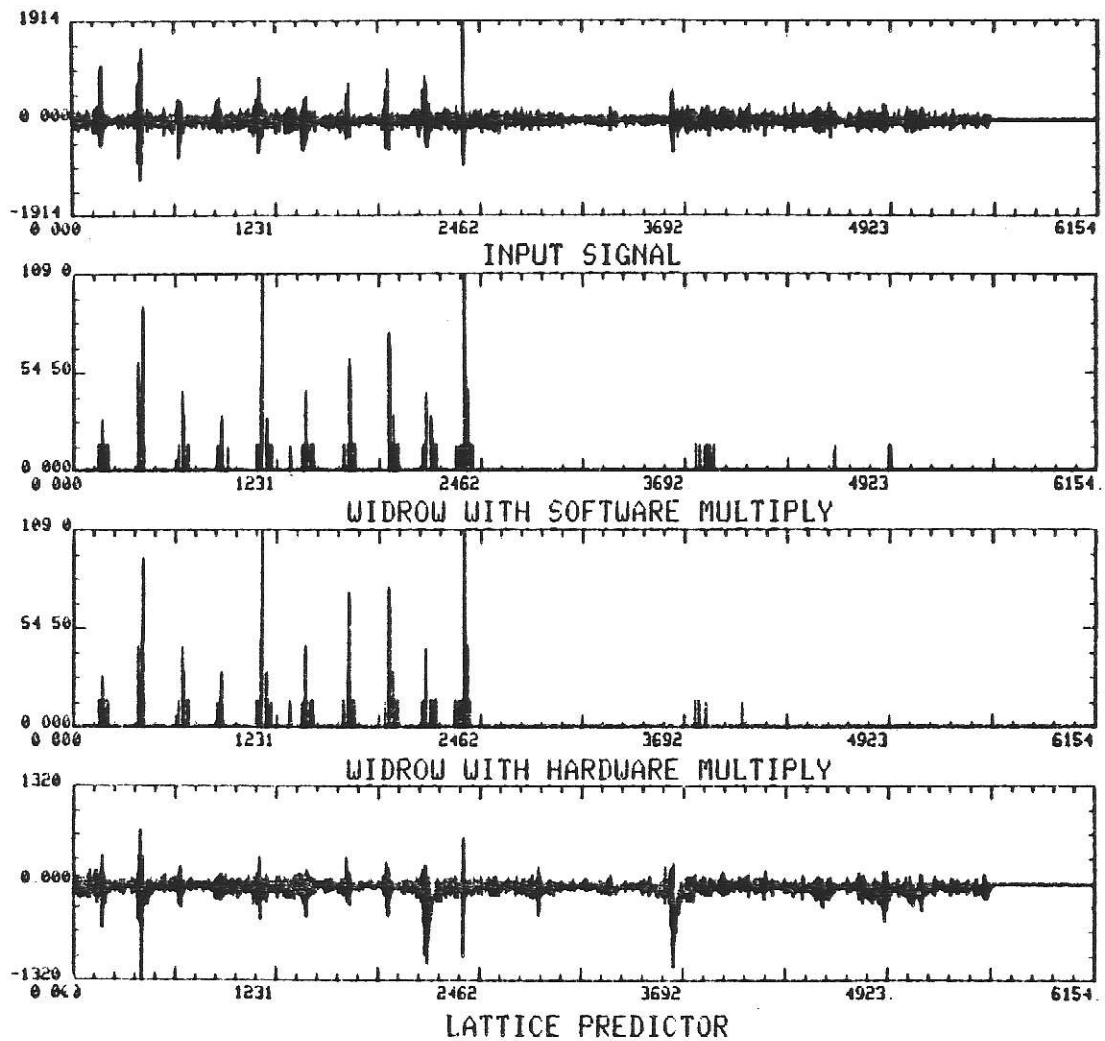

Figure 4.3: Sensor Data Input and Outputs of the ADP Programs

noise of the signal but had trouble with the wind gusts. The adaptive lattice predictor appeared to have better response with moderate gusts, but the strong gust (near 3692 on the time axis) produced a large output signal from the predictor. Without further classification by the predictor, this signal would be interpreted as an intruder. The tests did show that the NSC800 can perform the algorithms adequately.

CHAPTER V

CONCLUSIONS

As a result of the evaluation of the NSC800 microprocessor, a low power, single board computer is now available to perform various signal processing applications. The computer can also be used for projects involving data acquisition and/or numerical computation. The available memory space on the board is more than adequate for most applications of a dedicated computer system. It would be desirable to have a signed hardware multiply/divide circuit in CMOS in order to increase the throughput of the system.

The NSC800 met up to the advertised operating specifications with the exception of wide operating voltage. The power supply range of 5 volts $\pm$ 10 percent will reduce the potential noise immunity of any CMOS system it is used in. Further decrease in system speed will result from the increased gate delays of standard CMOS devices operating at 5 volts rather than 10 volts. The NSC800 does have the advantage of using the Z80 instruction set, which is still a powerful instruction set when compared to other 8-bit microprocessors. Even with a system clock speed of 2.5 MHz, the NSC800 is capable of performing many useful tasks. The P^2CMOS support logic is useful for high-speed, low-power logic applications and works well with the NSC800 in creating a fast computer system. It is hoped that National Semiconductor will continue development work with the NSC800 and eventually succeed in obtaining the full operational voltage range that was first promised.

The implementation of the two adaptive digital predictors was successful. The ALP algorithm appeared to performed better than the Widrow algorithm, but direct comparisons are not possible due to the lack of a moving average filter on the adaptive lattice predictor. The NSC800 is very capable as a digital processor for adaptive digital prediction. It appears to be an optimal solution for eight-bit digital signal processing applications. Once the unit price of the NSC800 decreases to a viable level, it would satisfy all the specifications of the 'ideal' digital proccessor CPU, as mentioned in Chapter I. Its use would then be highly encouraged.

ACKNOWLEDGEMENTS

This work was sponsored and funded by the Base and Installation Security Systems Program Office, Electronics Systems Division of the Air Force Systems Command, Hanscom Air Force Base, MA 01731.

The author would like to thank Sandia Laboratories for their support of this project. Keith Summers is to be commended for the schematics prepared for this thesis. Myron Flickner and Fred Ratcliffe are to be thanked for their work in setting up the NOVA interface. Thanks are also extended to Earl Creel for relieving the author of his teaching duties during the preparation of the thesis and to Karen Ashmore for her moral support. Appreciation is also due to Dr. G. Simons and Dr D. H. Lenhert, who served as committee members. Finally, the author would like to thank Dr. M.S.P. Lucas who served as the author's major professor and provided a great deal of assistance.

REFERENCES

1. N. Ahmed, et al., "On an Intrusion-Detection Approach via Adaptive Prediction", IEEE Trans. on Aerospace and Electronics Systems, Vol. AES-15, No. 3, pg 430, May 1979.

2. N. Ahmed and R. J. Fogler, "On an Adaptive Lattice Predictor and a Related Application.", IEEE Circuits and Systems Magazine, Vol. 1, No. 4, pg 17, May 1979.

3. D. J. Nickel, An Evaluation of Various Microprocessor Implementations of an Adaptive Digital Predictor for Intrusion Detection, A Master's Report, Kansas State University, 1979.

4. R. J. Hass, et al., "On a Microcomputer Implementation of an Intrusion-Detection Algorithm", IEEE Trans. on ASSP, Vol. ASSP-27, No. 6, pg 782, December 1979.

5. "NSC800 Microprocessor Family Handbook", National Semiconductor Corporation.

6. "Z80-CPU Assembly Language Programming Manual", Zilog Corporation.

7. B. Widrow, et al., "Stationary and Nonstationary Learning Characteristics of the Adaptive LMS Filter", Proc. IEEE. Vol. 64. p. 1151, August 1976.

APPENDIX 1

Z80 ASSEMBLER PROGRAM FOR THE WIDROW ADP WITH SOFTWARE MULTIPLY

This program is the Widrow LMS adaptive digital predictor which uses software multiply (WIDADP). It is essentially a copy of the listing created by Nickel [3] for his Master's report. Changes have been made in order to allow the program to run on the NSC800 SBC, which are:

1. All buffers and number storage has been relocated so that they start at 0400H rather than 4000H.
2. The location of the D/A and A/D converters have been changed to 1400H and 1402H respectively.

```
PASS 1 DONE
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0001
 ADDR  CODE       STMT SOURCE STATEMENT

                  0001 ;THIS IS THE WIDROW ADAPTIVE DIGITAL
                  0002 ;PREDICTOR ALGORITHM DEVELOPED BY D. NICKEL
                  0003 ;FOR HIS MASTER'S THESIS.  IT HAS BEEN SLIGHTLY
                  0004 ;MODIFIED FOR USE ON THE NSC800 SBC.  THIS
                  0005 ;INVOLVES MOVING THE RAM ADDRESSING IN THE
                  0006 ;PROGRAM TO 400H-7FFH.  THE ADC IS LOCATED AT
                  0007 ;1402H AND THE OUTPUT PORT IS NOW THE DAC WHICH
                  0008 ;IS LOCATED AT 1400H
                  0009 ;
                  0010 ;                  INITIALIZATION
'0000  066C       0011 WIDADP LD B,6CH ;SET UP ITERATION COUNTER
'0002  AF         0012        XOR A    ;CLEAR A
'0003  216B04     0013        LD HL,046BH ;LOAD HL WITH MEMORY POINTER
'0006  77         0014 L1     LD (HL),A ;CLEAR MEMORY BYTE
'0007  2B         0015        DEC HL ;DECREMENT HL
'0008  10FC       0016        DJNZ L1 ;DECREMENT ITERATION COUNTER
                  0017 ;                  AND REPEAT IF > 0
'000A  31FF04     0018        LD SP,4FFH ;INITIALIZE THE STACK POINTER.
                  0019 ;
                  0020 ;                  INPUT FROM A/D CONVERTER
'000D  AF         0021 START  XOR A ;CLEAR A
'000E  6F         0022        LD L,A ;CLEAR L
'000F  3A0214     0023        LD A,(1402H) ;INPUT SAMPLE FROM A/D
'0012  47         0024        LD B,A ;B=SAMPLE
'0013  3E7F       0025        LD A,7FH ;A=MASK
'0015  A8         0026        XOR B ;CONVERT SAMPLE TO 2'S COMPLEMENT FORM
'0016  67         0027        LD H,A ;H=SAMPLE
'0017  CB2C       0028        SRA H ;SCALE SAMPLE BY ARITHMETICALLY
'0019  CB1D       0029        RR L  ;SHIFTING RIGHT HL
'001B  222004     0030        LD (0420H),HL ;STORE F(M) IN MEMORY
                  0031 ;
                  0032 ; COMPUTE G=SUM(K=1,16) F(M-K)*B(M,K)
'001E  210000     0033        LD HL,0000H ;CLEAR HL
'0021  226404     0034        LD (0464H),HL ;CLEAR LOW ORDER 16 BITS OF G
'0024  226604     0035        LD (0466H),HL ;CLEAR HIGH ORDER 16 BITS OF G
'0027  DD212204   0036        LD IX,0422H ;LOAD INDEX REG. X WITH ADDRESS
                  0037 ;                    OF L.O. BYTE OF B(M,16)
'002B  FD210004   0038        LD IY,0400H ;LOAD INDEX REG.Y WITH ADDRESS
                  0039 ;                    OF L.O. BYTE OF F(M-16)
'002F  0610       0040        LD B,10H ;SET UP ITERATION COUNTER
'0031  DD6E00     0041 SUM1   LD L,(IX+0H) ;HL=B(M,K)
'0034  DD6601     0042        LD H,(IX+1H)
'0037  FD5E00     0043        LD E,(IY+0H) ;DE=F(M-K)
'003A  FD5601     0044        LD D,(IY+1H)
'003D  C5         0045        PUSH BC ;SAVE THE COUNTER
'003E  CD0002'    0046        CALL MULT ;CALL THE MULTIPLY SUBROUTINE
'0041  DD23       0047        INC IX ;INCREMENT BUFFER POINTER
'0043  DD23       0048        INC IX
'0045  FD23       0049        INC IY ;INCREMENT BUFFER POINTER
'0047  FD23       0050        INC IY
'0049  ED4B6404   0051        LD BC,(0464H) ;LOAD BC WITH L.O. 16 BITS
                  0052 ;                      OF G
'004D  EB         0053        EX DE,HL ;UPDATE L.O. 16 BITS OF A AND
'004E  09         0054        ADD HL,BC ;STORE IN MEMORY
'004F  226404     0055        LD (0464H),HL
'0052  EB         0056        EX DE,HL ;
'0053  ED4B6604   0057        LD BC,(0466H) ;LOAD BC WITH H.O. 16 BITS
                  0058 ;                      OF G
```

```
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0002
 ADDR  CODE       STMT SOURCE STATEMENT

'0057  ED4A       0059         ADC HL,BC ;UPDATE H.O. 16 BITS OF G AND
'0059  226604     0060         LD (0466H),HL ;STORE IN MEMORY
'005C  C1         0061         POP BC ;RETRIEVE COUNTER
'005D  10D2       0062         DJNZ SUM1 ;DECREMENT COUNTER AND REPEAT IF >0
                  0063 ;
                  0064 ;            COMPUTE E(M)=F(M)-G
                  0065 ;NOTE:. ENTER THIS ROUTINE WITH HL=G (H.O. 16 BITS)
'005F  EB         0066         EX DE,HL ;DE=G (H.O. 16 BITS)
'0060  2A2004     0067         LD HL,(0420H) ;HL=F(M)
'0063  AF         0068         XOR A ;CLEAR A,CLEAR CARRY
'0064  ED52       0069         SBC HL,DE ;HL=F(M)-G=E(M)
'0066  CB2C       0070         SRA H
'0068  CB1D       0071         RR L
'006A  CB2C       0072         SRA H
'006C  CB1D       0073         RR L ;DIVIDE BY 2**4 TO FORM
'006E  CB2C       0074         SRA H ;HL=E(M)/2**4
'0070  CB1D       0075         RR L
'0072  CB2C       0076         SRA H
'0074  CB1D       0077         RR L
'0076  226204     0078         LD (0462H),HL ;STORE E(M)/2**4 IN MEMORY
                  0079 ;
                  0080 ;     UPDATE THE WEIGHTS B(M+1,K)=U*B(M,K)+V*E(M)*F(M-K)
                  0081 ;NOTE: ENTER THIS ROUTINE WITH V*E(M)=E(M)/2**4 IN HL.
'0079  DD212204   0082         LD IX,0422H ;LOAD INDEX REG. X WITH THE ADDRESS
                  0083 ;                     OF  THE L.O. BYTE OF B(M,16)
'007D  FD210004   0084         LD IY,0400H ;LOAD INDEX REG. Y WITH THE ADDRESS
                  0085 ;                     OF THE L.O. BYTE OF F(M-16)
'0081  226A04     0086         LD (046AH),HL ;STORE V*E(M) IN MEMORY
'0084  0610       0087         LD B,10H ;SET UP ITERATION COUNTER
'0086  C5         0088 L2      PUSH BC ;SAVE COUNTER
'0087  2A6A04     0089         LD HL,(046AH) ;HL=V*E(M)
'008A  FD5E00     0090         LD E,(IY+0H) ;DE=F(M-16)
'008D  FD5601     0091         LD D,(IY+1H)
'0090  CD0002'    0092         CALL MULT ;CALL MULTIPLY SUBROUTINE
'0093  DD5E00     0093         LD E,(IX+0H) ;DE=B(M,K)
'0096  DD5601     0094         LD D,(IX+1H)
'0099  EB         0095         EX DE,HL
'009A  44         0096         LD B,H ;FORM B(M,K)/2**10
'009B  4C         0097         LD C,H
'009C  CB29       0098         SRA C
'009E  CB29       0099         SRA C
'00A0  AF         0100         XOR A ;CLEAR A
'00A1  CB10       0101         RL B ;ROTATE M.S.B. INTO CARRY
'00A3  3001       0102         JR NC,L3 ;FILL B WITH SIGN BIT
'00A5  2F         0103         CPL
'00A6  47         0104 L3      LD B,A
'00A7  AF         0105         XOR A ;CLEAR A
'00A8  ED42       0106         SBC HL,BC ;HL=U*B(M,K)=B(M,K)*(1-2**10)
'00AA  ED5A       0107         ADC HL,DE ;HL=B(M+1,K)
'00AC  DD7500     0108         LD (IX+0H),L ;STORE UPDATED B(M+1,K) IN MEMORY
'00AF  DD7401     0109         LD (IX+1H),H
'00B2  DD23       0110         INC IX ;INCREMENT POINTERS
'00B4  DD23       0111         INC IX
'00B6  FD23       0112         INC IY
'00B8  FD23       0113         INC IY
'00BA  C1         0114         POP BC ;RETRIEVE COUNTER
'00BB  10C9       0115         DJNZ,L2 ;DECREMENT COUNTER AND REPEAT IF > 0
                  0116 ;
```

```
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0003
 ADDR  CODE       STMT SOURCE STATEMENT

                  0117 ;      COMPUTE Q(M)=(1/16)*SUM(K=1,16) E(M-K)
'00BD  2A4204     0118         LD HL,(0442H) ;HL=E(M-16) :16
'00C0  EB         0119         EX DE,HL ;EXCHANGE DE AND HL
'00C1  2A6204     0120         LD HL,(0462H) ;HL=E(M)/16
'00C4  AF         0121         XOR A ;CLEAR A, CLEAR CARRY
'00C5  ED52       0122         SBC HL,DE ;HL=(E(M)-E(M-16)/16
'00C7  EB         0123         EX DE,HL ;EXCHANGE DE AND HL
'00C8  2A6804     0124         LD HL,(0468H) ;HL=OLD Q(M)
'00CB  19         0125         ADD HL,DE ;HL=NEW Q(M)
'00CC  226804     0126         LD (0468H),HL ;STORE NEW VALUE OF Q(M)
                  0127 ;                          IN MEMORY
'00CF  54         0128         LD D,H ;DE=HL=Q(M)
'00D0  5D         0129         LD E,L
'00D1  CD0002'    0130         CALL MULT ;CALL THE MULTIPLY SUBROUTINE
                  0131 ;                    TO FORM HL=Q(M)**2/16**2
'00D4  CB25       0132         SLA L
'00D6  CB14       0133         RL H
'00D8  CB25       0134         SLA L
'00DA  CB14       0135         RL H
'00DC  CB25       0136         SLA L ;MULTIPLY BY 16 TO FORM
'00DE  CB14       0137         RL H ; HL=Q(M)**2/16
'00E0  CB25       0138         SLA L
'00E2  CB14       0139         RL H
'00E4  7C         0140         LD A,H
'00E5  EE80       0141         XOR 80H ;CONDITION FOR DAC.
'00E7  320014     0142         LD (1400H),A ;OUTPUT RESULT TO DAC
                  0143 ;
                  0144 ; BLOCK MOVE OF E AND F BUFFERS
'00EA  114204     0145         LD DE,0442H ;LOAD TARGET ADDRESS
'00ED  214404     0146         LD HL,0444H ;LOAD SOURCE ADDRESS
'00F0  012000     0147         LD BC,0020H ;LOAD NUMBER OF BYTES TO BE MOVED
'00F3  EDB0       0148         LDIR ;BLOCK MOVE OF E BUFFER
'00F5  110004     0149         LD DE,0400H ;LOAD TARGET ADDRESS
'00F8  210204     0150         LD HL,0402H ;LOAD SOURCE ADDRESS
'00FB  012000     0151         LD BC,0020H ;LOAD NUMBER OF BYTES TO BE MOVED
'00FE  EDB0       0152         LDIR ;BLOCK MOVE OF F BUFFER
'0100  C30D00'    0153         JP START ;JUMP TO INPUT ROUTINE FROM A/D ROUTINE
                  0154 ;
                  0155 ; MULTIPLY SUBROUTINE
 >0200            0156         ORG 200H
'0200  5C         0157 MULT    LD E,H ;D=MPX , E=MPY
'0201  AF         0158         XOR A ;CLEAR A
'0202  67         0159         LD H,A ;CLEAR H
'0203  6F         0160         LD L,A ;CLEAR L
'0204  4F         0161         LD C,A ;CLEAR C
'0205  0608       0162         LD B,08H ;B=ITERATION COUNTER
'0207  CB7A       0163         BIT 7,D ;TEST SIGN OF MPX
'0209  2018       0164         JR NZ,ADJ1 ;IF NEGATIVE, GO TO ADJ1, ELSE CONTINUE
'020B  CB7B       0165 RET1    BIT 7,E ;TEST SIGN OF MPY
'020D  2018       0166         JR NZ,ADJ2 ;IF NEGATIVE, GO TO ADJ2 ELSE CONTINUE
'020F  CB3A       0167 RET2    SRL D ;SHIFT RIGHT LOGICAL MPX
'0211  3003       0168         JR NC,SKIP ;TEST L.S.B OF MPX
                  0169 ;                    IF 0, GO TO SKIP
'0213  7C         0170         LD A,H ;ELSE ADD MPY TO THE HIGH PART
'0214  83         0171         ADD A,E ; OF THE RESULT
'0215  67         0172         LD H,A
'0216  CB1C       0173 SKIP    RR H ;SHIFT RIGHT HL
'0218  CB1D       0174         RR L
```

```
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0004
 ADDR  CODE      STMT SOURCE STATEMENT

'021A  10F3      0175        DJNZ,RET2 ;DECREMENT ITERATION COUNTER AND
                 0176 ;                  REPEAT IF > 0
'021C  7C        0177        LD A,H ;SUBTRACT CONTENTS OF FUDGE REGISTER
'021D  91        0178        SUB C ; FROM HIGH PART OF RESULT
'021E  67        0179        LD H,A
'021F  AF        0180        XOR A
'0220  57        0181        LD D,A ;CLEAR DE
'0221  5F        0182        LD E,A
'0222  C9        0183        RET ;RETURN TO CALLING PROGRAM
'0223  7B        0184 ADJ1   LD A,E
'0224  4B        0185        LD C,E ;IF MPX < 0, ADD MPY TO FUDGE
'0225  18E4      0186        JR RET1
'0227  82        0187 ADJ2   ADD A,D ;IF MPY < 0, ADD MPX TO FUDGE
'0228  4F        0188        LD C,A
'0229  18E4      0189        JR RET2
                 0190        END

ERRORS=0000
```

APPENDIX 2

Z80 ASSEMBLER PROGRAM FOR THE WIDROW ADP WITH HARDWARE MULTIPLY

This program executes the Widrow LMS adaptive digital predictor with hardware multiply (WADPHM). It is fuctionally identical to the original software multiply version given in Appendix 1. The hardware multiply driver subroutine exchanges data with the main routine in exactly the same manner as its software counterpart. Due to the hardware multiply, the subroutine executes twice as fast (70.5 microseconds as compared to 147.25 microseconds) as the software multiply. For details, refer to Chapter IV.

```
PASS 1 DONE
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0001
 ADDR  CODE       STMT SOURCE STATEMENT

                  0001 ;THIS IS A MODIFICATION OF THE WIDROW ADAPTIVE
                  0002 ;DIGITAL PREDICTOR ALGORITHM DEVELOPED BY D. NICKEL
                  0003 ;FOR HIS MASTER'S THESIS.  IT HAS BEEN MODIFIED
                  0004 ;FOR USE ON THE NSC800 SBC WITH HARDWARE MULTIPLY.
                  0005 ;THE SOFTWARE MULTIPLY SUBROUTINE HAS BEEN REPLACED
                  0006 ;WITH A HARDWARE MULTIPLY DRIVER SUBROUTINE WHICH
                  0007 ;USES THE SAME DATA FORMAT AS THE ORIGINAL ROUTINE.
                  0008 ;OTHER CHANGES IN THE ORIGINAL SOFTWARE INCLUDE
                  0009 ;MOVING THE RAM ADDRESSING OF THE BUFFERS
                  0010 ;TO 0400H-046BH.  THE ADC IS LOCATED AT 1402H AND
                  0011 ;THE OUTPUT PORT IS NOW THE DAC WHICH IS LOCATED
                  0012 ;AT 1400H.
                  0013 ;
                  0014 ;                    INITIALIZATION
'0000  3E20       0015 WADPHM LD A,00100000B ;SET MDU'S CONTROL REG. TO INDICATE
'0002  320310     0016        LD (1003H),A ;2 UNITS ARE BEING USED. NO UNIT
                  0017 ;                      OPERATION, SEQUENCE COUNTER, SHIFT
                  0018 ;                      RATE SELECTOR ARE RESET AT POWER-ON
                  0019 ;                      RESET
'0005  066C       0020        LD B,6CH ;SET UP ITERATION COUNTER
'0007  AF         0021        XOR A    ;CLEAR A
'0008  216B04     0022        LD HL,046BH ;LOAD HL WITH MEMORY POINTER
'000B  77         0023 L1     LD (HL),A ;CLEAR MEMORY BYTE
'000C  2B         0024        DEC HL ;DECREMENT HL
'000D  10FC       0025        DJNZ L1 ;DECREMENT ITERATION COUNTER
                  0026 ;                AND REPEAT IF > 0
'000F  31FF04     0027        LD SP,04FFH ;INITIALIZE THE STACK POINTER.
                  0028 ;
                  0029 ;                  INPUT FROM A/D CONVERTER
'0012  AF         0030 START  XOR A ;CLEAR A
'0013  6F         0031        LD L,A ;CLEAR L
'0014  3A0214     0032        LD A,(1402H) ;INPUT SAMPLE FROM A/D
'0017  47         0033        LD B,A ;B=SAMPLE
'0018  3E7F       0034        LD A,7FH ;A=MASK
'001A  A8         0035        XOR B ;CONVERT SAMPLE TO 2'S COMPLEMENT FORM
'001B  67         0036        LD H,A ;H=SAMPLE
'001C  CB2C       0037        SRA H ;SCALE SAMPLE BY ARITHMETICALLY
'001E  CB1D       0038        RR L  ;SHIFTING RIGHT HL
'0020  222004     0039        LD (0420H),HL ;STORE F(M) IN MEMORY
                  0040 ;
                  0041 ; COMPUTE G=SUM(K=1,16) F(M-K)*B(M,K)
'0023  210000     0042        LD HL,0000H ;CLEAR HL
'0026  226404     0043        LD (0464H),HL ;CLEAR LOW ORDER 16 BITS OF G
'0029  226604     0044        LD (0466H),HL ;CLEAR HIGH ORDER 16 BITS OF G
'002C  DD212204   0045        LD IX,0422H ;LOAD INDEX REG. X WITH ADDRESS
                  0046 ;                    OF L.O. BYTE OF B(M,16)
'0030  FD210004   0047        LD IY,0400H ;LOAD INDEX REG.Y WITH ADDRESS
                  0048 ;                    OF L.O. BYTE OF F(M-16)
'0034  0610       0049        LD B,10H ;SET UP ITERATION COUNTER
'0036  DD6E00     0050 SUM1   LD L,(IX+0H) ;HL=B(M,K)
'0039  DD6601     0051        LD H,(IX+1H)
'003C  FD5E00     0052        LD E,(IY+0H) ;DE=F(M-K)
'003F  FD5601     0053        LD D,(IY+1H)
'0042  C5         0054        PUSH BC ;SAVE THE COUNTER
'0043  CD0002'    0055        CALL MULT ;CALL THE MULTIPLY SUBROUTINE
'0046  DD23       0056        INC IX ;INCREMENT BUFFER POINTER
'0048  DD23       0057        INC IX
'004A  FD23       0058        INC IY ;INCREMENT BUFFER POINTER
```

```
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0002
 ADDR   CODE       STMT SOURCE STATEMENT

'004C   FD23       0059         INC IY
'004E   ED4B6404   0060         LD BC,(0464H) ;LOAD BC WITH L.O. 16 BITS
                   0061 ;                        OF G
'0052   EB         0062         EX DE,HL ;UPDATE L.O. 16 BITS OF A AND
'0053   09         0063         ADD HL,BC ;STORE IN MEMORY
'0054   226404     0064         LD (0464H),HL
'0057   EB         0065         EX DE,HL ;
'0058   ED4B6604   0066         LD BC,(0466H) ;LOAD BC WITH H.O. 16 BITS
                   0067 ;                        OF G
'005C   ED4A       0068         ADC HL,BC ;UPDATE H.O. 16 BITS OF G AND
'005E   226604     0069         LD (0466H),HL ;STORE IN MEMORY
'0061   C1         0070         POP BC ;RETRIEVE COUNTER
'0062   10D2       0071         DJNZ SUM1 ;DECREMENT COUNTER AND REPEAT IF >0
                   0072 ;
                   0073 ;            COMPUTE E(M)=F(M)-G
                   0074 ;NOTE: ENTER THIS ROUTINE WITH HL=G (H.O. 16 BITS)
'0064   EB         0075         EX DE,HL ;DE=G (H.O. 16 BITS)
'0065   2A2004     0076         LD HL,(0420H) ;HL=F(M)
'0068   AF         0077         XOR A ;CLEAR A,CLEAR CARRY
'0069   ED52       0078         SBC HL,DE ;HL=F(M)-G=E(M)
'006B   CB2C       0079         SRA H
'006D   CB1D       0080         RR L
'006F   CB2C       0081         SRA H
'0071   CB1D       0082         RR L ;DIVIDE BY 2**4 TO FORM
'0073   CB2C       0083         SRA H ;HL=E(M)/2**4
'0075   CB1D       0084         RR L
'0077   CB2C       0085         SRA H
'0079   CB1D       0086         RR L
'007B   226204     0087         LD (0462H),HL ;STORE E(M)/2**4 IN MEMORY
                   0088 ;
                   0089 ;    UPDATE THE WEIGHTS B(M+1,K)=U*B(M,K)+V*E(M)*F(M-K)
                   0090 ;NOTE: ENTER THIS ROUTINE WITH V*E(M)=E(M)/2**4 IN HL.
'007E   DD212204   0091         LD IX,0422H ;LOAD INDEX REG. X WITH THE ADDRESS
                   0092 ;                       OF  THE L.O. BYTE OF B(M,16)
'0082   FD210004   0093         LD IY,0400H ;LOAD INDEX REG. Y WITH THE ADDRESS
                   0094 ;                       OF THE L.O. BYTE OF F(M-16)
'0086   226A04     0095         LD (046AH),HL ;STORE V*E(M) IN MEMORY
'0089   0610       0096         LD B,10H ;SET UP ITERATION COUNTER
'008B   C5         0097 L2      PUSH BC ;SAVE COUNTER
'008C   2A6A04     0098         LD HL,(046AH) ;HL=V*E(M)
'008F   FD5E00     0099         LD E,(IY+0H) ;DE=F(M-16)
'0092   FD5601     0100         LD D,(IY+1H)
'0095   CD0002'    0101         CALL MULT ;CALL MULTIPLY SUBROUTINE
'0098   DD5E00     0102         LD E,(IX+0H) ;DE=B(M,K)
'009B   DD5601     0103         LD D,(IX+1H)
'009E   EB         0104         EX DE,HL
'009F   44         0105         LD B,H ;FORM B(M,K)/2**10
'00A0   4C         0106         LD C,H
'00A1   CB29       0107         SRA C
'00A3   CB29       0108         SRA C
'00A5   AF         0109         XOR A ;CLEAR A
'00A6   CB10       0110         RL B ;ROTATE M.S.B. INTO CARRY
'00A8   3001       0111         JR NC,L3 ;FILL B WITH SIGN BIT
'00AA   2F         0112         CPL
'00AB   47         0113 L3      LD B,A
'00AC   AF         0114         XOR A ;CLEAR A
'00AD   ED42       0115         SBC HL,BC ;HL=U*B(M,K)=B(M,K)*(1-2**10)
'00AF   ED5A       0116         ADC HL,DE ;HL=B(M+1,K)
```

```
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0003
 ADDR   CODE       STMT SOURCE STATEMENT

'00B1   DD7500     0117        LD (IX+0H),L ;STORE UPDATED B(M+1,K) IN MEMORY
'00B4   DD7401     0118        LD (IX+1H),H
'00B7   DD23       0119        INC IX ;INCREMENT POINTERS
'00B9   DD23       0120        INC IX
'00BB   FD23       0121        INC IY
'00BD   FD23       0122        INC IY
'00BF   C1         0123        POP BC ;RETRIEVE COUNTER
'00C0   10C9       0124        DJNZ,L2 ;DECREMENT COUNTER AND REPEAT IF > 0
                   0125 ;
                   0126 ;     COMPUTE Q(M)=(1/16)*SUM(K=1,16) E(M-K)
'00C2   2A4204     0127        LD HL,(0442H) ;HL=E(M-16)/16
'00C5   EB         0128        EX DE,HL ;EXCHANGE DE AND HL
'00C6   2A6204     0129        LD HL,(0462H) ;HL=E(M)/16
'00C9   AF         0130        XOR A ;CLEAR A, CLEAR CARRY
'00CA   ED52       0131        SBC HL,DE ;HL=(E(M)-E(M-16)/16
'00CC   EB         0132        EX DE,HL ;EXCHANGE DE AND HL
'00CD   2A6804     0133        LD HL,(0468H) ;HL=OLD Q(M)
'00D0   19         0134        ADD HL,DE ;HL=NEW Q(M)
'00D1   226804     0135        LD (0468H),HL ;STORE NEW VALUE OF Q(M)
                   0136 ;                        IN MEMORY
'00D4   54         0137        LD D,H ;DE=HL=Q(M)
'00D5   5D         0138        LD E,L
'00D6   CD0002'    0139        CALL MULT ;CALL THE MULTIPLY SUBROUTINE
                   0140 ;                   TO FORM HL=Q(M)**2/16**2
'00D9   CB25       0141        SLA L
'00DB   CB14       0142        RL H
'00DD   CB25       0143        SLA L
'00DF   CB14       0144        RL H
'00E1   CB25       0145        SLA L ;MULTIPLY BY 16 TO FORM
'00E3   CB14       0146        RL H ; HL=Q(M)**2/16
'00E5   CB25       0147        SLA L
'00E7   CB14       0148        RL H
'00E9   7C         0149        LD A,H
'00EA   EE80       0150        XOR 80H ;CONDITION DATA FOR DAC.
'00EC   320014     0151        LD (1400H),A ;OUTPUT RESULT TO DAC
                   0152 ;
                   0153 ; BLOCK MOVE OF E AND F BUFFERS
'00EF   114204     0154        LD DE,0442H ;LOAD TARGET ADDRESS
'00F2   214404     0155        LD HL,0444H ;LOAD SOURCE ADDRESS
'00F5   012000     0156        LD BC,0020H ;LOAD NUMBER OF BYTES TO BE MOVED
'00F8   EDB0       0157        LDIR ;BLOCK MOVE OF E BUFFER
'00FA   110004     0158        LD DE,0400H ;LOAD TARGET ADDRESS
'00FD   210204     0159        LD HL,0402H ;LOAD SOURCE ADDRESS
'0100   012000     0160        LD BC,0020H ;LOAD NUMBER OF BYTES TO BE MOVED
'0103   EDB0       0161        LDIR ;BLOCK MOVE OF F BUFFER
'0105   C31200'    0162        JP START ;JUMP TO INPUT ROUTINE FROM A/D ROUTINE
                   0163 ;
                   0164 ; MULTIPLY SUBROUTINE
 >0200             0165        ORG 200H
'0200   AF         0166 MULT   XOR A ;CLEAR A ;D=MPX, H=MPY
'0201   4F         0167        LD C,A
'0202   CB7A       0168        BIT 7,D ;TEST SIGN OF MPX
'0204   2030       0169        JR NZ,ADJ1 ;IF NEGATIVE, GO TO ADJ1, ELSE CONTINUE
'0206   CB7C       0170 RET1   BIT 7,H ;TEST SIGN OF MPY
'0208   2030       0171        JR NZ,ADJ2 ;IF NEGATIVE, GO TO ADJ2 ELSE CONTINUE
'020A   3E64       0172 RET2   LD A,01100100B ;RESET SEQUENCE COUNTER, REG. Z
'020C   320310     0173        LD (1003H),A   ;AND MAKE NO OPERATION.
'020F   7A         0174        LD A,D ;PUT OPERAND IN UPPER X REG. .
```

```
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0004
 ADDR  CODE      STMT SOURCE STATEMENT

'0210  320010    0175        LD (1000H),A
'0213  AF        0176        XOR A ;CLEAR LOWER X REG. .
'0214  320010    0177        LD (1000H),A
'0217  7C        0178        LD A,H ;PUT OPERATOR IN UPPER Z REG. .
'0218  320110    0179        LD (1001H),A ;LOWER Z REG. IS ALREADY CLEAR.
'021B  3E69      0180        LD A,69H ;RESET SEQUENCE COUNTER, CLEAR Y REG.
'021D  320310    0181        LD (1003H),A ;AND START MULTIPLY.
'0220  00        0182        NOP
'0221  00        0183        NOP
'0222  00        0184        NOP
'0223  00        0185        NOP ;WAIT FOR MULTIPLY TO FINISH.
'0224  00        0186        NOP
'0225  00        0187        NOP
'0226  00        0188        NOP
'0227  3A0210    0189        LD A,(1002H) ;LOAD HIGH BYTE OF RESULT
'022A  67        0190        LD H,A          ;IN H REG. .
'022B  3A0210    0191        LD A,(1002H) ;LOAD NEXT HIGH BYTE OF RESULT
'022E  6F        0192        LD L,A        ;INTO L REG. .
'022F  7C        0193        LD A,H ;SUBTRACT CONTENTS OF FUDGE REGISTER
'0230  91        0194        SUB C ; FROM HIGH PART OF RESULT
'0231  67        0195        LD H,A
'0232  AF        0196        XOR A
'0233  57        0197        LD D,A ;CLEAR DE
'0234  5F        0198        LD E,A
'0235  C9        0199        RET ;RETURN TO CALLING PROGRAM
'0236  7C        0200 ADJ1   LD A,H
'0237  4C        0201        LD C,H ;IF MPX < 0, ADD MPY TO FUDGE
'0238  18CC      0202        JR RET1
'023A  82        0203 ADJ2   ADD A,D ;IF MPY < 0, ADD MPX TO FUDGE
'023B  4F        0204        LD C,A
'023C  18CC      0205        JR RET2
                 0206        END

ERRORS=0000
```

APPENDIX 3

Z80 ASSEMBLY PROGRAM LISTING FOR THE ADAPTIVE LATTICE PREDICTOR

Memory Organization

Location	Contents	Location	Contents
0400 HEX	e(1) low	0434	wl(9) low
0401	e(1) high	0435	wl(9) high
0402	e(2) low	0436	k(1) low
0403	e(2) high	0437	k(1) high
.	.	0438	k(2) low
.	.	0439	k(2) high
.	.	.	.
0410	e(9) low	.	.
0411	e(9) high	.	.
0412	w(1) low	0444	k(8) low
0413	w(1) high	0445	k(8) high
0414	w(2) low	0446	v(1) low
0415	w(2) high	0447	v(1) high
.	.	0448	v(2) low
.	.	0449	v(2) high
.	.	.	.
0422	w(9) low	.	.
0423	w(9) high	.	.
0424	wl(1) low	.	.
0425	wl(1) high	0454	v(8) low
.	.	0455	v(8) high
.	.		
.	.		

In order to follow the floating point arithmetic operations in this program, a standard discriptive notation has been included. In the general form it is given as (S/I/F), where the variables describe the number of sign, integer, and fraction bit respectively. One is refered to "A Block Floating-Point Notation for Signal Processes" (SAND79-1823) by James E. Simpson for details.

```
PASS 1 DONE
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0001
 ADDR  CODE       STMT SOURCE STATEMENT

                  0001 ;      LATZ80 IS AN ADAPTIVE LATTICE PREDICTOR PROGRAM
                  0002 ;WRITTEN IN Z80 ASSEMBLY LANGUAGE FOR THE NSC800 SBC.
                  0003 ;THE ROUTINE IS SET UP WITH AN EIGHT-STAGE LATTICE.
                  0004 ;INPUT DATA IS BROUGHT IN THROUGH THE 12-BIT ADC AND
                  0005 ;THEN CONDITIONED BY SOFTWARE FOR USE IN THE ALGORITHM.
                  0006 ;ALL MATHEMATICAL OPERATIONS OF THE LATTICE PROGRAM ARE
                  0007 ;DONE IN FIXED-POINT INTEGER ARITHMETIC WITH A SIXTEEN
                  0008 ;BIT WORD.  MULTIPLICATION AND DIVISION OPERATIONS ARE
                  0009 ;SIGNED BUT ARE DONE IN HARDWARE ON A CDP1855 MDU.  IT
                  0010 ;PERFORMS UNSIGNED MULTIPLICATIONS AND DIVISIONS HENCE,
                  0011 ;TWO HARDWARE DRIVER ROUTINES ARE NEEDED TO CONVERT THE
                  0012 ;SIGNED NUMBERS TO UNSIGNED NUMBERS AND BACK WHEN THESE
                  0013 ;OPERATIONS ARE DONE.  THE OUTPUT OF THE LATTICE FILTER
                  0014 ;IS SENT TO THE 8-BIT DAC FOR COMPARISON TO THE ORIGINAL
                  0015 ;SIGNAL.
                  0016 ;
                  0017 ;INITIALIZATION OF THE MDU, POINTERS, REGISTERS, AND
                  0018 ;ARRAYS.  THE PROGRAM STARTS AT THIS POINT UPON RESET.
                  0019 ;
 >0000            0020        ORG 0000H
'0000  3E20       0021 LATZ80 LD A,00100000B ;SET MDU'S CONTROL REG. TO INDICATE
'0002  320310     0022        LD (1003H),A ;  2 UNITS ARE BEING USED. NO UNIT
                  0023 ;                      OPERATION, SEQUENCE COUNTER, SHIFT
                  0024 ;                      RATE SELECTOR ARE RESET AT POWER-
                  0025 ;                      ON RESET.
'0005  0656       0026        LD B,56H ;SET UP ITERATION COUNTER.
'0007  AF         0027        XOR A        ;CLEAR A.
'0008  215504     0028        LD HL,0455H ;LOAD HL WITH MEMORY POINTER.
'000B  77         0029 L1     LD (HL),A ;CLEAR A MEMORY BYTE.
'000C  2B         0030        DEC HL     ;DECREMENT HL.
'000D  10FC       0031        DJNZ L1 ;DECREMENT ITERATION COUNTER AND REPEAT
                  0032 ;                IF > 0
'000F  31FF04     0033        LD SP,04FFH ;INITIALIZE THE STACK POINTER AND
'0012  FD210010   0034        LD IY,1000H ;POINTER FOR MDU REGISTERS.
                  0035 ;
                  0036 ;INITIALIZATION OF THE ARRAY POINTER AND STAGE LOOP
                  0037 ;COUNTER.  THIS IS THE START OF THE LATTICE ROUTINE.
                  0038 ;
'0016  DD210004   0039 LSTART LD IX,0400H ;INITIALIZE POINTER FOR FORWARD
                  0040 ;                      PREDICTION ERROR ARRAY.
'001A  0608       0041        LD B,08H ;INITIALIZE STAGE LOOP COUNTER.
'001C  2A0114     0042        LD HL,(1401H) ;LOAD HL WITH SAMPLE FROM ADC.
'001F  7D         0043        LD A,L
'0020  2F         0044        CPL
'0021  6F         0045        LD L,A
'0022  7C         0046        LD A,H ;CONVERT DATA TWO 2'S COMPLEMENT FORM.
'0023  EE7F       0047        XOR 7FH
'0025  67         0048        LD H,A ;INPUT NOW IN FORM (1/0/11)
'0026  CB2C       0049        SRA H ;SCALE SAMPLE BY ARITHMETICALLY
'0028  CB1D       0050        RR L  ;SHIFTING RIGHT HL. (2/0/11)
'002A  DD7500     0051        LD (IX+00H),L ;PLACE SAMPLE IN FORWARD PREDICTION
'002D  DD7401     0052        LD (IX+01H),H ;ERROR ARRAY LOCATION ZERO. (2/0/14)
'0030  DD7512     0053        LD (IX+12H),L ;PLACE SAMPLE IN BACKWARD PREDICTION
'0033  DD7413     0054        LD (IX+13H),H ;ERROR ARRAY LOCATION ZERO. (2/0/14)
                  0055 ;
                  0056 ;THE LATTICE STAGE LOOP STARTS HERE
                  0057 ;CALCULATION OF E(J+1) = E(J) - K(J)*W1(J)
                  0058 ;
```

```
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0002
 ADDR  CODE       STMT SOURCE STATEMENT

'0036  C5         0059 STGLOP PUSH BC ;SAVE STAGE COUNTER.
'0037  DD6E36     0060        LD L,(IX+36H) ;LOAD LATTICE COEFFICIENT INTO
'003A  DD6637     0061        LD H,(IX+37H) ;THE HL REGISTER. (2/0/14)
'003D  DD5E24     0062        LD E,(IX+24H) ;LOAD BACKWARD PREDICTION ERROR OF
'0040  DD5625     0063        LD D,(IX+25H) ;LAST ITERATION INTO DE REGISTER.
                  0064 ;                        (2/0/14)
'0043  CD0002'    0065        CALL MULT ;MULTIPLY THE TWO VALUES TOGETHER.
                  0066 ;                    (4/0/12)
'0046  CB25       0067        SLA L
'0048  CB14       0068        RL H
'004A  CB25       0069        SLA L
'004C  CB14       0070        RL H   ;SCALE THE RESULT. (2/0/14)
'004E  EB         0071        EX DE,HL ;MOVE RESULT TO DE REGISTER.
'004F  DD6E00     0072        LD L,(IX+00H) ;LOAD FORWARD PREDICTION ERROR
'0052  DD6601     0073        LD H,(IX+01H) ;INTO HL REGISTER. (2/0/14)
'0055  37         0074        SCF
'0056  3F         0075        CCF ;CLEAR THE CARRY FLAG.
'0057  ED52       0076        SBC HL,DE ;THIS OBTAINS THE DESIRED RESULT.
'0059  DD7502     0077        LD (IX+02H),L ;STORE RESULT IN NEXT CELL OF THE
'005C  DD7403     0078        LD (IX+03H),H ;FORWARD PREDICTION ERROR ARRAY.
                  0079 ;                        (2/0/14)
                  0080 ;CALCULATTION OF W(J+1) = W1(J) - K(J)*E(J)
                  0081 ;
'005F  DD6E36     0082        LD L,(IX+36H) ;LOAD LATTICE COEFFICIENT INTO
'0062  DD6637     0083        LD H,(IX+37H) ;THE HL REGISTER. (2/0/14)
'0065  DD5E00     0084        LD E,(IX+00H) ;LOAD THE FORWARD PREDICTOR ERROR
'0068  DD5601     0085        LD D,(IX+01H) ;VALUE IN THE DE REGISTER. (2/0/14)
'006B  CD0002'    0086        CALL MULT ;MULTIPLY THE TWO VALUES TOGETHER.
                  0087 ;                    (4/0/12)
'006E  CB25       0088        SLA L
'0070  CB14       0089        RL H
'0072  CB25       0090        SLA L
'0074  CB14       0091        RL H ;SCALE THE RESULT. (2/0/14)
'0076  EB         0092        EX DE,HL ;MOVE RESULT TO DE REGISTER.
'0077  DD6E24     0093        LD L,(IX+24H) ;LOAD BACKWARD PREDICTION ERROR OF
'007A  DD6625     0094        LD H,(IX+25H) ;LAST ITERATION INTO HL REGISTER.
'007D  37         0095        SCF   ;   (2/0/14)
'007E  3F         0096        CCF ;CLEAR THE CARRY FLAG.
'007F  ED52       0097        SBC HL,DE ;THIS OBTAINS THE DESIRED RESULT.
'0081  DD7514     0098        LD (IX+14H),L ;STORE RESULT IN NEXT CELL OF THE
'0084  DD7415     0099        LD (IX+15H),H ;BACKWARD PREDICTION ERROR ARRAY.
                  0100 ;                        (2/0/14)
                  0101 ;THE FOLLOWING ROUTINE CALCULATES
                  0102 ;V(J) = BETA*V(J) + BETA1*[ E(J)*E(J) + W1(J)*W1(J) ]
                  0103 ;
'0087  DD6E00     0104        LD L,(IX+00H) ;LOAD FORWARD PREDICTION ERROR
'008A  DD6601     0105        LD H,(IX+01H) ;INTO THE HL REGISTER. (2/0/14)
'008D  5D         0106        LD E,L
'008E  54         0107        LD D,H ;ALSO PUT IT IN THE DE REGISTER. (2/0/14)
'008F  CD0002'    0108        CALL MULT ;OBTAIN E(J)*E(J). (4/0/12)
'0092  E5         0109        PUSH HL ;SAVE RESULT UNTIL LATER.
'0093  DD6E24     0110        LD L,(IX+24H) ;LOAD BACKWARD PREDICTION ERROR OF
'0096  DD6625     0111        LD H,(IX+25H) ;THE LAST ITERATION IN HL REGISTER.
'0099  5D         0112        LD E,L ;        (2/0/14)
'009A  54         0113        LD D,H ;ALSO PLACE IT IN THE DE REGISTER. (2/0/14)
'009B  CD0002'    0114        CALL MULT ;OBTAIN W1(J)*W1(J). (4/0/12)
'009E  D1         0115        POP DE ;RETRIEVE V(J)*V(J). (4/0/12)
'009F  19         0116        ADD HL,DE ;OBTAINS [ E(J)*E(J) + W1(J)*W1(J) ]
```

```
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0003
 ADDR  CODE      STMT SOURCE STATEMENT

'00A0  CB25      0117        SLA L ;      (4/0/12)
'00A2  CB14      0118        RL H
'00A4  CB25      0119        SLA L
'00A6  CB14      0120        RL H  ;SCALE RESULT. (2/0/14)
'00A8  EB        0121        EX DE,HL ;PLACE RESULT IN DE REGISTER. (2/0/14)
'00A9  2A3C01'   0122        LD HL,(BETA1) ;LOAD BETA1 INTO HL REGISTER.
                 0123 ;                          (1/0/15)
'00AC  CD0002'   0124        CALL MULT ;GETS BETA1*[ E(J)*E(J) + W1(J)*W1(J) ].
'00AF  E5        0125        PUSH HL ;SAVE RESULT FOR LATER USE. (3/0/13)
'00B0  DD5E46    0126        LD E,(IX+46H) ;LOAD VARIANCE TERM
'00B3  DD5647    0127        LD D,(IX+47H) ;INTO DE REGISTER. (2/0/14)
'00B6  2A3A01'   0128        LD HL,(BETA) ;LOAD BETA INTO HL REGISTER. (1/0/15)
'00B9  CD0002'   0129        CALL MULT ;OBTAIN BETA*V(J). (3/0/13)
'00BC  D1        0130        POP DE ;RETRIEVE PREVIOUSLY CALCULATED TERM.
                 0131 ;                  (3/0/13)
'00BD  19        0132        ADD HL,DE ;OBTAINS FINAL RESULT. (3/0/13)
'00BE  CB25      0133        SLA L
'00C0  CB14      0134        RL H ;ADJUST RESULT. (2/0/14)
'00C2  DD7546    0135        LD (IX+46H),L ;PUT FINAL RESULT
'00C5  DD7447    0136        LD (IX+47H),H ;IN VARIANCE ARRAY. (2/0/14)
                 0137 ;
                 0138 ;THIS ROUTINE CALCULATES THE FILTER COEFFICIENT FOR
                 0139 ;THIS STAGE OF THE LATTICE FILTER.
                 0140 ;K(J) = K(J) + ALPHA*[ E(J+1)*W1(J) + E(J)*W(J+1) ]/V(J)
                 0141 ;FIRST, DETERMINE IF THE VARIANCE HAS BECOME TOO SMALL,
                 0142 ;I.E., LESS THAN EPSILON.  V(J) IS IN THE HL REGISTER.
                 0143 ;
'00C8  EB        0144        EX DE,HL ;PLACE V(J) IN THE DE REGISTER. (2/0/14)
'00C9  2A3E01'   0145        LD HL,(EPSLN) ;LOAD EPSILON INTO HL REGISTER.
'00CC  37        0146        SCF      ;  (2/0/14)
'00CD  3F        0147        CCF ;CLEAR CARRY FLAG.
'00CE  ED52      0148        SBC HL,DE ;FIND EPSILON - V(J).
'00D0  210000    0149        LD HL,0000H ;SET COEFF. ADJUST TO ZERO. (2/0/14)
'00D3  F20B01'   0150        JP P TOOLOW ;EPSILON?  IF SO, SKIP CALCULATION.
'00D6  DD6E00    0151        LD L,(IX+00H) ;LOAD E(J) IN HL REGISTER.
'00D9  DD6601    0152        LD H,(IX+01H) ;  (2/0/14)
'00DC  DD5E14    0153        LD E,(IX+14H) ;LOAD W(J+1) IN DE REGISTER.
'00DF  DD5615    0154        LD D,(IX+15H) ;  (2/0/14)
'00E2  CD0002'   0155        CALL MULT ;OBTAIN E(J)*W(J+1). (4/0/12)
'00E5  E5        0156        PUSH HL ;SAVE RESULT FOR LATER USE.
'00E6  DD6E24    0157        LD L,(IX+24H) ;LOAD W1(J) IN HL REGISTER. (2/0/14)
'00E9  DD6625    0158        LD H,(IX+25H)
'00EC  DD5E02    0159        LD E,(IX+02H) LOAD E(J+1) IN DE REGISTER. (2/0/14)
'00EF  DD5603    0160        LD D,(IX+03H)
'00F2  CD0002'   0161        CALL MULT ;OBTAIN E(J+1)*W1(J). (4/0/12)
'00F5  D1        0162        POP DE ;RETRIEVE PRIOR RESULT. (4/0/12)
'00F6  19        0163        ADD HL,DE ;OBTAIN E(J+1)*W1(J) + E(J)*W(J+1).
                 0164 ;                  (4/0/12)
'00F7  DD5E46    0165        LD E,(IX+46H) ;LOAD V(J) IN DE REGISTER. (2/0/14)
'00FA  DD5647    0166        LD D,(IX+47H)
'00FD  CD8002'   0167        CALL DIVIDE ;[ E(J+1)*W1(J) + E(J)*W(J+1) ]/V(J).
                 0168 ;                  (2/0/14)
'0100  EB        0169        EX DE,HL ;PUT QUOTIENT IN DE REGISTER. (2/0/14)
'0101  2A3801'   0170        LD HL,(ALPHA) ;LOAD ALPHA IN HL REGISTER. (1/0/15)
'0104  CD0002'   0171        CALL MULT ;OBTAIN K(J) MODIFIER. (3/0/13)
'0107  CB25      0172        SLA L
'0109  CB14      0173        RL H ;RESCALE THE RESULT.  (2/0/14)
'010B  DD5E36    0174 TOOLOW LD E,(IX+36H) ;LOAD LATTICE FILTER COEFFICIENT
```

```
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0004
 ADDR  CODE       STMT SOURCE STATEMENT

'010E  DD5637     0175         LD D,(IX+37H) ;INTO THE DE REGISTER. (2/0/14)
'0111  19         0176         ADD HL,DE ;OBTAIN NEW LATTICE FILTER CONSTANT.
'0112  DD7536     0177         LD (IX+36H),L ;STORE NEW FILTER CONSTANT
'0115  DD7437     0178         LD (IX+37H),H ;IN FILTER CONSTANT ARRAY. (2/0/14)
                  0179 ;
                  0180 ;UPDATE THE BACKWARD PREDICTOR ERROR ARRAY FOR THE LAST
                  0181 ;ITERATION WITH THE VALUE OBTAINED IN THIS ITERATION.
                  0182 ;
'0118  DD6E12     0183         LD L,(IX+12H) ;LOAD BACKWARD PREDICTION ERROR FOR
'011B  DD6613     0184         LD H,(IX+13H) ;THIS ITERATION IN THE HL REGISTER.
                  0185 ;                          (2/0/14)
'011E  DD7524     0186         LD (IX+24H),L ;STORE IN ARRAY CELL OF BACKWARD
'0121  DD7425     0187         LD (IX+25H),H ;PREDICTION ERROR OF LAST ITERATION.
                  0188 ;                          (2/0/14)
'0124  C1         0189         POP BC ;RESTORE STAGE COUNTER.
                  0190 ;
                  0191 ;CHECK FOR COMPLETION OF STAGES.  IF SO, OUTPUT THE
                  0192 ;FORWARD PREDICTION ERROR TO THE DAC (HIGH 8 BITS).
                  0193 ;
'0125  DD23       0194         INC IX
'0127  DD23       0195         INC IX ;POINT TO NEXT STAGE OF LATTICE.
'0129  05         0196         DEC B
'012A  C23600'    0197         JP NZ STGLOP ;IF NOT DONE, GO TO NEXT STAGE.
'012D  DD7E01     0198         LD A,(IX+01H) ;LOAD HIGH BYTE OF PREDICTION ERROR.
'0130  EE80       0199         XOR 80H ;CONDITION DATA FOR DAC.
'0132  320014     0200         LD (1400H),A ;STORE OUT TO DAC.
'0135  C31600'    0201         JP LSTART ;GO TO START OF LATTICE LOOP.
                  0202 ;
                  0203 ;THE FOLLOWING ARE FILTER CONSTANTS:
                  0204 ;
'0138  8F02       0205 ALPHA   DEFW 028FH ;ALPHA = .02  (1/0/15)
'013A  717D       0206 BETA    DEFW 7D71H ;BETA = 0.98  (1/0/15)
'013C  8F02       0207 BETA1   DEFW 028FH ;BETA1 = 7FFF - BETA + 1 = .02 (1/0/15)
'013E  0100       0208 EPSLN   DEFW 0001H ;EPSILON = 0.000064 (2/0/14)
                  0209 ;
                  0210 ;THIS SUBROUTINE IS THE HARDWARE MULTIPLY DRIVER
                  0211 ;SOFTWARE.  THE SUBROUTINE TAKES TWO SIGNED 16-BIT
                  0212 ;INTEGER NUMBERS IN THE HL ADND DE REGISTERS AND
                  0213 ;RETURNS A 16-BIT SIGNED NUMBER IN THE HL REGISTER.
                  0214 ;
 >0200            0215         ORG 200H
'0200  44         0216 MULT    LD B,H
'0201  4D         0217         LD C,L ;PUT HL IN BC REGISTER.
'0202  210000     0218         LD HL,0000H ;CLEAR THE HL REGISTER.
'0205  CB78       0219         BIT 7,B ;TEST SIGN OF NUMBER IN BC REGISTER.
'0207  2801       0220         JR Z ARND1 ;IF POSITIVE, SKIP FUDGE FOR DE.
'0209  19         0221         ADD HL,DE ;PUT DE IN FUDGE REGISTER.
'020A  CB7A       0222 ARND1   BIT 7,D ;TEST SIGN OF NUMBER IN DE REGISTER.
'020C  2801       0223         JR Z ARND2 ;IF POSITIVE, SKIP FUDGE FOR BC.
'020E  09         0224         ADD HL,BC ;ADD BC TO FUDGE REGISTER.
'020F  3E68       0225 ARND2   LD A,01101000B ;RESET SEQUENCE COUNTER, REG. Y,
'0211  320310     0226         LD (1003H),A   ;AND MAKE NO OPERATION.
'0214  FD7200     0227         LD (IY+00H),D ;PUT MULTIPLIER IN X REGISTER.
'0217  FD7300     0228         LD (IY+00H),E
'021A  FD7001     0229         LD (IY+01H),B ;PUT MULTIPLICAND IN Z REGISTER.
'021D  FD7101     0230         LD (IY+01H),C
'0220  44         0231         LD B,H
'0221  4D         0232         LD C,L ;PUT FUDGE REGISTER IN BC REGISTER.
```

```
                                        SD SYSTEMS Z80 ASSEMBLER PAGE 0005
 ADDR  CODE       STMT SOURCE STATEMENT

'0222  3E61       0233        LD A,01100001B ;RESET SEQUENCE COUNTER AND
'0224  320310     0234        LD (1003H),A   ;START THE MULTIPLY.
'0227  00         0235        NOP
'0228  00         0236        NOP
'0229  00         0237        NOP
'022A  00         0238        NOP ;WAIT FOR THE MULTIPLY TO COMPLETE.
'022B  FD6602     0239        LD H,(IY+02H) ;LOAD HIGH WORD OF RESULT
'022E  FD6E02     0240        LD L,(IY+02H) ;INTO THE HL REGISTER.
'0231  37         0241        SCF
'0232  3F         0242        CCF ;CLEAR THE CARRY FLAG.
'0233  ED42       0243        SBC HL,BC ;SUBTRACT FUDGE REGISTER FROM RESULT.
'0235  C9         0244        RET
                  0245 ;
                  0246 ;THIS SUBROUTINE IS THE HARDWARE DIVIDE DRIVER SOFTWARE.
                  0247 ;IT TAKES TWO SIGNED 16-BIT NUMBERS (DIVISOR IN THE
                  0248 ;DE REGISTER AND DIVIDEND IN THE HL REGISTER) AND
                  0249 ;RETURNS A 16-BIT QUOTIENT IN THE HL REGISTER.  THE
                  0250 ;LOWER 16 BITS OF THE DIVIDEND ARE SET TO ZERO.
                  0251 ;
 >0280            0252        ORG 280H
'0280  0600       0253 DIVIDE LD B,00H ;SET SIGN HOLDER TO ZERO (POSITIVE).
'0282  CB7C       0254        BIT 7,H ;DETERMINE SIGN OF DIVIDEND.
'0284  2809       0255        JR Z POSTVE ;IF POSITIVE SKIP CHANGE OF SIGN.
'0286  06FF       0256        LD B,0FFH ;SET SIGN HOLDER TO NEGATIVE (FFH)
'0288  2B         0257        DEC HL ;CONVERT FROM 2'S COMPLEMENT.  SUBTRACT ONE,
'0289  7C         0258        LD A,H
'028A  2F         0259        CPL ;PERFORM 1'S COMPLEMENT ON RESULT.
'028B  67         0260        LD H,A
'028C  7D         0261        LD A,L
'028D  2F         0262        CPL
'028E  6F         0263        LD L,A
'028F  3E64       0264 POSTVE LD A,01100100B ;RESET SEQUENCE COUNTER,Y REG.,
'0291  320310     0265        LD (1003H),A   ;AND NO OPERATION.
'0294  FD7402     0266        LD (IY+02H),H ;LOAD DIVIDEND IN Y REGISTER OF MDU.
'0297  FD7502     0267        LD (IY+02H),L
'029A  FD7200     0268        LD (IY+00H),D ;LOAD DIVISOR IN X REGISTER OF MDU.
'029D  FD7300     0269        LD (IY+00H),E
'02A0  3E62       0270        LD A,01100010B ;RESET SEQUENCE COUNTER AND
'02A2  320310     0271        LD (1003H),A   ;START THE DIVIDE.
'02A5  00         0272        NOP
'02A6  00         0273        NOP
'02A7  00         0274        NOP
'02A8  00         0275        NOP ;WAIT FOR DIVIDE TO COMPLETE.
'02A9  FD6601     0276        LD H,(IY+01H) ;LOAD HL REGISTER WITH THE RESULT.
'02AC  FD6E01     0277        LD L,(IY+01H)
'02AF  04         0278        INC B ;CHECK THE SIGN HOLDER.
'02B0  C0         0279        RET NZ ;IF HOLDER ZERO,RESULT IS CORRECT; RETURN.
'02B1  7C         0280        LD A,H ;HOLDER FF, CONVERT TO NEGATIVE NUMBER.
'02B2  2F         0281        CPL
'02B3  67         0282        LD H,A
'02B4  7D         0283        LD A,L ;TAKE 1'S COMPLEMENT OF HL REGISTER.
'02B5  2F         0284        CPL
'02B6  6F         0285        LD L,A
'02B7  23         0286        INC HL ;ADD ONE TO OBTAIN 2'S COMPLEMENT.
'02B8  C9         0287        RET
                  0288 ;
                  0289        END

ERRORS=0000
```

AN EVALUATION OF THE NSC800 8-BIT MICROPROCESSOR FOR DIGITAL SIGNAL PROCESSING APPLICATIONS

by

MAC A. CODY

B. S., Kansas State University, 1979

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
1981

ABSTRACT

As use of signal processing algorithms in dedicated applications has become more commonplace, there has been a concurrent search for a microprocessor that can execute such algorithms in the field. The search has been partially successful, but the microprocessors investigated have proven undesirable due to a number of factors. These have chiefly been power consumption and cost considerations. The need for a microprocessor that could operate at high speed with low power use and low cost per unit was apparent.

This thesis covers the evaluation of a new microprocessor, the National Semiconductor NSC800, which may answer this need. In order to carry out the evaluation, a low-power, single board computer based on the NSC800 and its family of support devices is constructed. Next, studies are made of the performance of the computer and the NSC800 as hardware devices. Discussion of the difficulties involved in obtaining a working computer is made along with analysis of the results of hardware test routines. Finally, the microprocessor is evaluated in the role of a digital signal processor. Implementation of adaptive Widrow and lattice digital predictor algorithms is reviewed and software is developed for execution on the computer. The results of their performance and the limitations imposed by the support hardware are discussed.

With the completion of the evaluation above, a summary of the potential uses of the single board computer is made. A final analysis of the performance of the NSC800 is also presented.