

AD-A118 072

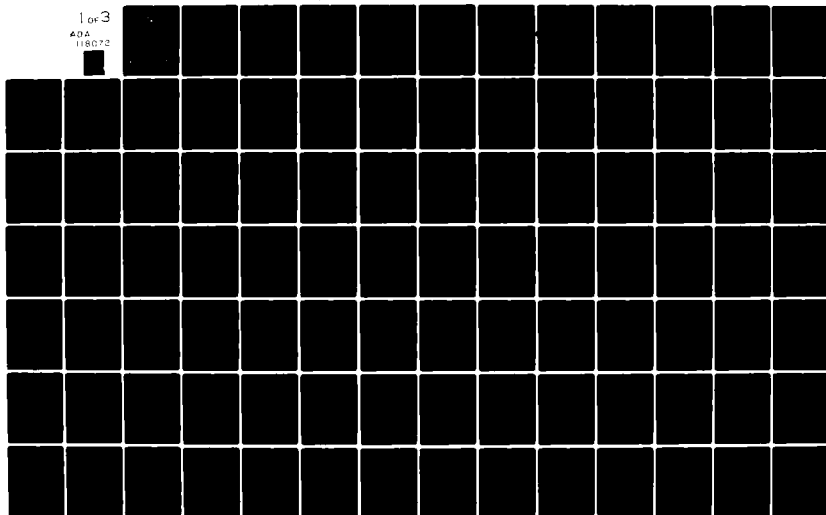
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCH00--ETC F/6 14/3
AN INFLIGHT RECORDER PROTOTYPE FOR THE INFLIGHT PHYSIOLOGICAL D--ETC(U)
FEB 82 R E WEISNER
AFIT/GCS/EE/82M-5

NL

UNCLASSIFIED

1 of 3

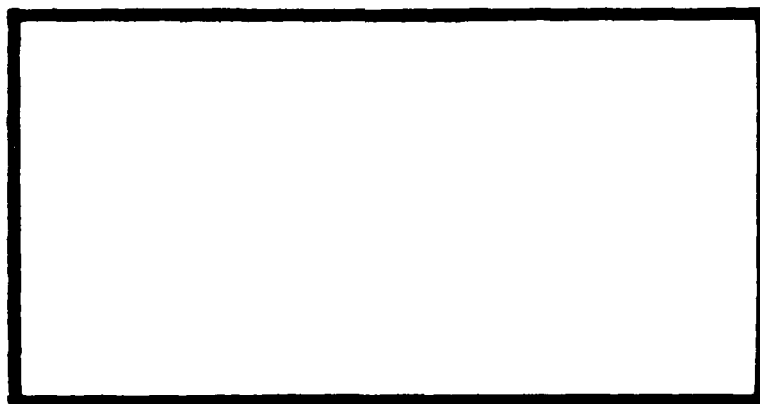
ADA
118072



198c

①

AD A118072



DTIC FILE COPY

DTIC
ELECTE
AUG 11 1982
S E D

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

82 08 11 067

AFIT/GCS/EE/82M-5

①

AN INFLIGHT RECORDER PROTOTYPE
FOR THE
INFLIGHT PHYSIOLOGICAL DATA
ACQUISITION SYSTEM III

THESIS

AFIT/GCS/EE/82M-5

Robert E. Meisner
Capt USAF

Approved for public release; distribution unlimited

AFIT/GCS/EE/82M-5

AN INFLIGHT RECORDER PROTOTYPE
FOR THE
INFLIGHT PHYSIOLOGICAL DATA
ACQUISITION SYSTEM III

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

by

Robert E. Meisner
Capt USAF

Graduate Computer Systems

February 1982

Accession For	
NTIS GRA&I	X
DTIC TAB	
Unannounced	
Justification	
By _____	
Distri _____	
Avail _____	
Dist _____	
A	



Approved for public release; distribution unlimited

Preface

This thesis is another in the long line of efforts aimed at building a better Inflight Recorder (IR) for the Inflight Physiological Data Acquisition System (IFPDAS). Previous theses analyzed different aspects of the IR problems and made recommendations for construction of an IR prototype. Some recommendations were followed, while others were updated to take advantage of new advances in IC designs. The primary product of this thesis is a hardware prototype for the IR. With the hardware built, continuing theses can concentrate on software development.

The people who helped bring the IR prototype to fruition are too numerous to mention individually. They include faculty, students, technicians and corporate representatives. There are, however, a few people who deserve individual recognition for the special attention they gave me. First of all I must thank Capt Hall and Lt Shackford of the School of Aerospace Medicine for insuring that the project was properly financed. Thanks also to Mike West of the Air Force Avionics Lab. His expertise and willingness to help were invaluable (and I truly mean invaluable) in developing Magnetic Bubble Memory hardware and software described in this thesis. I can not forget Orville Wright. His timely procurement of hard to find parts was instrumental to finishing this thesis. Thanks to Major Alan Ross, Dr Mathew Kabrisky, and Captain Larry

Kizer for their guidance throughout the thesis. I wholeheartedly recommend them as advisors to future degree candidates. Although not a member of my thesis committee, I appreciate the time that Major Walt Seward took to advise and critique my work.

Credit for the quality of this thesis must also go to my poker/bowling buddies. Their hard fought attempts to lead me into financial ruin provided an important link with reality. More seriously, I would like to acknowledge the help and understanding given me by my loving wife, Celeste, and daughter, Elizabeth. There is a direct correlation between their support for my endeavors and my academic accomplishments. I look forward to graduation so that we can spend more time together.

Contents

	Page
Preface	ii
List of Figures	vi
List of Tables	vii
List of Abbreviations	viii
Abstract	xi
I. Introduction	1
Background	1
Current System	2
Previous Studies	3
Problem Statement	4
Scope and Assumptions	5
Approach	6
Sequence of Presentation	6
II. Hardware Requirements Analysis	8
Required Characteristics	8
Desirable Characteristics	10
Preliminary Architecture	12
IC Technology	14
Evaluation	18
Main Processor	18
Secondary Storage	25
Program Memory	32
Data Acquisition Ports	34
Buffer Memory	34
Conclusions	36
III. Prototype Construction	38
Operating Voltage	39
IC Technology Mix	39
Board Layout	41
Bus Structure	42
CPU	44
System Clock	44
System Reset	46
Wait State Generator	47
Bus Demultiplexer	49
Primary Memory	51
Program Memory	52
Buffer Memory	54

Peripheral Devices	56
Timers	58
General I/O	60
A/D Converter	61
MBM	63
Interrupt Structure	65
Conclusion	68
IV. Hardware Verification Program	69
Buffer Memory	70
Timers	71
Timer 1	71
Timer 0	72
General I/O	73
A/D Converter	74
BMC	76
Conclusion	77
V. Conclusions and Recommendations	85
Conclusions	86
Recommendations	88
Bibliography	91
Appendix A: IR Prototype Schematic	93
Appendix B: EEPROM Programmer	99
Appendix C: MBM Interactive Development System	155
Appendix D: IFEDAS IR Debugging Tool	236
Appendix E: Manufacturers' Data Sheets	256
Vita	257

List of Figures

Figure	Page
1. A Preliminary IR PMS	13
2. The Proposed New IR	36
3. Major Component Map of IR Prototype	41
4. CPU	45
5. Wait State Generator	48
6. EEPROM	50
7. Conventional HNVM3008 Interface	53
8. RAM	55
9. I/O Ports, Timers, and A/D Converter	59
10. MBM	64
11. IR Prototype Verification Program	78
12. IR Prototype Schematic	94
13. EEPROM Programmer Schematic	106
14. EEPROM Programmer Flowchart	109
15. EEPROM Programmer Software	110
16. BPK-72 to S-100 Interface Schematic	160
17. MIDS Software	162
18. MBM Software	196
19. RDT I/O Buffers	242
20. Data Bus Monitor	245
21. Address Bus Monitor	245
22. IR Reset Function	248
23. Single Step Function	250
24. Memory/Peripheral I/O Circuit	252
25. RDT IC Functional Groupings	254

List of Tables

Table	Page
I. Sensor Sampling Rates	11
II. Comparison of Logic Families	15
III. Comparison of Microprocessors	20
IV. Microprocessor Criteria Ratings	24
V. Comparison of Secondary Storage Devices	26
VI. IC Family Voltage Characteristics	40
VII. IR Bus Connector Definition	43
VIII. I/O Port Mapping	57
IX. IR Interrupt Structure	66
X. EEPROM Programmer Selectable Ports	102
XI. EEPROM Programmer IC Listing	104
XII. S-100 to EEPROM Programmer Interface Definition	105
XIII. Selectable MBM I/O Ports	158
XIV. BPK-72 to S-100 IC Listing	159
XV. BPK-72 to S-100 Interface Definition	159
XVI. RDT IC Listing	255

List of Abbreviations

AFIT	US Air Force Institute of Technology, Wright-Patterson AFB, OH
ALE	Address Latch Enable Strobe (active high)
A/D	Analog-to-Digital
A##	Reference to Specific Bit (##) of the Address Bus
BACK*	Bus Request Acknowledgment Signal (active low)
BMC	Bubble Memory Controller
BREQ*	Bus Request Signal (active low)
CCD	Charge Coupled Device
CDOS	Cromemco Disk Operating System
CLK	System Clock
CMOS	Complementary Metal-Oxide Semiconductor
CPM	Control Program for Microprocessors
CPU	Central Processing Unit
CS*	Chip Select (active low)
D#	Reference to Specific Bit (#) of the Data Bus
ECG	Electrocardiogram
EEPROM	Electrically-Erasable Programmable Read Only Memory
EPROM	Erasable Programmable Read Only Memory
FIFO	First-In / First-Out
FF	Flip-Flop
GND	Electrical Ground
Gx	Lateral Acceleration
Gy	Vertical Acceleration
Gz	Longitudinal Acceleration

HMOS	High-performance Metal-Oxide Semiconductor
IFPDAS	Inflight Physiological Data Acquisition System
IC	Integrated Circuit
INTA*	Interrupt Acknowledge (active low)
INTR*	Maskable Interrupt (active low)
IO/M*	Type of Machine Reference; high signal implies access to input/output device, low implies memory access.
IR	Inflight Recorder
I/O	Input / Output
KIPS	Thousands of Instructions Per Second
LSTTL	Low-power Schottky Transistor-Transistor Logic
LTTL	Low-power Transistor-Transistor Logic
MBM	Magnetic Bubble Memory
MIDS	Magnetic Bubble Memory Interactive Development System
NMI*	Non-Maskable Interrupt (active low)
PMS	Processor-Memory-Switch
PROM	Programmable Read Only Memory
P2CMOS	Poly-Planar Complementary Metal-Oxide Semiconductor
RAM	Random Access Memory
RDT	Inflight Recorder Debugging Tool
RD*	Read Strobe (active low)
RFSH*	Dynamic Memory Refresh Signal (active low)
ROM	Read Only Memory
RSTx*	Maskable Interrupts, x = A, B, or C (active low)
S0, S1	Microprocessor Machine Cycle Status
SAM	US Air Force School of Aerospace Medicine, Brooks AFB, TX

TTL	Transistor-Transistor Logic
WR*	Write Strobe (active low)
XWAIT*	Processor Wait Request Signal (active low)

Abstract

A prototype for the Inflight Recorder component of the Inflight Physiological Data Acquisition System was built. The Inflight Recorder is a remote data acquisition computer for sampling physiological data. Characteristics of the recorder's design were solid-state, microprocessor controlled, expandability, 16 sensor inputs, and 122 samples per second. Demonstration of battery operation for four hours and unobstructive size characteristics awaits further testing.

Following a hardware requirements analysis, the prototype was built using Complementary Metal Oxide Semiconductor (CMOS) integrated circuits. Components featured in the design were a CMOS microprocessor; Electrically Erasable Programmable Read Only Memories (EEPROM); a monolithic, 16 channel, analog to digital converter; and Magnetic Bubble Memories (MBM).

In addition to building the IR prototype, several development tools were constructed. One was a EEPROM Programmer. Another was an MBM Interactive Development System. A third was a hardware front panel for debugging IR software. User's manuals for these tools appear in appendices to the thesis.

An Inflight Recorder Prototype for the Inflight Physiological Data Acquisition System III

I Introduction

One of the missions of the United States Air Force School of Aerospace Medicine (SAM) is to develop effective life support systems for the crews of high performance aircraft. To accomplish this task, SAM collects environmental and physiological data during actual sorties. Upon mission completion, this data is added to a historical data base and correlated with data from past missions. This data collection and analysis system is known as the Inflight Physiological Data Acquisition System (IFPDAS). Since the IFPDAS is the primary method for collecting inflight environmental and physiological data, it is an important tool for evaluating the effectiveness of Air Force life support systems.

Background

The IFPDAS is composed of three subsystems. These are the Inflight Recorder (IR), the field processing facility, and the laboratory processing facility. The IR is the data collection component of the IFPDAS, while the other two subsystems function as data analyzers (Ref 8). The current IFPDAS is of limited usefulness because of present IR

weaknesses. These weaknesses and the development of a prototype to overcome them, motivates this thesis research.

Current System. From its inception, the IFPDAS has been plagued by an inadequate IR. This was true for the IFPDAS I and is still true for the current model, IFPDAS II. The inadequacy of the IR results from its hardware configuration as a cassette tape recorder interfaced to a signal sampling device. This configuration caused several problems, which were revealed in the IFPDAS I (Ref 16:1-2). Five of the problems were:

1. The cassette drive mechanism stopped during high-G maneuvers, causing discontinuities in data recording.
2. The IR was capable of recording only the following seven signals:
 - a. a time code for correlating samples,
 - b. pilot voice,
 - c. Electrocardiogram (ECG),
 - d. cabin pressure,
 - e. oxygen consumption,
 - f. expired flow, and
 - g. vertical acceleration.Additions and changes to these seven inputs were impossible without a hardware redesign.
3. All data manipulation was done with analog signals. This degraded the samples as noise was introduced during each stage of data manipulation.
4. The IR was constructed with discrete components, making it less reliable than a system based on integrated circuits (IC's).
5. Additionally, discrete components added to system bulk, undesirably restricting pilot movement.

In an effort to correct some of the problems outlined above, The Pacific Missile Test Center, Microelectronics Branch, at Point Mugu Naval Air Station, California,

redesigned the IR. The result, currently being used in the IFPDAS II, is a more capable and reliable IR. Increased capabilities are a result of the addition of environmental and body temperature sensors. However, it should be noted that some design tradeoffs were made to accomodate the additional sensors. The result is that either body temperature and ECG, or environmental parameters can be recorded, exclusive of each other (Ref 8). Reliability is increased in two ways, one being through the increased use of IC's. Another is the result of digitizing some of the recorded data. The sensors that are recorded digitally are the body temperature and ECG. While this new IR is an improvement over the old, all of the five problems listed above still exist.

Previous Studies. Previous studies have shown that a solid-state IR is feasible, using commercially available hardware. The progression from the initial feasibility question through the most recent thesis effort is outlined in the following chronology of US Air Force Institute of Technology (AFIT) studies.

1. Jolda and Wanzek (DEC 77) - showed a solid-state IR is feasible using Magnetic Bubble Memory (MBM);
2. Hill (DEC 78) - investigated storage requirements and techniques for sampling 12 physiological sensors;
3. Moore (JUN 80) - simulated IR operation on a Rockwell 6500 microcomputer with MBM, while analyzing storage requirements; and

4. Svetz (DEC 80) - considered a hardware design for the IR and wrote software for a ground based system to analyze IFPDAS data.

While a one sentence synopsis of each of the above theses is terse, it describes the most important aspect that each contributes to the ongoing search for an improved IR. Based on recommendations made by these studies, the next logical step is to construct a prototype of a new IR.

Problem Statement

This thesis is aimed at replacing the weak link in the IFPDAS by building an IR prototype. Unlike previous models, the new IR will be a solid-state, microprocessor-controlled device. This new design offers the following solutions to the problems listed in the Current System description.

1. Moving parts, which stop during high-G maneuvers, will be replaced by solid-state components.
2. Sampling limitations will be alleviated by two means. One will be that any 0-5V sensor can be plugged into any of the sensor ports. Another will be use of a microprocessor to control sampling methods through characteristically flexible software.
3. All data will be stored and manipulated in its digital form. Therefore, the only error in the sample data will be a consequence of the analog-to-digital (A/D) conversion. Once in digital form, data manipulation will be free from error introduced by analog noise.
4. System reliability will increase because IC's will be used where possible. Use of discrete components will be minimized.
5. While the IR prototype will not directly solve the bulk problem, it will provide a model for estimating the size of a new IR. Though the new IR will be physically larger, its basic structure as a microprocessor with peripheral I/O ports (input - signal sensors; output - secondary

storage) allows the system to be broken down into several smaller devices and distributed to convenient body locations. If distribution is judicious, pilot movement will be less restricted with the new IR.

Scope and Assumptions

For this project, a prototype was considered to be a box capable of overcoming the first four problems mentioned above, as well as providing a model for estimating system bulk. To realize a working prototype, IR hardware was designed and built using commercially available IC's. In addition, software was written to show that IR components were functional.

Because of the limited time for completing this thesis, custom design of the physiological sensors was not done. Instead, the IR was designed to interface directly to any sensor having a full scale output range of 0-5V. This 0-5V assumption was natural since all IR sensors in the IFPDAS II meet this criterion (Ref 8). In addition, the new IR benefited from this assumption because its design was not restricted by a closed set of sensors. Therefore, the new IR can be tailored for a specific application simply by changing the sensors and writing appropriate software drivers.

A second assumption involving the sensors dealt with data accuracy. The assumption was that eight bits of digital data could accurately represent the sampled analog inputs. IR sensor sampling and ground based signal reconstruction were investigated by Jolda and Wanzek. Based on their observations an eight bit word, capable of recording units

from 0 to 255, was sufficient for IFPDAS use (Ref 16:27-41). Further analysis of sample accuracy was not done in this thesis.

It must also be noted that IR construction was not restricted by requirements to interface the new IR to current IFPDAS analysis hardware. This flexibility allowed for an optimum hardware design based only on the requirements outlined in Chapter II. This thesis assumed that appropriate data processing equipment would be procured should current equipment prove inadequate for supporting the new IR.

Approach

Building the IR prototype involved constructing hardware and writing software. Before either hardware or software work began, requirements were identified, an architecture was developed, and IC's were chosen for the new IR. This process matched a set of commercially available IC's to the major components of the IR. Once devices were identified, hardware design and construction began. As construction proceeded, software was developed to show that each new system component was properly interfaced.

Sequence of Presentation

Chapter II is a hardware requirements analysis whose purpose it is to define a set of IC's for building the new IR. Using the chips defined in Chapter II, Chapter III describes the circuit design for constructing the new IR prototype. Then Chapter IV outlines a program that

demonstrates the operation of IR components. Finally, Chapter V lists conclusions and makes recommendations for a flyable IR. In addition, IR prototype support tools developed during the project are described in the appendices.

II Hardware Requirements Analysis

The purpose of this functional analysis is to define a microprocessor architecture that will satisfy the requirements for a new generation IR. This analysis begins by deriving a list of required and desired characteristics to guide construction of the IR prototype. Then, a preliminary hardware architecture for performing physiological data acquisition is described. Finally, commercially available IC's are mapped onto the preliminary architecture, defining the set of solid-state components used in the new IR.

Required Characteristics

The characteristics required for the new IR were derived from a set of requirements identified by Capt Hall and Lt Shackford of SAM (Ref 8). Generally, requirements were derived to solve the problems inherent in the IFPDAS II (see Current System, Chapter I). In addition, prior theses showed that it was feasible to construct a system with characteristics based on the following requirements.

The first requirement is that the IR resist mechanical failures. This requirement is a consequence of the fact that the tape recorder portion of the current IR fails during high-G aircraft maneuvers. This failure results from the tape transport's mechanical nature. As forces on the recorder become excessive, the tape transport stops. A

solution to this problem is to replace the mechanical components with solid-state components.

The second requirement is that the IR be totally man portable. To preclude interference with emergency pilot egress, there can be no physical connections between the IR and the cockpit environment. This interconnecting restriction forces a remote operating capability on the IR. Consequently, the IR must carry its own power supply, resulting in a required characteristic for battery operation.

The third requirement is that the IR be unrestrictive. That is, it must be small enough to be attached to crew members without restricting movement. This requirement results from comments made by pilots who have worn the IR of the IFPDAS I. Their comments indicate that since the IR is located on their chests, its two inch thickness hindered movement. A consensus on the bulkiness of the IR of the IFPDAS II, which is only an inch and a half thick, is not available at the time of this writing. Regardless, the new IR must not obstruct pilot movements.

When considered together, the previous two requirements imply that the IR should be as small as possible. That is, given that it must be totally man portable, a small IR is less restrictive than a large IR. Requirements do not give a set of measurements to bound the IR; instead, the requirement for an unrestrictive property is specified. So, while small size is a by-product of portable and unrestrictive, it will not be categorized as a required characteristic.

The final requirement is that the IR be flexible. This is motivated by the fact that SAM wishes to record data other than the limited set available from the current IR. According to Capt. Hall, changing current sensor inputs and sampling rates is tedious to the point that variations are not made (Ref 8). A more flexible IR must possess two characteristics. One is that it be microprocessor controlled, so that sampling order and rate are easily changed by software reprogramming. Another is that sensor interfacing be simple and direct. For the purposes of this thesis, sensor interfacing is simplified based on the Scope and Assumptions discussion of Chapter I.

In summary, because of the requirements that the new IR be failure resistant, man portable, unrestrictive, and flexible; the IR must possess the following characteristics:

1. solid-state,
2. battery operated,
3. unobstructive, and
4. microprocessor controlled.

Desirable Characteristics

In addition to the required characteristics outlined above, SAM listed several other features that should be incorporated into the IR prototype. These desired characteristics differ from those that are required, in that they are only goals for guiding the prototype design. Failure to meet all desired characteristics is not critical to construction of the new IR.

TABLE I
Sensor Sampling Rates
(Refs 11:9-13; 21:15)

Sensor	Sampling Rate (samples per second)	
	sensor	cluster
Triaxial Acceleration	8	24
Cabin Pressure	2	2
Inspired Flow Rate	20	20
Expired Flow Rate	20	20
Inspired Oxygen Concentration	20	20
Expired Oxygen Concentration	20	20
Body Temperature	2	16
Total		122

Most of the desired characteristics involve sensor inputs. First, the IR prototype should be capable of 16 sensor inputs. In addition, the initial set of sensors should be:

1. triaxial acceleration (Gx, Gy, Gz),
 2. cabin pressure,
 3. inspired flow rate,
 4. expired flow rate,
 5. inspired oxygen concentration,
 6. expired oxygen concentration, and
 7. body temperature (at eight points).
- (Ref 8)

These sixteen sensors dictate a rate of 122 samples per second. Individual rates are broken out in Table I, based on derivations done in the Hill and Moore theses. Noting that future applications call for the use of various other

sensors, another desirable characteristic is that IR components be able to support expanding capabilities. The sensors being considered include additional environmental and several electrocardiogram sensors. Yet another desirable characteristic is for an operating duration of four hours, that being the length of a useful data acquisition mission (Ref 8). In summary, there are four desirable characteristics for the IR prototype:

1. 16 sensors,
2. room-for-expansion,
3. 122 samples per second, and
4. four hour operation.

Preliminary Architecture

Hardware architectures illustrated in this thesis rely on the Processor-Memory-Switch (PMS) technique developed by Gordon Bell and Allen Newell. They developed the PMS technique as a "compact and useful" method for describing digital computers. Basically, PMS diagrams show gross hardware structure by illustrating the capacity of system components, information paths between components, and distribution of control between components. The level of detail at which these PMS attributes are defined depends on particular applications. A more in depth description of the PMS technique is found in Bell and Newell's book - Computer Structures: Readings and Examples (Ref 1:15-36,615-27).

Component designations used in PMS diagrams within this thesis come from standard abbreviations applied by Bell and

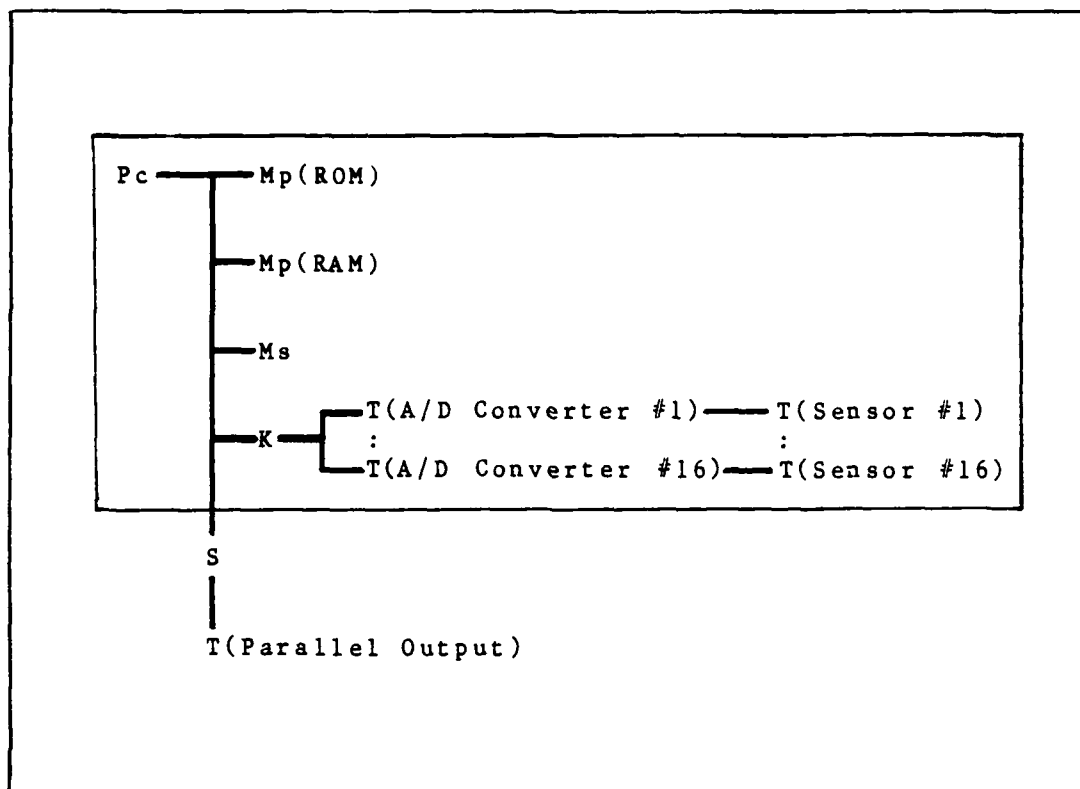


Figure 1. A Preliminary IR PMS.

Newell. They are: Pc - Central processor, Mp - primary memory, Ms - secondary memory, K - controller, T - transducer, and S - switch.

Figure 1 is a PMS diagram showing a general overview of the hardware configuration to be employed in the new IR. The remainder of this requirements analysis is directed at mapping available IC's onto the portions of this configuration that fall within the outlined box. The

remainder of this chapter analyzes the:

1. main processor - Pc,
2. secondary storage - Ms,
3. program memory - Mp(ROM),
4. data acquisition ports - T(A/D Converter), and
5. buffer storage - Mp(RAM).

Other portions of the diagram are not examined in the following analysis, but can easily be built using commercially available components.

IC Technology

Before proceeding with analysis of individual components, an IC technology family must be chosen to implement the design. A proper technology is critical in light of the unobstructive and battery operated characteristics required for the new IR. Since batteries are bulky, space requirements are best minimized by reducing system power requirements. Therefore, the primary criterion for choosing a technology must be to minimize power consumption.

Currently, Complementary Metal-Oxide Semiconductor (CMOS) devices draw far less power than other technologies. Table II illustrates this fact by comparing the power dissipated by Transistor-Transistor Logic (TTL), Low-power Schottky TTL (LSTTL), Low-power TTL (LTTL), and CMOS technologies. The reason CMOS consumes such low power is that only a low level leakage current flows through CMOS gates when they are in a steady state. Larger currents are drawn only while gates are switching from one state to

TABLE II

Comparison of Logic Families (*)

(Refs 3:1-3,1-5; 28:6-3 - 6-5)

	TTL	LSTTL	LTTL	CMOS
Typical Power Dissipation (per gate)	10 mW	2 mW	1 mW	.01 mW (static) 1 mW (1Mhz)
Propagation Delay	9 ns	10 ns	33 ns	50 ns (5V) 30 ns (10V)
Input Voltages				
Low Level (max)	.8 V	.8 V	.7 V	1.5 V (5V) 2.0 V (10V)
High Level (min)	2 V	2 V	2 V	3.5 V (5V) 8.0 V (10V)
Output Voltages				
Low Level (max)	.4 V	.5 V	.4 V	.5 V (5V) 1.0 V (10V)
High Level (min)	2.4 V	2.7 V	2.4 V	4.5 V (5V) 9.0 V (10V)
Noise Margin (guaranteed)	.4 V	.3 V	.3 V	1.0 V (5V) 1.0 V (10V)

(*) NAND gates used as standard for comparison.

another. Consequently, as the frequency of gate switching increases so does power consumption. In practice, however, only a few gates switch at any one time, leaving most gates in a static state. Therefore, at any one time, most gates draw only leakage current resulting in low overall power consumption (Ref 18:585).

Besides its low power characteristic, CMOS displays several other advantages that make it a preferred candidate for the IR. One is that CMOS generates very little heat as a

consequence of its low power operation. This characteristic allows the IR to operate within the life vest of a pilot without bulky ventilation devices.

Another advantage of CMOS over other technologies is its relative immunity to noise. This immunity is known as the noise margin and is defined as the difference between the guaranteed voltage limits of a driving gate and voltage requirements of a driven gate for a particular logic state (Ref 27:40). Applying this definition to the figures of Table II, CMOS tolerates 2.5 times the noise that TTL does. The figures of Table II are guaranteed by the manufacturer to be absolutely safe operating tolerances. But in practice, both TTL and CMOS exhibit higher tolerances. Typically, TTL tolerates 1.5 volts in its logic 1 state, and 1.15 volts in its 0 state (Ref 27:41). Because CMOS changes states at close to half of its supply voltage, it typically tolerates noise at levels up to 45 percent of the supply voltage (Refs 3:6-60; 29:94). With a five volt supply this equates to a 2.25 volt tolerance. So, because CMOS has a higher noise margin than other technologies, the power supply of the IR can be simplified without affecting system operation.

So far, discussion has centered on a +5V power supply for operating the different technologies. As Table II indicates, CMOS can also be biased at higher voltages. Advantages gained by operating CMOS at a higher voltage are reduced propagation delay and increased noise margin. The

magnitude of these advantages is shown in Table II for CMOS gates operated at +10V.

In addition to the advantages offered by CMOS, there is one important disadvantage that must be considered. The disadvantage, as illustrated in Table II, is that current CMOS devices are slower than devices of other technologies. These slower speeds result more from monetary considerations than from theoretical limits. But, as CMOS manufacturing costs decrease relative to more popular technologies, more CMOS devices with improved performance will become commercially available (Refs 6:24; 26:90,94). One recent example of an improved performance CMOS technology is National Semiconductor's poly-planar CMOS (P2CMOS) process (Ref 24:3-1). A NAND gate fabricated with P2CMOS techniques has a propagation delay of only 18 nanoseconds when operated from a five volt supply (Ref 24:A-55). But, even though CMOS technologies are capable of better speed performance, most CMOS chips available today are comparatively slow.

Consideration of CMOS device speeds is important to the IR for two reasons. One is that the main processor of the IR must be fast enough to sample and store data at an acceptable rate. Analysis, completed later in this chapter (see Main Processor subsection), shows that such a processor does exist. Another reason involves the circuits which connect the components within the processor. These "glue" circuits must keep up with the processor so that control signals are not lost. Since a wide variety of fast CMOS devices do not

exist, interconnecting IC's hold potential problems for the IR design. Fortunately, these problems can be reduced by using either of the more popular LTTL or LSTTL technologies. Which technology to use depends on speed requirements, with LTTL being preferred because of its lower power consumption.

Evaluation

The following analysis yields a set of IC's for use as the major circuit components in the new IR. Though components have been analyzed in prior theses, there are two reasons why they must be reevaluated. Most importantly, many new chips have been marketed in the past year. Several of them are directly applicable to the IR. Another reason for reevaluation is that previous analyses were biased by availability of and familiarity with the development tools located in the AFIT Digital Engineering Lab. These biases forced unnecessary restrictions on previous analyses. The only restrictions placed on the following analysis are the required and desired characteristics outlined above. This new evaluation yields a chip set which is better suited for the new IR than those suggested in prior theses.

Main Processor. The central component of the new IR is a CMOS general-purpose microprocessor. The following discussion analyzes three microprocessors: the National Semiconductor NSC800, the Motorola MC146805E2, and the RCA CDP1802. They were chosen because they are the only 8-bit CMOS general-purpose microprocessors on the market today (Refs 20:150-4,162-5; 13:501-45). Each is a solid-state, 40

pin IC, capable of battery operation. These features satisfy all required characteristics, qualifying them for use in the new IR.

Before analyzing desirable characteristics, it should be noted that benchmarking would have been a useful tool for comparing the microprocessors. Programming identical IR data manipulation algorithms, would have provided valuable information for choosing the best applicable microprocessor. However, with the time allotted to order hardware and complete this thesis, such a comparison was not feasible. Still, one microprocessor was judged best for IR applications.

Continuing the evaluation, only two of the desirable characteristics, 122 samples per second and room-for-expansion, apply to the microprocessors. Considering the desire to process 122 samples per second, Table III indicates that even the slowest processor can execute 1279 instructions per sample. While no minimum number of instructions per sample has been projected, 1279 certainly allows for reading and storing a byte of data with some level of processing in between. It follows that, if the slowest processor is capable of processing 122 samples per second, then all of the processors in Table III are acceptable.

Expansion capabilities of the new IR depend upon both instruction cycle time and available address space. Possible areas of expansion include sampling at higher data rates and increasing the complexity of preprocessing algorithms. Both

TABLE III

Comparison of Microprocessors

(Refs 22:2,3,22-23;
24:4-2,4-7 - 4-9;
26:17,19,24,31)

	NSC800	MC6805	RCA1802
Operating Frequency (max at 5V)	5M	5M	2.5M
Clock Cycles per Machine Cycle	2	5	8
Machine Cycle Frequency (max at 5V)	2.5M	1M	.31M
Cycles per Instr (fastest instr)	4	2	2
Instr Cycle Frequency (KIPS)	625	500	156
Instr per 1/122 sec	5120	4100	1279
Address Range	64K	8K	64K

areas are limited by the speed of the host microprocessor. In addition, increased algorithm complexity generally implies the need for additional program storage. Comparing the processor speed and address space characteristics outlined in Table III, the NSC800 is the best performer. The MC6805 is chosen as the second best since its execution speed is faster than the RCA1802 and its address space can be expanded to equal that of the RCA1802 through memory banking.

Memory banking techniques allow software to activate blocks of memory one at a time. The largest block size

available to the MC6805 is 8K bytes. Implementing eight 8K banks, the MC6805 has access to 64K bytes of memory, equaling that of the RCA1802. A summary of expansion capability rankings is provided later in Table IV.

Prior to examining the room-for-expansion criterion, all three microprocessors were judged capable of supporting IR requirements. To this point there has been no compelling reason to choose one microprocessor over the others. Therefore, an additional set of criteria is introduced to aid in the selection process.

The following paragraphs introduce three additional criteria for selecting a microprocessor. First, the data manipulation capabilities of each microprocessor are compared. Next, processor support, including documentation and variety of support chips, is evaluated. Then, the popularity of the microprocessors is discussed relative to software maintenance problems.

Besides sampling and preprocessing physiological data, a primary function of the IR is to block and save the data in secondary storage. While each of the microprocessors has adequate single byte I/O and arithmetic instructions, only the NSC800 has block manipulating instructions. The advantage of block moves is that a single instruction causes transfer of an entire data block. In addition, a single transfer instruction executes somewhat faster than a loop when moving the same amount of data. Consider, for example,

the following loop for outputting a block of data:

LOOP:	LD	A,(HL)	7	**	
	OUT	(PORT),A	11	**	Instruction Time
	INC	HL	6	**	In Machine Cycles
	DJNZ	LOOP	13	**	

			37		- Total Machine Cycles

Using the block output instruction, OTIR, the above loop can be executed in one instruction, requiring only 21 machine cycles per byte. Other block manipulating instructions have similar advantages. (Ref 24:5-5 - 5-32)

The I/O subset of the block move instructions have immediate applications in the IR. Since the primary function of the IR is real time data acquisition, the less time spent saving data, the more time is available for processing samples. Consequently, the block move instructions of the NSC800 make it the most capable data handler for IR applications.

Another important factor, especially during system design, is manufacturer support. This support is critical in two areas, processor documentation and variety of support chips. All three manufacturers have good descriptions of their microprocessors in both manuals and data sheets (Refs 22; 24; 26). However, each manufacturer's family of support chips varies. RCA offers the widest variety, due probably to the fact that their microprocessor has been commercially available considerably longer than the others. The second best variety comes from National Semiconductor, who has adequate variety for simple applications, including remote data acquisition (Refs 24:8-13, 8-15 - 8-17).

One way to reduce software maintenance problems is to choose a popular microprocessor. This is true for two reasons. One is that popular processors have a wider variety of software development tools. This wider variety increases the probability that one will be acceptable for IR applications. Another reason is that increased popularity implies that more people are qualified to program the microprocessor. Consequently, the search for, or the training of, programmers is simplified.

An indicator of microprocessor popularity is the number of commercially available microcomputers which employ them. Electronic Design publishes an annual guide for single board microcomputer comparisons. Counting the numbers of CPU boards based on the three processors in this analysis, the MC6805 did not appear, the NSC800 was the basis for one board, and the RCA1802 was the basis for three. However, since microprocessor popularity is being judged as it relates to software, and since the NSC800 executes the Z80 instruction set (Ref 24:A-3), all references to the Z80 can be applied to the NSC800 count. The Z80 is used in 36 products (Ref 19:88-95).

Before saying that the NSC800 has the most popular software instruction set and the MC6805 has the least popular, one other factor must be considered. Many microprocessors have instruction sets that are extensions of ancestors within the microprocessor family tree. Therefore, upgrading to a related processor is a simple task when

TABLE IV
Microprocessor Criteria Ratings (*)

Criteria	<u>NSC800</u>	<u>MC6805</u>	<u>RCA1802</u>
Required/Desired Characteristics	1	1	1
Expansion Capability	1	2	3
Data Manipulation	1	2	2
Support Chips	2	3	1
Popularity	1	2	3

(*) Acceptable = 3, Better = 2, Best = 1

compared to learning a new language. Ancestors of the NSC800/Z80 include the 8080 and the 8085. Those for the MC6805 are the 6800 series processors and the 6502. The RCA1802 has no popular ancestor (Refs 5:44-52; 30:175-84). Referring to ancestors in the microcomputer guide, the NSC800 has the most popular software structure, followed by the MC6805, with the RCA1802 being a distant third (Ref 19:88-95).

Table IV summarizes the microprocessor comparisons made in the previous discussion. For each criterion the processors are ranked, with 1 being the best rating. The 1 rankings for the required/desired characteristics indicate only that all microprocessors fulfilled the criteria. Expansion capability is ranked separately because it was the only desired criterion where one processor was judged more

capable than the others. The remaining criteria in the table are those added to the selection process.

In conclusion, the NSC800 is chosen as the main processor for the new IR. Referring to Table IV, the NSC800 has the best performance characteristics for all but one of the evaluation criteria. Neither of the other two processors performs as well as the NSC800, based on the criteria developed above.

Secondary Storage. The need for a large secondary memory, on the order of one megabyte, was projected by the Hill and Moore theses. A one megabyte memory is the single largest component in the IR. Therefore, it is potentially the biggest power user and space consumer in the system.

Three possible solid-state memories that are available for mass storage are MBM, RAM, and Charge Coupled Devices (CCD). The densest possible examples of each are compared in the following analysis. According to the 1981 IC Master catalog, the densest commercially available devices are the Intel 7110 MBM, the Fairchild F264 CCD, and the Harris HM6564 Static RAM.

Table V shows some characteristics of the three devices that are targetted for use as secondary storage. The number of chips indicates only what is required to store one megabyte of data. The number does not include chips required for processor interfacing. Since both the CCD's and RAM's are 64 kilobit devices, they require 128 chips to store one megabyte (Refs 7:1; 10:3-94). The bubble memory and its five

TABLE V
Comparison of Secondary Storage Devices

	MBM	CCD	RAM
# Chips	48	128	128
Area (sq in)	48	46	268
Device Active Time (percent)	.13	.06	.06
Power			
Standby	8.36W	8.32W	.03W
Operating	11.27W	8.35W	.06W
Non-volatile	yes	no	no

support IC's can store one megabit. To store one megabyte, an MBM device requires $6 * 8 = 48$ chips (Ref 2:1-1).

The area consumed by each one megabyte store is estimated by adding the areas taken up by each chip. Included in the estimate is two-tenths of an inch space between each chip. To simplify calculations, this space is divided among the IC's by including a one-tenth of an inch border around each chip. The following calculations derive the area values of Table V:

CCD chip size - .3 x .8
 Single chip area
 with .1 border - .4 x .9 = .36
 Total area required - .36 * 128 = 46.08 (Ref 7:8)

RAM chip size - .9 x 2.0
 Single chip area
 with .1 border - 1.0 * 2.1 = 2.1
 Total area required - 2.1 * 128 = 268.8 (Ref 10:8-9)

MBM chip sizes - 725X	-	.3 x .7	
7242	-	.3 x 1.0	
7230	-	.6 x 1.2	
7110	-	1.7 x 2.0	
Single chip areas			
with .1 border	-	.4 * .8	= .32
		.4 * 1.1	= .44
		.7 * 1.3	= .91
		1.8 * 2.1	= 3.78
Total area required	-		
8 * ((3 * .32) + .44 + .91 + 3.78)		= <u>48.3</u>	
(Ref 2:2-4; 17:11).			

All three devices chosen for secondary memory have standby power ratings. Standby power indicates the amount of power drawn by an IC which is not actively operating. Typically this power is less than that dissipated when the chip is being accessed. So, the amount of time that an IC is operating has a direct affect on the total power dissipated by that IC.

The device active time within Table V indicates the percentage of time that a memory device will be operating during a data acquisition mission. Three assumptions were made to obtain these percentages. First, the need to store data at a four kilobytes per minute rate was established. This rate uses 960K bytes, or 94 percent of the one megabyte total, during a four hour mission. A second assumption was that only one unit within a secondary device operates at one time. That is, at any time only one IC will be operating within the RAM and CCD devices, and only one module (six IC's) will operate in the MBM device. The last assumption was that secondary memory operates at its maximum rated speed, consistent with low power consumption. Note, however,

that this maximum speed must be tempered by the speed of the main processor.

A useable upper limit on secondary memory speed is the maximum output transfer rate of the main processor. The fastest way to output a data block through the NSC800 is with the OTIR instruction. With a 2.5 MHz clock frequency, the OTIR instruction takes 8.4 microseconds to transfer one byte (Ref 24:5-27). This speed translates to a maximum output transfer rate of 120,000, or approximately 117K bytes per second.

Two of the three secondary storage devices operate faster than 120 KHz. The fastest device is RAM, which can be interfaced directly to the main processor without wait states (Refs 10:3-100; 24:4-16). The second fastest device is CCD. It consumes minimum power when operated at its minimum frequency of one megahertz (Ref 7:6). Noting that CCD's are serial devices, an operating frequency of one megahertz translates to a transfer rate of $1,000,000 / 8 = 125,000$, or approximately 122K, bytes per second. With a speed matching buffer between the NSC800 and CCD, the CCD peripheral operates at the 120 KHz maximum of the NSC800. The NSC800 does not affect the MBM transfer speed. The BPK-72 Bubble Memory Prototype Kit User's Manual sets the maximum MBM rate at 50K bytes per second in its minimum power consuming configuration (Ref 2:2-7). In summary, the data transfer rates used for Table V calculations are: RAM - 117K, CCD - 117K, and MBM - 50K bytes per second.

Calculations for device active times are based on the transfer rates listed above. The method for determining active time percentages is to divide the 4K bytes per minute storage requirement by the transfer rate of each device. In general, device active time = $4 / (\text{rate} * 60) * 100$, when expressed as a percentage. Substituting the storage rate defined in the preceeding paragraph yields the values in Table V.

The next to the last set of entries in Table V are power ratings. They were obtained by adding up the power consumed by each IC within a storage device. Individual power ratings were taken from "typical" values reported by the manufacturer. Differences in reporting IC characteristics led to the employment of three different methods for determining the power ratings. Intel explicitly listed typical power consumption figures for each device in the MBM module. Fairchild provided a graph showing typical power dependence on the operating clock pulse width. Finally, Harris listed typical current and voltage characteristics to which the power formula, voltage times current, was applied.

The standby power values of Table V are calculated with all IC's in an inactive state. Calculations yield

$$\text{RAM} = 128 * .25\text{mW} = 32\text{mW} = .032\text{W (Ref 10:3-97),}$$

$$\text{CCD} = 128 * 65\text{mW} = 8.32\text{W (Ref 7:6), and}$$

$$\begin{aligned} \text{MBM} = & \quad 8 * .29\text{W} = 2.32\text{W} \\ & + 8 * .225\text{W} = 1.80\text{W} \\ & + 8 * .5\text{W} = 4.00\text{W} \\ & + 8 * .03\text{W} = .24\text{W} \\ & + 16 * 0.0\text{W} = \underline{0.00\text{W}} \\ & = 8.36\text{W (Ref 2:2-8).} \end{aligned}$$

Operating power values depend on the conditions set above for transfer rates and numbers of parallel operating IC's. For RAM storage only one IC operates. Since RAM operates at the 120 KHz transfer rate of the NSC800, the HM6564 has a typical current drain of $120\text{KHz} * (40\text{mA} / 1\text{MHz})$, or 4.8mA. At five volts, 4.8mA translates to a power draw of 24 mW, so,

$$\text{RAM} = (127 * .25\text{mW}) + 24\text{mW} = .056\text{W} \text{ (Ref 10:3-97)}.$$

Operating at one megahertz an F265 draws 90mW, leading to

$$\text{CCD} = (127 * 65\text{mW}) + 90\text{mW} = 8.345\text{W} \text{ (Ref 7:6)}.$$

Finally, multiplexing MBM modules one at a time results in six active IC's, and a power draw of

$$\begin{aligned} \text{MBM} = & (7 * .29\text{W}) + 1.48\text{W} & = 3.51\text{W} \\ & + (7 * .225\text{W}) + .225\text{W} & = 1.97\text{W} \\ & + (7 * .5\text{W}) + .5 & = 4.00\text{W} \\ & + (7 * .03\text{W}) + .48 & = .69\text{W} \\ & + (14 * 0\text{W}) + (2 * .55\text{W}) & = \underline{1.10\text{W}} \\ = & & 11.27\text{W} \\ & & \text{(Ref 2:2-8)}. \end{aligned}$$

At this point an observation relative to power ratings is useful. The observation is that secondary storage is not used frequently enough for the active power ratings to be a useful comparison parameter. The largest and longest operating power consumer, MBM, illustrates this point. The MBM operates at 8.36W, 99.87 percent of the time, and at 11.27W the rest of the time. So, overall, the MBM dissipates

$$(99.87 * 8.36\text{W}) + (0.13 * 11.27\text{W}) = 8.364\text{W}.$$

This calculation shows that active power contributes on the order of only a few milli-watts to overall power consumption. Likewise, the other active power ratings of Table V do not

significantly affect overall power consumption. Consequently, the active power values of Table V are ignored in the following analysis.

CCD, MBM, and RAM were chosen for analysis as secondary storage devices because they satisfied both solid-state and microprocessor controlled requirements. In addition, all three can be operated on batteries. The lithium battery pack proposed by Hill (Ref 11:84) is capable of powering both the MBM and RAM devices. With a slight modification to include a -5V supply the Hill pack could also power the CCD's.

Comparing the three storage devices relative to the unobstructive requirement, forces RAM's to be dropped from further consideration. As seen in Table V, RAM's consume 5.6 times the area of either MBM's or CCD's. Allowing for reduced power requirements and using Hill's proposed power pack, the area factor is still $RAM = 4.4 * MBM$. This is deduced from the extreme assumption that the 3 x 2 x 5 inch power pack can be eliminated to allow space for four 3 x 5 inch RAM boards. Since a RAM storage device would increase the size of the IR much more than the other devices, it will not be used as secondary IR memory.

The only significant difference between the MBM and CCD devices as they relate to IR applications is their volatility. This difference is important when considered with the battery operated requirement and the power consumption characteristics of the devices. To minimize battery requirements a maximum operating time must be

designed into the IR. This time must be longer for the volatile CCD's, since the IR must stay powered up until the collected data can be dumped to a more permanent device. On the other hand, MBM storage saves the data even after power is removed. Consequently, the IR power supply can be designed to operate only during the data acquisition task. Because of its non-volatile nature, MBM is chosen over the CCD as IR secondary memory.

In conclusion, the MBM system is chosen as secondary storage for the new IR. RAM is eliminated because it is much bulkier than MBM. CCD is not used because it is volatile and requires a power source even after a data acquisition task is complete.

Program Memory. From the beginning of the requirements definition it has been assumed that program memory should be non-volatile. Basically, there are two reasons for having a non-volatile program memory. One is system flexibility. This offers the advantages of being able to program the IR long before a mission and to use the same program for several missions without reprogramming after battery changes. Another reason is that non-volatility increases IR reliability. Volatile memories are susceptible to change during a mission, destroying program execution. In addition, a non-volatile set of chips need only be programmed once for a particularly popular mission and they can be used for years without reprogramming. Every time a memory is reprogrammed, there is potential for introduction of errors. So, given

that program memory will be non-volatile, the types to be considered in this analysis are: ROM, PROM, EPROM, and EEPROM.

ROM and PROM are not suitable for IR program storage. They are rejected because they can be programmed only once and changes can not be made. This permanence feature is undesirable since the capability for program changes is inherent in the room-for-expansion criterion. Since non-volatile memories exist that can be reprogrammed, there is no need for further consideration of ROM's and PROM's.

Two common types of non-volatile, reprogrammable memories are EPROM and EEPROM. Basically they differ in the way they are erased. EPROM's generally require UV light, while EEPROM's are erased electrically. Both types come in CMOS, with 1K byte EEPROM's having the largest currently available capacity; although, industry rumors are that National Semiconductor will soon market the 27C16, a 2K byte EPROM (Ref 29:96).

Due mainly to the larger capacity of the EEPROM, it will be used in the new IR. The EEPROM of choice is the Hughes Aircraft HNVM3008, since it is the only 1K byte EEPROM that is commercially available.

Data Acquisition Ports. An analog interface to the physiological sensors was previously analyzed in the Moore thesis. Based on four requirements:

1. 12 channel input, minimum,
2. 144 conversions per second, minimum,
3. conversion error of less than 1 percent, and
4. low power consumption,

Moore chose the National Semiconductor ADC0817CC, a 16 channel monolithic A/D converter (Ref 21:10).

Since Moore's thesis was published, user requirements have changed slightly. New requirements are for 16 input channels and a 122 samples per second conversion rate. Both of these desired characteristics are supported by the ADC0817. So, for the reasons initially chosen by Moore, and as tempered by new requirements, the ADC0817 is used for the analog data acquisition ports of the new IR.

Buffer Memory. Buffer memory is used primarily as an area where physiological data is collected and preprocessed before being transferred to secondary memory. This activity implies the need for a read/write memory, or RAM. In the following analysis, the three IR characteristics of - battery operated, unobtrusive size, and 16 sensors - drive RAM selection.

As derived earlier, the battery operated and unobstructive characteristics imply the need to minimize power consumption. A table of RAM characteristics, provided for Hitachi memories, indicates that CMOS static RAM's have low power consumption relative to other types of RAM (Ref 14:4). Even dynamic RAM, which characteristically draws less

power than static RAM's (Ref 9:123), draws more power than Hitachi's CMOS RAM. Using RAM from another manufacturer for comparison verifies this relationship. As an example, the Intel 2117 dynamic RAM consumes 462 mW when operating and 20 mW in standby (Ref 4:1-26). By comparison, the Hitachi HM6116LP draws 300 mW operating and .5 mW standby (Ref 14:72). So, to minimize power consumption, buffer memory should be CMOS static RAM.

The third characteristic, the need for 16 sensors, provides a basis for estimating RAM size. Using projections made by Moore, there should be enough RAM to store 25 blocks of data. This allows buffer space for one and a half times the 16 data channels, leaving one block for program scratchpad memory. The eight additional data blocks are used to start new buffers once old ones are full and awaiting output to secondary storage (Ref 14:20-2).

The choice of MBM as secondary storage sets the buffer block size to 64 bytes. In its minimum power consuming configuration with error correction enabled, MBM transfers data in 512 bit (64 byte) blocks (Ref 2:2-8,3-6). Combined with the need for 25 data blocks, buffer memory should be a minimum of $25 * 64 = 1600$ bytes, or effectively 2K bytes.

Several manufacturers offer 2K byte, CMOS, static RAM's. Of them, Hitachi offers the most flexible line of chips (Ref 13:2813). That is, they offer a wide range of power drains and access times (Ref 14:66-75). The lowest power consuming model offered is the HM6116LP and is chosen for IR RAM.

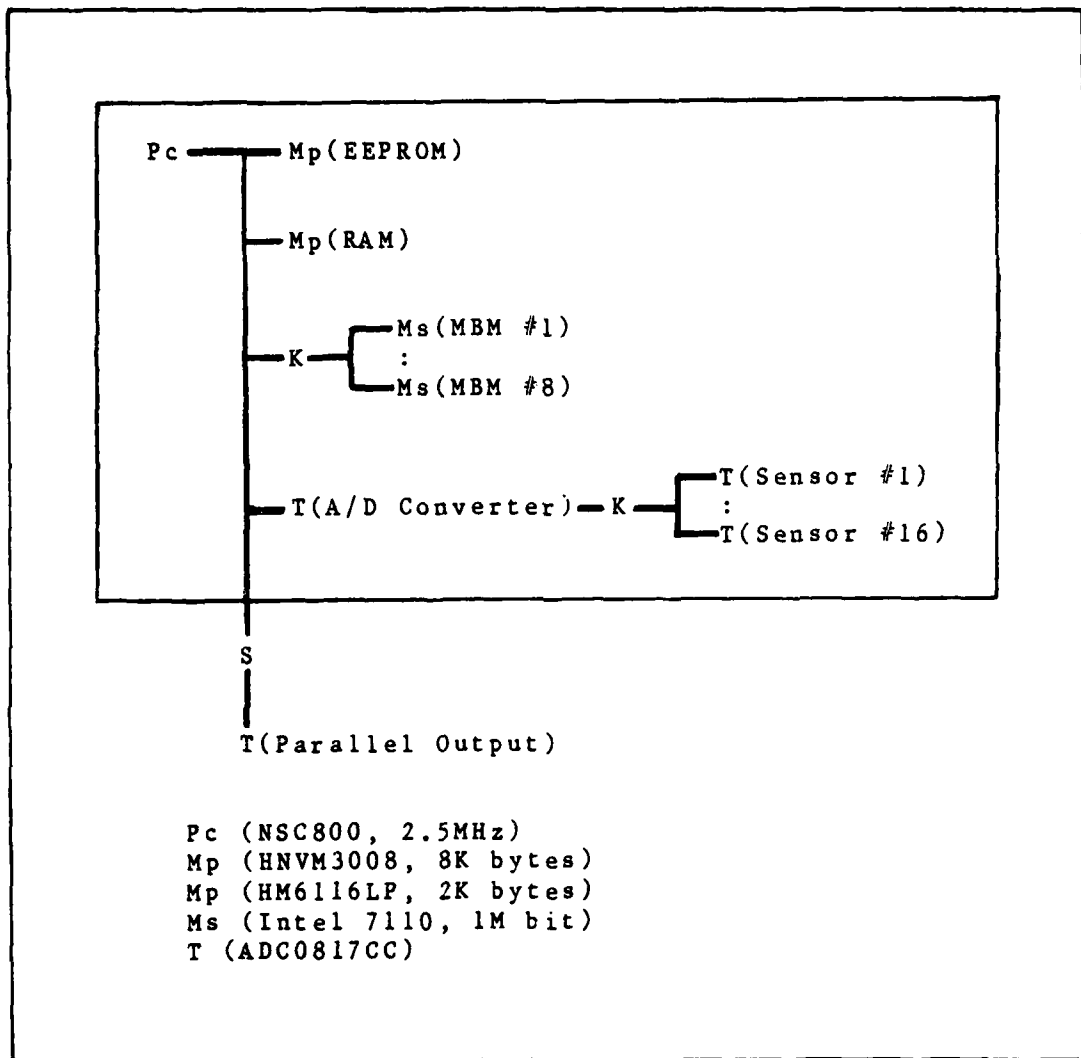


Figure 2. The Proposed New IR.

Conclusions

Component choices made in the above analyses contribute to a more definitive IR architecture than that outlined in Figure 1. Figure 2 is a PMS diagram showing the chip level components for the new IR. Having defined components for the

new IR, the next step is to build a prototype to prove the proposed design meets user requirements. The next chapter describes the IC level hardware architecture of the new IR prototype.

III Prototype Construction

The proposed new IR, diagrammed in Figure 2, is an architecture for satisfying the required and desired characteristics outlined in Chapter II. The prototype IR described in this chapter provides a tool for determining how well the proposed architecture functions. Tests run with the prototype provide information for tailoring the proposed IR before final circuit boards are produced.

The IR prototype implements the architecture of Figure 2 with two minor but important changes. One is that RAM is increased to 8K bytes. This allows space for investigating effects of buffering data in blocks larger than 64 bytes, the minimum required by the MBM. The other change is that only one MBM module is implemented. The original purchase of Intel Bubble Prototyping Kits (BPK-72) allows two separate 128K byte storage units to be built. To obtain a 256K byte memory, either an Intel iSBC 254 board must be purchased, or an interface must be designed to combine the BPK-72 boards. Because of time constraints neither option was pursued.

The schematic diagrams used for discussing the components of the IR in this chapter are the same ones used to build the IR prototype. While the diagrams are scattered throughout this chapter, they are combined into one five page figure in Appendix A.

Operating Voltage

The prototype IR requires +5V, +12V, and -12V for proper operation. The +12V supply is required by both the MBM and A/D Converter peripherals. The -12V supply is used only by the A/D Converter peripherals (Refs 2:2-8; 10). Other parts of the system use +5V.

The main processor and its support IC's are fabricated with P2CMOS technology. While P2CMOS chips operate over a voltage range of +3V to +12V (Ref 24:4-6), they are biased at +5V in the IR. There are three reasons for choosing a +5V power supply. One is that at +5V the NSC800 operates at 2.5 MHz, as analyzed in Chapter II. Another reason for choosing a +5V power supply is to keep power consumption low, since power dissipated by a CMOS gate is directly proportional to its operating voltage (Refs 3:6-5; 9:32-33; 18:585). The third reason involves interfacing the various chips within the IR. Both the Bubble Memory Controller (BMC) and the EEPROM must operate at +5V (Refs 2:2-8; 12). While voltage translation circuitry could be used between components of the IR, interfacing is simplified and chip count is reduced if all components operate at a single voltage.

IC Technology Mix

All IR circuitry, except for the MBM, is constructed with CMOS derivative IC'S. The MBM peripheral was built using customized circuits provided by the manufacturer. Intel provides a BMC for interfacing the MBM to various

TABLE VI

IC Family Voltage Characteristics
(Refs 24:A-4; 15:12; 3:1-185)

Voltage Type	P2CMOS	CMOS	HMOS
Logical 0 Input (max)	1.5	1.5	.8
Logical 1 Input (min)	3.5	3.5	2.0
Logical 0 Output (max)	.4	.4	.4
Logical 1 Output (min)	4.5	4.6	2.4

CPU's. The technology used to fabricate the BMC is Intel's High-performance MOS (HMOS). Connecting HMOS to CMOS is straight forward but requires some precautions.

HMOS is a NMOS derivative technology (Ref 9:30). Its direct current voltage characteristics are summarized in Table VI along with characteristics for P2CMOS and CMOS. The devices used to obtain the ratings in Table VI are the NSC800, the ADC0817, and the Intel 7220. As seen in Table VI, the three technologies are directly compatible at the logic 0 level. That is, the voltage level output by any one technology is below the input threshold of the others. However, potential problems arise at the logic 1 level. While both CMOS technologies have output voltages above the input threshold of HMOS, the opposite is not true. An HMOS output of 2.4V is not guaranteed to be recognized as a high input to either P2CMOS or CMOS. So, in cases where HMOS provides input by either P2CMOS or CMOS, a pull up resistor

with a value in the neighborhood of 10K ohms should be included in the circuit (Ref 3:6-8; 8:42). Pull up resistors used in the IR are 10K ohms.

Board Layout

The IR prototype exists on a wirewrap card. Figure 3 is a map that shows the major component parts of the IR as they appear on the card. To aide in tracing system bugs, the wiring used in the IR is color coded. The code used is:

Red	- +5V,
Black	- Ground,
Blue	- Data Bus,
Yellow	- Address (7 - 0) Bus
Orange	- Address (15 - 8) Bus
Green	- Control Bus
White	- Other

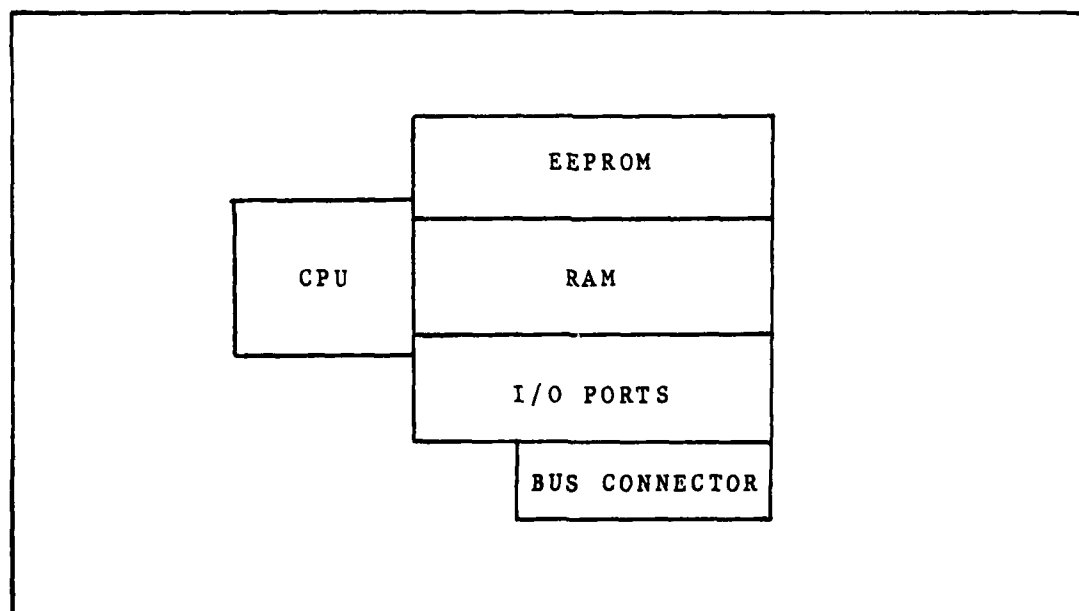


Figure 3. Major Component Map of IR Prototype.

Bus Structure

The three system busses of the IR carry data, address, and control information. These three busses intersect at the CPU. To keep them operating at their full rated speed, connection of devices to the busses must be scrutinized.

Since CMOS has a high input impedance, it draws only leakage current while it is in a steady state. Consequently, fan-out for CMOS to CMOS interfaces is typically 50 devices (Ref 18:67-8). However, circuit capacitance puts a practical limit on the number of devices that a CMOS gate can drive. CMOS gate inputs add capacitive loads to circuits. As the number of inputs increase, so does the time that it takes to charge the additional capacitance. The result is increased propagation delays between the output and input of gates (Ref 3:6-4 - 6-7,6-17).

The drive capacity for the NSC800 family of chips is rated at 100 pico-farads (Ref 24). Since input capacitance specifications are not available for many of the IC's used in the IR, each input is assumed to add a 15 pico-farad load to the circuit (Ref 3:6-17). This assumption limits the fan-out of CMOS gates used in the IR to six, leaving a small margin for error. In all but one case, fan-out is less than or equal to six within the IR. The connection of eight EEPROM's to system bus buffers violates the limit. However, that portion of the IR works consistently and is a logical block to which no hardware additions are anticipated.

TABLE VII
IR Bus Connector Definition

Pin Number	Signal Function	Pin Number	Signal Function
1	+5V	50	Open
2	GND	49	S0
3	+12V	48	CLK
4	Data 0	47	PowerSave*
5	Data 1	46	Data 4
6	Data 2	45	Data 5
7	Data 3	44	Data 6
8	WR*	43	Data 7
9	Addr 0	42	kD*
10	Addr 1	41	Addr 4
11	Addr 2	40	Addr 5
12	Addr 3	39	Addr 6
13	XWAIT*	38	Addr 7
14	Addr 8	37	IO/M*
15	Addr 9	36	Addr 12
16	Addr 10	35	Addr 13
17	Addr 11	34	Addr 14
18	ResetIn*	33	Addr 15
19	BREQ*	32	ResetOut
20	NMI*	31	BACK*
21	RSTB*	30	RSTA*
22	INTR*	29	RSTC*
23	INTA*	28	Open
24	S1	27	GND
25	-12V	26	+5V

A 50 pin connector is provided with the IR prototype so that external devices can easily be added. Table VII defines the pin-out for the connector. Restricting the bus to 50 pins resulted from the availability of connectors during late phases of the project. While more control lines could be defined for a more general bus, the 50 pin bus is sufficient

for most applications, including the Recorder Debugging Tool (see Appendix D) interface.

The reason for providing a connection to the IR bus structure is so that development and debugging circuits can easily be added. When adding such circuits, a designer must be aware that the connector is not buffered and every gate interfaced through the connector loads the CPU. Therefore, it is recommended that, at a minimum, every pin used as an input by an external device be buffered by a CMOS gate. This presents only a single CMOS load to the CPU. Because of the conservative loading design of the IR, adding single CMOS loads should not effect IR operation.

CPU

The CPU for the IR prototype consists of the NSC800 and its clock, reset, wait state, and bus demultiplexing circuitry. Figure 4 is a schematic showing how these circuits are integrated to form the CPU. This schematic is referenced throughout the discussion of CPU components.

System Clock. Operating at +5V, the microprocessor used in the prototype has a maximum rated speed of 2.5 MHz. This operating frequency is controlled by an external timing circuit which must be twice as fast. The NSC800 contains an on-chip oscillator which divides the external timing signals to produce a square wave clock signal. This clock is the basis for machine cycle timing within the NSC800. The circuit used to produce the 5MHz external clock is one suggested in the NSC800 Microprocessor Family Handbook. It

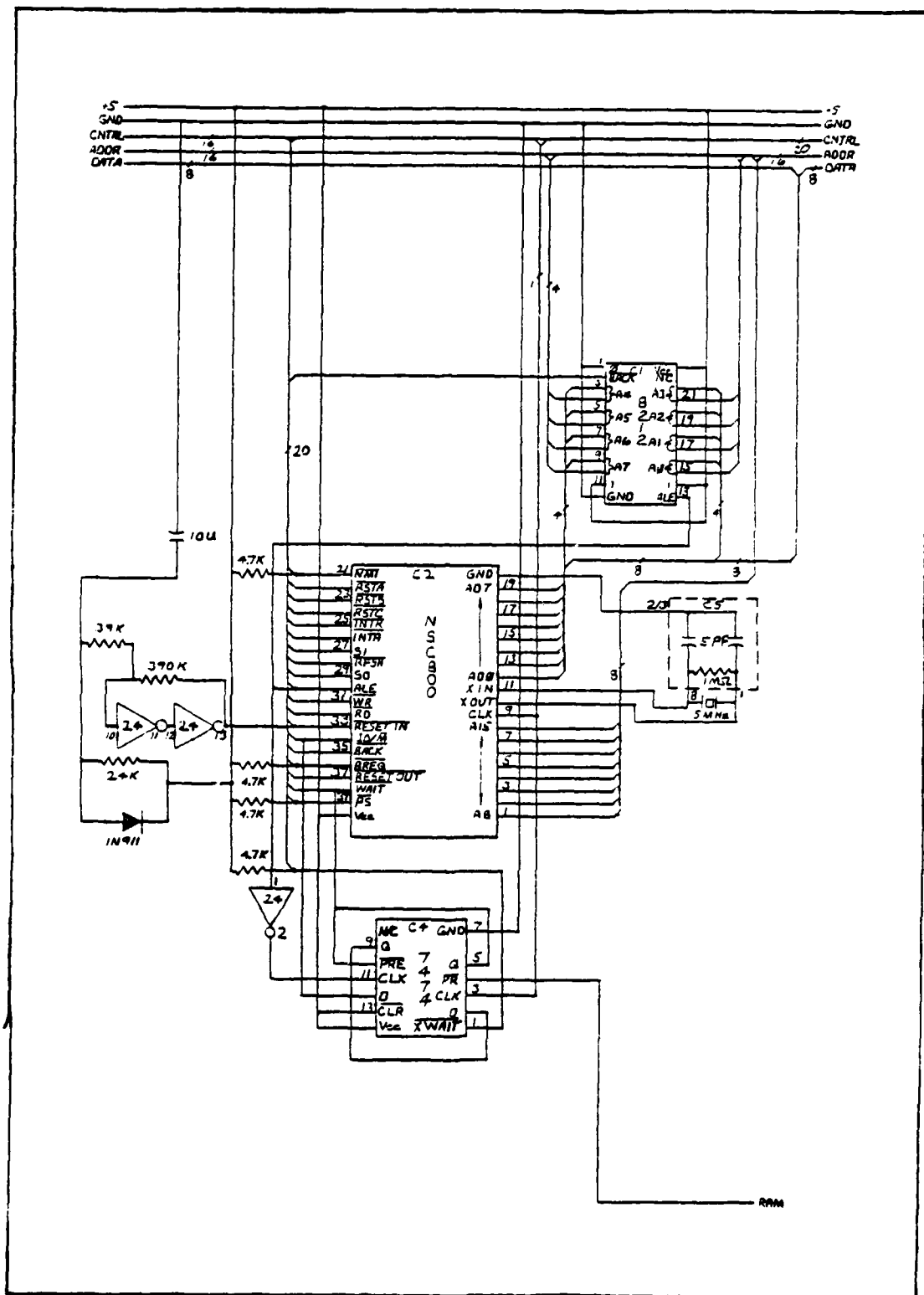


Figure 4. CPU

consists of a 5 MHz crystal, a one mega-ohm resistor and two 4.7 pico-farad capacitors (Ref 24:4-8 - 4-9, A-10).

System Reset. Another circuit mentioned in the NSC800 Handbook is one to provide orderly power-up for the system. Since the NSC800 has an on-chip Schmitt trigger, the manufacturer claims that a simple Resistor-Capacitor network, connected to RESET-IN*, provides a proper power-up reset function. Following manufacturer directions, repeated experimentation with various combinations of resistors and capacitors could not produce a clean power-up sequence.

As the IR powered up, the NSC800 reset many times. These multiple resets were observed by monitoring portions of the data, address, and control busses with a logic analyzer. Two phenomena indicated that the NSC800 had an unstable period while it was resetting. One was that RESET-OUT toggled randomly as system power approached +5V. The other was that instruction execution began at location 0000H each time RESET-OUT went low, with various numbers of machine cycles being completed before RESET-OUT returned high. Once RESET-OUT stabilized at 0V, the processor operated predictably.

To correct the reset problem a Schmitt trigger was added to the reset circuit between the Resistor-Capacitor network and the RESET-IN* pin of the NSC800. The Schmitt trigger circuit consists of two inverters and a feedback network. The circuit used in the IR was adapted from one described in

Douglas Hall's book (Ref 9:35-6). After the external Schmitt trigger was added, the NSC800 reset properly.

Wait State Generator. Another component of the CPU is the wait state generator. Wait states must be added to memory read cycles whenever program memory is accessed. Access time for the HNVM 3008's is slow enough to require one extra machine cycle for transferring data to the NSC800. In addition to generating wait states for EEPROM accesses, the wait state generator must also insure that machine cycles are not added when other memory, or any peripheral, is addressed. The last function supported by the wait state generator is to gate wait state requests from external devices, such as the Recorder Debugging Tool, to the NSC800.

An inverter, a pull-up resistor, and the two data flip-flops (FF) of IC number C4 form the wait state generator. Basically their function is to hold the WAIT* pin of the NSC800 high until a wait state is required. Two conditions are sufficient requirements for generating wait states. One is when the XWAIT* pin of the IR bus is pulled low by an external device. When this happens, the FF controlling WAIT* is cleared and multiple wait states are generated until XWAIT* returns high. The other condition for generating wait states is when a EEPROM address is accessed. During these times only one wait state is necessary. To add one wait state, the WAIT* pin of the NSC800 must be pulled low for one machine cycle following the latching (ALE = 1) of a EEPROM address.

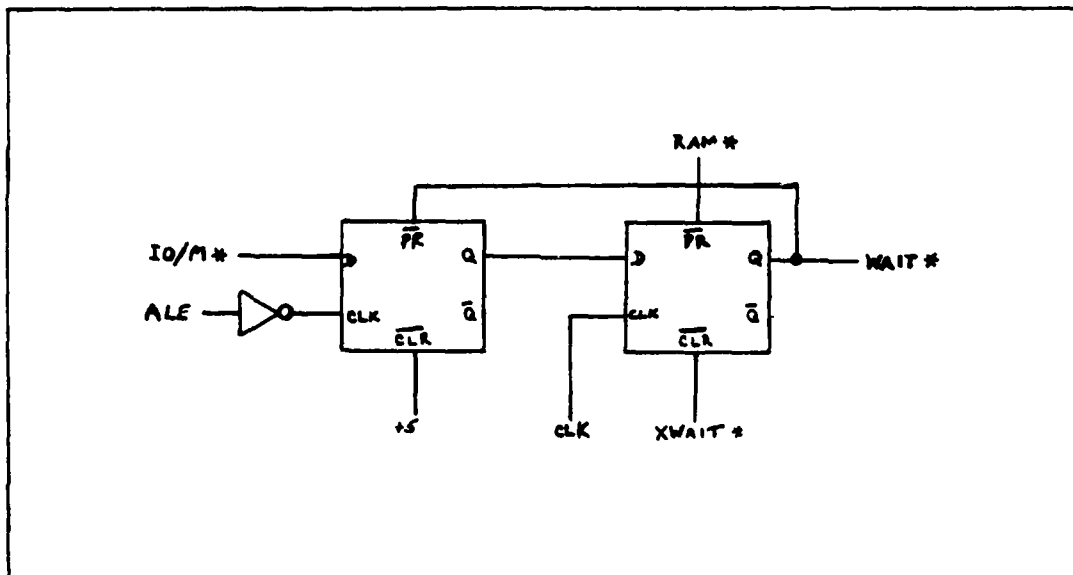


Figure 5. Wait State Generator

Generation of a single wait state is explained using Figure 5, a more explicit drawing than that of Figure 4. The wait state generator works by passing the current value at the "D" input of the left FF to the "Q" output of the right one. Consequently, when a peripheral is addressed, $WAIT^* = IO/M^* = 1$ and no wait states are generated. When memory is addressed, a zero passes from the left to the right FF. But, if the memory access is to RAM, the right FF is preset by RAM^* before $WAIT^* = 0$ is recognized by the NSC800; and again no wait states are generated. If EEPROM is addressed, $RAM^* = 1$, $WAIT^* = 0$, and the NSC800 adds an extra machine cycle to its memory access operation.

To keep from adding more than one wait state, $WAIT^*$ has one clock cycle in which to reset. This cycle is between the falling edges of the system clock one cycle before the added

wait state and during the wait state (Ref 24:8-8). To complete this reset within the allotted time, the left FF is immediately preset whenever WAIT* becomes zero. The resulting high later passes to the right FF on the rising edge of the next NSC800 clock pulse. Under this scenario WAIT* is held low long enough to add only one machine cycle to EEPROM access operations.

Bus Demultiplexer. The low order byte of the address bus is multiplexed with the data bus in the NSC800. The two are separated by a widely used circuit employing an 82PC12, eight bit I/O Port (Refs 4:6-56; 24:8-4 - 8-6). During the first machine cycle of a memory or peripheral access, an address appears on the multiplexed bus. Before the cycle completes, an ALE pulse causes the address to latch into the 82PC12. Following this latching sequence, the multiplexed bus is dedicated to use as the data bus.

One other function performed by the 82PC12 is to place the low order portion of the address bus in a tri-state mode whenever BACK* = 0. This feature is intended for systems which employ direct memory accessing. While the IR prototype does not currently employ direct memory accessing internally, the capability is used by an external device - the Recorder Debugging Tool. Connecting BACK* to the 82PC12 is transparent to normal IR operation and could be removed. But, the connection is important for debugging purposes, so it remains.

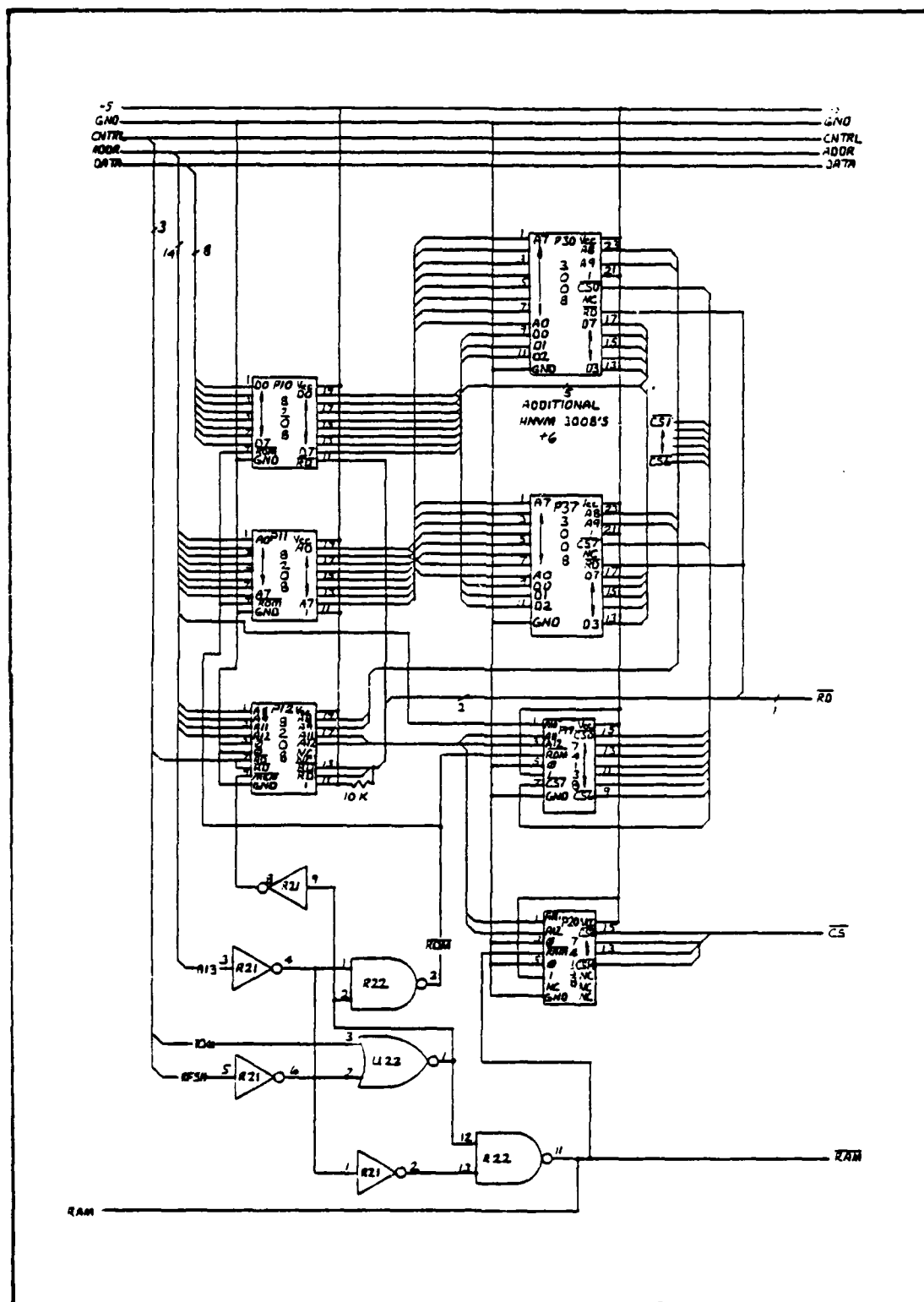


Figure 6. EEPROM.

Primary Memory

Primary memory is split between EEPROM and RAM. The 8K EEPROM space is used as program storage and occupies the address space from 0000H to 1FFFH. RAM is provided for use as buffer storage between the A/D converter and the MBM peripheral. It occupies addresses 2000H through 3FFFH.

The two types of primary memory are physically separate components within the IR prototype. With the exception of one RD* line, signals from each component are interfaced to the CPU through their own set of buffers. The RD* line is shared between EEPROM and RAM only to reduce fan-out of the line from the CPU. EEPROM requires nine RD* connections. With the fan-out limit of six, EEPROM needs two buffered RD* lines. Since EEPROM does not use all 12 of the available loads, one is connected to RAM. This connection deletes the requirement for RAM to load the CPU's heavily used RD* line.

In addition to their physical separation, EEPROM and RAM are logically separated by their addresses. That is, address bit 13 (A13) determines which primary memory component is enabled. When A13 is zero, EEPROM is accessed; and when it is one, RAM is accessed. While A13 determines which primary memory component is enabled, the IO/M* and RFSH* control signals determine when they are enabled. All three signals are combined by the logic in the lower left corner of Figure 6 to provide proper enable pulses. Basically, the logic will output an active low memory enable signal whenever the CPU wishes to access memory (IO/M* = 0) during times other than

refresh cycles (RFSH* = 1). The inverse of this memory enable allows A13 and its complement to pass through NAND gates and choose the primary memory component to be accessed.

IR hardware does not contain logic to protect software from attempts to address memory locations above 3FFFH. While the NSC800 is capable of addressing 64K bytes of memory, EEPROM and RAM occupy only the low order 16K bytes. So, any IR memory location can be addressed using only 14 bits. Address decoding logic ignores the remaining two address bits, truncating A14 and A15 from addresses greater than 3FFFH. It is the software designer's responsibility to insure that programs limit their accesses to the available 0000H to 3FFFH address space.

Program Memory. With the exception of the logic gates described above and IC number R20, Figure 6 shows the EEPROM component of the primary memory. P10, P11, and P12 are buffers; P19 is an address decoder; and P30 through P37 are EEPROM's.

P10 and P11 are only enabled when a EEPROM address is accessed. Since one of the RD* control lines is shared with RAM, P12 is enabled whenever either memory component is accessed. Direction control on P11 and P12 is hardwired to pass information from the CPU to memory. RD* supplies direction control for the data bus buffer, P10. Even though the current IR design does not support writing to the EEPROM's, direction control for the data bus buffer can not

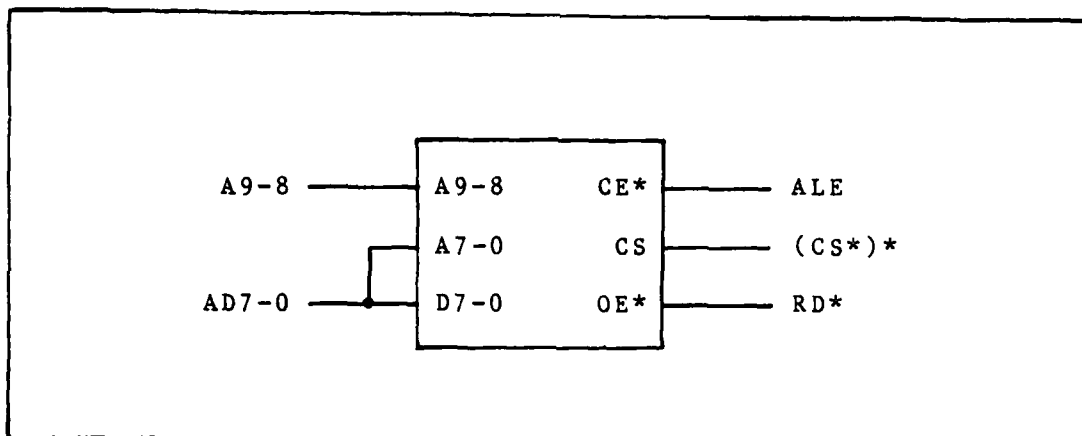


FIGURE 7. Conventional HNVM3008 Interface

be hardwired. Doing so causes bus contention problems with the CPU.

Bus contention stems from the multiplexed nature of the data bus. The EEPROM data bus buffer is enabled whenever $A_{13} = 0$, $IO/M^* = 0$ and $RFSH^* = 1$. These conditions are true at the beginning of each instruction cycle which accesses EEPROM. However, during the first part of the cycle the multiplexed bus contains a valid address. Hardwiring the direction control would cause interference during this portion of an instruction cycle. So, EEPROM is only granted control of the data bus while RD^* is low.

HNVM3008's are used for storage in the EEPROM portion of primary memory. The way they are interfaced to the CPU is unconventional by manufacturer standards. The manufacturer's pin-out descriptions of the HNVM3008 leads to the design shown in Figure 7.

A more optimum design is used in the IR. A comparison of Figures 6 and 7 show the differences between the two. One difference is that the IR design does not use the HNVM3008's on-chip address latch to demultiplex the address/data bus. Instead, the separated data and address busses, provided by the CPU, are used. This reduces the number of loads on the address/data bus by one half. Consequently, the demultiplexed address bus is used to replace one of the address/data bus loads on the CPU. This trade off is desirable since the fan-out from the CPU is greater for the address/data bus than for the demultiplexed address bus.

Another benefit of the customized interface for the HNVM3008's is that chip select (CS) pulses from the address decoding logic do not have to be inverted. This reduces chip count in the IR by at least one, and possibly two. Another, although minor, benefit of the IR configuration is that one less control signal is required. That is, the ALE signal is not used by program memory.

Buffer Memory. The buffer memory component of primary memory consists of the circuit diagrammed in Figure 8, along with IC number R20 of Figure 6. In Figure 8, R13 through R15 provide full buffering of lines connected to the CPU. Memory itself consists of four 2K byte static RAM's. Consequently, four CS* signals and an 11 bit address are sufficient for addressing any byte within buffer memory. The four CS* signals are generated by a three-to-eight line decoder, since smaller decoders are not available in P2CMOS. IR production

designs may find it beneficial to change to a two-to-four line decoder fabricated in another CMOS technology.

Peripheral Devices

IR peripheral devices are shown in the schematics of Figure 9 and 10. Accessing of peripherals is done using I/O mapped addressing. During I/O operations, the address of the selected peripheral appears on both the low and high order bytes of the address bus (Ref 24:A-9). This duplication of the peripheral address allows use of the high order byte for selecting specific I/O ports, and minimizes loading of the heavily used address/data bus.

With the exception of BBH, peripheral addresses are broken down into a 3 bit channel address and a five bit port address. The channel address is essentially an encoded chip select for enabling one of the three chips that contain addressable ports and registers. The NSC810 is the peripheral chip with the most addressable entities, requiring five bits to access all of them. Hence, five bits are used to address any port on a specified channel. I/O address BBH is internally reserved by the NSC800 as an interrupt control register (Ref 24:A-17).

Bits A15, A14, and A13 carry the I/O channel address. U25 decodes these address bits into chip selects. The timers and I/O ports are attached to channel 1 - A15,A14,A13 = 001. Channel 2, 010, contains the A/D Converter and channel 4, 100, contains the MBM. These channel addresses are listed along with their associated port addresses in Table VIII.

Table VIII

I/O Port Mapping

(Refs 2:3-1 - 3-3;
3:1-189; 24:A-32)

Binary Address	Type Port	Function
0010 0000	R/W	Port A
0010 0001	R/W	Port B
0010 0010	R/W	Port C
0010 0100	W	Port A Data Direction Reg.
0010 0101	W	Port B Data Direction Reg.
0010 0110	W	Port C Data Direction Reg.
0010 0111	W	Port A Mode Definition Reg.
0010 1000	W	Port A - Bit Clear
0010 1001	W	Port B - Bit Clear
0010 1010	W	Port C - Bit Clear
0010 1100	W	Port A - Bit Set
0010 1101	W	Port B - Bit Set
0010 1110	W	Port C - Bit Set
0011 0000	R/W	Timer 0 (LSB)
0011 0001	R/W	Timer 0 (MSB)
0011 0010	R/W	Timer 1 (LSB)
0011 0011	R/W	Timer 1 (MSB)
0011 0100	W	Stop Timer 0
0011 0101	W	Start Timer 0
0011 0110	W	Stop Timer 1
0011 0111	W	Start Timer 1
0011 1000	R/W	Timer 0 Mode
0011 1001	R/W	Timer 1 Mode
010X 0000	R	A/D Converter Port 0
010X .	R	.
010X .	R	A/D Converter Port .
010X .	R	.
010X 1111	R	A/D Converter Port 15
100X XXX0	R/W	MBM Data Port
100X XXX1	R	MBM Status Register
100X XXX1	W	MBM Command Register
1011 1011	W	Interrupt Control Register

X's appear in the table where address bits are ignored for a particular channel. A/D converter port assignments correspond to the low order four bits of the peripheral address and are compressed in the table. Note that port 100X XXX1 has two different definitions, depending on whether data is being read or written.

Timers. The IR prototype uses both of the timers contained on the NSC810. Timer 0 is wired for generation of fixed interval interrupts, while Timer 1 provides a clock for the A/D converter.

The CPU provides a clock input frequency of 2.5 MHz to both timers. This input frequency exceeds the maximum for proper timer operation. Therefore, clock inputs must be prescaled. The NSC810 allows for independent scaling of clock inputs to both counters. Scaling factors for Timer 0 are 1, 2, and 64. Those for Timer 1 are 1 and 2. Since maximum timer input frequency is 2 MHz, a scaling factor of at least two allows for proper operation of the timers (Ref 24:A-27,A-36). Chapter IV describes how software controls operation of the timers.

Output from Timer 0 is connected to the RSTA* interrupt pin of the CPU. To provide interrupts that meet hardware design objectives (see Interrupt Structure), Timer 0 must be programmed as an accumulative timer. In this mode, output from the timer is activated at fixed intervals. The length of these intervals range from 800 nanoseconds to 1.7 seconds. This range supports the design requirement for an 8.3

millisecond sample interval. It also allows a wide time range for changing sensor sampling rates.

Timer 1 is used in its square wave generator mode to provide the clock input for the A/D converter peripheral. Valid input frequencies for the ADC0817 range from 10 KHz to 1200 KHz, with manufacturer specifications being computed at 640 KHz (Ref 3:1-186 - 1-187). In its square wave mode, with a prescaled clock input of two, Timer 1 can provide a range of output frequencies from 1250 KHz down to 19 Hz. Note, however, that the distribution of frequencies is not uniform throughout the range. Valid frequencies are clustered more heavily towards the low end. The sequence of valid Timer 1 output frequencies follow the pattern:

1250K / 1	=	1250	KHz,
1250K / 2	=	625	KHz,
1250K / 3	=	416.5	KHz,
		...	
1250K / 65,536	=	19	Hz.

General I/O. In addition to the two timers described above, the NSC810 provides two general purpose I/O ports for the IR prototype. Each port is eight bits long and can be addressed at the bit level. In addition, the direction of data flow, in or out, is selectable for each bit. Therefore, one port can carry both input and output at the same time. Another feature of the NSC810 is that Port A is capable of strobed I/O. This allows handshaking between the IR and an external CPU for such functions as dumping data from the IR to a database, or for programming EEPROM's without removing them from the IR. (Ref 24:A-31 - A-33)

The general purpose I/O ports have not been hardwired to take advantage of any particular capability of the NSC810. The ports were wirewrapped only far enough to verify that they communicate properly with the CPU. Configuring the ports is best handled in parallel with software development.

A/D Converter. The fact that the NSC810 has an 8085 hardware architecture simplified interfacing of the A/D converter. National Semiconductor's CMOS Databook contains a schematic for interfacing the ADC0817 to an 8085 microprocessor (Ref 3:1-193). Construction of the A/D Converter peripheral followed National Semiconductor's proposal. Still, clarification of a few of the connections is appropriate.

The ADC0817 uses the low order four bits of the peripheral address to select the sensor channel to be converted. To minimize bus loading on the multiplexed data/address bus, the channel select is obtained from bits 8 through 11 of the address bus.

Two factors determine the voltages to be used as references in the A/D Converter. One is the bias voltage of the ADC0817, and the other is the output voltage range of the analog sensors (Ref 3:1-191). Since the bias voltages are ground and +5V, and since the analog sensors are conditioned for 0V to 5V outputs, reference voltages for the ADC0817 are 0V and +5V. The low reference is obtained by a direct connection to ground. Capitalizing on the fact that output from a CMOS gate comes very close to the bias voltage of the

chip, the high voltage is obtained from the output of an inverter.

Using the output of an inverter proved adequate for showing that the ADC0817 worked properly. An oscilloscope trace of the inverter output showed a constant 5V signal being coupled with 0.1V of noise. Assuming that the $0.1 / 256 = .4$ millivolt error introduced by the noisy reference is acceptable, use of the inverter as a positive reference is adequate for the IR prototype. However, using inverter references in a flyable IR is risky, as it depends on at least two variables. One is that output from an inverter gate is not guaranteed to equal the chip supply voltage. Another is that the supply voltage in a flyable IR may degrade with prolonged use of the batteries, resulting in a decreased reference voltage. If allowances are not made for these two variables in the flyable IR, then the voltage reference circuit must be redesigned.

The end of conversion signal generated by the ADC0817 provides a conversion complete interrupt to the CPU. A peculiarity exists in this structure. That is, the end of conversion signal remains active until another conversion is started. So, the conversion complete interrupt can not be reset between sampling tasks without additional hardware. Using a data FF to buffer the interrupt, the end of conversion signal could pass to the CPU and be reset whenever the converted data was read. While this method of controlling the interrupt is simple, it requires an

additional chip. In keeping with the minimized bulk requirement of the IR, a hardware solution is abandoned in favor of software. Chapter IV discusses the software solution to the conversion complete interrupt problem.

MBM. The schematic for the MBM peripheral appears in Figure 10. Interface of the peripheral is simplified by the fact that all data transfers take place through the BMC. The hardware architecture of the BMC for interfacing microprocessors looks similar to that of many peripherals. That is, interfacing the BMC to a processor requires connection of the data bus, address bus, read and write strobes, chip select, and system reset. Additional pins are provided for interrupt and direct memory access processing.

As mentioned previously, the BMC is an HMOS IC. Consequently, two precautions are taken to insure accurate communications with the P2CMOS CPU. One is that 10K ohm pull up resistors are used at connections where HMOS provides input to P2CMOS. The other is that P2CMOS outputs are loaded with only one HMOS input.

The MBM is wired to take advantage of the interrupt processing capability of the BMC. Active high signals for buffer half full and operation complete are fed through buffers to the RSTA* and INTR* pins of the NSC800. The need for these interrupts is explained in the next section of this chapter.

In addition to the signals mentioned above for interfacing the BMC to the CPU, the MBM requires a 4 MHz

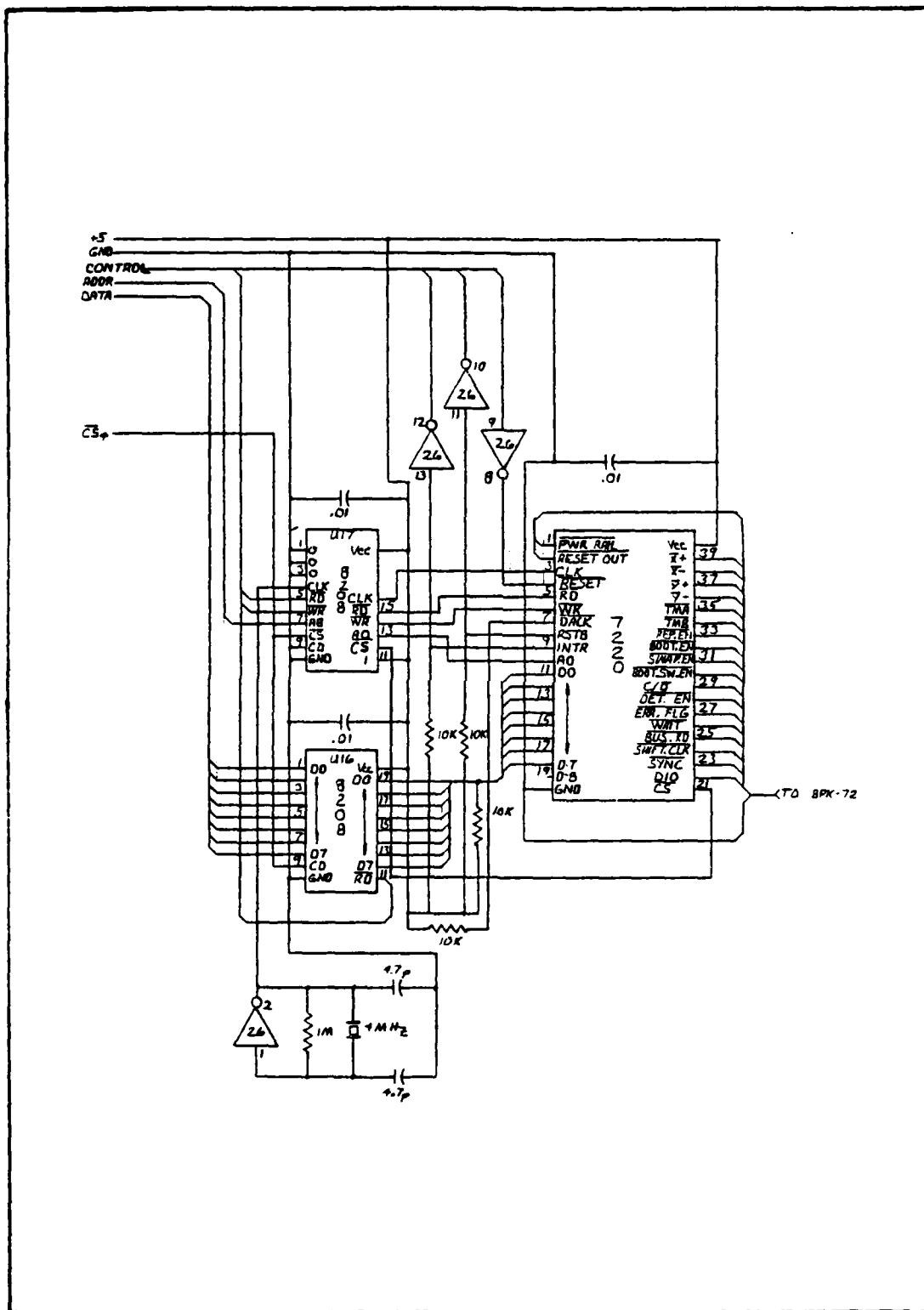


Figure 10. MBM

clock having a 50 percent duty cycle. The circuit appearing below U16 in Figure 10 is a crystal controlled oscillator for providing the required clock. The circuit is an adaptation of the one used for the NSC800 clock input. U17 provides buffering to produce a constant load on oscillator output. The oscillator with buffering provides a stable clock.

The MBM and drive circuitry is not shown in Figure 10. Instead, only the connections that must be made from the BMC to the MBM board are shown. The MBM is mounted on a BPK-72 printed circuit board, which has previously been tested using the MBM Interactive Development System. Design of the MBM peripheral for the IR prototype involved removing the BMC from the BPK-72 and placing it with the other components of the IR. A cable connects the MBM to the BMC for completing communications within the MBM peripheral. Additional connections, not shown in Figure 10, carry power to the BPK-72.

Interrupt Structure

IR prototype design implements a hierarchy of interrupts. Basically, there are two reasons why the IR needs an interrupt capability. One is that it allows data samples to be started at fixed, known intervals. Another reason is that interrupts allow software tasks to run concurrently. That is, several tasks can be initiated before any one completes. Interrupt usage is clarified in following paragraphs where the rationale for specific interrupts are explained.

TABLE IX

IR Interrupt Structure

Interrupt Priority	CPU Signal	Interrupt Function
1	RSTA*	Fixed Interval Generator
2	RSTB*	MBM FIFO Half Full
3	RSTC*	A/D Conversion Complete
4	INTR*	MBM Operation Complete or MBM Error

Five levels of interrupts are provided by the prioritized interrupt request pins of the NSC800. Of the five, only RSTA*, RSTB*, RSTC*, and INTR* are used. To reduce hardware requirements, the interrupt structure relies on the NSC800's Mode 1 processing scheme. In Mode 1 the response to a recognized interrupt is a jump to one of the NSC800's dedicated restart addresses. Other interrupt processing modes require external hardware to generate a restart sequence (Ref 24:4-16 - 4-21,A-15 - A-17). Table XI outlines the interrupt structure used in building the IR prototype.

The interrupt with the highest priority is the one generated by Timer 0 of the NSC810, RAM-I/O-Timer chip. It has the highest priority so that sampling intervals can be precisely defined. As soon as Timer 0 interrupts, software

starts the A/D conversion of the next required sensor. Should other interrupts be allowed to preempt the timer, sampling intervals would have unpredictable lengths. Consequently, the collected data would have an unknown skew from sample to sample.

The interrupt with the second highest priority is the one indicating that the MBM FIFO buffer is only half full. Once an MBM operation has started, "the user must keep up" with the FIFO data buffer (Ref 2:3-8,3-17). "Keep up" means avoiding FIFO underflow during writes, and overflow during reads. Underflow and overflow problems stem from the fact that the FIFO is only 40 bytes long, whereas, the shortest MBM transfer is 64 bytes. In a system where only one bubble is operating, as projected for the IR in Chapter II, the maximum transfer rate is 50K bytes per second (Ref 2:3-5). This translates to one byte every 20 microseconds. During an MBM write which begins by filling the FIFO, the half full interrupt activates whenever 22 bytes are empty (Ref 2:3-8). This allows approximately 360 microseconds (18×20) before an underflow occurs. Similarly, during a read operation the half full interrupt indicates that 22 bytes are available for input (Ref 2:3-8), allowing 360 microseconds before a FIFO overflow. In either case there is a time margin available for servicing MBM FIFO half full interrupts.

Priority level 3 interrupts are less time critical than the interrupts of higher priority. With projections derived from Chapter II, the IR prototype has approximately 8.1

milliseconds in which to service A/D conversion complete interrupts. At 122 samples per second, there are 8.3 milliseconds between the starts of samples. Allowing for the typical conversion time of 100 microseconds (Ref 3:1-187), there are $8.2 - .1 = 8.1$ milliseconds between the time that an interrupt occurs and the time that the next sample must be initiated.

The interrupt with the lowest priority is the one indicating that an MBM operation has either completed normally or with an error. Both interrupts originate from the same MBM pin. BMC status tells which event caused the interrupt. During normal operation, servicing of these interrupts is not critical.

Conclusion

This chapter has described the theoretical and practical considerations involved in constructing the IR prototype. Details of hardware construction for each component were highlighted. The next chapter details software techniques for driving this newly constructed IR prototype.

IV Hardware Verification Program

This chapter describes software used to verify the design and construction of the IR prototype. The program used to exercise the IR prototype is called IRTST. It is located at the end of the chapter, in Figure 11. Throughout this chapter, software descriptions are made with reference to IRTST.

Verification of design and construction involves exercising at least one capability of each component in the system. While IRTST is not a comprehensive test of every capability, it does show that the system components are interfaced properly. In addition, it provides a basis for understanding how the components operate. Reference material is available in Appendix E for expanding this basis and for tailoring the components to meet future prototype software requirements.

In general the flow of execution through IRTST is:

1. initialize the components,
2. fill a buffer with information obtained alternately from an input port and the A/D Converter,
3. dump the filled buffer to the BMC FIFO,
4. read the BMC FIFO, and
5. compare the input and output of the BMC FIFO.

The operations of step 2 are accomplished under interrupt control. Every time a byte of information is moved to the buffer, it is displayed on an output port and the system is

halted for about a second. Timer interrupts restart the system from its halted state.

Throughout program execution, values for indicating program status are written to an output port. A monitor on the output port reveals the following sequence:

1. FF - system reset,
2. A1 - this value was hardwired on the input port for the test,
3. XX - byte obtained from analog sensor #7,
4. ... subsequence 2 and 3 are repeated 40 times (the size of the BMC FIFO),
5. 55 - constant output for 3 seconds to indicate that the BMC FIFO has been written and is about to be read,
6. D0 - successful completion, or
FF - FIFO write and read do not match.

Monitoring this sequence helps to verify that the program is executing properly and that IR components are functioning as expected.

During construction of the IR prototype, programs were written to assist in debugging hardware as it was added to the system. The fact that all of these test programs, including IRTST, executed correctly shows that both the CPU and program memory function properly. The software provided by IRTST verifies operation of the other components.

Buffer Memory

The sequence of indicators outlined above shows that RAM functions properly. The main reason for this conclusion is that the code, which produces the outputs in steps 2 and 3, relies on subroutine calls and interrupt servicing. Both of these tasks use a program stack to temporarily store return addresses. If RAM were not working, invalid addresses would

be retrieved from the stack, resulting in unpredictable program behavior.

Another factor for concluding that RAM functions properly involves buffering of data. At address 0164H, an output buffer is dumped to the BMC FIFO. Later, at 0178H, the FIFO is read into a separate input buffer. Then the output and input buffers are compared. The fact that IRTST ends with a D0 status reinforces the belief that RAM operates properly.

Timers

The NSC810 is equipped with two general purpose timers, each having six software selectable modes of operation. Both timers are used in the IR prototype. Output from Timer 1 is the master clock input for the A/D Converter. Timer 0 provides a fixed interval interrupt for the CPU.

Before either timer is used it must be initialized. For Timer 0 this involves writing a control byte to the Timer Mode Register. For Timer 1, it involves setting the direction of data flow for pins 1, 2, and 5 of the NSC810 in addition to setting the Timer Mode Register (Ref 24:6-7 - 6-12, A-34 - A-38). Code appearing between addresses 0113H and 012DH shows how the timers were initialized for testing the IR prototype.

Timer 1. The five instructions used to configure and start Timer 1 are all that are needed to provide a clock for the A/D Converter. The first two instructions, at 011FH,

configure the timer as a square wave generator. The next two instructions initialize the generator's output frequency, while the instruction at 0129H starts the generator. The output frequency provided by IRTST is as close as possible to the typical operating frequency of the ADC0817. With the input frequency to the timer being divided in half by the mode setting, and with a timer count value of one, Timer 1 output is 625 KHz. During testing this output was verified with an oscilloscope.

Timer 0. To provide fixed interval interrupts to the CPU, Timer 0 is configured as an Event Counter. The event counter works by generating an active output whenever a user loaded count reaches zero. Timer output is deactivated by reading the count value. (Ref 24:6-8)

The six instructions starting at address 0113H, initialize Timer 0 in two important ways. One is that they produce an active output every 0.95 ($= 2.5 \text{ MHz} / 64 / 40960$) seconds. The other way is that timer output is active when low. The polarity of Timer 0 output is important since it is connected directly to the RSTA* pin of the NSC800. This connection also forces any IR programs that enable RSTA* to include interrupt servicing routines.

Once an RSTA* interrupt is recognized, the NSC800 jumps to location 003CH for its next instruction. At that point IRTST software contains a jump instruction to the Timer 0 interrupt servicing routine, T0\$HNDL. Since the timer interrupt is only being used to awaken the CPU from a halt

state, TOSHNDL needs only to deactivate Timer 0 output and reenable NSC800 interrupts. Upon exiting TOSHNDL, control returns to address 019FH, followed by a return to program execution.

The statement - that control passes to 019FH after Timer 0 interrupt processing - is made with confidence. The interrupt frequency is intentionally low to simplify the verification process. All tasks within the IRTST program take much less than 0.95 seconds to execute. Therefore, the CPU is always in a halt state at 019EH before Timer 0 interrupts. Interrupt frequency will be much higher in prototype software, possibly causing return addresses to be unpredictable.

General I/O

As mentioned in Chapter III, wiring of I/O ports was deferred until prototype software requirements are defined. At this time it is impossible to predict the mix of I/O pins required for a flyable IR. Therefore, verifying general I/O operation is restricted to showing that both input and output are available through the NSC810.

While the NSC810 provides 22 pins for general purpose I/O, only 16 are available within the IR. The other six are used for Timer 1 and strobed I/O. The 16 available pins are split between Port A and Port B. However, the bits of each group are individually addressable in any combination of input and output (Ref 24:6-3 - 6-4, A-31 - A-33). This

flexibility is another reason why design of system I/O was postponed.

IRTST does not test every capability of the NSC810 I/O ports. Instead, Port A is initialized for strobed output, and Port B is initialized for input. To verify operation of the output port, a one byte monitor is connected to Port A during testing. An AlH is hardwired to Port B, insuring that input values are known constants.

As with the timers, the I/O ports of the NSC810 must be initialized before they are used. Important tasks during initialization are to set the direction of data flow through each pin of the two ports and to set the type of I/O to be performed by Port A. Type of I/O does not have to be set for Port B since it is capable of only basic parallel I/O. However, Port A has an additional capability for strobed I/O. When strobed I/O is enabled, an additional task of initializing the data direction for the strobe control pins must be done. The instructions between address 0103H and 0111H perform the initialization tasks outlined in this paragraph.

A/D Converter

Obtaining data samples from the A/D Converter can be as easy as reading and writing an I/O port. To begin the conversion process, a program selects the desired channel via an output instruction to the proper address. The single instruction at 0147H is an example. The data byte output is irrelevant to the conversion process. At some later time,

when the conversion is complete, the program reads the sample value from the A/D Converter. The instruction at 0301H illustrates reading the sample. However, in a more general case, the input address does not have to match the output address. Since the A/D Converter only has one register in which to hold sampling results, any address read will retrieve the value of the last sample started.

While obtaining sample data is straightforward, coordinating the A/D Converter's interrupts is more challenging. As alluded to in Chapter III, the IR prototype does not contain hardware for clearing conversion complete interrupts. From the time one sampling task is complete to the time that another is started, the conversion complete interrupt remains active. Because the interrupt can not be reset, it must be managed differently from interrupts such as RSTA* which can.

The conversion complete interrupt is assigned to the RSTC* pin of the CPU. The method for managing RSTC* is to keep it disabled within the NSC800 until a sample is requested. This management takes place in three different locations within IRTST. First, during system initialization RSTC* is disabled. This is accomplished by writing a zero to the RSTC* bit within the Interrupt Enable Register of the NSC800 (Ref 24:A-17). The module at address 012FH shows how the RSTC* interrupt is disabled while other interrupts are enabled. A second place where RSTC* is managed is at 0149H. There the conversion complete interrupt is enabled just after

a sample conversion is requested. The conversion complete interrupt servicing routine, ADC\$HNDL, is the last place where RSTC* is managed. Again, RSTC* interrupts are disabled while others are enabled, exactly as was done during system initialization.

In addition to enabling the RSTC* interrupt at location 0149H, RSTA* was also enabled. This action is a consequence of the fact that bits within the Interrupt Enable Register can not be individually addressed. Still, RSTA* is enabled with confidence, knowing that it is always enabled except when it is being serviced. A more complex algorithm for enabling interrupts may be required for the increased interrupt activity in the flyable IR.

BMC

All requests for MBM I/O pass through the BMC. Because the MBM peripheral does not work, a first step in tracing the malfunction is to verify communications between the CPU and the BMC. One simple test for determining proper communications is to write and read a test pattern using the FIFO registers within the BMC.

The BMC contains many registers, but only a single address line. Therefore, a channel command word must be written to the BMC telling which register is to be accessed (Ref 2:3-1 - 3-3). The two instructions at 0160H illustrate how the BMC is initialized for accessing the FIFO. Once the BMC points to the FIFO, it is available to the system as a

general purpose FIFO (Ref 2:3-8). The instructions at 0164H show how a data buffer is dumped to the FIFO using an NSC800 block I/O command. Similarly, the FIFO is read at 0174H.

Conclusion

The program illustrated and described in this chapter verifies operation of the major hardware components in the IR prototype. While the MBM is not fully operational, IRTST verifies that proper communications exists between the CPU and the BMC. All other components operate as expected for the set of capabilities exercised by IRTST.

```

0000'      .Z80
           ASEG
;TITLE:  IR TEST - SYSTEM TEST FOR IR PROTOTYPE
;AUTHOR:  CAPT R E MEISNER
;DATE:  4 MAR 82
;SYSTEM:  IFPDAS IR
;OPERATION:  THIS PROGRAM DEMONSTRATES OPERATION OF THE FOLLOWING
;            COMPONENTS OF THE IR PROTOTYPE:
;
;            COMPONENT          S/W EXERCISE
;
;            CPU                - PROGRAM EXECUTION
;            PROGRAM MEMORY     - PROGRAM STORAGE
;            RAM                 - STACK, DATA BUFFER
;            TIMERS              - FIXED INTERVAL INTERRUPT
;            INPUT               - READ PORT B
;            OUTPUT              - WRITE TO PORT A
;            A/D CONVERTER       - SAMPLE SENSOR #7
;            MBM CONTROLLER      - READ & WRITE FIFO
;
;            TRANSFER OF DATA BETWEEN THE BUBBLE MEMORY CONTROLLER
;            (BMC) AND THE MBM IS NOT EXERCISED BY THIS PROGRAM
;
;***** CONSTANTS *****
00BB      IER      EQU      0BBH      ;I/O PORT FOR INTERRUPT ENABLE REG
000A      IERVAL   EQU      0AH       ;ENABLE RSTA AND RSTC INTERRUPTS
0003      STBOUT   EQU      03H       ;STROBED OUTPUT MODE TO ACTIVE BUS
0000      DDIN     EQU      00H       ;INPUT DEFINITION FOR DDR
00FF      DDOUT    EQU      0FFH      ;OUTPUT DEFINITION FOR DDR
0023      DDCTRL   EQU      23H       ;DIRECTION DEF FOR PORT C CONTROL
0010      TOMODE   EQU      19H       ;MODE FOR TIMER 0 - EVENT COUNTER,
;RD/WR ONE BYTE, PRESCALER = 64
001D      TIMODE   EQU      6DH       ;MODE FOR TIMER 1, SQUARE WAVE GEN
;RD/WR TWO BYTES, PRESCALER = 2
0000      TOSCL0   EQU      00H       ;LO BYTE -- COUNT VALUE
00A0      TOSCL1   EQU      0A0H      ;HI BYTE -- FOR 0
0001      TISCL0   EQU      01H       ;LO BYTE -- COUNT VALUE
0000      TISCL1   EQU      00H       ;HI BYTE -- FOR 1
PAGE

```

Figure 11. IR Prototype Verification Program (page 1 of 7).

```

;*** NSC810 PORT ASSIGNMENTS ***

0020      PORTA  EQU    20H
0021      PORTB  EQU    21H
0024      DDRA   EQU    24H
0025      DDRB   EQU    25H
0026      DDRC   EQU    26H
0027      MDRA   EQU    27H
0030      TOLB   EQU    30H
0031      TOHB   EQU    31H
0032      T1LB   EQU    32H
0033      T1HB   EQU    33H
0034      TOSTOP EQU    34H
0035      TOSTRT EQU    35H
0036      T1STOP EQU    36H
0037      T1STRT EQU    37H
0038      TMRO   EQU    38H
0039      TMR1   EQU    39H

;*** A/D CONVERTER SENSOR ADDRESSES ***

0040      ADC0   EQU    40H
;... AND OTHERS ...
0047      ADC7   EQU    47H
;... AND OTHERS ...
004F      ADCF   EQU    4FH

;*** MBM I/O PORT ASSIGNMENTS

0088      BM$DATA EQU    88H      ;MBM DATA (I/O)
0089      BM$CMD  EQU    89H      ;MBM COMMAND (OUT ONLY)
0089      BM$STAT EQU    89H      ;MBM STATUS (IN ONLY)

;*** REGISTER ADDRESS COUNTER (RAC) ASSIGNMENTS

0000      FIFO   EQU    00H      ;FIFO I/O REGISTER

;*** MBM COMMAND CODES

0012      BM$RD   EQU    12H      ;READ BUBBLE DATA
0013      BM$WR   EQU    13H      ;WRITE BUBBLE DATA

;***** END CONSTANTS *****
PAGE

```

Figure 11. IR Prototype Verification Program (page 2 of 7).

```

;***** VARIABLES *****

                ORG     2000H           ;BEGINNING ADDRESS OF RAM
2000            DS      64             ;*** DEFINE SYSTEM ***
2040            STACK EQU $             ;***   STACK   ***
2040            SAVER: DS      1         ;ONE BYTE TEMPORARY SAVE AREA

                ORG     3100H           ;(ADDRESS WITHIN ANOTHER RAM)
0028            FIFOLN EQU    40D       ;LENGTH OF FOLLOWING FIFO BUFFERS
3100            FIFOUT: DS    40D       ;BMC FIFO OUTPUT BUFFER
3128            FIFOIN: DS    40D       ;BMC FIFO INPUT BUFFER

;***** END VARIABLES *****


                ORG     0000H
0000  C3 0100    JP      START          ;START AT BEGINNING ON SYSTEM RESET

                ORG     002CH
002C  C3 0300    JP      ADC$HNDL       ;RSTC INTERRUPT ENTRY
                                           ;A/D CONVERTER INTERRUPT

                ORG     0034H
0034  76        HALT                    ;RSTB INTERRUPT ENTRY
                                           ;NOT YET SUPPORTED

                ORG     0038H
0038  76        HALT                    ;INTR INTERRUPT ENTRY
                                           ;NOT YET SUPPORTED

                ORG     003CH
003C  C3 0200    JP      T0INT$HNDL     ;RSTA INTERRUPT ENTRY
                                           ;TIMER 0 INTERRUPT

                ORG     0100H
0100  31 2040    START: LD    SP,STACK   ;INIT STACK PNTR

```

Figure 11. IR Prototype Verification Program (page 3 of 7).

```

;*** * * * *
;*** SET UP NSC810 I/O PORTS ***
;*** * * * *

0103 3E FF          LD    A,DDOUT      ;*** INIT ALL PORT A ***
0105 D3 24          OUT    (DDRA),A    ;*** BITS AS OUTPUT ***

0107 3E 00          LD    A,DDIN       ;*** INIT ALL PORT B ***
0109 D3 25          OUT    (DDRB),A    ;*** BITS AS INPUT ***

010B 3E 23          LD    A,DDCTRL     ;*** INIT DIRECTION OF ***
010D D3 26          OUT    (DDRC),A    ;*** CONTROL BITS ***

010F 3E 03          LD    A,STBOUT     ;*** INIT PORT A FOR ***
0111 D3 27          OUT    (MDRA),A    ;*** STROBED OUTPUT ***

;*** * * * *
;*** SET UP NSC810 TIMERS ***
;*** * * * *

0113 3E 19          LD    A,TOMODE     ;*** SET UP TIMER 0 AS ***
0115 D3 38          OUT    (TMR0),A    ;*** EVENT COUNTER ***

0117 3E 00          LD    A,TOSCLO     ;*** INIT ***
0119 D3 30          OUT    (TOLB),A    ;*** TIMER ***
011B 3E A0          LD    A,TOSCL1     ;*** 0 ***
011D D3 31          OUT    (TOHB),A    ;*** COUNT ***

011F 3E 6D          LD    A,TIMODE     ;*** SET UP TIMER 1 AS ***
0121 D3 39          OUT    (TMR1),A    ;*** SQUARE WAVE GEN ***

0123 3E 01          LD    A,T1SCLO     ;*** INIT TIMER ***
0125 D3 32          OUT    (T1LB),A    ;*** 1 COUNT ***

0127 D3 35          OUT    (TOSTRT),A  ;*** START THE ***
0129 D3 37          OUT    (T1STRT),A  ;*** COUNTERS ***

012B DB 30          IN     A,(TOLB)     ;*** INSURE TIMER 0 ***
012D DB 31          IN     A,(TOHB)     ;*** INTERRUPTS ARE RESET ***

```

PAGE

Figure 11. IR Prototype Verification Program (page 4 of 7).

```

;*** * * * *
;*** SET UP INTERRUPT STRUCTURE ***
;*** * * * *

```

```

012F 3E 0A      LD      A,IERVAL      ;ENABLE SYSTEM INTERRUPTS
0131 E6 FD      AND      OFDH          ;TURN OFF RSTC
0133 D3 BB      OUT      (IER),A      ;SET INTERRUPT ENABLE REG
0135 ED 56      IN       1            ;SET INTR FOR RSTX TYPE INTERRUPTS
0137 FB      EI

```

```

;*** * * * *
;*** IR TEST LOOP ***
;*** * * * *

```

```

0138 21 3100      LD      HL,FIFOUT      ;SET PNTR TO FIFO OUTPUT BUFFER
013B 06 28      LD      B,FIFOLN        ;INIT BUFFER LENGTH COUNTER

013D DB 21      LOOP:  IN       A,(PORTB)  ;READ PORTB
013F 77          LD      (HL),A          ;SAVE VALUE JUST READ
0140 D3 20      OUT      (PORTA),A      ;WRITE VALUE
0142 23          INC      HL            ;INC BUFFER PNTR
0143 05          DEC      B             ;DEC BUFFER BYTE COUNT
0144 CD 0199     CALL     WAIT

0147 D3 47      OUT      (ADC7),A        ;START A/D CONVERSION
0149 3E 0A      LD      A,IERVAL        ;*** ENABLE RSTA & ***
014B D3 BB      OUT      (IER),A        ;*** RSTC INTERRUPTS ***
014D CD 0199     CALL     WAIT

0150 3A 2040     LD      A,(SAVER)        ;*** SAVE VALUE ***
0153 77          LD      (HL),A          ;*** JUST READ ***
0154 D3 20      OUT      (PORTA),A      ;WRITE VALUE
0156 23          INC      HL            ;INC BUFFER PNTR
0157 CD 0199     CALL     WAIT
015A 10 E1      DJNZ     LOOP            ;DEC BUFFER BYTE COUNT AND
                                           ;LOOP UNTIL BUFFER FULL

015C 3E 55      LD      A,55H           ;*** OUTPUT FIFO TEST ***
015E D3 20      OUT      (PORTA),A      ;*** STARTED INDICATOR ***

0160 3E 00      LD      A,FIFO          ;*** SET BMC PNTR ***
0162 D3 89      OUT      (BMCMD),A      ;*** TO FIFO ***

```

Figure 11. IR Prototype Verification Program (page 5 of 7).

```

0164 06 28          LD      B,FIFOLN      ;*** DUMP BUFFER ***
0166 21 3100        LD      HL,FIFOUT     ;*** TO ***
0169 ED B3          OTIR                    ;*** BMC FIFO ***

016B CD 0199        CALL    WAIT           ;WAIT A WHILE
016E CD 0199        CALL    WAIT
0171 CD 0199        CALL    WAIT

0174 06 28          LD      B,FIFOLN      ;*** FILL BUFFER ***
0176 21 3128        LD      HL,FIFOIN     ;*** FROM ***
0179 ED B2          INIR                    ;*** BMC FIFO ***

017B 11 3100        LD      DE,FIFOUT     ;*** INIT COMPARE ***
017E 21 3128        LD      HL,FIFOIN     ;*** LOOP DRIVING ***
0181 01 0028        LD      BC,40D        ;*** PARAMETERS ***

0184 1A             LD      A,(DE)         ;GET FIFO OUTPUT BUF VALUE
0185 ED A1          CMPLP: CPI             ;COMPARE OUTPUT TO INPUT BUFFER
0187 13             INC      DE            ;BUMP PNTR
0188 20 09          JR      NZ,ERRFF      ;ERROR - BUFFERS NOT THE SAME
018A EA 0185        JP      PE,CMPLP      ;LOOP UNTIL END OF BUFFERS

                                ;BUFFERS COMPARED OK
018D 3E D0          LD      A,ODOH         ;*** OUTPUT SATISFACTORY ***
018F D3 20          OUT     (PORTA),A      ;*** COMPLETION INDICATOR ***
0191 F3             DI
0192 76             HALT

0193 3E FF          ERRFF: LD      A,OFFH   ;*** OUTPUT BAD COMPARISON ***
0195 D3 20          OUT     (PORTA),A      ;*** INDICATOR ***
0197 F3             DI
0198 76             HALT

                                ;*** * * * * *
                                ;*** WAIT FOR AN INTERRUPT ***
                                ;*** * * * * *

0199 F5             WAIT: PUSH    AF
019A 76             HALT
019B F1             POP     AF
019C C9             RET

```

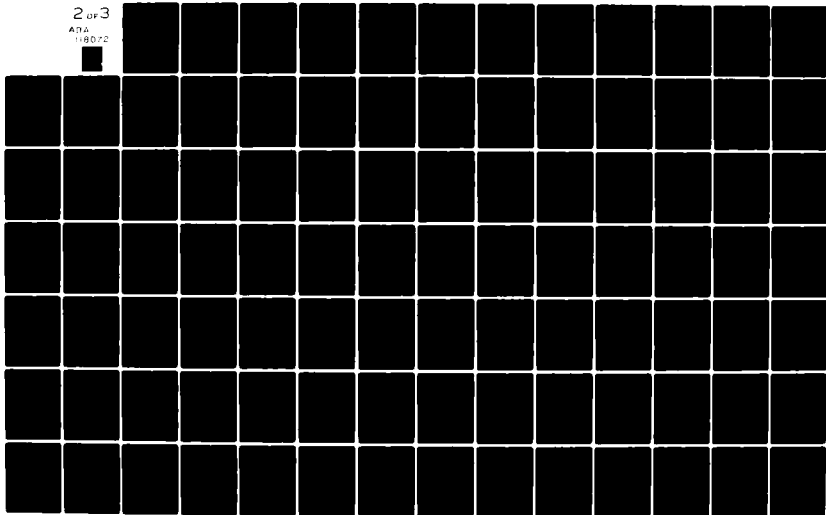
PAGE

Figure 11. IR Prototype Verification Program (page 6 of 7).

AD-A118 072 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCH00--ETC F/6 14/3
AN INFLIGHT RECORDER PROTOTYPE FOR THE INFLIGHT PHYSIOLOGICAL D--ETC(U)
FEB 82 R E WEISNER
UNCLASSIFIED AFIT/GCS/EE/82M-5 NL

2 of 3

ADA
118072



```

;*** * * * *
;*** TIMER 0 INTERRUPT HANDLER ***
;*** * * * *

                                ORG     0200H
0200      TOINT$HNDL:          PUSH     AF
0200      F5

0201      DB 30                IN      A,(TOLB)      ;*** RESET TIMER 0 ***
0203      DB 31                IN      A,(TOHB)      ;*** INTERRUPT ***
0205      FB                    EI

0206      F1                    POP      AF
0207      ED 4D                RETI

;*** * * * *
;*** A/D CONVERTER INTERRUPT HANDLER ***
;*** * * * *

                                ORG     0300H
0300      ADC$HNDL:          PUSH     AF
0300      F5

0301      DB 47                IN      A,(ADC7)      ;*** SAVE CONVERTER ***
0303      32 2040              LD      (SAVER),A      ;*** VALUE ***

0306      3E 0A                LD      A,IERVAL      ;*** DISABLE ***
0308      E6 FD                AND     OFDH          ;*** ONLY RSTC ***
030A      D3 BB                OUT     (IER),A        ;*** INTERRUPTS ***
030C      FB                    EI

030D      F1                    POP      AF
030E      ED 4D                RETI

                                END

```

Figure 11. IR Prototype Verification Program (page 7 of 7).

V Conclusions and Recommendations

With two exceptions, the IR prototype constructed in this thesis conforms to the hardware architecture previously defined in Figure 2. One exception is that testing of the MBM peripheral is not complete. The other is that RAM is increased to allow software experiments to vary MBM data buffer sizes. In addition to wirewrapping a prototype, thesis results include construction of several tools to support system development.

Appendices to this thesis contain documentation and user's manuals for IR prototype support tools. Tools that can be found in the appendices are:

- Appendix B - EEPROM Programmer,
- Appendix C - MBM Interactive Development System, and
- Appendix D - IR Debugging System.

The EEPROM programmer is used to dump software from floppy discs to HNVM3008 EEPROM's. The MBM Interactive Development System is primarily a tool for teaching new users capabilities and limitations of the Intel 7110 MBM. The capabilities taught are those pertinent to the IR. Additionally, the development system can be used to troubleshoot and verify MBM operation. The last support tool, the IR Debugging System, is a hardware front panel for the IR prototype. It provides a minimum level of software debugging support when connected to the IR prototype.

Conclusions

The scope of this thesis allowed design requirements to be developed, and a prototype to be constructed. Time constraints forced an end to this thesis before a definitive analysis could show that the IR prototype adheres to the four required characteristics outlined in Chapter II. The two requirements for the IR to be solid-state and microprocessor controlled are incorporated into the hardware architecture. More work must be done before definitive statements can be made about the unobstructive and battery operated requirements.

While it is too early to say that the IR will be unobstructive, its thickness should be smaller than the two inches that pilots found restrictive in the IFPDAS I. This estimate is based on the likely assumption that the IR will consist of IC's housed on a printed circuit board. An upper bound on the length and width of a flyable IR is the current area of the wirewrapped prototype. This area, 13 x 4.5 inches, is projected to include the MBM interface but not the MBM's themselves. Density of IC's in the flyable should be greater than that of the wirewrapped prototype. An additional factor that could reduce board area is the possibility that some IC's can be eliminated once software is developed for the IR. The upper bound just developed for the IR does not include space for the MBM storage peripheral. A discussion of MBM space requirements follows under

Recommendations. Estimates of the unobstructive property of the new IR must wait for further system development.

Until software is running on the IR prototype, battery operated requirements can only be rough estimates. One important estimating factor is that power dissipation in CMOS components varies with operating frequency. Even if the operating frequency of each component could be projected, NSC800 documentation does not contain the figures required to accurately estimate power consumption. Another important factor is that current draws vary within MBM's, depending on I/O frequency and numbers of parallel operating bubbles. Therefore, accurate estimates on battery requirements must wait until the system can be exercised by software.

In addition to the statements made about required characteristics, the following observations relate to desired characteristics. There are 16 A/D converter channels available which accept 0V - 5V conditioned inputs. Output from any channel to the CPU is available 100 microseconds after conversion is started, allowing a maximum sampling frequency of 10,000 per second. In addition to being able to increase the sampling rate of sensors, other design characteristics leave room for expansion. A discussion of how each IR component can be expanded is found in the component subsections of Chapter II. An evaluation of the last desired characteristic of four hour operation depends on a solution to battery operated requirements. Therefore, all

desirable characteristics with the exception of four hour operation have been achieved in the IR prototype.

Recommendations

As mentioned above, the IR prototype is not fully operational. Until it is, unobstructive size and battery operated requirements can not properly be evaluated. Therefore, the first step should be to complete prototype construction by debugging the MBM peripheral. The peripheral has been wirewrapped as specified by the design in Chapter III. Communications between the CPU and BMC has also been verified. However, initial tests could not access the MBM itself. Software to debug and ultimately drive the MBM peripheral can be adapted from modules found in Appendix C.

Another high priority task should be to develop a software prototype for the IR. Once software is developed, hardware component requirements can be optimized. This optimization should result in a reduction of the number of IC's used in the flyable IR. Another reason for completing software early in the next thesis cycle is that it will allow the system requirements for battery operation and system bulk to be evaluated. Then a descision can be made about continued IR development.

An important point must be made with reference to the MBM secondary storage peripheral. Continued development with the current 1M bit MBM's will probably result in an IR that is too bulky. However, Intel has announced that 4M bit

bubbles will be available for general sampling during the first half of 1983 (Ref 25). With the lead time for development of systems in an academic environment, work should not be discontinued to await release of the next generation of bubbles. Instead, development should continue along the lines set down in this thesis. That is, any redesign of the MBM peripheral should remain modular so that new bubbles can easily be interfaced once they become available. Meanwhile, an IR can be developed with less than a 1M byte capacity for reduced data acquisition tasks and to prove the concept of the new generation IR.

As stated previously, the IR Debugging Tool provides only minimum front panel support for software development. A recommendation for improving the front panel is to add hardware breakpoints. Currently, the only way to insure that the machine halts at a point of interest is to single step to that point. As programs get longer, this becomes increasingly tedious. Besides, single stepping interferes with a program's interaction with interrupts.

Another lesson learned during software exercising of the prototype is that programs burned into EEPROM's are cumbersome to debug. This results from the fact that changes can not be made to software during testing. Instead the EEPROM must be reprogrammed and the test restarted for each bug found. Developing a capability to replace EEPROM's with RAM during software development would cure this problem. Then programs loaded into the RAM could be altered during

testing through use of the IR Debugging Tool's memory write capability. Relying on the fact that HNVM3008's have an industry standard pin-out should minimize disruption of IR prototype hardware.

Currently, EEPROM's must be removed from the IR whenever reprogramming is desired. Future designs should incorporate methods for programming the EEPROM's while they remain in the IR. However, doing so should not add hardware to the IR itself. The benefit of programming the EEPROM's without removing them from the IR is that the possibility for system errors is reduced. Errors are reduced in two ways. One is that the possibility of misplacing IC's is eliminated. The other is that permanent mechanical contacts have less chances of loosening to cause unpredictable results.

Now that the IR prototype is nearing completion, consideration should be given to other components of the IFPDAS. As development continues, the answers to three general questions become important. How will users develop software for the IR? What field processing capabilities does SAM need? How will SAM get the data from the field into their laboratory data base? Until the IR can be integrated into the IFPDAS, its use is restricted to proving feasibility of design.

Bibliography

1. Bell, Gordon C. and Allen Newell. Computer Structures: Readings and Examples. New York: McGraw-Hill Book Company, 1971.
2. BPK-72 Bubble Memory Prototype Kit Users Manual, Santa Clara, CA: Intel Corp, 1981.
3. CMOS Databook. Santa Clara, CA: National Semiconductor Corp, 1978.
4. Component Data Catalog. Santa Clara, CA: Intel Corp, 1980.
5. Electrical Research Association. The Engineering of Microprocessor Systems. Oxford, England: Pergamon Press Ltd, 1979.
6. Fullager, David. "CMOS Comes of Age," IEEE Spectrum, 17:24-7 (December 1980).
7. F264 - 65,536x1 Dynamic Serial Memory. Product Specification. Mountain View, CA: Fairchild Camera and Instrument Corp, October 1980.
8. Hall, Capt. and Lt. Shackford. IFPDAS point of contact at SAM (personal interview). Brooks AFB, TX, 5-6 May 1981.
9. Hall, Douglas V. Microprocessors and Digital Systems. New York: McGraw-Hill Book Company, 1980.
10. Harris Digital Data Book, Volume 2. Melbourne, FL: Harris Corp, 1981.
11. Hill, Robert E. Aircrew Modularized Inflight Data Acquisition System. MS Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1978.
12. HNVM 3008, 8K CMOS EEPROM. Product Description. Newport Beach, CA: Hughes Solid State Products, March 1981.
13. IC Master 1981, Volume 2. Garden city, NY: United Technical Publications, Inc, 1981.
14. IC Memories. San Jose, CA: Hitachi America Ltd, 1980.

15. Intel 7220-1, Bubble Memory Controller. Product Description. Santa Clara, CA: Intel Corp, 1981.
16. Jolda, Joseph G. and Stephen J Wanzek. Aircrew Inflight Physiological Data Acquisition System II. MS Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1977.
17. Magnetic Bubble Storage Data Catalog. Santa Clara, CA: Intel Corp, February 1981.
18. Mano, M Morris. Digital Logic and Computer Design. Englewood Cliffs, NJ: Prentiss-Hall, Inc, 1979.
19. "Microcomputer Data Manual," Electronic Design, 29: 88-99 (March 19, 1981).
20. "Microprocessor Data Manual," Electronic Design, 28: 107-208 (November 22, 1980).
21. Moore, Kenneth L. Aircrew Inflight Physiological Data Acquisition System. MS Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, June 1980.
22. MC146805E2. Product Specification. Austin, TX: Motorola Semiconductor Products Inc, 1980.
23. Nassi, I. and B. Shneiderman. "Flowchart Techniques for Structured Programming," ACM SIGPLAN Notices, 8: 12-26 (August 1973).
24. NSC800 Microprocessor Family Handbook. Santa Clara, CA: National Semiconductor Corp, 1981.
25. Ramndanes, Carol. Marketing Representative, Non-Volatile Memory Division, Intel Corporation (personal interview). Santa Clara, CA, 24 February 1982.
26. RCA COS/MOS Memories, Microprocessors, and Support Systems. Somerville, NJ: RCA Corp, 1979.
27. Texas Instruments IC Applications Staff. Designing With TTL Integrated Circuits, edited by Robert L Morris and John R Miller. New York: McGraw-Hill Book Company, 1971.
28. TTL Data Book (Second Edition). Dallas TX: Texas Instruments Inc, 1976.
29. Twaddle, William. "Special Report: CMOS IC's", EDN, 26: 88-100 (June 24, 1981).
30. Zaks, Rodney. Microprocessors, from Chips to Systems. USA: Sybex Inc, 1980.

Appendix A

IR Prototype Schematic

This appendix contains the schematic diagram of the IR prototype. Instead of using a one page foldout, the schematic is broken into logical pieces and distributed over five pages. To combine the pages, lines ending at the right-hand margin of one page are continued at the left margin of the next page.

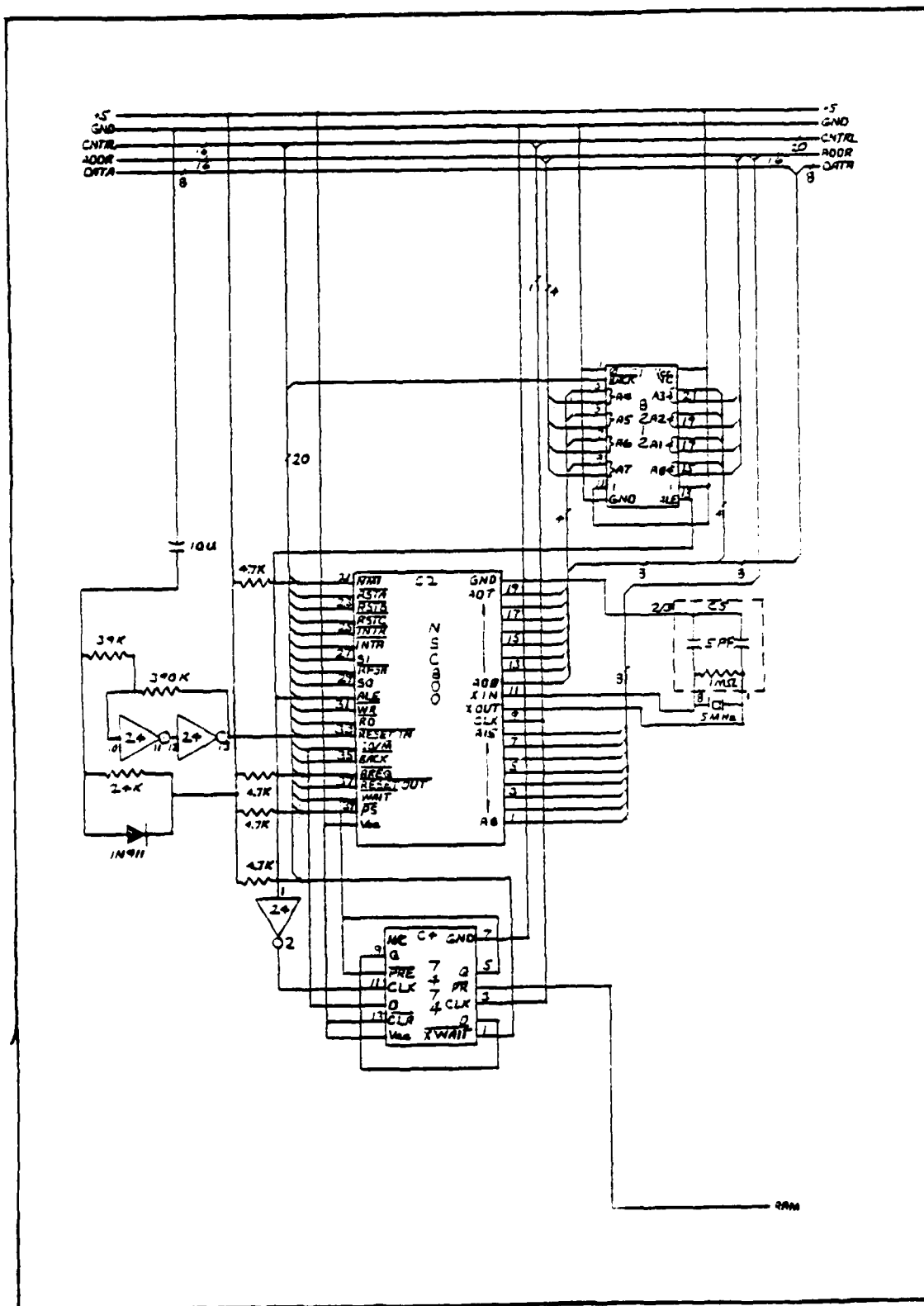


Figure 12. IR Prototype Schematic (page 1 of 5).

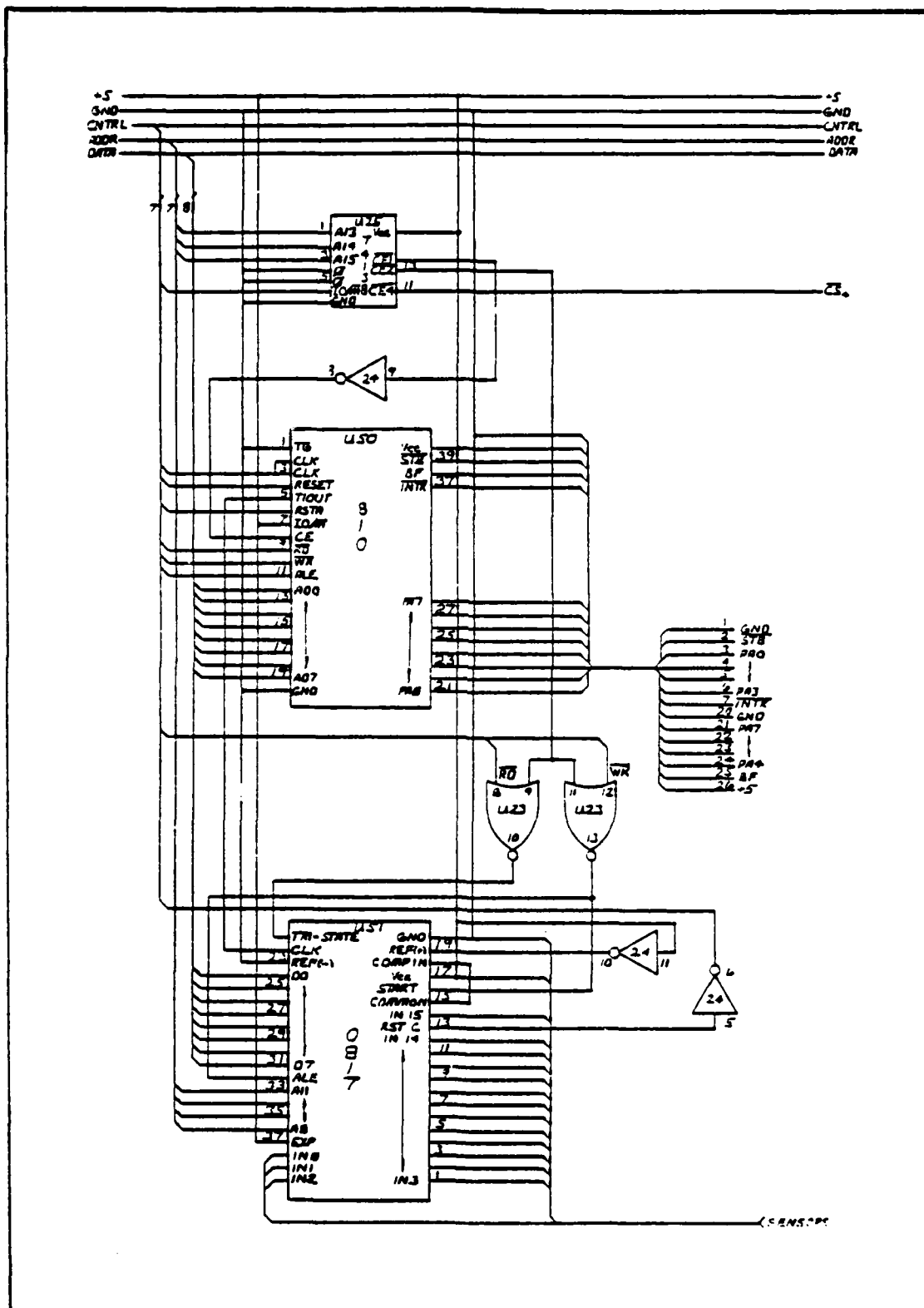


Figure 12. IR Prototype Schematic (page 4 of 5).

Figure 12. IR Prototype Schematic (page 5 of 5).

Appendix B

EEPROM Programmer

Contents

I.	Introduction	100
II.	Schematic Diagram	101
III.	Software	108
IV.	User's Manual	147
	System Start-up	148
	Commands	148
	ERASE	149
	PROGRAM	149
	VERIFY	151
	DUMP	153
	Errors	154

EEPROM Programmer

I. Introduction

This Appendix describes and documents operation of the EEPROM Programmer designed to support Hughes Solid-state Products HNVM 3008 EEPROM's. Documentation consists of a schematic diagram and an associated software listing. Following the software listing is a user's manual which describes the Programmer's capabilities and summarizes its operating procedures. Another important source of information, the HNVM 3008 data sheet, is located in Appendix E.

The EEPROM Programmer described in this document is a flexible tool for supporting HNVM 3008 EEPROM's. This flexibility results from two design considerations. One is that the hardware is based on the S-100 bus. Another is that software runs under control of the Cromemco Disk Operating System (CDOS) and consequently the Control Program for Microprocessors (CPM) Operating System. Further explanations of these design decisions are contained in the following sections of this document.

II. Schematic Diagram

The hardware used to implement the Programmer is illustrated in the schematic diagram of Figure 12. To facilitate understanding of the schematic, Table XI lists the functions of the IC's used to construct the Programmer. More detailed information on individual IC's is available from The TTL Data Book and The Intel Component Data Catalog (Refs 11; 4).

The Programmer is wirewrapped on a Cromemco Z-2D prototyping card, and therefore, can be easily transported to any S-100 based system. Table XII illustrates which S-100 pins are used by the Programmer. Since the interface to the S-100 bus is fully buffered, each line in Table XII presents only a single TTL load to the bus.

Another aid to transportability is the onboard switch selection of the five most significant bits of the Programmer port addresses. This allows Programmer hardware addresses to be chosen which do not interfere with the permanent I/O addresses of the host computer. The EEPROM Programmer addresses are selected by opening and closing appropriate switches. Closed switches indicate zero bit settings, and open switches indicate ones. The most significant bit of the address switches is plainly marked on the wire-wrap card. Beware that changes to these address switches require that corresponding changes be made to Programmer software.

TABLE X
EEPROM Programmer Selectable Ports

Port Address	Function
BBBB B000	EEPROM Data Bus
BBBB B001	EEPROM Address LSB(yte)
BBBB B010	EEPROM Address MSB(yte)
BBBB B011	I/O Command/Status
BBBB B100	EEPROM Control Bus

Table X lists the EEPROM Programmer ports which are affected by hardware address settings. The three least significant bits, denoted by B's in Table X, are switch selectable, allowing 32 choices for port addresses. One possible selection is 00110, yielding software addresses 30H through 34H. Switch settings to coordinate these addresses with the hardware are - from the most significant bit - closed, closed, open, open, and closed.

The only other requirement for EEPROM Programmer operation is the need for an external 20V power supply. This power source is regulated on the Programmer card to provide either 5V or 17V to the positive supply terminal of the EEPROM. Switching between the two volatges is governed by software.

An important note with reference to the HNVM 3008 is that proper programming depends on a continuous voltage to the positive supply pin of the EEPROM. Output from the regulator which supplies EEPROM power must not go to ground during voltage transitions between 5V and 17V. The LM317 and its associated circuitry provides these continuous power transistions. Consequently, voltage changes from 5V to 17V and from 17V to 5V, produce output waveforms that are step functions.

TABLE XI
EEPROM Programmer IC Listing

Device Type	Functional Designation	Schematic Reference
74365	Hex Bus Drivers	U1,U2
8216	4-bit Bidirectional Bus Driver	U3,U4
7404	Hex Inverters	U5
74156	3-to-8 Line Decoder	U6
7485	4-bit Magnitude Comparator	U7,U8
7400	Quad 2-input NAND Gates	U12
8255	Programmable Peripheral Interface	U11
8212	8-bit I/O Port	U13

TABLE XII

S-100 to EEPROM Programmer Interface Definition

S-100 Pin	Signal Function	S-100 Pin	Signal Function
29	Addr 5	75	RESET
30	Addr 4	79	Addr 0
31	Addr 3	80	Addr 1
35	Data Out 1	81	Addr 2
36	Data Out 0	82	Addr 6
38	Data Out 4	83	Addr 7
39	Data Out 5	88	Data Out 2
40	Data Out 6	89	Data Out 3
41	Data In 2	90	Data Out 7
42	Data In 3	91	Data In 4
43	Data In 7	92	Data In 5
45	OUT	93	Data In 6
46	INP	94	Data In 1
50	GND	95	Data In 0
51	+8V		

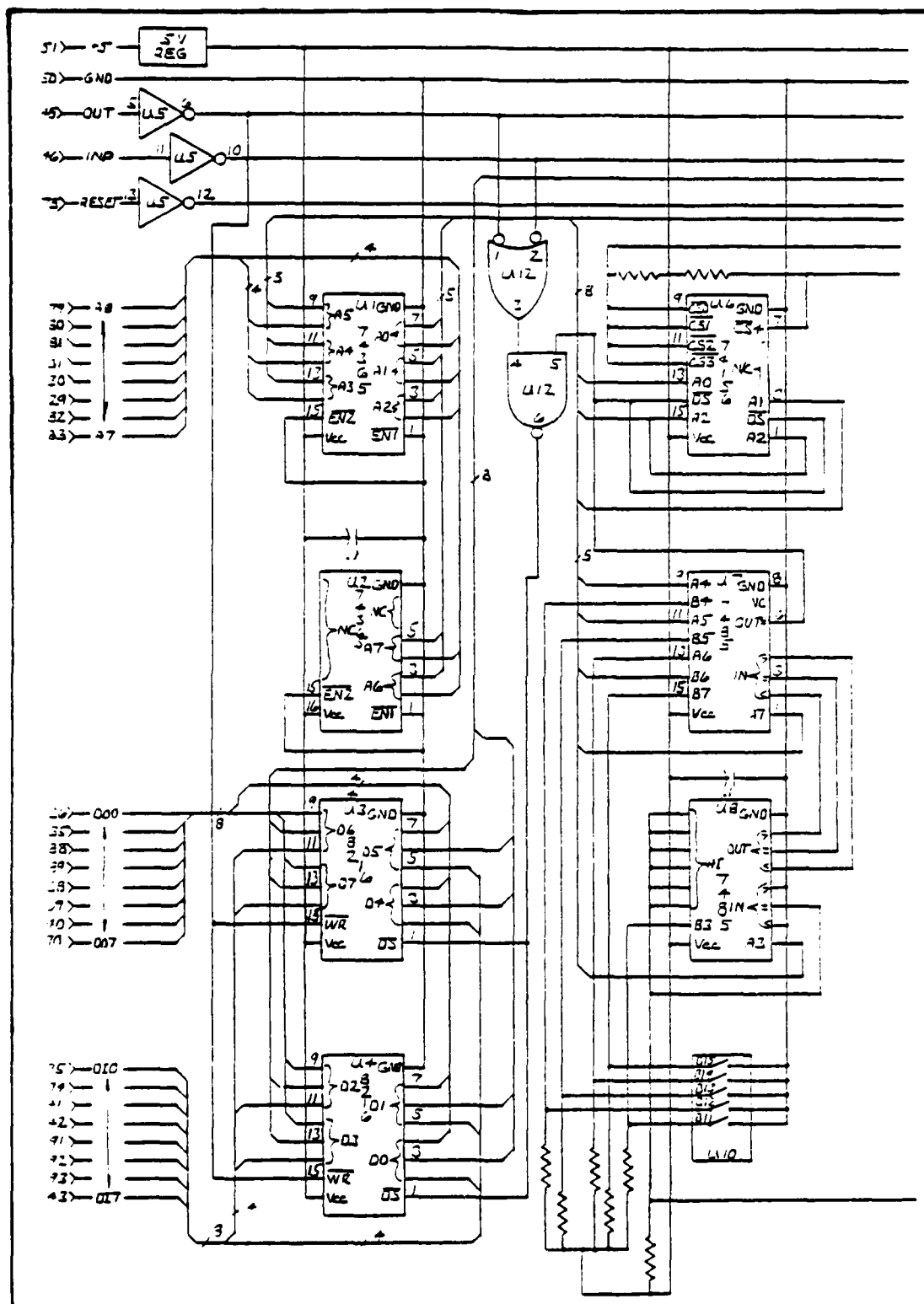


Figure 13. EEPROM Programmer Schematic (page 1 of 2).

Figure 13. EEPROM Programmer Schematic (page 2 of 2).

III. Software

Figure 15 is a software listing of the program used to drive the EEPROM Programmer hardware. Its basic flow is outlined in the Nassi-Shneiderman chart (Ref 23) of Figure 14. The software was written in Z-80 assembler language with system calls to CDOS for I/O support. Since system calls are restricted to those between 1 and 27, the software is transportable to CPM based systems without modification. This transportability results from identical execution of the operating systems for calls in the range of 1 to 27.

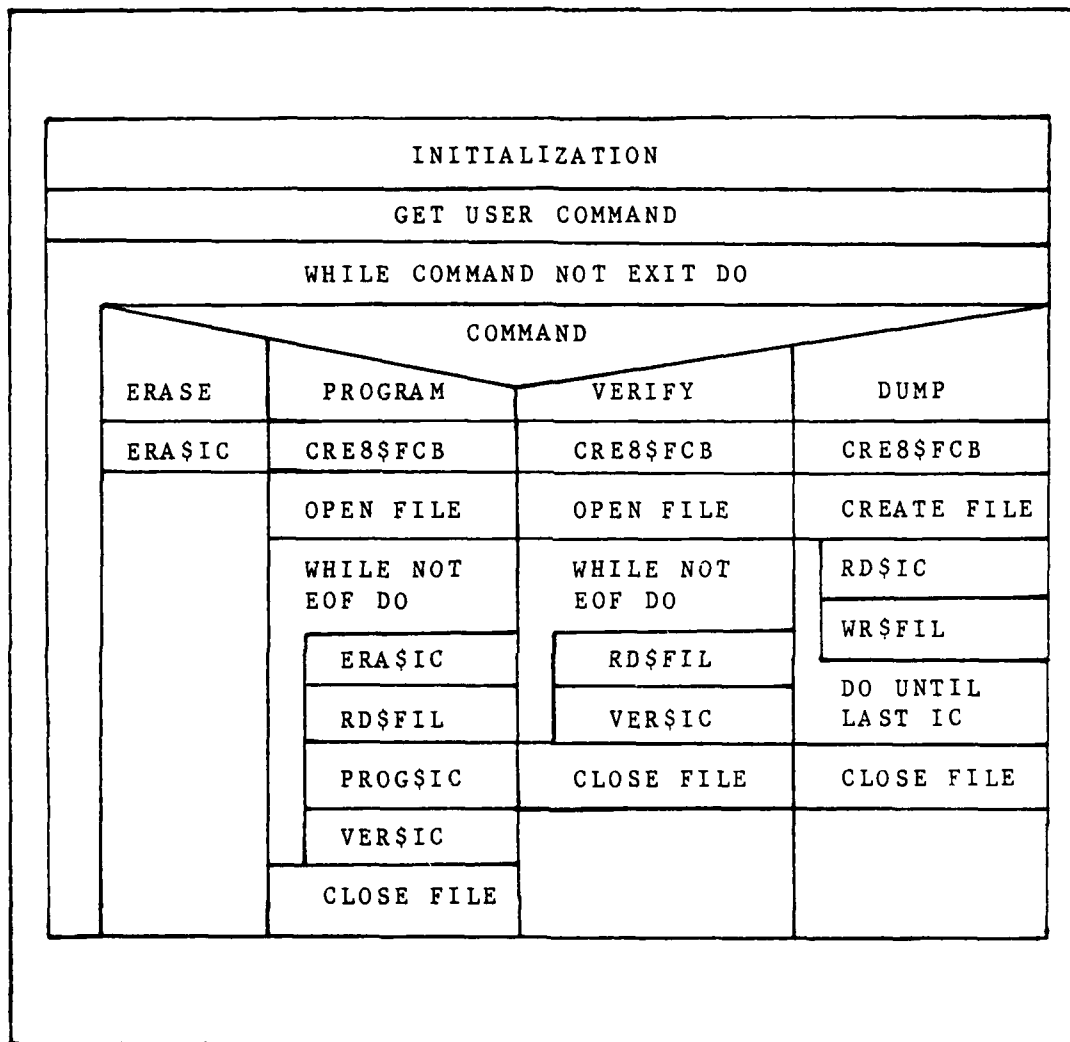


Figure 14. EEPROM Programmer Flowchart

```
.Z80
.COMMENT Z
;AUTHOR: CAPT R E MEISNER
;DATE: 25 AUG 81
;SYSTEM: CROMEMCO Z2D (4 MHZ) / CDOS 2.36
;DESCRIPTION: THIS ROUTINE SUPPORTS HUGHES HMVM 3008 EEPROM'S BY
;              PROVIDING THE FOLLOWING OPERATIONS:
;              ERASE - ERASE AN IC,
;              PROGRAM - DUMP A FILE TO IC(S),
;              VERIFY - INSURE FILE AND IC(S) DATA MATCH, AND
;              DUMP - DUMP IC(S) TO A FILE.
;OPERATION:
```

THIS PROGRAM IS EXECUTED BY RUNNING "EEPROM" FROM THE CDOS MACHINE LEVEL. ONCE INITIATED, EEPROM WILL GUIDE THE USER THROUGH OPERATION OF THE PROGRAM WITH APPROPRIATE CONSOLE DIRECTIVES. WHEN DONE, THE USER CAN EXIT GRACEFULLY BACK TO THE CDOS LEVEL.

***** EEPROM PORT REQUIREMENTS *****

PORT ADDRESSES ARE SWITCH SELECTABLE BY SETTING THE HIGH ORDER 5 BITS OF THE PORT ADDRESS ON THE PROGRAMMER BOARD. THE LOWER 3 BITS HAVE THE FOLLOWING DEFINITIONS:

- 0 - EEPROM DATA BUS
- 1 - EEPROM ADDRESS LSB(YTE)
- 2 - EEPROM ADDRESS MSB(YTE)
- 3 - 8255 COMMAND/STATUS PORT
- 4 - EEPROM CONTROL BUS

***** EEPROM CONTROL LINE DEFINITIONS *****

- D7 - N/A
- D6 - N/A
- D5 - N/A
- D4 - N/A
- D3 - VDD CONTROL (0 = 17V, 1 = 5V)
- D2 - CE (ACTIVE LOW)
- D1 - OE (ACTIVE LOW)
- D0 - CS (ACTIVE HIGH)

Z
PAGE

Figure 15. EEPROM Programmer Software (page 1 of 37).

.COMMENT X

 *** NOTE TO MAINTENANCE PROGRAMMERS ***

SEVERAL SUBROUTINES IN THIS PROGRAM CONTAIN TIME SENSITIVE INSTRUCTION SEQUENCES. CONSULT THE HUGHES SOLID STATE PRODUCTS HNVM 3008 DATA SHEET BEFORE MAKING CHANGES. THE CRITICAL SUBROUTINES ARE:

ERASE, IC\$RD, AND PROG\$IC.

OTHER SUBROUTINES CAN BE FREELY BE CHANGED WITHOUT AFFECTING TIMING REQUIREMENTS.

ALSO, THROUGHOUT THIS PROGRAM THE ASSUMPTION IS MADE THAT THE EEPROM SUPPLY VOLTAGE IS NORMALLY SET AT 5V. IT IS ONLY INCREASED TO 17V WHEN REQUIRED FOR ERASING OR PROGRAMMING.

X

0000'		ASEG		
		ORG	0100H	
0100		ENTRY\$PT:		
0100	ED 73 014A	LD	(OLDSP),SP	;SAVE OLD STACK POINTER
0104	31 014A	LD	SP,STACK	;INITIALIZE NEW STACK
0107	C3 0D60	JP	START	
010A		DS	64	;64 BYTE STACK
014A		STACK EQU	\$	;TOP OF STACK
014A	0000	OLDSP: DW	0	;OLD STACK POINTER SAVE AREA
		PAGE		

Figure 15. EEPROM Programmer Software (page 2 of 37).

```

;***** CONSTANTS *****

FFFF      NEG1    EQU    -1
0000      ZERO    EQU    0
0001      ONE     EQU    1
0010      MAXERR  EQU    16      ;*** MAXIMUM NUMBER OF VERIFY ***
                                   ;*** ERRORS THAT WILL BE DISPLAYED ***
0080      RECSIZ  EQU    128     ;RECORD SIZE = DISK SECTOR SIZE
0008      BF      EQU    8       ;*** BLOCKING FACTOR FOR 1K ***
                                   ;*** (BF * RECSIZ = 1024) ***

;ASCII CHARACTERS

0020      BLANK   EQU    ' '
003A      COLON   EQU    ':'
002E      PERIOD  EQU    '.'
002F      SLASH   EQU    '/'

;CDOS SYSTEM CALL PARAMETERS

0005      CDOS    EQU    0005H    ;CDOS ENTRY POINT
0001      RDCHR   EQU    1        ;READ A CHARACTER FROM CONSOLE
0002      PRTCHR  EQU    2        ;PRINT A CHARACTER ON THE CONSOLE
0009      PRTLN   EQU    9        ;PRINT BUFFER LINE ON CONSOLE
000A      RDLN    EQU    10       ;INPUT BUFFER LINE FROM CONSOLE
0024      PRTEND  EQU    '$'      ;END PRINT BUFFER
000F      OPNFI   EQU    15       ;OPEN DISK FILE
0010      CLSFI   EQU    16       ;CLOSE DISK FILE
0014      RDFIL   EQU    20       ;READ A DISK SECTOR
0015      WRFIL   EQU    21       ;WRITE A DISK SECTOR
0016      CR8FI   EQU    22       ;CREATE A DISK FILE
0019      CURDK   EQU    25       ;GET CURRENT DISK INDICATOR
005C      FCB     EQU    05CH     ;BEGINNING OF FILE CONTROL BLOCK
0080      CDOS$DB EQU    080H     ;DEFAULT DISK BUFFER ADDRESS

;***** FILE CONTROL BLOCK DESCRIPTION *****
005C      FCB$DK  EQU    FCB+0    ;DISK DESCRIPTOR      *
005D      FCB$FN  EQU    FCB+1    ;FILE NAME          *
0065      FCB$FT  EQU    FCB+9    ;FILE TYPE          *
0068      FCB$EX  EQU    FCB+12   ;FILE EXTENT        *
006B      FCB$RC  EQU    FCB+15   ;RECORD COUNT       *
006C      FCB$MP  EQU    FCB+16   ;CLUSTER ALLOCATION MAP *
007C      FCB$NR  EQU    FCB+32   ;NEXT RECORD        *
;*****

```

Figure 15. EEPROM Programmer Software (page 3 of 37).

```

;CARRAIGE CONTROL

000D      CR      EQU      00DH      ;ASCII CARRAIGE RETURN
000A      LF      EQU      00AH      ;ASCII LINE FEED

;I/O PORT ADDRESSES

0001      CIO      EQU      001H      ;CONSOLE I/O PORT
0020      PROMA     EQU      020H      ;EEPROM DATA PORT
0021      PROMB     EQU      021H      ;EEPROM ADDRESS LSB
0022      PROMC     EQU      022H      ;EEPROM ADDRESS MSB
0023      PCNTRL    EQU      023H      ;PERIPHERAL CONTROLLER PORT FOR PORTS A, B, C
0024      PROMD     EQU      024H      ;EEPROM CONTROL PORT

;CHANNEL COMMAND WORDS FOR PROGRAMMING THE PERIPHERAL CONTROLLER

0080      CCW1      EQU      10000000B ;PORTS A, B, C = LATCHED OUTPUT
0090      CCW2      EQU      10010000B ;*** PORTS B, C = LATCHED OUTPUT ***
                                           ;*** PORT A = INPUT ***

;EEPROM CONTROL LINES AS DEFINED FOR PORT D

0008      V$5       EQU      008H      ;** SUPPLY **          ** 5V **
00F7      V$17      EQU      0FFH-V$5  ;** VOLTAGE **         ** 17V **
0004      D$CE       EQU      004H      ;*** CHIP ***          ** DISABLE **
00FB      E$CE       EQU      0FFH-D$CE ;*** ENABLE ***        ** ENABLE **
0002      D$OE       EQU      002H      ;** OUTPUT **          ** DISABLE **
00FD      E$OE       EQU      0FFH-D$OE ;** ENABLE **          ** ENABLE **
0001      E$CS       EQU      001H      ;*** CHIP ***          ** ENABLE **
00FE      D$CS       EQU      0FFH-E$CS ;*** SELECT ***        ** DISABLE **

;CONSOLE MESSAGES

014C      0D 0A 0A      MSG1: DB CR,LF,LF
014F      20 20 20 20      DB ' WHAT OPERATION DO YOU WISH TO PERFORM?',CR,LF
017B      45 28 52 29      DB 'E(R)ASE, (P)ROGRAM, (V)ERIFY, (D)UMP, OR E(X)IT'
01AA      0D 0A 24      DB CR,LF,PRTE
01AD      0D 0A      MSG2: DB CR,LF
01AF      46 49 4C 45      DB 'FILENAME? ',PRTE
01BB      0D 0A 0A      MSG3: DB CR,LF,LF
01BE      50 4C 45 41      DB 'PLEASE ANSWER THE FOLLOWING QUESTIONS IN HEXIDECIMAL'
01F2      0D 0A 4E 4F      DB CR,LF,'NOTE: THE FIRST 2 ADDRESSES MUST BE ON '
021C      4B 49 4C 4F      DB 'KILOBYTE BOUNDARIES',CR,LF,LF
0232      53 54 41 52      DB 'STARTING ADDRESS OF PROGRAM ON FILE? ',PRTE

```

Figure 15. EEPROM Programmer Software (page 4 of 37).

0259	0D 0A 46 49	MSG4:	DB	CR,LF,'FIRST ADDRESS TO BE PROGRAMMED/VERIFIED?'
0285	24		DB	PRTEEND
0286	0D 0A 4C 41	MSG5:	DB	CR,LF,'LAST ADDRESS TO BE PROGRAMMED/VERIFIED?'
02B1	24		DB	PRTEEND
02B2	0D 0A 0A	MSG6:	DB	CR,LF,LF
02B5	52 45 4D 4F		DB	'REMOVE OLD IC / INSERT NEXT IC',CR,LF,LF
02D6	50 52 45 53		DB	'PRESS ANY KEY WHEN READY',CR,LF,LF,PRTEEND
02F2	0D 0A 44 4F	MSG7:	DB	CR,LF,'DO YOU HAVE MORE EEPROMS? (Y/N)',PRTEEND
0314	0D 0A 56 45	MSG8:	DB	CR,LF,'VERIFICATION COMPLETED WITH NO ERRORS',PRTEEND
033C	0D 0A	MSGERA:	DB	CR,LF
033E	45 52 41 53		DB	'ERASING',PRTEEND
0346	2D 50 52 4F	MSGPRG:	DB	'-PROGRAMMING',PRTEEND
0353	2D 56 45 52	MSGVER:	DB	'-VERIFYING',CR,LF,PRTEEND
0360	0D 0A 2A 2A	ERR1:	DB	CR,LF,'*** ERROR *** FILE NOT FOUND',CR,LF,PRTEEND
0381	0D 0A 2A 2A	ERR2:	DB	CR,LF,'*** ERROR *** PROM DID NOT ERASE',CR,LF,PRTEEND
03A6	0D 0A 2A 2A	ERR3:	DB	CR,LF,'*** ERROR *** FILE COULD NOT BE CREATED'
03CF	0D 0A 24		DB	CR,LF,PRTEEND
03D2	2A 2A 2A 20	ERR4\$0:	DB	'*** VERIFY ERROR - PROM FILE/PROM',CR,LF
03F8	2A 2A 2A 20		DB	'*** ADDRESS VALUES',CR,LF,PRTEEND
041E	2A 2A 2A 20	ERR4\$1:	DB	'***',PRTEEND
0434	20 2F 20 24	ERR4\$2:	DB	' / ',PRTEEND
0438	0D 0A 2A 2A	ERR5:	DB	CR,LF,'*** ERROR *** DISK RECORD COULD NOT BE WRITTEN'
0468	0D 0A 24		DB	CR,LF,PRTEEND
046B	0D 0A 2A 2A	ERR6:	DB	CR,LF,'*** ERROR *** RELATIVE MAGNITUDE OF ADDRESSES'
049B	49 53 20 49		DB	'IS INVALID',CR,LF,PRTEEND
04A8	0D 0A 2A 2A	ERR7:	DB	CR,LF,'*** ERROR *** INVALID ADDRESS',CR,LF,PRTEEND
04CA	0D 0A 2A 2A	ERR8:	DB	CR,LF,'*** ERROR *** DISK FILE READ ERROR OR'
04F2	55 4E 45 58		DB	'UNEXPECTED EOF',CR,LF,PRTEEND

;***** END CONSTANTS *****

Figure 15. EEPROM Programmer Software (page 5 of 37).

```

;***** VARIABLES *****
0503      NXTADD: DS      2      ;NEXT EEPROM ADDR TO BE PROGRAMMED
0505      FLSTAD: DS      2      ;STARTING ADDR OF PROGRAM ON THE FILE
0507      FSTADD: DS      2      ;FIRST EEPROM ADDR TO BE PROGRAMMED
0509      LSTADD: DS      2      ;LAST EEPROM ADDR TO BE PROGRAMMED
050B      ERRADD: DS      2      ;SAVE AREA FOR AN ERROR ADDR
050D      01      ERRCNT: DB      1      ;TEMPORARY ERROR COUNTER
050E      50      CONBUF: DB      80      ;BUFFER LENGTH
050F      00              DB      0      ;NUMBER OF CHARACTERS READ
0510              DS      80      ;CONSOLE INPUT BUFFER
0560      DSKBUF: DS      BF*RECSIZ ;DISK BUFFER - HOLDS 'BF' RECORDS
0960      PROMBF: DS      1024      ;EEPROM BUFFER - HOLDS EEPROM IMAGE

;***** END VARIABLES *****
PAGE

```

Figure 15. EEPROM Programmer Software (page 6 of 37).

```

0D60 3E 0E          START: LD  A,V$5+D$CE+D$0E ;*** DISABLE EEPROM ***
0D62 D3 24          OUT   (PROMD),A ;*** CONTROL LINES ***

0D64              GET$OPR:
0D64 0E 09          LD    C,PRTLN ;* * * * *
0D66 11 014C        LD    DE,MSG1 ;* PROMPT USER FOR OPERATION *
0D69 CD 0005        CALL   CDOS ;* * * * *

0D6C 0E 01          LD    C,RDCHR ;*** GET USER ***
0D6E CD 0005        CALL   CDOS ;*** RESPONSE ***

0D71 FE 52          CP    'R'
0D73 CA 0D90        JP    Z,E$OPR ;GO ERASE
0D76 FE 50          CP    'P'
0D78 CA 0DA3        JP    Z,P$OPR ;GO PROGRAM
0D7B FE 56          CP    'V'
0D7D CA 0E18        JP    Z,V$OPR ;GO VERIFY
0D80 FE 44          CP    'D'
0D82 CA 0E8E        JP    Z,D$OPR ;GO DUMP
0D85 FE 58          CP    'X' ;EXIT?
0D87 20 DB          JR    NZ,GET$OPR ;NO, INVALID INPUT

0D89 ED 7B 014A     LD    SP,(OLDSP) ;YES, RESTORE STACK
0D8D C3 0000        JP    0 ; RETURN TO CDOS

;***** ERASE IC *****

0D90 0E 09          E$OPR: LD    C,PRTLN ;*** INSTRUCT USER ***
0D92 11 02B2        LD    DE,MSG6 ;*** TO TURN ON ***
0D95 CD 0005        CALL   CDOS ;*** PROGRAMMER ***

0D98 0E 01          LD    C,RDCHR ;**** WAIT UNTIL ****
0D9A CD 0005        CALL   CDOS ;**** DONE ****

0D9D CD 1131        CALL   ERA$IC

0DA0 C3 0D60        JP    START ;ALLOW USER ANOTHER OPERATION

```

PAGE

Figure 15. EEPROM Programmer Software (page 7 of 37).

***** ERASE, PROGRAM, & VERIFY IC *****

```

ODA3  CD 0F01      P%OPR: CALL  CRE8$FCB

ODA6  0E 0F          LD    C,OPNFI  ;* * * * *
ODA8  11 005C        LD    DE,FCB   ;* OPEN DISK FILE *
ODAB  CD 0005        CALL  CDOS     ;* * * * *

ODAE  FE FF          CP    NEG1      ;WAS OPEN SUCCESSFUL?
ODBO  C2 0DBE        JP    NZ,P%C1   ;YES
ODB3  0E 09          LD    C,PRTLN   ;NO, * * * * *
ODB5  11 0360        LD    DE,ERR1   ; * PRINT ERROR *
ODB8  CD 0005        CALL  CDOS     ; * * * * *
ODBB  C3 0D60        JP    START     ;ALLOW USER ANOTHER TRY

ODBE  CD 0F6E        P%C1: CALL  SET$ADDR
ODC1  FE FF          CP    NEG1      ;ADDR ENTRY ERROR?
ODC3  CA 0E0D        JP    Z,P%DN    ;YES, ALLOW USER ANOTHER TRY

ODC6  CD 10BD        CALL  POS$FIL
ODC9  FE FF          CP    NEG1      ;FILE POSITIONING ERROR?
ODCB  28 40          JR    Z,P%DN    ;YES, ALLOW USER ANOTHER TRY

*** LOOP UNTIL ALL IC'S ARE PROGRAMMED ***

ODCD  CD 10FF        P%NI: CALL  RD$FIL
ODDO  FE 00          CP    ZERO      ;WERE ANY RECORDS READ?
ODD2  20 31          JR    NZ,P%E8    ;NO, MUST BE READ ERROR

ODD4  0E 09          LD    C,PRTLN   ;** INSTRUCT USER **
ODD6  11 02B2        LD    DE,MSG6   ;** TO TURN ON **
ODD9  CD 0005        CALL  CDOS     ;** PROGRAMMER **

ODDC  0E 01          LD    C,RDCHR   ;**** WAIT UNTIL ****
ODDE  CD 0005        CALL  CDOS     ;**** DONE ****

ODE1  CD 1131        CALL  ERA$IC
ODE4  FE 00          CP    ZERO      ;WAS ERASE SUCCESSFUL?
ODE6  20 25          JR    NZ,P%DN   ;NO, GO CLOSE FILE AND GET OUT

ODE8  CD 11B4        P%C2: CALL  PROG$IC

ODEB  CD 120F        CALL  VER$IC
ODEE  FE FF          CP    NEG1      ;WERE THERE PROGRAMMING ERRORS?
ODF0  28 1B          JR    Z,P%DN    ;YES, GO CLOSE FILE AND GET OUT

```

Figure 15. EEPROM Programmer Software (page 8 of 37).

```

0DF2 ED 4B 0503 LD BC,(NXTADD) ;* * * * *
0DF6 51 LD D,C ;* LOAD DE WITH NXTADD *
0DF7 58 LD E,B ;* * * * *
0DF8 ED 4B 0509 LD BC,(LSTADD) ;* * * * *
0DFC 61 LD H,C ;* LOAD HL WITH LSTADD *
0DFD 68 LD L,B ;* * * * *
0DFE A7 AND A ;*** COMPUTE ***
0DFF ED 52 SBC HL,DE ;*** LSTADD - NXTADD ***
0E01 38 0A JR C,P$DN ;(0 IMPLIES DONE
0E03 18 C8 JR P$NI ;)=0 IMPLIES NOT DONE

;*** END LOOP ***

0E05 0E 09 P$E8: LD C,PRTLN ;* * * * *
0E07 11 04CA LD DE,ERR8 ;* PRINT DISK READ ERROR *
0E0A CD 0005 CALL CDOS ;* * * * *

0E0D 0E 10 P$DN: LD C,CLSFL ;* * * * *
0E0F 11 005C LD DE,FCB ;* CLOSE DISK FILE *
0E12 CD 0005 CALL CDOS ;* * * * *
0E15 C3 0D60 JP START ;ALLOW USER ANOTHER OPERATION

PAGE

```

Figure 15. EEPROM Programmer Software (page 9 of 37).

```

;***** VERIFY IC *****
0E18  CD 0F01      V$OPR: CALL  CRE8$FCB

0E1B  0E 0F        LD    C,OPNFL      ;* * * * *
0E1D  11 005C      LD    DE,FCB      ;* OPEN DISK FILE *
0E20  CD 0005      CALL  CDOS        ;* * * * *

0E23  FE FF        CP    NEG1        ;WAS OPEN SUCCESSFUL?
0E25  20 0B        JR    NZ,V$C1     ;YES
0E27  0E 09        LD    C,PRTLN     ;NO, * * * * *
0E29  11 0360      LD    DE,ERR1     ; * PRINT ERROR *
0E2C  CD 0005      CALL  CDOS        ; * * * * *
0E2F  C3 0D60      JP    START      ;ALLOW USER ANOTHER TRY

0E32  CD 0F6E      V$C1: CALL  SET$ADDR
0E35  FE FF        CP    NEG1        ;ADDR ENTRY ERROR?
0E37  CA 0E83      JP    Z,V$DN      ;YES, ALLOW USER ANOTHER TRY

0E3A  CD 108D      CALL  POS$FIL
0E3D  FE FF        CP    NEG1        ;FILE POSITIONING ERROR?
0E3F  28 42        JR    Z,V$DN      ;YES, ALLOW USER ANOTHER TRY

;*** LOOP UNTIL ALL IC'S ARE VERIFIED ***

0E41  CD 10FF      V$NI: CALL  RD$FIL
0E44  FE 00        CP    ZERO        ;WERE ANY RECORDS READ?
0E46  20 33        JR    NZ,V$E8     ;NO, MUST BE DISK READ ERROR

0E48  0E 09        LD    C,PRTLN     ;*** INSTRUCT USER ***
0E4A  11 02B2      LD    DE,MSG6     ;*** TO TURN ON ***
0E4D  CD 0005      CALL  CDOS        ;*** PROGRAMMER ***

0E50  0E 01        LD    C,RDCHR     ;**** WAIT UNTIL ****
0E52  CD 0005      CALL  CDOS        ;**** DONE ****

0E55  0E 02        LD    C,PRTCHR    ;* * * * *
0E57  1E 0D        LD    E,CR       ;* MOVE CURSOR *
0E59  CD 0005      CALL  CDOS        ;* TO NEXT *
0E5C  1E 0A        LD    E,LF       ;* LINE *
0E5E  CD 0005      CALL  CDOS        ;* * * * *

0E61  CD 120F      CALL  VER$IC
0E64  FE FF        CP    NEG1        ;WERE THERE PROGRAMMING ERRORS?
0E66  28 1B        JR    Z,V$DN      ;YES, GO CLOSE FILE AND GET OUT

```

Figure 15. EEPROM Programmer Software (page 10 of 37).

```

0E68 ED 4B 0503      LD BC,(NXTADD)      ;* * * * *
0E6C 51              LD D,C              ;* LOAD DE WITH NXTADD *
0E6D 58              LD E,B              ;* * * * *
0E6E ED 4B 0509      LD BC,(LSTADD)      ;* * * * *
0E72 61              LD H,C              ;* LOAD HL WITH LSTADD *
0E73 68              LD L,B              ;* * * * *
0E74 A7              AND A               ;*** COMPUTE ***
0E75 ED 52           SBC HL,DE           ;*** LSTADD - NXTADD ***
0E77 38 0A           JR C,V$DN          ;<0 IMPLIES DONE
0E79 18 C6           JR V$NI            ;>0 IMPLIES NOT DONE

;*** END LOOP ***

0E7B 0E 09           V$E8: LD C,PRTLN      ;* * * * *
0E7D 11 04CA         LD DE,ERR8          ;* PRINT DISK READ ERROR *
0E80 CD 0005         CALL CDOS           ;* * * * *

0E83 0E 10           V$DN: LD C,CLSFL     ;* * * * *
0E85 11 005C         LD DE,FCB          ;* CLOSE DISK FILE *
0E88 CD 0005         CALL CDOS          ;* * * * *
0E8B C3 0D60         JP START           ;ALLOW USER ANOTHER OPERATION

PAGE

```

Figure 15. EEPROM Programmer Software (page 11 of 37).

```

;***** DUMP IC *****

0E8E  CD 0F01      D$OPR: CALL  CRE8$FCB

0E91  0E 16              LD    C,CR8FL      ;* * * * *
0E93  11 005C          LD    DE,FCB        ;*  CREATE A DISK FILE  *
0E96  CD 0005          CALL  CDOS          ;* * * * *

0E99  FE FF              CP    NEG1          ;WAS CREATE SUCCESSFUL?
0E9B  20 0B              JR    NZ,D$RA      ;YES
0E9D  0E 09              LD    C,PRTLN      ;NO, * * * * *
0E9F  11 03A6          LD    DE,ERR3      ;  * PRINT ERROR *
0EA2  CD 0005          CALL  CDOS          ;  * * * * *
0EA5  C3 0D60          JP    START        ;ALLOW USER ANOTHER TRY

;*** LOOP UNTIL ALL IC'S ARE DUMPED ***

0EA8  0E 09      D$RA: LD    C,PRTLN      ;*** INSTRUCT USER ***
0EAA  11 02B2      LD    DE,MSG6        ;*** TO TURN ON ***
0EAD  CD 0005      CALL  CDOS          ;*** PROGRAMMER ***

0EB0  0E 01              LD    C,RDCHR      ;*** WAIT UNTIL ***
0EB2  CD 0005          CALL  CDOS          ;***  DONE  ***

0EB5  CD 1182          CALL  IC$RD

0EB8  3E 08              LD    A,BF        ;INIT * LOOP COUNTER
0EBA  21 0960          LD    HL,PROMBF    ;  * PROMBF PNTR

;*** LOOP UNTIL PROMBF IS WRITTEN ***

0EBD  11 0080      D$WA: LD    DE,CDOS$DB  ;SET * CDOS DISK BUFFER PNTR
0EC0  01 0080      LD    BC,RECSIZ      ;  * BLOCK MOVE COUNTER
0EC3  ED 80          LDIR

0EC5  F5              PUSH  AF          ;SAVE LOOP COUNTER
0EC6  0E 15              LD    C,WRFIL      ;* * * * *
0EC8  11 005C          LD    DE,FCB        ;*  WRITE A DISK RECORD *
0ECB  CD 0005          CALL  CDOS          ;* * * * *
0ECE  FE 00              CP    ZERO        ;WRITE COMPLETED OK?
0ED0  20 1B              JR    NZ,D$ERR5    ;NO
0ED2  F1              POP    AF          ;RESTORE LOOP COUNTER
0ED3  3D              DEC    A          ;END OF LOOP?
0ED4  20 E7              JR    NZ,D$WA      ;NO, WRITE ANOTHER RECORD

;*** END INNER LOOP ***

```

Figure 15. EEPROM Programmer Software (page 12 of 37).

```

0ED6 0E 09          D$C3: LD    C,PRTLN      ;* * * * *
0ED8 11 02F2        LD    DE,MSG7          ;* ASK FOR ANOTHER EEPROM *
0EDB CD 0005        CALL   CDOS            ;* * * * *

0EDE 0E 01          LD    C,RDCHR          ;*** AWAIT ***
0EE0 CD 0005        CALL   CDOS            ;*** RESPONSE? ***
0EE3 FE 59          CP     'Y'             ;MORE EEPROM'S?
0EE5 28 C1          JR     Z,D$RA          ;YES
0EE7 FE 4E          CP     'N'             ;INVALID INPUT?
0EE9 20 EB          JR     NZ,D$C3         ;YES
0EEB 18 09          JR     D$DN            ;NO, MUST BE DONE

;*** END OUTER LOOP ***

0EED F1             D$ERR5: POP    AF        ;CLEAR GARBAGE OFF STACK
0EEE 0E 09          LD    C,PRTLN          ;* * * * *
0EF0 11 0438        LD    DE,ERR5          ;* PRINT WRITE ERROR *
0EF3 CD 0005        CALL   CDOS            ;* * * * *

0EF6 0E 10          D$DN:  LD    C,CLSFL     ;* * * * *
0EF8 11 005C        LD    DE,FCB           ;* CLOSE DISK FILE *
0EFB CD 0005        CALL   CDOS            ;* * * * *
0EFE C3 0D60        JP     START          ;ALLOW USER ANOTHER OPERATION

```

PAGE

Figure 15. EEPROM Programmer Software (page 13 of 37).

```

;*****
;* THIS ROUTINE CREATES A FILE CONTROL BLOCK FOR THE FILE
;* REQUESTED BY THE USER THROUGH CONSOLE INPUT.
;*
;* INPUT:  N/A
;*
;* OUTPUT: FCB - CREATED FOR REQUESTED FILE NAME
;*
;*****

0F01      CRE8$FCB:
0F01      F5          PUSH    AF          ;SAVE REGS
0F02      C5          PUSH    BC
0F03      D5          PUSH    DE
0F04      E5          PUSH    HL

0F05      0E 09       LD      C,PRTLN    ;*****
0F07      11 01AD     LD      DE,MSG2    ;* PROMPT USER FOR FILENAME *
0F0A      CD 0005     CALL    CDOS       ;*****

0F0D      0E 0A       LD      C,RDLN     ;*****
0F0F      11 050E     LD      DE,CONBUF   ;* GET USER RESPONSE *
0F12      CD 0005     CALL    CDOS       ;*****

;***** SET DISK DRIVE IN FCB *****

0F15      3A 0511     LD      A,(CONBUF+3) ;GET SECOND CHAR OF USER RESPONSE
0F18      FE 3A       CP      COLON      ;DID USER SPECIFY DISK DRIVE?
0F1A      28 0B       JR      Z,CR8$SD    ;YES

0F1C      0E 19       LD      C,CURDK    ;NO, *** GET CURRENT ***
0F1E      CD 0005     CALL    CDOS       ; *** DISK DRIVE ***
0F21      3C          INC      A         ;CHANGE IT TO FCB FORMAT
0F22      32 005C     LD      (FCBDK),A  ;SET CURRENT DRIVE IN FCB
0F25      18 0B       JR      CR8$C1

0F27      3A 0510     CR8$SD: LD      A,(CONBUF+2) ;GET USER SPECIFIED DRIVE
0F2A      E6 03       AND      03H      ;CONVERT TO FCB FORMAT
0F2C      32 005C     LD      (FCBDK),A  ;SET FCB

0F2F      3E 20       CR8$C1: LD      A,BLANK ;*****
0F31      32 005D     LD      (FCBFN),A  ;* BLANK OUT *
0F34      01 000A     LD      BC,10     ;* FILE NAME *
0F37      11 005E     LD      DE,FCBFN+1 ;* AND EXTENT *
0F3A      21 005D     LD      HL,FCBFN  ;* IN THE FCB *
0F3D      ED B0       LDIR             ;*****

```

Figure 15. EEPROM Programmer Software (page 14 of 37).

```

;***** SET FILE NAME IN FCB *****

0F3F 21 0510      LD      HL,CONBUF+2    ;SET POINTER TO POSSIBLE FILE NAME
0F42 3A 0511      LD      A,(CONBUF+3)   ;GET SECOND CHAR OF USER RESPONSE
0F45 FE 3A        CP      COLON          ;DID USER SPECIFY DISK DRIVE?
0F47 20 02        JR      NZ,CR8$C2     ;NO, SO POINTER IS CORRECT
0F49 23           INC      HL            ;YES, *** BUMP POINTER PAST DISK ***
0F4A 23           INC      HL            ; *** DRIVE TO FILE NAME ***

0F4B 11 005D      CR8$C2: LD      DE,FCBFH    ;SET DESTINATION POINTER
0F4E 3E 2E        CR8$TA: LD      A,PERIOD    ;*** AT EXTENT ***
0F50 BE           CP      (HL)            ;*** MARKER? ***
0F51 28 08        JR      Z,CR8$FT       ;YES
0F53 AF          XOR      A              ;*** AT END OF ***
0F54 BE           CP      (HL)            ;*** USER INPUT? ***
0F55 28 0E        JR      Z,CR8$NR       ;YES, SO LEAVE EXTENT BLANK
0F57 ED A0        LDI      CR8$TA         ;NO, MOVE A CHAR TO FCB
0F59 18 F3        JR      CR8$TA         ;GO TRY ANOTHER CHAR MOVE

;***** SET FILE TYPE (EXTENSION) IN FCB *****

0F5B 23           CR8$FT: INC      HL        ;BUMP POINTER TO EXTENT NAME
0F5C 11 0065      LD      DE,FCBFT       ;SET DESTINATION POINTER
0F5F ED A0        LDI      *             ;* * * * *
0F61 ED A0        LDI      *             ;* MOVE EXTENT NAME TO FCB *
0F63 ED A0        LDI      *             ;* * * * *

0F65 AF          CR8$NR: XOR      A        ;*** INITIALIZE ***
0F66 32 007C      LD      (FCBNR),A      ;*** NEXT RECORD PNTR ***

0F69 E1           POP      HL            ;RESTORE REGS
0F6A D1           POP      DE
0F6B C1           POP      BC
0F6C F1           POP      AF
0F6D C9           RET

PAGE

```

Figure 15. EEPROM Programmer Software (page 15 of 37).

```

;*****
;*
;* THIS ROUTINE CONVERTS ADDRESSES INPUT THROUGH THE CONSOLE
;* FROM ASCII TO PURE BINARY, AND STORES THEM IN APPROPRIATE
;* SAVE AREAS.
;*
;* INPUT:  N/A
;*
;* OUTPUT: REG A =  0, IF ADDR'S ENTERED PROPERLY
;*           = -1, IF ADDR'S INVALID
;*           FLSTAD - ***  THESE  ***
;*           FSTADD - *** ADDRESSES ***
;*           LSTADD - ***   ARE   ***
;*           NXTADD - ***   SET   ***
;*
;*****

SET$ADDR:
0F6E          PUSH    BC           ;SAVE REGS
0F6E          C5
0F6F          PUSH    DE
0F70          E5
0F71          DD  E5
0F73          FD  E5

0F75          LD      C,PRTLN      ;*** PROMPT USER ***
0F77          LD      DE,MSG3      ;*** FOR FILE ***
0F7A          CALL    CDOS         ;*** STARTING ADDR ***

0F7D          LD      C,RDLN      ;*****
0F7F          LD      DE,CONBUF    ;* AWAIT RESPONSE *
0F82          CALL    CDOS         ;*****

0F85          LD      HL,FLSTAD    ;** SAVE RESPONSE **
0F88          CALL    AS$TO$BI     ;** IN FLSTAD **
0F8B          CP      NEG1         ;INVALID DIGITS INPUT?
0F8D          JP      Z,SET$RT     ;YES

0F90          CALL    SET$KB       ;CHECK ADDR FOR KILOBYTE BOUNDARY
0F93          CP      NEG1         ;INVALID ADDR?
0F95          JP      Z,SET$RT     ;YES

0F98          LD      C,PRTLN      ;*** PROMPT USER ***
0F9A          LD      DE,MSG4      ;*** FOR EEPROM ***
0F9D          CALL    CDOS         ;*** STARTING ADDR ***

```

Figure 15. EEPROM Programmer Software (page 16 of 37).

OFA0	0E 0A	LD	C,RDLN	;*****
OFA2	11 050E	LD	DE,CONBUF	;* AWAIT RESPONSE *
OFA5	CD 0005	CALL	CDOS	;*****
OFA8	21 0507	LD	HL,FSTADD	;** SAVE RESPONSE **
OFAB	CD 1056	CALL	AS%TO%BI	;** IN FSTADD **
OFAD	FE FF	CP	NEG1	;INVALID DIGITS INPUT?
OFB0	CA 1021	JP	Z,SET%RT	;YES
OFB3	FE 01	CP	ONE	;ANY DIGITS INPUT?
OFB5	20 12	JR	NZ,SET%FA	;NO
OFB7	DD 21 0505	LD	IX,FLSTAD	;** THE START ADDR IN THE FILE **
OFBB	FD 21 0507	LD	IY,FSTADD	;** (FLSTAD) MUST BE LESS THAN THE **
OFBF	CD 103F	CALL	SET%CMP	;** START ADDR OF THE PROM (FSTADD) **
OFC2	FE FF	CP	NEG1	;IS FLSTAD (= FSTADD)?
OFC4	CA 1019	JP	Z,SET%ER	;NO, ERROR
OFC7	18 08	JR	SET%C3	;YES
OFC9	ED 5B 0505	SET%FA: LD	DE,(FLSTAD)	;*** SET FSTADD ***
OFCD	ED 53 0507	LD	(FSTADD),DE	;*** EQUAL TO FLSTAD ***
OFD1	CD 1029	SET%C3: CALL	SET%KB	;CHECK ADDR FOR KILOBYTE BOUNDARY
OFD4	FE FF	CP	NEG1	;INVALID ADDR?
OFD6	CA 1021	JP	Z,SET%RT	;YES
OFD9	0E 09	LD	C,PRTLN	;*** PROMPT USER ***
OFDB	11 0286	LD	DE,MSG5	;*** FOR EEPROM ***
OFDE	CD 0005	CALL	CDOS	;*** ENDING ADDR ***
OFE1	0E 0A	LD	C,RDLN	;*****
OFE3	11 050E	LD	DE,CONBUF	;* AWAIT RESPONSE *
OFE6	CD 0005	CALL	CDOS	;*****
OFE9	21 0509	LD	HL,LSTADD	;** SAVE RESPONSE **
OFEC	CD 1056	CALL	AS%TO%BI	;** IN LSTADD **
OFEF	FE FF	CP	NEG1	;INVALID DIGIT INPUT?
OFF1	28 2E	JR	Z,SET%RT	;YES
OFF3	DD 21 0507	LD	IX,FSTADD	;** PROM START ADDR (FSTADD) **
OFF7	FD 21 0509	LD	IY,LSTADD	;** MUST BE LESS THAN THE **
OFFB	CD 103F	CALL	SET%CMP	;** PROM END ADDR (LSTADD) **
OFFE	FE FF	CP	NEG1	;IS FSTADD (= LSTADD)?
1000	CA 1019	JP	Z,SET%ER	;NO, ERROR
1003	ED 5B 0507	LD	DE,(FSTADD)	;** INIT NXTADD **
1007	ED 53 0503	LD	(NXTADD),DE	;** TO FSTADD **

Figure 15. EEPROM Programmer Software (page 17 of 37).

```

100B 0E 02          SET$C5: LD      C,PRTCHR      ;* * * * *
100D 1E 0D          LD      E,CR                ;* MOVE CONSOLE *
100F CD 0005        CALL    CDOS                ;* CURSOR TO  *
1012 1E 0A          LD      E,LF                ;* NEW LINE  *
1014 CD 0005        CALL    CDOS                ;* * * * *
1017 18 08          JR      SET$RT

1019 0E 09          SET$ER: LD      C,PRTLN      ;* * * * *
101B 11 0468        LD      DE,ERR6            ;* PRINT ADDR ERROR *
101E CD 0005        CALL    CDOS                ;* * * * *

1021 FD E1          SET$RT: POP     IY            ;RESTORE REGS
1023 DD E1          POP     IX
1025 E1             POP     HL
1026 D1             POP     DE
1027 C1             POP     BC
1028 C9             RET

;*** CHECK ADDR TO BE SURE IT ***
;*** IS ON A KILOBYTE BOUNDARY ***

1029                SET$KB:
1029 7E             LD      A,(HL)              ;LOAD MSB
102A E6 03          AND     03H                ;ARE BITS LESS THAN 1024 SET?
102C 20 06          JR      NZ,S$K$ER          ;YES
102E 23             INC     HL                  ;*** LOAD ***
102F 7E             LD      A,(HL)              ;*** LSB ***
1030 A7             AND     A                  ;ARE ANY LSB BITS SET
1031 20 01          JR      NZ,S$K$ER          ;YES
1033 C9             RET

1034 0E 09          S$K$ER: LD      C,PRTLN      ;* * * * *
1036 11 04A8        LD      DE,ERR7            ;* PRINT BOUNDARY ERROR *
1039 CD 0005        CALL    CDOS                ;* * * * *
103C 3E FF          LD      A,NEG1              ;SET BOUNDARY ERROR FLAG
103E C9             RET

```

Figure 15. EEPROM Programmer Software (page 18 of 37).

```

;*** CHECK RELATIVE MAGNITUDES OF ADDRESSES ***
;***** IX SHOULD POINT TO SMALLER VALUE *****

```

```

103F      SET$CMP:
103F      FD 7E 00      LD      A,(IY)      ;LOAD MSB
1042      DD BE 00      CP      (IX)      ;(IY) : (IX)
1045      38 0C      JR      C,S$C$ER      ;C, IMPLIES ERROR
1047      20 08      JR      NZ,S$C$OK      ;), MEANS LSB CAN BE IGNORED
1049      FD 7E 01      LD      A,(IY+1)    ;LOAD LSB
104C      DD BE 01      CP      (IX+1)    ;(IY+1) : (IX+1)
104F      38 02      JR      C,S$C$ER      ;C, IMPLIES ERROR
1051      AF      S$C$OK: XOR      A      ;SET (IX) (= (IY) FLAG
1052      C9      RET
1053      3E FF      S$C$ER: LD      A,NEG1    ;SET (IX) > (IY) FLAG
1055      C9      RET
PAGE

```

Figure 15. EEPROM Programmer Software (page 19 of 37).

```

;* *****
;*
;* THIS ROUTINE CONVERTS ASCII ADDRESSES INTO BINARY.
;*
;* INPUT:  HL - PNTR TO SAVE AREA FOR CONVERTED ADDR
;*         CONBUF - THE CONSOLE BUFFER CONTAINING ASCII
;*               TO BE CONVERTED
;*
;* OUTPUT: (HL) - WORD WITH BINARY ADDRESS
;*         REG A = 1, IF (SEMI)-VALID HEX INPUT BY USER
;*               = 0, IF NO HEX CHAR'S WERE INPUT
;*               = -1, IF INVALID INPUT BY USER
;*
;* *****

```

1056		AS%TO\$BI:		
1056	C5	PUSH	BC	;SAVE REGS
1057	E5	PUSH	HL	
1058	DD E5	PUSH	IX	
105A	FD E5	PUSH	IY	
105C	AF	XOR	A	;*****
105D	47	LD	B,A	;* CLEAR BC *
105E	4F	LD	C,A	;*****
105F	77	LD	(HL),A	*** ZERO THE SAVE AREA ***
1060	23	INC	HL	*** AND SET HL PNTR TO ***
1061	77	LD	(HL),A	*** LSB OF SAVE AREA ***
1062	FD 21 050F	LD	IY,CONBUF+1	;IY POINTS TO * OF CHAR IN CONBUF
1066	FD 4E 00	LD	C,(IY)	;SET BC TO * OF CHAR IN CONBUF
1069	B9	CP	C	;IS CONBUF EMPTY?
106A	28 37	JR	Z,A%B\$RT	;YES
106C	FD E5	PUSH	IY	;NO, *** SET IX PNTR ***
106E	DD E1	POP	IX	; *** TO LAST CHAR ***
1070	DD 09	ADD	IX,BC	; *** IN CONBUF ***
1072	FD 36 00 30	LD	(IY),'0'	;SET IN CASE ODD * OF CHAR IN CONBUF
1076	06 02	LD	B,2	;INIT LOOP COUNTER
1078	79	LD	A,C	*** DID USER RESPOND WITH ***
1079	FE 03	CP	3	*** LESS THAN 3 DIGITS? ***
107B	30 01	JR	NC,A%B\$C2	;NO
107D	05	DEC	B	;YES, SET LOOP COUNTER TO 1
107E	CD 10AA	A%B\$C2: CALL	A%B\$CONV	;CONVERT LS NIBBLE
1081	FE FF	CP	NEG1	;INVALID HEX INPUT?

Figure 15. EEPROM Programmer Software (page 20 of 37).

```

1083 28 16      JR      Z,A$B$ER      ;YES
1085 ED 67      RRD      ;NO, SAVE LS NIBBLE
1087 DD 2B      DEC      IX           ;BUMP THE ASCII PNTR

1089 CD 10AA    CALL     A$B$CONV     ;CONVERT MS NIBBLE
108C FE FF      CP       NEG1        ;INVALID HEX INPUT?
108E 28 08      JR      Z,A$B$ER      ;YES
1090 ED 67      RRD      ;SAVE MS NIBBLE
1092 DD 2B      DEC      IX           ;BUMP THE ASCII PNTR

1094 2B         DEC      HL           ;SET PNTR TO MSB OF ADDR SAVE AREA
1095 10 E7      DJNZ     A$B$C2       ;JUMP BACK IF NOT DONE
1097 3E 01      LD       A,ONE        ;SET INPUT OK FLAG
1099 18 08      JR      A$B$RT

109B 0E 09      A$B$ER: LD      C,PORTLN ;* * * * *
109D 11 04A8    LD       DE,ERR7      ;* PRINT INVALID ADDR ERROR *
10A0 CD 0005    CALL     CDOS         ;* * * * *

10A3 FD E1      A$B$RT: POP     IY      ;RESTORE REGS
10A5 DD E1      POP     IX
10A7 E1         POP     HL
10A8 C1         POP     BC
10A9 C9         RET

10AA           A$B$CONV:
10AA DD 7E 00    LD       A,(IX)      ;LOAD CHAR TO BE CONVERTED
10AD FE 47      CP       'F'+1        ;*** FILTER SOME ***
10AF 30 09      JR      NC,A$B$CE     ;*** BAD INPUTS ***
10B1 FE 3A      CP       '9'+1        ;* * * * *
10B3 38 02      JR      C,A$B$C5     ;* CONVERT ASCII *
10B5 D6 07      SUB      7           ;* TO HEXIDECIMAL *
10B7 E6 0F      A$B$C5: AND     0FH    ;* * * * *
10B9 C9         RET

10BA 3E FF      A$B$CE: LD      A,NEG1 ;SET INVALID DIGIT FLAG
10BC C9         RET
PAGE

```

Figure 15. EEPROM Programmer Software (page 21 of 37).

```

;*****
;* THIS ROUTINE POSITIONS A DISK FILE SO THAT THE FIRST RECORD
;* IN DSKBUF IS THE ONE TO BE PROGRAMMED INTO THE FIRST EEPROM.
;*
;* INPUT:  FLSTAD - START ADDR OF PROG ON DISK
;*         FSTADD - FIRST ADDR TO BE PROGRAMMED
;*
;* OUTPUT: DISK FILE IS POSITIONED SO THAT NEXT READ GETS
;*         PROPER RECORD.
;*         REG A = 0, IF FILE POSITIONED W/O ERRORS
;*              = -1, IF FILE POSITIONING ERROR
;*
;*****

```

```

10BD          POS$FIL:
10BD          C5          PUSH    BC          ;SAVE REGS
10BE          D5          PUSH    DE
10BF          E5          PUSH    HL
10C0          DD E5       PUSH    IX

10C2          3A 0505     LD      A,(FLSTAD)   ;INIT ** HL WITH **
10C5          67          LD      H,A          ; ** START **
10C6          3A 0506     LD      A,(FLSTAD+1) ; ** ADDR **
10C9          6F          LD      L,A          ; ** ON FILE **
10CA          DD 21 0507  LD      IX,FSTADD    ; * PNTR TO EEPROM FIRST ADDR

10CE          11 0080     POS$NR: LD      DE,RECSIZ ;*** ADD REC SIZE ***
10D1          19          ADD     HL,DE        ;*** TO FLSTAD ***

10D2          DD 7E 00     LD      A,(IX)      ;LOAD MSB
10D5          BC          CP      H          ;*** FLSTAD > FSTADD ***
10D6          38 20       JR      C,POS$DN     ;*** FSTADD IS W/IN NEXT RECORD ***
10D8          20 06       JR      NZ,POS$RD    ;FLSTAD = FSTADD ... LOAD LSB
10DA          DD 7E 01     LD      A,(IX+1)    ;LOAD LSB
10DD          BD          CP      L          ;*** FLSTAD > FSTADD ***
10DE          38 18       JR      C,POS$DN     ;*** FSTADD IS W/IN NEXT RECORD ***

10E0          0E 14       POS$RD: LD      C,RDFIL ;*****
10E2          11 005C     LD      DE,FCB      ;* READ A DISK RECORD *
10E5          CD 0005     CALL    CDOS        ;*****
10E8          FE 00       CP      ZERO        ;READ COMPLETE?
10EA          28 E2       JR      Z,POS$NR     ;YES, GO LOOK AT NEXT RECORD

10EC          0E 09       LD      C,PRTLN     ;NO, *****
10EE          11 04CA     LD      DE,ERR8     ; * PRINT ERROR OR UNEXPECTED EOF *
10F1          CD 0005     CALL    CDOS        ; *****

```

Figure 15. EEPROM Programmer Software (page 22 of 37).

10F4	3E FF	LD	A,NEG1	;SET ERROR FLAG
10F6	18 01	JR	POS\$RT	
10F8	AF	POS\$DN:	XOR A	;SET NO ERRORS FLAG
10F9	DD E1	POS\$RT:	POP IX	;RESTORE REGS
10FB	E1		POP HL	
10FC	D1		POP DE	
10FD	C1		POP BC	
10FE	C9		RET	
		PAGE		

Figure 15. EEPROM Programmer Software (page 23 of 37).

```

;*****
;*
;* THIS ROUTINE FILLS DSKBUF WITH DATA READ FROM A DISK FILE.
;* THE MAXIMUM SIZE BLOCK READ IS 1024 BYTES (THE SIZE OF THE
;* HNUM 3008 EEPROM).
;*
;* INPUT:  N/A
;*
;* OUTPUT: DSKBUF - FILLED WITH RECORDS JUST READ
;*          REG A = -1, IF NO RECORDS READ
;*          = 0, IF RECORDS READ OK
;*
;*****

10FF      RD$FIL:
10FF      C5          PUSH    BC          ;SAVE REGS
1100      D5          PUSH    DE
1101      E5          PUSH    HL

1102      06 08      LD      B,BF          ;INITIALIZE LOOP COUNTER
1104      11 0560    LD      DE,DSKBUF    ;INITIALIZE DSKBUF PNTR

;*** READ LOOP ***

110      D5          RD$RA: PUSH    DE          ;SAVE DSKBUF PNTR
1108      0E 14      LD      C,RD$FIL    ;*****
110A      11 005C    LD      DE,FCB      ;* READ A DISK RECORD *
110D      CD 0005    CALL    CDOS        ;*****
1110      D1          POP     DE          ;RESTORE DSKBUF PNTR

1111      FE 01      CP      ONE          ;EOF?
1113      28 0E      JR      Z,RD$EF      ;YES
1115      C5          PUSH    BC          ;SAVE LOOP COUNTER
1116      01 0080    LD      BC,RECSIZ   ;NO, *** MOVE DATA FROM ***
1119      21 0080    LD      HL,CDOS$DB  ; *** CDOS DISK BUFFER ***
111C      ED B0      LDIR           ; *** TO DSKBUF ***
111E      C1          POP     BC          ;RESTORE LOOP COUNTER

111F      10 E6      DJNZ    RD$RA        ;GO READ ANOTHER RECORD UNTIL BUF FULL
1121      18 09      JR      RD$C3

;*** END LOOP ***

1123      3E 08      RD$EF: LD      A,BF          ;*** DOES LOOP COUNTER INDICATE ***
1125      B0          CP      B          ;*** AT LEAST ONE RECORD READ? ***
1126      20 04      JR      NZ,RD$C3      ;YES

```

Figure 15. EEPROM Programmer Software (page 24 of 37).

1128	3E FF	LD	A,NEG1	;NO, SET REG A AS NO RECS READ
112A	18 01	JR	RD\$RT	
112C	AF	RD\$C3:	XOR A	;SET REG A AS RECS READ
112D	E1	RD\$RT:	POP HL	;RETORE REGS
112E	D1		POP DE	
112F	C1		POP BC	
1130	C9		RET	
		PAGE		

Figure 15. EEPROM Programmer Software (page 25 of 37).

```

;* *****
;*
;* THIS ROUTINE CLEARS A EEPROM TO ZEROS THROUGH THE FOLLOWING
;* SEQUENCE OF CONTROL LINE MANIPULATIONS:
;*
;*          CS = 0
;*          CE = 1
;*          OE = 1
;*          VDD = 0,
;* FOLLOWED BE OE BEING PULSED FROM 1 TO 0. AFTER THESE
;* MANIPULATIONS, THE EEPROM IS CHECKED TO BE SURE IT CONTAINS
;* ALL ZEROS.
;*
;* INPUT:  N/A
;*
;* OUTPUT: EEPROM IS CLEARED
;*          REG A = 0 - IF EEPROM ERASED
;*          = 1 - IF EEPROM NOT ERASED
;*
;* *****

```

1131		ERA\$IC:		
1131	C5	PUSH	BC	;SAVE REGS
1132	D5	PUSH	DE	
1133	E5	PUSH	HL	
1134	0E 09	LD	C,PRTLH	;*****
1136	11 033C	LD	DE,MSGERA	;* NOTIFY USER OF ERASE IN PROGRESS *
1139	CD 0005	CALL	CDOS	;*****
113C	3E 0E	LD	A,V\$5+D\$CE+D\$OE	*** INIT CONTROL LINES ***
113E	D3 24	OUT	(PROMD),A	*** BEFORE APPLYING 17V ***
1140	E6 F7	AND	V\$17	*** CLEAR VDD BIT, **
1142	D3 24	OUT	(PROMD),A	*** RESULT - VDD = 17V **
1144	CD 117B	CALL	V\$STABL	;WAIT FOR VOLTAGE TO STABLIZE
1147	E6 FD	AND	E\$OE	*** PULSE OE LOW, FORCING ***
1149	D3 24	OUT	(PROMD),A	*** THE ERASE TO BEGIN ***
				;OE PULSE WIDTH IS 100 MICRO-SECS
114B	06 1E	LD	B,30	*** THIS LOOP DELAYS **
114D	10 FE	DJNZ	\$	*** FOR 99 OUT OF **
114F	00	NOP		*** THE 100 REQUIRED **
1150	F6 02	OR	D\$OE	*** SET OE BACK ***
1152	D3 24	OUT	(PROMD),A	*** TO INACTIVE ***

Figure 15. EEPROM Programmer Software (page 26 of 37).

```

1154 F6 08 OR V$5 ;*** SET VDD BIT, ***
1156 D3 24 OUT (PROMD),A ;*** RESULT - VDD = 5V ***
1158 CD 117B CALL V$STABL ;WAIT FOR VOLTAGE TO STABLIZE

115B CD 1182 CALL IC$RD ;FILL THE EEPROM BUFFER

;*** VERIFY THAT IC WAS ERASED (ALL ZEROS) ***

115E 01 0400 LD BC,BF*RECSIZ ;INITIALIZE ** BUFFER LENGTH **
1161 21 0960 LD HL,PROMBF ; ** BUFFER POINTER **
1164 AF XOR A ; ** COMPARISON REG **
1165 ED A1 ERA$CP: CPI
1167 20 05 JR NZ,ERA$E2 ;JUMP IF BUFFER DID NOT CONTAIN ZERO
1169 EA 1165 JP PE,ERA$CP ;JUMP IF NOT END OF BUFFER
116C 18 09 JR ERA$RT ;RETURN TO CALLER WITH ERROR NOT SET

116E 0E 09 ERA$E2: LD C,PRTLN ;*** NOTIFY USER ***
1170 11 0381 LD DE,ERR2 ;*** THAT EEPROM ***
1173 CD 0005 CALL CDOS ;*** DID NOT ERASE ***
1176 3C INC A ;SET ERROR FLAG FOR RETURN TO CALLER

1177 E1 ERA$RT: POP HL ;RESTORE REGS
1178 D1 POP DE
1179 C1 POP BC
117A C9 RET

;*** THIS ROUTINE ESSENTIALLY A WAIT LOOP TO ***
;*** ALLOW THE 5V - 17V SWITCH TO STABLIZE. ***
;*** WAIT TIME IS APPROX 60 MICRO-SECONDS. ***

117B V$STABL:
117B C5 PUSH BC
117C 06 10 LD B,10H
117E 10 FE DJNZ $
1180 C1 POP BC
1181 C9 RET

```

PAGE

Figure 15. EEPROM Programmer Software (page 27 of 37).

```

;*****
;*
;* THIS ROUTINE READS THE CONTENTS OF A EEPROM INTO ITS
;* DEDICATED BUFFER AREA
;*
;* INPUT:  N/A
;*
;* OUTPUT: PROMBF - CONTAINS EEPROM IMAGE
;*
;*****

1182      IC$RD:
1182      F5          PUSH    AF          ;SAVE REGS
1183      C5          PUSH    BC
1184      D5          PUSH    DE
1185      E5          PUSH    HL

1186      3E 90      LD      A,CCW2      ;*** PROGRAM PORTS B, C - OUTPUT ***
1188      D3 23      OUT     (PCNTRL),A  ;*** A - INPUT ***

118A      3E 0D      LD      A,V$5+D$CE+E$CS ;*** VDD = 5V, DISABLE CE, ***
118C      D3 24      OUT     (PROMD),A   ;*** ENABLE OE & CS ***

118E      0E 20      LD      C,PROMA     ;INITIALIZE * PORT ADDR FOR INPUTS
1190      11 0000    LD      DE,0       ; * EEPROM ADDR
1193      21 0960    LD      HL,PROMBF   ; * EEPROM BUFFER PNTR

;*** LOOP UNTIL END OF EEPROM IS READ ***

1196      F5          IC$RA: PUSH    AF          ;SAVE CONTROL LINE STATUS
1197      7A          LD      A,D          ;*****
1198      D3 22      OUT     (PROMC),A     ;* SET EEPROM *
119A      7B          LD      A,E          ;* ADDR BUS *
119B      D3 21      OUT     (PROMB),A     ;*****
119D      F1          POP     AF          ;RESTORE CONTROL LINE STATUS

119E      E6 FB      AND     E$CE         ;*** ACTIVATE ***
11A0      D3 24      OUT     (PROMD),A     ;*** CHIP ENABLE ***

11A2      ED 40      IN      B,(C)        ;READ EEPROM DATA BUS

11A4      F6 04      OR      D$CE         ;*** DEACTIVATE ***
11A6      D3 24      OUT     (PROMD),A     ;*** CHIP ENABLE ***

11A8      70          LD      (HL),B      ;PUT BYTE JUST READ INTO PROMBF

```

Figure 15. EEPROM Programmer Software (page 28 of 37).

```

11A9 23          INC    HL          ;BUMP PROMBF PNTR
11AA 13          INC    DE          ;BUMP EEPROM ADDR
11AB CB 52       BIT    2,D         ;DID ADDR OVERFLOW INTO BIT 11?
                                   ;(IE, ADDR > 1K)
11AD 28 E7       JR     Z,IC$RA     ;NO, GO READ ANOTHER BYTE

                                   ;*** END LOOP ***

11AF E1          POP     HL          ;RESTORE REGS
11B0 D1          POP     DE
11B1 C1          POP     BC
11B2 F1          POP     AF
11B3 C9          RET

PAGE

```

Figure 15. EEPROM Programmer Software (page 29 of 37).

```

;*****
;*
;* THIS ROUTINE TRANSFERS DATA FROM THE DISK FILE BUFFER AREA
;* TO THE EEPROM
;*
;* INPUT:  DSKBUF - BUFFER CONTAINING DATA TO BE PROGRAMMED.
;*          NXTADD - NEXT ADDR TO BE PROGRAMMED, ASSUMED TO BE
;*                ON A KILOBYTE BOUNDARY.
;*          LSTADD - LAST ADDR TO BE PROGRAMMED.
;*
;* OUTPUT: EEPROM IS PROGRAMMED
;*
;*****

11B4      PROG$IC:
11B4      F5          PUSH    AF          ;SAVE REGS
11B5      C5          PUSH    BC
11B6      D5          PUSH    DE
11B7      E5          PUSH    HL

11B8      0E 09       LD      C,PRTLN    ;*****
11BA      11 0346     LD      DE,MSGPRG  ;* NOTIFY OF PROGRAMMING IN PROGRESS *
11BD      CD 0005     CALL   CDOS        ;*****

11C0      CD 12C1     CALL   INIT$BCT    ;INIT *****
11C3      60          LD      H,B        ; * EEPROM PROGRAM COUNTER *
11C4      69          LD      L,C        ; *****
11C5      ED 5B 0503  LD      DE,(NXTADD) ; *****
11C9      43          LD      B,E        ; ** NEXT **
11CA      4A          LD      C,D        ; ** ADDR **
11CB      C5          PUSH    BC        ; *****
11CC      11 0560     LD      DE,DSKBUF  ; * DSKBUF PNTR

11CF      3E 80       LD      A,CCW1     ;*** PROGRAM PORTS A, B, C ***
11D1      D3 23       OUT     (PCNTRL),A ;*** FOR LATCHED OUTPUT ***

11D3      3E 0E       LD      A,V$5+D$0E+D$CE ;*** VDD = 5V, ***
11D5      D3 24       OUT     (PROMD),A   ;*** DISABLE CS, 0E, CE ***

11D7      E6 F7       AND     V$17       ;** CLEAR VDD BIT, **
11D9      D3 24       OUT     (PROMD),A   ;** RESULT - VDD = 17V **
11DB      CD 117B     CALL   V$STABL     ;WAIT FOR VOLTAGE TO STABLIZE

```

Figure 15. EEPROM Programmer Software (page 30 of 37).

;*** LOOP UNTIL EEPROM IS PROGRAMMED ***

11DE	C1	PRG\$NB:	POP	BC	;RESTORE NXTADD
11DF	F5		PUSH	AF	;SAVE CONTROL LINE STATUS
11E0	78		LD	A,B	;*****
11E1	D3 22		OUT	(PRMC),A	;* SET EEPROM *
11E3	79		LD	A,C	;* ADDR LINES *
11E4	D3 21		OUT	(PRMB),A	;*****
11E6	1A		LD	A,(DE)	;*** SET EEPROM ***
11E7	D3 20		OUT	(PRMA),A	;*** DATA LINES ***
11E9	13		INC	DE	;BUMP DSKBUF PNTR
11EA	F1		POP	AF	;RESTORE CONTROL LINE STATUS
11EB	03		INC	BC	;** BUMP AND **
11EC	C5		PUSH	BC	;** STORE NXTADD **
11ED	E6 FB		AND	E\$CE	;*** ENABLE ***
11EF	D3 24		OUT	(PRMD),A	;*** CE ***
11F1	06 1E		LD	B,30	;CE PULSE WIDTH IS 100 MICRO-SECS
11F3	10 FE		DJNZ	\$	;** THIS LOOP DELAYS **
11F5	00		NOP		;** FOR 99 OUT OF THE **
					;** 100 REQUIRED **
11F6	F6 04		OR	D\$CE	;*** DISABLE ***
11F8	D3 24		OUT	(PRMD),A	;*** CE ***
11FA	01 0000		LD	BC,ZERO	;*****
11FD	37		SCF		;* DECREMENT EEPROM PROGRAM COUNTER *
11FE	ED 42		SBC	HL,BC	;*****
1200	20 DC		JR	NZ,PRG\$NB	;BRANCH BACK UNTIL DONE

;*** END LOOP ***

1202	F6 08		OR	V\$5	;*** SET VDD BIT, ***
1204	D3 24		OUT	(PRMD),A	;*** RESULT - VDD = 5V ***
1206	CD 117B		CALL	V\$STABL	;WAIT FOR VOLTAGE TO STABLIZE
1209	C1		POP	BC	;CLEAN UP STACK
120A	E1		POP	HL	;RESTORE REGS
120B	D1		POP	DE	
120C	C1		POP	BC	
120D	F1		POP	AF	
120E	C9		RET		

PAGE

Figure 15. EEPROM Programmer Software (page 31 of 37).

```

;*****
;*
;* THIS ROUTINE COMPARES THE DATA CONTAINED IN THE DISK FILE
;* AND EEPROM BUFFERS, MAKING SURE THEY ARE EQUAL
;*
;* INPUT:  DSKBUF - BUFFER FILLED WITH DATA TO BE COMPARED
;*          TO PROM BUFFER.
;*
;* OUTPUT: APPROPRIATE MESSAGES
;*          REG A = 0, IF NO ERRORS ENCOUNTERED
;*          = -1, IF COMPARISON ERRORS
;*
;*****
VER$IC:
120F      05          PUSH    BC          ;SAVE REGS
1210      D5          PUSH    DE
1211      E5          PUSH    HL

1212      0E 09       LD      C,PRTLN    ;*****
1214      11 0353     LD      DE,MSGVER  ;* NOTIFY OF VERIFY IN PROGRESS *
1217      CD 0005     CALL    CDS       ;*****

121A      CD 1182     CALL    IC$RD

121D      AF          XOR      A          ;INIT ** ERROR **
121E      32 050D     LD      (ERRCNT),A ; ** COUNTER **
1221      CD 12C1     CALL    INIT$BCT   ; * COMPARE COUNTER

1224      3A 0503     LD      A,(NXTADD) ;*****
1227      67          LD      H,A        ;*
1228      3A 0504     LD      A,(NXTADD+1);*
122B      6F          LD      L,A        ;*
122C      A7          AND      A          ;* BUMP NXTADD FOR *
122D      ED 4A       ADC      HL,BC     ;* NEXT EEPROM *
122F      7C          LD      A,H        ;*
1230      32 0503     LD      (NXTADD),A ;*
1233      7D          LD      A,L        ;*
1234      32 0504     LD      (NXTADD+1),A;*****

1237      11 095F     LD      DE,PROMBF-1;INITIALIZE * EEPROM BUFFER PNTR
123A      21 0560     LD      HL,DSKBUF  ; * DISK BUFFER PNTR

;*** LOOP UNTIL ENTIRE BLOCK IS VERIFIED ***

123D      13          VER$NB: INC    DE          ;BUMP PROMBF PNTR TO NEXT BYTE

```

Figure 15. EEPROM Programmer Software (page 32 of 37).

```

123E 1A          LD      A,(DE)          ;LOAD A BYTE FOR COMPARISON
123F ED A1       CPI          ;COMPARE EEPROM AND DISK BUFFERS
1241 C4 1269     CALL     NZ,VER$E4      ;BYTES NOT EQUAL - CALL ERROR ROUTINE
1244 3A 050D     LD      A,(ERRCNT)     ;*** HAVE WE PRINTED MAX ***
1247 FE 10       CP      MAXERR        ;*** NUMBER OF ERRORS? ***
1249 28 0E       JR      Z, VER$C2      ;YES, THATS ENOUGH, IGNORE REST OF BUF

124B AF          XOR      A             ;* * * * *
124C B8          CP      B             ;* IF BLOCK COUNTER *
124D 20 EE       JR      NZ,VER$NB      ;* NOT EQUAL 0 THEN *
124F B9          CP      C             ;* CONTINUE LOOP *
1250 20 EB       JR      NZ,VER$NB      ;* * * * *

;*** END LOOP ***

1252 3A 050D     LD      A,(ERRCNT)     ;*** WERE ANY ERRORS ***
1255 FE 00       CP      ZERO          ;*** ENCOUNTERED? ***
1257 28 04       JR      Z,VER$C3      ;NO
1259 3E FF       VER$C2: LD      A,NEG1  ;YES, SET ERROR FLAG
125B 18 08       JR      VER$RT

125D 0E 09       VER$C3: LD      C,PRTLH ;* * * * *
125F 11 0314     LD      DE,MSG8       ;* PRINT NO ERRORS FOUND *
1262 CD 0005     CALL     CDOS         ;* * * * *

1265 E1          VER$RT: POP      HL     ;RESTORE REGS
1266 D1          POP      DE
1267 C1          POP      BC
1268 C9          RET

;*** PRINT AN ERROR LINE ***

1269 F5          VER$E4: PUSH     AF     ;SAVE REGS
126A C5          PUSH     BC
126B D5          PUSH     DE
126C E5          PUSH     HL

;***** INITIALIZE TOP OF STACK *****
;*** FOR INTERFACE TO FOLLOWING ***
;***** PRINT ALGORITHM *****

126D D5          PUSH     DE
126E E5          PUSH     HL
126F A7          AND      A           ;RESET CPU CARRY FLAG
1270 01 0561     LD      BC,DSKBUF+1  ;* * * * *
1273 ED 42       SBC      HL,BC       ;* COMPUTE EEPROM ERROR ADDR *
1275 22 050B     LD      (ERRADD),HL  ;* ADDR = (ERRADD + 1) - DSKBUF - 1 *

```

Figure 15. EEPROM Programmer Software (page 33 of 37).

1278	21 050C	LD	HL,ERRADD+1	;* ADDR = HL - DSKBUF - 1 *
127B	E5	PUSH	HL	;* * * * *
127C	3A 050D	LD	A,(ERRCNT)	;*** HAVE THERE BEEN ***
127F	A7	AND	A	;*** ANY ERRORS YET? ***
1280	20 08	JR	NZ,VER\$C5	;YES, SO SKIP ERROR BANNER
1282	0E 09	LD	C,PRTLN	;* * * * *
1284	11 03D2	LD	DE,ERR4\$0	;* PRINT DIFFERENCE BANNER *
1287	CD 0005	CALL	CDOS	;* * * * *
128A	0E 09	VER\$C5: LD	C,PRTLN	;* * * * *
128C	11 041E	LD	DE,ERR4\$1	;* MOVE CURSOR *
128F	CD 0005	CALL	CDOS	;* * * * *
1292	E1	POP	HL	;* * * * *
1293	CD 12E5	CALL	PRT\$BYT	;* PRINT EEPROM *
1296	2B	DEC	HL	;* ERROR ADDR *
1297	CD 12E5	CALL	PRT\$BYT	;* * * * *
129A	11 0434	LD	DE,ERR4\$2	;*** PRINT ***
129D	CD 0005	CALL	CDOS	;*** SLASH ***
12A0	E1	POP	HL	;GET FILE PNTR
12A1	2B	DEC	HL	;ADJUST TO PROPER BYTE
12A2	CD 12E5	CALL	PRT\$BYT	;PRINT BYTE VALUE
12A5	CD 0005	CALL	CDOS	;*** PRINT A SLASH ***
12A8	E1	POP	HL	;*** PRINT PROM ***
12A9	CD 12E5	CALL	PRT\$BYT	;*** BYTE VALUE ***
12AC	0E 02	LD	C,PRTCHR	;* * * * *
12AE	1E 0D	LD	E,CR	;* MOVE CURSOR *
12B0	CD 0005	CALL	CDOS	;* TO NEXT *
12B3	1E 0A	LD	E,LF	;* LINE *
12B5	CD 0005	CALL	CDOS	;* * * * *
12B8	3C	INC	A	;*** BUMP ERROR ***
12B9	32 050D	LD	(ERRCNT),A	;*** COUNTER ***
12BC	E1	POP	HL	;RESTORE REGS
12BD	D1	POP	DE	
12BE	C1	POP	BC	
12BF	F1	POP	AF	
12C0	C9	RET		

PAGE

Figure 15. EEPROM Programmer Software (page 34 of 37).

```

;* *****
;*
;* THIS ROUTINE COMPUTES A BLOCK LENGTH THAT IS THE MAXIMUM OF
;* EITHER LSTADD - NXTADD + 1
;* OR
;* BF * RECSIZ (CURRENTLY 1024)
;*
;* INPUT:  NXTADD - NEXT EEPROM ADDR TO BE PROGRAMMED
;*          OR VERIFIED
;*          LSTADD - LAST EEPROM ADDR TO BE PROGRAMMED
;*          OR VERIFIED
;*
;* OUTPUT: BC PAIR - BLOCK LENGTH
;*
;* *****
INIT$BCT:
12C1          PUSH    HL          ;SAVE REGS
12C1    E5
12C2    3A 0509    LD      A,(LSTADD)  ;*****
12C5    67          LD      H,A        ;*
12C6    3A 050A    LD      A,(LSTADD+1) ;*
12C9    6F          LD      L,A        ;*      COMPUTE
12CA    3A 0503    LD      A,(NXTADD)  ;*      BLOCK COUNTER
12CD    47          LD      B,A        ;*
12CE    3A 0504    LD      A,(NXTADD+1) ;*
12D1    4F          LD      C,A        ;*      (IE, BC)
12D2    A7          AND      A        ;*      =
12D3    ED 42      SBC      HL,BC      ;* LSTADD - NXTADD + 1
12D5    23          INC      HL        ;*
12D6    44          LD      B,H        ;*
12D7    4D          LD      C,L        ;*****

12D8    A7          AND      A        ;*****
12D9    21 0400    LD      HL,BF*RECSIZ ;* IS COMPUTED BLOCK COUNTER > 1024?
12DC    ED 42      SBC      HL,BC      ;*****
12DE    30 03      JR      NC,INT$RT  ;NO, SO WE HAVE A SHORT BLOCK
12E0    01 0400    LD      BC,BF*RECSIZ ;YES,. SET BLOCK COUNTER TO MAX

12E3    E1          INT$RT: POP    HL          ;RESTORE REGS
12E4    C9          RET
PAGE

```

Figure 15. EEPROM Programmer Software (page 35 of 37).

```

;*****
;*
;* THIS ROUTINE CONVERTS A BYTE TO TWO ASCII CHARACTERS AND
;* PRINTS THEM ON THE CONSOLE.
;*
;* INPUT:  HL - POINTS TO BYTE TO BE PRINTED
;*
;* OUTPUT: TWO HEX DIGITS ARE PRINTED ON CONSOLE
;*
;*****

12E5      PRT$BYT:
12E5      F5          PUSH    AF          ;SAVE REGS
12E6      C5          PUSH    BC
12E7      D5          PUSH    DE
12E8      46          LD      B,(HL)      ;SAVE BYTE TO BE PRINTED

12E9      ED 6F       RLD              ;LOAD REG A WITH 1ST HEX DIGIT
12EB      CD 12F8     CALL    PRT$DIG    ;GO PRINT DIGIT
12EE      ED 6F       RLD              ;LOAD 2ND HEX DIGIT
12F0      CD 12F8     CALL    PRT$DIG    ;GO PRINT DIGIT

12F3      70          LD      (HL),B     ;RESTORE BYTE THAT WAS PRINTED
12F4      D1          POP     DE         ;RESTORE REGS
12F5      C1          POP     BC
12F6      F1          POP     AF
12F7      C9          RET

;*** CONVERT & PRINT A HEX DIGIT ***

12F8      PRT$DIG:
12F8      E6 0F       AND      0FH       ;GET RID OF HIGH ORDER GARBAGE
12FA      FE 0A       CP      0AH       ;IS HEX DIGIT = 0 - 9
12FC      38 06       JR      C,PRT$C5   ;YES
12FE      D6 09       SUB      9         ;NO, *** CONVERT TO ***
1300      F6 40       OR      040H      ; *** ASCII A - F ***
1302      18 02       JR      PRT$C6
1304      F6 30       PRT$C5: OR      030H      ;CONVERT TO ASCII 0 - 9

1306      0E 02       PRT$C6: LD      C,PRTCHR ;*****
1308      5F          LD      E,A        ;* PRINT THE DIGIT *
1309      CD 0005     CALL    CDOS      ;*****
130C      C9          RET

END      ENTRY$PT

```

Figure 15. EEPROM Programmer Software (page 36 of 37).

Macros:

Symbols:

A\$B\$C2	107E	A\$B\$C5	10B7	A\$B\$CE	10BA	A\$B\$CD	10AA
A\$B\$ER	109B	A\$B\$RT	10A3	AS\$TO\$	1056	BF	0008
BLANK	0020	CCW1	0080	CCW2	0090	CDOS	0005
CDOS\$D	0080	CIO	0001	CLSFL	0010	COLON	003A
CONBUF	050E	CR	000D	CR8\$C1	0F2F	CR8\$C2	0F4B
CR8\$FT	0F5B	CR8\$NR	0F65	CR8\$SD	0F27	CR8\$TA	0F4E
CR8\$FL	0016	CRE8\$F	0F01	CURDK	0019	D\$C3	0ED6
D\$CE	0004	D\$CS	00FE	D\$DN	0EF6	D\$ERR5	0EED
D\$OE	0002	D\$OPR	0E8E	D\$RA	0EA8	D\$WA	0EBD
DSKBUF	0560	E\$CE	00FB	E\$CS	0001	E\$OE	00FD
E\$OPR	0D90	ENTRY\$	0100	ERA\$CP	1165	ERA\$E2	116E
ERA\$IC	1131	ERA\$RT	1177	ERR1	0360	ERR2	0381
ERR3	03A6	ERR4\$0	03D2	ERR4\$1	041E	ERR4\$2	0434
ERR5	0438	ERR6	046B	ERR7	04A8	ERR8	04CA
ERRADD	050B	ERRCNT	050D	FCB	005C	FCBDK	005C
FCBEX	0068	FCBFN	005D	FCBFT	0065	FCBMP	006C
FCBNR	007C	FCBRC	006B	FLSTAD	0505	FSTADD	0507
GET\$OP	0D64	IC\$RA	1196	IC\$RD	1182	INIT\$B	12C1
INT\$RT	12E3	LF	000A	LSTADD	0509	MAXERR	0010
MSG1	014C	MSG2	01AD	MSG3	018B	MSG4	0259
MSG5	0286	MSG6	02B2	MSG7	02F2	MSG8	0314
MSGERA	033C	MSGPRG	0346	MSGVER	0353	NEG1	FFFF
NXTADD	0503	OLDSP	014A	ONE	0001	OPNFL	000F
P\$C1	0DBE	P\$C2	0DE8	P\$DN	0E0D	P\$E8	0E05
P\$NI	0DCD	P\$OPR	0DA3	PCNTRL	0023	PERIOD	002E
POS\$DN	10F8	POS\$FI	10BD	POS\$NR	10CE	POS\$RD	10E0
POS\$RT	10F9	PRG\$NB	11DE	PROG\$I	11B4	PROMA	0020
PROMB	0021	PROMBF	0960	PROMC	0022	PROMD	0024
PRT\$BY	12E5	PRT\$C5	1304	PRT\$C6	1306	PRT\$DI	12F8
PRTCHR	0002	PRTEND	0024	PRTLN	0009	RD\$C3	112C
RD\$EF	1123	RD\$FIL	10FF	RD\$RA	1107	RD\$RT	112D
RDCHR	0001	RDFIL	0014	RDLN	000A	RECSIZ	0080
S\$C\$ER	1033	S\$C\$OK	1051	S\$K\$ER	1034	SET\$AD	0F6E
SET\$C3	0FD1	SET\$C5	100B	SET\$CM	103F	SET\$ER	1019
SET\$FA	0FC9	SET\$KB	1029	SET\$RT	1021	SLASH	002F
STACK	014A	START	0D60	V\$17	00F7	V\$5	0008
V\$C1	0E32	V\$DN	0E83	V\$E8	0E7B	V\$NI	0E41
V\$OPR	0E18	V\$STAB	117B	VER\$C2	1259	VER\$C3	125D
VER\$C5	128A	VER\$E4	1269	VER\$IC	120F	VER\$NB	123D
VER\$RT	1265	WRFIL	0015	ZERO	0000		

No Fatal error(s)

Figure 15. EEPROM Programmer Software (page 37 of 37).

IV. User's Manual

The EEPROM Programmer described in the following manual is an S-100 based peripheral device used to support HNVM 3008 EEPROM's. The hardware and its associated software executes the following operations:

1. Erase,
2. Program,
3. Verify, and/or
4. Dump.

Operation of the Programmer is simple and requires only that the user be able to log onto the system and initiate execution of the program call EEPROM. Programmer software prompts the user for subsequent inputs.

The Programmer operates on only one EEPROM at a time. Jobs requiring more than one EEPROM are managed by software. At appropriate times, software prompts the user to remove old EEPROM's and insert new ones to continue operation. A result of this method of operation is that the length of the longest program that can be manipulated by the Programmer is essentially unlimited. However, another result is that ordering of EEPROM insertions is critical, since operations proceed from the low addresses to the high ones.

System Start-up

The following sequence describes how to get started with the EEPROM Programmer.

1. Insure EEPROM Programmer card is seated in the motherboard.
2. Flip the switch at bottom of zero insertion force socket to "OFF", disabling the Programmer.
3. Insure zero insertion force socket is empty.
4. Type "EEPROM" on console (ie, start program execution).
5. Console will prompt for additional information.

** NOTE **

There are two times when it is safe to insert/remove EEPROM's to or from the Programmer socket. One is when prompted by the console. Another is when software prompts the user to perform any operation. At these times the software disables the 20V power supply, thereby reducing the chances of destroying a EEPROM.

Commands

After initiating EEPROM Programmer software, the console prompts the user to enter an execution command with the following message:

WHAT OPERATION DO YOU WISH TO PERFORM?
E(R)ASE, (P)ROGRAM, (V)ERIFY, (D)UMP, OR E(X)IT

To execute any one of the five listed commands, the user must enter the letter contained within the parentheses of the desired operation.

Together the four commands - Erase, Program, Verify, and Dump - provide a flexible system for supporting HNVM 3008 EEPROM's. The Exit command is provided for easy return to the operating system. The following discussion describes the first four commands in more detail.

ERASE. This command is the simplest of the four, requiring only the EEPROM to be erased. Its execution destroys data held in a EEPROM by clearing all bits to zero.

PROGRAM. This command is used to program one or more EEPROM's. Programming of a EEPROM is accomplished by:

1. erasing EEPROM contents,
2. re-writing appropriate bytes, and
3. checking the new contents.

Programs to be dumped to EEPROM(s) must reside on a floppy disk file and be in the format of a .COM file. COM is the default file extension used by CDOS to indicate an executable program. For files longer than one kilobyte, the console instructs users to change EEPROM's as one is filled and others remain to be filled. Programming continues until either end-of-file is encountered, or the last user specified EEPROM address is written.

Note, when programming EEPROM's for use with the IFPDAS, the origin of software linked into a COM file should be hexadecimal address 0000H. This address corresponds to the physical beginning of the EEPROM address space within the IR. The last EEPROM address is 1FFFH.

To simplify recovery from errors encountered during programming, the Program command allows for calculating

relative starting points. That is, programming can begin at some point within a disk file instead of always starting at the first byte. Relative starting points are calculated from user responses to questions about the starting address of the program on the disk file, and from the first EEPROM address to be programmed.

This programming flexibility places several restrictions on Programmer software. The first one is that the starting address of programs on a disk file must be on a kilobyte boundary. In addition, the first EEPROM address to be programmed must also be on a kilobyte boundary. In effect this means that programming can not start in the middle of an EEPROM. Another restriction is that the first EEPROM address to be programmed must be greater than or equal to the beginning address of the program on the disk file. Finally, the last EEPROM address to be programmed must be greater than or equal to the first EEPROM address.

The following example illustrates the relative starting flexibility described above. A user wishes to program a series of EEPROM's from a disk file which starts at hexadecimal address 0000H and ends at 4082H. To accomplish this the user would initiate the Program command and respond to the console prompts as specified below:

FILENAME? filename

PLEASE ANSWER THE FOLLOWING QUESTIONS IN HEXIDECIMAL
NOTE: THE FIRST 2 ADDRESSES MUST BE ON KILOBYTE
BOUNDARIES

STARTING ADDRESS OF PROGRAM ON FILE? 0
FIRST ADDRESS TO BE PROGRAMMED/VERIFIED? (return)
LAST ADDRESS TO BE PROGRAMMED/VERIFIED? 4082 .

Note that a "RETURN" may be entered as the first address to be programmed when it is equal to the starting address of the program on file.

In this example programming proceeds error-free until software detects an error at address 4055H. At this point the first 16 EEPROM's were properly programmed. Therefore, programming can proceed from the 17th EEPROM (ie, from address 4000H). So, to continue programming at the 17th EEPROM the user would again perform a Program operation with responses to console prompts as specified below:

FILENAME? filename

PLEASE ANSWER THE FOLLOWING QUESTIONS IN HEXIDECIMAL
NOTE: THE FIRST 2 ADDRESSES MUST BE ON KILOBYTE
BOUNDARIES

STARTING ADDRESS OF PROGRAM ON FILE? 0
FIRST ADDRESS TO BE PROGRAMMED/VERIFIED? 4000
LAST ADDRESS TO BE PROGRAMMED/VERIFIED? 4082 .

Barring continued, unrecoverable errors, the 17 EEPROM's will contain the entire 4083 byte program; even though one was reprogrammed.

VERIFY. This command is used to compare a disk file to one or more EEPROM's. The result of the comparison is a report that either the EEPROM's match the file or they do not. If they don't match, up to 16 differences will be illustrated and verification will terminate.

As implied in the previous paragraph, the verification process may involve comparing more than one EEPROM against a

file which is larger than one kilobyte. In these cases software prompts the user to change EEPROM's at appropriate times. During this change, the user must be aware of the order in which the EEPROM's are inserted for verification. The order of comparison is from the EEPROM with the lowest physical address to the one with the highest. Logically this order corresponds with the direction in which the disk file is read.

As was the case with the program command, the verify command allows relative starting addresses. Relative addressing is accomplished in the same manner for both the verify and program commands, with appropriate subroutines being shared between them.

As an example of where the relative addressing facility would be used, consider the following scenario. The fourth and fifth EEPROM's within a six kilobyte program are suspected of being swapped. So, to find which is the fourth EEPROM, one is chosen and compared to the 0C00H to 0FFFH address space of the program contained on the disk file. Assuming that the program on the disk file starts at address 0, the user would initiate the Verify command and respond to console prompts as specified below:

FILENAME? filename

PLEASE ANSWER THE FOLLOWING QUESTIONS IN HEXIDECIMAL
NOTE: THE FIRST 2 ADDRESSES MUST BE ON KILOBYTE
BOUNDARIES

STARTING ADDRESS OF PROGRAM ON FILE? 0
FIRST ADDRESS TO BE PROGRAMMED/VERIFIED? C00
LAST ADDRESS TO BE PROGRAMMED/VERIFIED? FFF .

The result of these responses is that the chosen EEPROM is compared to the fourth kilobyte (decimal - 3072 to 4095) of the program on the disk file. If verification completes without errors then the chosen EEPROM is indeed the fourth of the sequence.

DUMP. This command is used to dump one or more EEPROM's to a disk file. In this way a floppy disk file can be created to contain the contents of a set of EEPROM's.

** NOTE **

Since the dump command creates a new disk file to hold EEPROM(s) data, a user response to the software request for a filename must be a unique file.

Usually programs are larger than one kilobyte and stretch across several EEPROM's. Still one file can be created to hold an entire program. The dump command allows for dumping sequences of EEPROM's by asking:

DO YOU HAVE MORE EEPROM'S? (Y/N)

after the dumping of each EEPROM is completed. To include another EEPROM in the dump sequence the user replies with a "Y". Software will instruct the user what to do next. Note also that the program sequence on the disk file is entirely determined by the order in which EEPROM's are inserted in the the Programmer socket. The first EEPROM appears first on the file.

Errors

EEPROM Programmer software recognizes many user errors. When an error occurs, a message is printed, execution of the current command ceases, and program control is returned to the Programmer command entry level. At the command entry level the user can retry the erroneous command, or try a different command. Error messages and some of their causes are:

FILE COULD NOT BE CREATED - disk directory is full or requested file already exists;

FILE NOT FOUND - the requested file name is not on the specified disk;

DISK RECORD COULD NOT BE WRITTEN - either the directory is full or no more file space is available;

DISK READ ERROR OR UNEXPECTED EOF - an attempt to read a disk record resulted in an error with no record transferred;

PROM DID NOT ERASE - the current EEPROM can not be cleared to zero's;

VERIFY ERROR - data contained on the disk file and in the current EEPROM do not match;

RELATIVE MAGNITUDE OF ADDRESS IS INVALID - beginning addresses must be less than ending addresses;

INVALID ADDRESS - addresses must be in hexadecimal; some addresses must also be on kilobyte boundaries.

Appendix C

MBM Interactive Development System

Contents

I.	Introduction	156
II.	S-100 Interface	157
III.	Software	161
	MIDS Software	162
	MBM Software	196
IV.	User's Manual	228
	System Start-up	228
	Command Summary	229
	Display Command Menu	230
	Initialize MBM Buffer	230
	Set Interrupt I/O Processing	230
	Set Polled I/O Processing	230
	Print MBM Buffer on Console	231
	Read BMC Address Register (and Print)	231
	Read FIFO (and Print)	231
	Print BMC Status	231
	Set BMC Register Values	231
	Print BMC Register Values	232
	Write FIFO	233
	Exit to CDOS	233
	Command Features	233
	MBM Initialization	234
	Interrupt Processing	234
	Errors	235

MBM Interactive Development System

I. Introduction

This Appendix documents operation of the MBM Interactive Development System (MIDS), designed to support Intel 7110 MBM's. Documentation consists of an Intel BPK-72 to S-100 hardware interface schematic, and MIDS software listings. Following the listings is a user's manual which describes the System's capabilities and summarizes its operating procedures. Before using MIDS, users must be familiar with MBM operating characteristics as outlined in the BPK-72 Bubble Memory Prototype Kit User's Manual (Ref 2).

MIDS is a flexible tool for supporting Intel 7110 MBM's. This flexibility results from two design considerations. One is that the hardware is based on the S-100 bus. Another is that software runs under control of the Cromemco Disk Operating System (CDOS) and consequently the Control Program for Microprocessors (CPM) Operating System. Further explanations of these design decisions are contained in following sections of this document.

II. S-100 Interface

Adaptation of the BPK-72 bus structure to the S-100 bus is illustrated in the schematic diagram of Figure 16. To facilitate understanding of the schematic, Table XIV lists the functions of the IC's used to construct the bus interface. More detailed information on individual IC's is available from The TTL Data Book (Ref 28).

The BPK-72 to S-100 interface is assembled on an S-100 wirewrap card. Attached to the card is a 44 pin connector for seating the BPK-72. This construction allows easy transportation of MBM hardware to any S-100 based system. Another aid to transportability is the full buffering of the interface circuitry to present only a single TTL load to the S-100 bus. Table XV lists the subset of S-100 pins required by the BPK-72.

Yet another aid to transportability is the onboard switch selection of the seven most significant bits of the MBM peripheral port addresses. This allows MBM hardware addresses to be chosen which do not interfere with the permanent I/O addresses of the host computer. These addresses are selected by opening and closing appropriate switches. Closed switches indicate zero bit settings, and open switches indicate ones. The most significant address

TABLE XIII
Selectable MBM I/O Ports

Port Address	Function
BBBB BBB0	Bi-directional Data Bus
BBBB BBB1	Command Port (output only)
BBBB BBB1	Status Port (input only)

bit corresponds to pin 1 of the IC socket that houses the address swithes.

Table XIII lists MBM port addresses and their related function. The least significant bit of the addresses is hardwired. The user selectable bits are denoted with B's. When setting these B's, be sure that corresponding changes are made to MIDS software.

TABLE XIV

BPK-72 to S-100 IC Listing

Device Type	Functional Designation	Schematic Reference
7402	Quad 2-input NOR Gates	U1
7404	Hex Inverters	U2
74244	Octal Buffers	U3
8216	4-bit Bidirectional Bus Driver	U4, U5
7485	4-bit Magnitude Comparator	U6, U7

TABLE XV

BPK-72 to S-100 Interface Definition

S-100 Pin	Signal Function	S-100 Pin	Signal Function
1	+8V	73	INT
2	+18V	75	RESET
25	CLK (4 MHz)	79	Addr 0
29	Addr 5	80	Addr 1
30	Addr 4	81	Addr 2
31	Addr 3	82	Addr 6
35	Data Out 1	83	Addr 7
36	Data Out 0	88	Data Out 2
38	Data Out 4	89	Data Out 3
39	Data Out 5	90	Data Out 7
40	Data Out 6	91	Data In 4
41	Data In 2	92	Data IN 5
42	Data In 3	93	Data In 6
43	Data In 7	94	Data In 1
45	OUT	95	Data In 0
46	INP	96	INTA
50	GND		

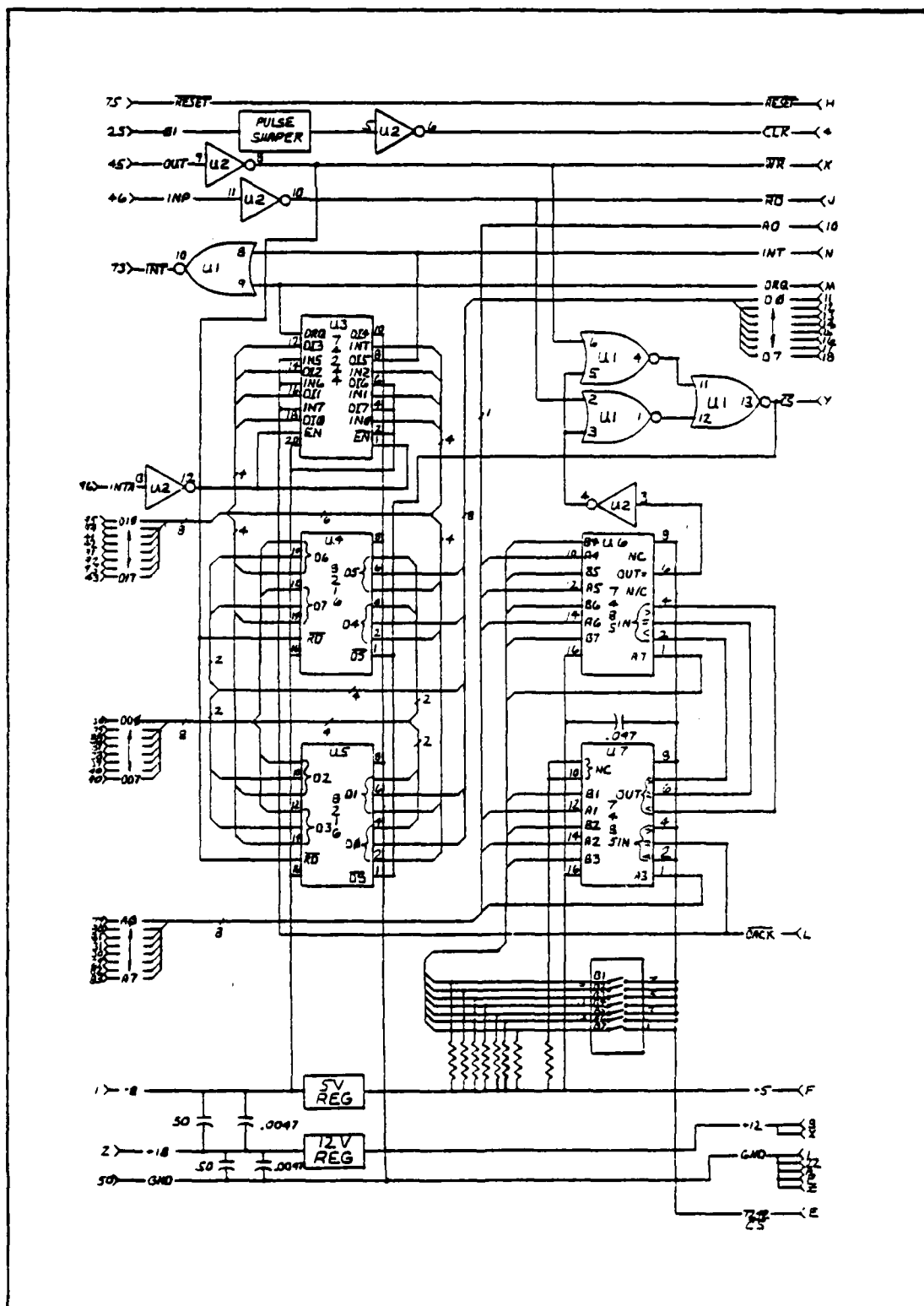


Figure 16. BPK-72 to S-100 Interface Schematic.

III. Software

The MIDS software listings are attached. The first listing, Figure 17, supports the interactive feature of MIDS by accepting and directing user requests for system operations. The second listing, Figure 18, is a subprogram containing MBM driver routines. Both programs were written in Z-80 assembler language with system calls to CDOS for I/O support.

Because CDOS system calls are restricted to those between 1 and 27, the software is transportable to CPM based systems without modification. This transport feature results from identical execution of the operating systems for calls in the range of 1 to 27.

0000'

```

.Z80
ASEG
ORG 100H
;TITLE: MIDS - MBM INTERACTIVE DEVELOPMENT SYSTEM
;AUTHOR: CAPT R E MEISNER
;DATE:
;SYSTEM: CROMEMCO Z2D / CDOS 2.36
;DESCRIPTION: THIS PROGRAM IS AN INTERACTIVE DEBUGGER FOR
;              THE INTEL 7110 MAGNETIC BUBBLE MEMORY (MBM).
;OPERATION: THIS PROGRAM CONTAINS THE INTERACTIVE ROUTINES
;            FOR COMMUNICATING WITH AN MBM DEBUGGER USER.
;            TO OPERATE IT MUST BE LINKED TO APPROPRIATE MBM
;            DRIVER ROUTINES. DRIVER ROUTINES ARE CONTAINED
;            IN MBM.REL, AN OBJECT FILE OF THE FOLLOWING
;            ROUTINES.
EXTRN MBM$STAT
EXTRN MBM$ICLR
EXTRN MBM$ISET
EXTRN MBM$WBRM
EXTRN MBM$INIT
EXTRN MBM$READ
EXTRN MBM$WRIT
EXTRN MBM$RSEK
EXTRN MBM$RXBR
EXTRN MBM$WXBR
EXTRN MBM$WZBL
EXTRN MBM$RFSB
EXTRN MBM$ABRT
EXTRN MBM$WSEK
EXTRN MBM$RZBL
EXTRN MBM$RCDT
EXTRN MBM$FFRE
EXTRN MBM$PURG
EXTRN MBM$SRES
;      IN ADDITION, MBM.REL CONTAINS THE FOLLOWING
;      COMMON VARIABLES:
EXTRN MBM$BMCR
EXTRN MBM$PSIZ
PAGE

```

Figure 17. MIDS Software (page 1 of 34).

```

;DICTIONARY: THE FOLLOWING IS A LIST OF SOME OF THE
;             ABBREVIATIONS USED IN THIS SOURCE LISTING.
;   BLR - BLOCK LENGTH REGISTER
;   BMC - BUBBLE MEMORY CONTROLLER
;   BUF - BUFFER
;   CHAR - CHARACTER
;   CNTR - COUNTER
;   ICD - INTERNALLY CORRECT DATA (MBM COMMAND)
;   INIT - INITIALIZE
;   LSB - LEAST SIGNIFICANT BIT/BYTE
;   MBM - MAGNETIC BUBBLE MEMORY
;   MSB - MOST SIGNIFICANT BIT/BYTE
;   NBR - NUMBER
;   PNTR - POINTER
;   RCD - READ CORRECTED DATA (MBM COMMAND)
;   REG - REGISTER
;   XFER - TRANSFER

```

```

;***** CONSTANTS *****

```

FFFF	NEG1	EQU	-1	
0000	ZERO	EQU	0	
0001	ONE	EQU	1	
0002	TWO	EQU	2	
0003	THREE	EQU	3	
0004	FOUR	EQU	4	
000D	CR	EQU	0DH	;ASCII CARRAIGE RETURN
000A	LF	EQU	0AH	;ASCII LINE FEED
0059	YES	EQU	'Y'	*** YES/NO RESPONSES ***
004E	NO	EQU	'N'	*** TO USER PROMPTS ***
0020	SPACE	EQU	' '	
0055	FILLER	EQU	55H	;FILL CHAR FOR FIFO BUF

```

;MBM REGISTER ADDRESS COUNTER (RAC) ASSIGNMENTS

```

000E	ADRO	EQU	0EH	;ADDRESS REG (LSB)
000F	ADRI	EQU	0FH	;ADDRESS REG (MSB)
0000	FIFO	EQU	00H	;FIFO I/O REG

```

;MBM CONTROLLER (BMC) STATUS REG BIT POSITIONS

```

0005	OPFBPS	EQU	5	;OP FAIL
0006	OPCBPS	EQU	6	;OP COMPLETE

Figure 17. MIDS Software (page 2 of 34).

```

0007          BSYBPS EQU 7          ;BMC BUSY

          ;MBM CONTROLLER (BMC) ENABLE REG BIT POSITIONS

0000          INBPS EQU 0          ;INTERRUPT ENABLE (NORMAL)
0001          IEBPS EQU 1          ;INTERRUPT ENABLE (ERROR)
0002          DMABPS EQU 2          ;DMA ENABLE
0003          XFRBPS EQU 3          ;MAX FSA TO BMC XFER RATE
0004          WBLBPS EQU 4          ;WRITE BOOTLOOP ENABLE
0005          RCDBPS EQU 5          ;ENABLE READ CORRECTED DATA
0006          ICDBPS EQU 6          ;ENABLE INTERNALLY CORRECTED DATA
0007          IPBPS EQU 7          ;INTERRUPT ENABLE (PARITY)

          ;MBM CONTROLLER ENABLE REG BIT SETTINGS

0020          RCDBIT EQU 20H        ;READ CORRECTED DATA
0040          ICDBIT EQU 40H        ;INTERNALLY CORRECT DATA

          ;OTHER MBM RELATED CONSTANTS

0028          BM$DATA EQU 28H        ;DATA I/O PORT
0029          BM$CND EQU 29H        ;COMMAND OUT PORT

          ;CDOS SYSTEM CALL PARAMETERS

0005          CDOS EQU 0005H        ;CDOS ENTRY POINT
0001          RDCHR EQU 1          ;READ A CHAR FROM THE CONSOLE
0002          PRCHR EQU 2          ;WRITE A CHAR TO THE CONSOLE
0009          PRTLN EQU 9          ;PRINT BUF LINE ON CONSOLE
000A          RDLN EQU 10          ;READ LINE FROM CONSOLE INTO BUF
0024          PRTEnd EQU '$'        ;END PRINT BUF PNTR

          ;CONSOLE MESSAGES

0100 0D 0A 24          CRLF: DB CR,LF,PRTEnd
0103 0D 0A 0A          PREMSG: DB CR,LF,LF
0106 20 20 20 57          DB ' WELCOME TO THE INTEL 7110 INTERACTIVE DEBUGGER'

```

Figure 17. MIDS Software (page 3 of 34).

0137	0D 0A 0A	DB	CR,LF,LF
013A	41 4E 59 54	DB	'ANYTIME YOU WISH TO SEE A COMMAND MENU, TYPE H (HELP)'
016F	0D 0A	DB	CR,LF
0171	20 20 20 20	DB	'
01A2	0D 0A 0A	DB	CR,LF,LF
01A5	54 48 45 20	DB	'THE SYSTEM IS CURRENTLY SET FOR POLLED I/O PROCESSING'
01DA	0D 0A 24	DB	CR,LF,PRTEEND
01DD	0D 0A 43 4F	PROMPT: DB	CR,LF,'COMMAND - ',PRTEEND
01EA	0D 0A 09 20	CMDERR: DB	CR,LF,' *** ERROR *** INVALID COMMAND',PRTEEND
020D	0D 0A 09 20	INPERK: DB	CR,LF,' *** ERROR *** INVALID INPUT',PRTEEND
022E	20 20 3C 3C	BSYWRN: DB	' (((BUSY))) ',PRTEEND
023F	20 20 3C 3C	OPCMPL: DB	' (((OP COMPLETE))) ',PRTEEND
0257	20 20 3C 3C	OPERR: DB	' (((OP FAIL))) CAUTION: BMC REGISTER',CR,LF
0282	09 09 09 20	DB	' VALUES MAY BE INVALID',PRTEEND
029C	20 20 53 54	STATHD: DB	' STATUS = ',PRTEEND
02A8	0D 0A 09 4E	BLXMSG: DB	CR,LF,' NUMBER OF PAGES PER I/O BLOCK = ',PRTEEND
02CC	0D 0A 0A 09	EN1MSG: DB	CR,LF,LF,' ENABLE NORMAL INTERRUPTS? '
02EA	28 59 2F 4E	DB	' (Y/N/Return) ',PRTEEND
02F8	0D 0A 09 20	EN2MSG: DB	CR,LF,' INTERRUPT ON ERRORS? (Y/N/Return) ',PRTEEND
0323	0D 0A 09 20	EN3MSG: DB	CR,LF,' MAXIMUM TRANSFER RATE? (Y/N/Return) ',PRTEEND
034E	0D 0A 09 20	EN5MSG: DB	CR,LF,' READ CORRECTED DATA? (Y/N/Return) ',PRTEEND
0379	0D 0A 09 20	EN6MSG: DB	CR,LF,' INTERNALLY CORRECT DATA? (Y/N/Return) ',PRTEEND
03A4	0D 0A 0A 09	RE1MSG: DB	CR,LF,LF,' (*(** (* REINITIALIZING MBM '
03C7	50 45 52 49	DB	' PERIPHERAL *)*)*)*)',CR,LF,LF,PRTEEND
03E0	0D 0A 0A 09	MSG4#1: DB	CR,LF,LF,' WHICH BUBBLE? ',PRTEEND
03FC	0D 0A 09 52	MSG4#2: DB	CR,LF,' RECORD NUMBER (3 HEX DIGITS)? ',PRTEEND
041E	20 20 49 4E	MSG1#1: DB	' INITIAL VALUE (1 HEX BYTE)? ',PRTEEND
043D	0D 0A 09 09	MSG1#2: DB	CR,LF,' INCREMENT (1 HEX BYTE)? ',PRTEEND
045B	20 20 42 4D	MSGQ: DB	' BMC ADDRESS REG = ',PRTEEND
0470	20 20 4E 4F	DESMG: DB	' NOTE: 1ST BYTE OF FIFO IS DESTROYED BY THIS '
049F	43 4F 4D 4D	DB	' COMMAND',CR,LF,PRTEEND
04A9	0D 0A 09 4F	STADDR: DB	CR,LF,' OPERATION STARTED AT MBM ADDRESS ',PRTEEND
04CE		BLROMSG:	
04CE	0D 0A 20 20	DB	CR,LF,' BLR LSB = ',PRTEEND
04E0		BLR1MSG:	
04E0	0D 0A 20 20	DB	CR,LF,' BLR MSB = ',PRTEEND
04F2		ENRMSG:	
04F2	0D 0A 20 20	DB	CR,LF,' ENABLE REG = ',PRTEEND
0504		ADROMSG:	
0504	0D 0A 20 20	DB	CR,LF,' ADR LSB = ',PRTEEND
0516		ADR1MSG:	
0516	0D 0A 20 20	DB	CR,LF,' ADR MSB = ',PRTEEND
0528	0D 0A	MENU: DB	CR,LF
052A	09 09 2A 2A	DB	' ***** MBM COMMAND MENU *****',CR,LF,LF
054B	30 20 2D 20	DB	' 0 - WRITE BOOTLOOP REGISTER MASKED '

Figure 17. MIDS Software (page 4 of 34).

0570	31 20 2D 20	DB	'1 - INITIALIZE',CR,LF
0580	32 20 2D 20	DB	'2 - READ BUBBLE
0597	33 20 2D 20	DB	'3 - WRITE BUBBLE',CR,LF
05A9	34 20 2D 20	DB	'4 - READ SEEK
05BE	35 20 2D 20	DB	'5 - READ BOOTLOOP REGISTER',CR,LF
05DA	36 20 2D 20	DB	'6 - WRITE BOOTLOOP REGISTER
05FB	37 20 2D 20	DB	'7 - WRITE BOOTLOOP',CR,LF
060F	38 20 2D 20	DB	'8 - READ FSA STATUS
0629	39 20 2D 20	DB	'9 - ABORT',CR,LF
0634	41 20 2D 20	DB	'A - WRITE SEEK
064A	42 20 2D 20	DB	'B - READ BOOTLOOP',CR,LF
065D	43 20 2D 20	DB	'C - READ CORRECTED DATA
067B	44 20 2D 20	DB	'D - RESET FIFO',CR,LF
068B	45 20 2D 20	DB	'E - MBM PURGE
06A0	46 20 2D 20	DB	'F - SOFTWARE RESET',CR,LF,LF
06B5	48 20 2D 20	DB	'H - DISPLAY COMMAND MENU
06D3	49 20 2D 20	DB	'I - INITIALIZE MBM BUFFER',CR,LF
06EE	4A 20 2D 20	DB	'J - SET INTERRUPT I/O PROCESSING
0713	4B 20 2D 20	DB	'K - SET POLLED I/O PROCESSING',CR,LF
0732	50 20 2D 20	DB	'P - PRINT MBM BUFFER ON CONSOLE
0757	51 20 2D 20	DB	'Q - READ BMC ADDR REG (AND PRINT)',CR,LF
077A	52 20 2D 20	DB	'R - READ FIFO (AND PRINT)
0799	53 20 2D 20	DB	'S - PRINT BMC STATUS',CR,LF
07AF	55 20 2D 20	DB	'U - SET BMC REG VALUES
07CC	56 20 2D 20	DB	'V - PRINT BMC REG VALUES',CR,LF
07E6	57 20 2D 20	DB	'W - WRITE FIFO
07FC	58 20 2D 20	DB	'X - EXIT TO CDOS',CR,LF,PRTEMD

;***** END CONSTANTS *****

;***** VARIABLES *****

080F	00	INTFLG: DB	0	;INTERRUPT ENABLED FLAG
00CC		BUFLEN EQU	204D	;LENGTH OF MBM I/O BUFFER
0810		MBMBUF: DS	204D	;MBM I/O BUFFER
08DC	50	CONBF: DB	80D	;BUFFER LENGTH
08DD	00		DB 0	;NBR OF CHAR READ
08DE			DS 80D	;CONSOLE INPUT BUFFER

;***** END VARIABLES *****

PAGE

Figure 17. MIDS Software (page 5 of 34).

```

092E  ED 73 0978      START: LD      (OLDSP),SP      ;SAVE OLD STACK PNTR
0932  31 0978          LD      SP,STACK          ;INIT NEW STACK
0935  C3 097A          JP      CONTINUE
0938                      DS      64              ;64 BYTE STACK
0978                      STACK EQU $              ;TOP OF STACK
0978  0000            OLDSP: DW      0              ;OLD STACK PNTR SAVE AREA

097A                      CONTINUE:
097A  0E 09            LD      C,PRTLN            ;* * * * *
097C  11 0103          LD      DE,PREMSG         ;* PRINT PREAMBLE *
097F  CD 0005          CALL    CDOS              ;* * * * *

0982                      GET$OPR:
0982  0E 09            LD      C,PRTLN            ;* * * * *
0984  11 01D0          LD      DE,PROMPT         ;* PROMPT USER WITH COMMAND - *
0987  CD 0005          CALL    CDOS              ;* * * * *

098A  0E 01            LD      C,RDCHR           ;*** GET USER ***
098C  CD 0005          CALL    CDOS              ;*** RESPONSE ***

098F  FE 30            CP      '0'
0991  CC 0A77          CALL    Z,OPR$0           ;WRITE BOOTLOOP REG MASKED
0994  CA 0982          JP      Z,GET$OPR

0997  FE 31            CP      '1'
0999  CC 0A85          CALL    Z,OPR$1           ;INITIALIZE
099C  CA 0982          JP      Z,GET$OPR

099F  FE 32            CP      '2'
09A1  CC 0A8E          CALL    Z,OPR$2           ;READ DATA
09A4  CA 0982          JP      Z,GET$OPR

09A7  FE 33            CP      '3'
09A9  CC 0A82          CALL    Z,OPR$3           ;WRITE DATA
09AC  CA 0982          JP      Z,GET$OPR

09AF  FE 34            CP      '4'
09B1  CC 0ACB          CALL    Z,OPR$4           ;READ SEEK
09B4  CA 0982          JP      Z,GET$OPR

09B7  FE 35            CP      '5'
09B9  CC 0ADB          CALL    Z,OPR$5           ;READ BOOTLOOP REG
09BC  CA 0982          JP      Z,GET$OPR

```

Figure 17. MIDS Software (page 6 of 34).

09BF	FE 36	CP	'6'	
09C1	CC 0AE9	CALL	Z,OPR\$6	;WRITE BOOTLOOP REG
09C4	CA 0982	JP	Z,GET\$OPR	
09C7	FE 37	CP	'7'	
09C9	CC 0AF7	CALL	Z,OPR\$7	;WRITE BOOTLOOP
09CC	CA 0982	JP	Z,GET\$OPR	
09CF	FE 38	CP	'8'	
09D1	CC 0B00	CALL	Z,OPR\$8	;READ FSA STATUS
09D4	CA 0982	JP	Z,GET\$OPR	
09D7	FE 39	CP	'9'	
09D9	CC 0B09	CALL	Z,OPR\$9	;ABORT
09DC	CA 0982	JP	Z,GET\$OPR	
09DF	FE 41	CP	'A'	
09E1	CC 0B12	CALL	Z,OPR\$A	;WRITE SEEK
09E4	CA 0982	JP	Z,GET\$OPR	
09E7	FE 42	CP	'B'	
09E9	CC 0B22	CALL	Z,OPR\$B	;READ BOOTLOOP
09EC	CA 0982	JP	Z,GET\$OPR	
09EF	FE 43	CP	'C'	
09F1	CC 0B2B	CALL	Z,OPR\$C	;READ CORRECTED DATA
09F4	CA 0982	JP	Z,GET\$OPR	
09F7	FE 44	CP	'D'	
09F9	CC 0B34	CALL	Z,OPR\$D	;RESET FIFO
09FC	CA 0982	JP	Z,GET\$OPR	
09FF	FE 45	CP	'E'	
0A01	CC 0B3D	CALL	Z,OPR\$E	;MBM PURGE
0A04	CA 0982	JP	Z,GET\$OPR	
0A07	FE 46	CP	'F'	
0A09	CC 0B46	CALL	Z,OPR\$F	;SOFTWARE RESET
0A0C	CA 0982	JP	Z,GET\$OPR	
0A0F	FE 48	CP	'H'	
0A11	CC 0B4F	CALL	Z,OPR\$H	;HELP
0A14	CA 0982	JP	Z,GET\$OPR	
0A17	FE 49	CP	'I'	
0A19	CC 0B5E	CALL	Z,OPR\$I	;INITIALIZE MBM BUFFER
0A1C	CA 0982	JP	Z,GET\$OPR	

Figure 17. MIDS Software (page 7 of 34).

0A1F	FE 4A	CP	'J'	
0A21	CC 0B00	CALL	Z,OPR\$J	;SET INTERRUPT I/O PROCESSING
0A24	CA 0982	JP	Z,GET\$OPR	
0A27	FE 4B	CP	'K'	
0A29	CC 0BDB	CALL	Z,OPR\$K	;SET POLLED I/O PROCESSING
0A2C	CA 0982	JP	Z,GET\$OPR	
0A2F	FE 50	CP	'P'	
0A31	CC 0BES	CALL	Z,OPR\$P	;PRINT MBM BUFFER
0A34	CA 0982	JP	Z,GET\$OPR	
0A37	FE 51	CP	'Q'	
0A39	CC 0C61	CALL	Z,OPR\$Q	;PRINT BMC ADDR REG
0A3C	CA 0982	JP	Z,GET\$OPR	
0A3F	FE 52	CP	'R'	
0A41	CC 0C87	CALL	Z,OPR\$R	;READ FIFO
0A44	CA 0982	JP	Z,GET\$OPR	
0A47	FE 53	CP	'S'	
0A49	CC 0D47	CALL	Z,PRT\$BMS	;PRINT BMC STATUS
0A4C	CA 0982	JP	Z,GET\$OPR	
0A4F	FE 55	CP	'U'	
0A51	CC 0CB4	CALL	Z,OPR\$U	;SET BMC REGS VALUES
0A54	CA 0982	JP	Z,GET\$OPR	
0A57	FE 56	CP	'V'	
0A59	CC 0CDF	CALL	Z,OPR\$V	;PRINT BMC REG VALUES
0A5C	CA 0982	JP	Z,GET\$OPR	
0A5F	FE 57	CP	'W'	
0A61	CC 0D2C	CALL	Z,OPR\$W	;WRITE FIFO
0A64	CA 0982	JP	Z,GET\$OPR	
0A67	FE 58	CP	'X'	
0A69	CA 0D40	JP	Z,OPR\$X	;EXIT
0A6C	0E 09	LD	C,PRTLN	;* * * * *
0A6E	11 01EA	LD	DE,CMDERR	;* INVALID COMMAND *
0A71	CD 0005	CALL	CDOS	;* * * * *
0A74	C3 0982	JP	GET\$OPR	

PAGE

Figure 17. MIDS Software (page 8 of 34).

```

;*** * * * *
;*** WRITE BOOTLOOP REG MASKED ***
;*** * * * *

0A77 F5 OPR$0: PUSH AF
0A78 E5        PUSH HL
0A79 21 0810   LD HL,MBMBUF ;SET PNTR TO MBM BUF
0A7C CD 0000*  CALL MBM$WBRM ;WRITE BOOTLOOP REG MASKED
0A7F CD 0D47   CALL PRT$BMS ;PRINT MBM STATUS
0A82 E1        POP HL
0A83 F1        POP AF
0A84 C9        RET

;*** * * * *
;*** INITIALIZE ***
;*** * * * *

0A85 F5 OPR$1: PUSH AF
0A86 CD 0000*  CALL MBM$INIT ;INITIALIZE
0A89 CD 0D47   CALL PRT$BMS ;PRINT MBM STATUS
0A8C F1        POP AF
0A8D C9        RET

;*** * * * *
;*** READ DATA ***
;*** * * * *

0A8E F5 OPR$2: PUSH AF
0A8F C5        PUSH BC
0A90 E5        PUSH HL

0A91 3E 55     LD A,FILLER ;INIT FILL CHAR
0A93 06 CC     LD B,BUFLEN ; LOOP CNTR
0A95 21 0810   LD HL,MBMBUF ; BUF PNTR
0A98 77        LD (HL),A ;* * * * *
0A99 23        INC HL ;* FILL THE BUF *
0A9A 10 FC     DJNZ $-2 ;* * * * *

0A9C ED 4B 0003* LD BC,(MBM$BMCR+3) ;GET STARTING PAGE NBR
0AA0 21 0810   LD HL,MBMBUF ;LOAD INPUT BUF PNTR
0AA3 CD 0000*  CALL MBM$READ ;READ MBM PAGE(S)
0AA6 CD 0D47   CALL PRT$BMS ;PRINT MBM STATUS

```

Figure 17. MIDS Software (page 9 of 34).

```

0AA9 60          LD    H,B          ;*** PRINT ***
0AAA 69          LD    L,C          ;*** READ ***
0AAB CD 0D84     CALL  PRTSAD       ;*** START ADDR ***

0AAE E1          POP    HL
0AAF C1          POP    BC
0AB0 F1          POP    AF
0AB1 C9          RET

```

```

;*** * * * * *
;*** WRITE DATA ***
;*** * * * * *

```

```

0AB2 F5          OPR$3: PUSH  AF
0AB3 C5          PUSH  BC
0AB4 E5          PUSH  HL

0AB5 ED 4B 0003* LD    BC,(MBM$BMCR+3) ;GET STARTING PAGE NBR
0AB9 21 0810     LD    HL,MBMBUF ;LOAD OUTPUT BUF PNTR
0ABC CD 0000*    CALL  MBM$WRIT ;WRITE A PAGE
0ABF CD 0D47     CALL  PRT$BMS ;PRINT MBM STATUS
0AC2 60          LD    H,B          ;*** PRINT ***
0AC3 69          LD    L,C          ;*** WRITE ***
0AC4 CD 0D84     CALL  PRTSAD       ;*** START ADDR ***

0AC7 E1          POP    HL
0AC8 C1          POP    BC
0AC9 F1          POP    AF
0ACA C9          RET

```

PAGE

Figure 17. MIDS Software (page 10 of 34).

```

;*** * * * *
;*** READ SEEK ***
;*** * * * *

OACB  F5          OPR$4:  PUSH    AF

OACC  CD 0ED7      CALL    SETADR    ;SET BMC ADDR REG VALUES
OACF  FE FF        CP      NEG1      ;INVALID INPUT?
OAD1  28 06        JR      Z,04$RT   ;YES
OAD3  CD 0000*     CALL    MBM$RSEK   ;GO SEEK
OAD6  CD 0D47      CALL    PRT$BMS    ;PRINT MBM STATUS

OAD9  F1          04$RT:  POP      AF
OADA  C9          RET

```

```

;*** * * * *
;*** READ BOOTLOOP REG ***
;*** * * * *

OADB  F5          OPR$5:  PUSH    AF
OADC  E5          PUSH    HL
OADD  Z1 0810      LD      HL,MBMBUF ;SET BUF PNTR FOR CALL
OAE0  CD 0000*     CALL    MBM$RXBR   ;READ BOOTLOOP REG
OAE3  CD 0D47      CALL    PRT$BMS    ;PRINT MBM STATUS
OAE6  E1          POP      HL
OAE7  F1          POP      AF
OAE8  C9          RET

```

```

;*** * * * *
;*** WRITE BOOTLOOP REG ***
;*** * * * *

OAE9  F5          OPR$6:  PUSH    AF
OAEA  E5          PUSH    HL
OAEB  Z1 0810      LD      HL,MBMBUF ;SET BUF PNTR FOR CALL
OAEF  CD 0000*     CALL    MBM$WXBR   ;WRITE BOOTLOOP REG
OAF1  CD 0D47      CALL    PRT$BMS    ;PRINT MBM STATUS
OAF4  E1          POP      HL
OAF5  F1          POP      AF
OAF6  C9          RET

```

PAGE

Figure 17. MIDS Software (page 11 of 34).

```

;*** * * * * *
;*** WRITE BOOTLOOP ***
;*** * * * * *

0AF7 F5 OPR$7: PUSH AF
0AF8 CD 0000* CALL MBM$WZBL ;WRITE BOOTLOOP
0AFB CD 0D47 CALL PRT$BMS ;PRINT MBM STATUS
0AFE F1 POP AF
0AFF C9 RET

;*** * * * * *
;*** READ FSA STATUS' ***
;*** * * * * *

0B00 F5 OPR$8: PUSH AF
0B01 CD 0000* CALL MBM$RFSA ;READ FSA STATUS'
0B04 CD 0D47 CALL PRT$BMS ;PRINT MBM STATUS
0B07 F1 POP AF
0B08 C9 RET

;*** * * * *
;*** ABORT ***
;*** * * * *

0B09 F5 OPR$9: PUSH AF
0B0A CD 0000* CALL MBM$ABRT ;ABORT CURRENT INSTRUCTION
0B0D CD 0D47 CALL PRT$BMS ;PRINT MBM STATUS
0B10 F1 POP AF
0B11 C9 RET

;*** * * * * *
;*** WRITE SEEK ***
;*** * * * * *

0B12 F5 OPR$A: PUSH AF
0B13 CD 0ED7 CALL SETADR ;SET BMC ADDR REG VALUES
0B14 FE FF CP NEG1 ;INVALID INPUT?
0B18 28 06 JR Z,A$RT ;YES
0B1A CD 0000* CALL MBM$WSEK ;GO SEEK

```

Figure 17. MIDS Software (page 12 of 34).

```

OB1D  CD 0D47          CALL  PRT$BMS          ;PRINT MBM STATUS

OB20  F1               A$RT: POP      AF
OB21  C9               RET

;*** * * * *
;*** READ BOOTLOOP ***
;*** * * * *

OB22  F5               OPR$B: PUSH    AF
OB23  CD 0000*         CALL    MBM$RZBL      ;READ BOOTLOOP
OB26  CD 0D47         CALL    PRT$BMS      ;PRINT MBM STATUS
OB29  F1               POP      AF
OB2A  C9               RET

;*** * * * *
;*** READ CORRECTED DATA ***
;*** * * * *

OB2B  F5               OPR$C: PUSH    AF
OB2C  CD 0000*         CALL    MBM$RCDT      ;READ CORRECTED DATA
OB2F  CD 0D47         CALL    PRT$BMS      ;PRINT MBM STATUS
OB32  F1               POP      AF
OB33  C9               RET

;*** * * * *
;*** FIFO RESET ***
;*** * * * *

OB34  F5               OPR$D: PUSH    AF
OB35  CD 0000*         CALL    MBM$FFRE      ;RESET BMC FIFO
OB38  CD 0D47         CALL    PRT$BMS      ;PRINT MBM STATUS
OB3B  F1               POP      AF
OB3C  C9               RET

```

Figure 17. MIDS Software (page 13 of 34).

```

;*** * * * ***
;*** PURGE ***
;*** * * * ***

OB3D  F5          OPR$E:  PUSH  AF

OB3E  CD 0000*    CALL  MBM$PURG      ;PURGE MBM SYSTEM
OB41  CD 0D47     CALL  PRT$BMS      ;PRINT MBM STATUS
OB44  F1          POP   AF
OB45  C9          RET

;*** * * * * * * * * *
;*** SOFTWARE RESET ***
;*** * * * * * * * * *

OB46  F5          OPR$F:  PUSH  AF
OB47  CD 0000*    CALL  MBM$SRES      ;RESET MBM SYSTEM
OB4A  CD 0D47     CALL  PRT$BMS      ;PRINT MBM STATUS
OB4D  F1          POP   AF
OB4E  C9          RET

;*** * * * ***
;*** HELP ***
;*** * * * ***

OB4F  F5          OPR$H:  PUSH  AF
OB50  C5          PUSH  BC
OB51  D5          PUSH  DE

OB52  0E 09       LD     C,PRTLN      ;* * * * *
OB54  11 0528     LD     DE,MENU      ;* PRINT COMMAND MENU *
OB57  CD 0005     CALL  CDOS          ;* * * * *

OB5A  D1          POP   DE
OB5B  C1          POP   BC
OB5C  F1          POP   AF
OB5D  C9          RET

PAGE

```

Figure 17. MIDS Software (page 14 of 34).

```

;*** * * * *
;*** INITIALIZE MBM BUFFER ***
;*** * * * *

OB5E    F5          OPR$I:  PUSH    AF
OB5F    C5          PUSH    BC
OB60    D5          PUSH    DE
OB61    E5          PUSH    HL

OB62    0E 09       LD      C,PRTLN      ;*** PROMPT USER ***
OB64    11 041E     LD      DE,MSGI$1    ;*** FOR INITIAL ***
OB67    CD 0005     CALL    CDOS        ;*** VALUE ***

OB6A    0E 0A       LD      C,RDLN      ;* * * * *
OB6C    11 08DC     LD      DE,CONBF     ;* AWAIT RESPONSE *
OB6F    CD 0005     CALL    CDOS        ;* * * * *

OB72    3A 08DD     LD      A,(CONBF+1)  ;GET NBR OF CHAR INPUT
OB75    FE 02       CP      TWO         ;WERE THERE 2?
OB77    20 33       JR      NZ,I$ERR     ;NO
OB79    CD 0BB9     CALL    CONV$2      ;CONVERT THE 2 CHAR TO BINARY
OB7C    32 0810     LD      (MBMBUF),A  ;SET 1ST BUFFER VALUE

OB7F    0E 09       LD      C,PRTLN      ;*** PROMPT USER ***
OB81    11 043D     LD      DE,MSGI$2    ;*** FOR INCREMENT ***
OB84    CD 0005     CALL    CDOS        ;*** VALUE ***

OB87    0E 0A       LD      C,RDLN      ;* * * * *
OB89    11 08DC     LD      DE,CONBF     ;* AWAIT RESPONSE *
OB8C    CD 0005     CALL    CDOS        ;* * * * *

OB8F    3A 08DD     LD      A,(CONBF+1)  ;GET NBR OF CHAR INPUT
OB92    FE 02       CP      TWO         ;WERE THERE 2?
OB94    20 16       JR      NZ,I$ERR     ;NO
OB96    CD 0BB9     CALL    CONV$2      ;CONVERT THE 2 CHAR TO BINARY
OB99    4F          LD      C,A         ;SAVE INCREMENT VALUE
OB9A    3A 0810     LD      A,(MBMBUF)  ;SET INITIAL VALUE
OB9D    06 CB       LD      B,BUFLEN-1  ; LOOP COUNTER/MOVE LENGTH
OB9F    21 0811     LD      HL,MBMBUF+1 ; MBM BUFFER PNTR

OBA2    81          I$LP5:  ADD      A,C  ;BUMP PREVIOUS MBMBUF BYTE
OBA3    77          LD      (HL),A      ;SAVE IN MBMBUF
OBA4    23          INC      HL         ;BUMP MBMBUF PNTR
OBA5    10 FB       DJNZ    I$LP5      ;LOOP UNTIL DONE

OBA7    CD 0BE5     CALL    OPR$P      ;PRINT FIFO BUFFER ON CONSOLE

```

Figure 17. MIDS Software (page 15 of 34).

```

OBAA 18 08                JR      I$RT

OBAC 0E 09                I$ERR: LD      C,PRTLN      ;* * * * *
OBAE 11 020D              LD      DE,INPERR      ;* PRINT INVALID INPUT MSG *
OBB1 CD 0005              CALL     CDOS          ;* * * * *

OBB4 E1                  I$RT:  POP      HL
OBB5 D1                  POP      DE
OBB6 C1                  POP      BC
OBB7 F1                  POP      AF
OBB8 C9                  RET

; *** CONVERT 2 ASCII BYTES TO BINARY ***

OBB9 3A 08DE              CONV$2: LD      A,(CONBF+2)      ;GET MOST SIGNIFICANT NIBBLE
OBBC CD 0F7D              CALL     A$B$CONV      ;CONVERT IT TO BINARY
OBBF 47                  LD      B,A          ;* * * * *
OBC0 CB 20              SLA      B          ;* SET MOST SIGNIFICANT *
OBC2 CB 20              SLA      B          ;* NIBBLE WHILE ZEROING *
OBC4 CB 20              SLA      B          ;* THE LEAST SIGNIFICANT *
OBC6 CB 20              SLA      B          ;* * * * *
OBC8 3A 08DF              LD      A,(CONBF+3)      ;GET LEAST SIGNIFICANT NIBBLE
OBCB CD 0F7D              CALL     A$B$CONV      ;CONVERT IT TO BINARY
OBCE B0                  OR      B          ;MERGE WITH MSN(IBBLE)
OBCF C9                  RET

;*** * * * *
;*** SET INTERRUPT I/O PROCESSING ***
;*** * * * *

OBD0 F5                  OPR$J: PUSH     AF
OBD1 AF                  XOR      A          ;*** SET ***
OBD2 3C                  INC      A          ;*** INTERRUPT ***
OBD3 32 080F              LD      (INTFLG),A      ;*** FLAG ***
OBD6 CD 0000*            CALL     MBM$ISET
OBD9 F1                  POP      AF
OBDA C9                  RET

;*** * * * *
;*** SET POLLED I/O PROCESSING ***
;*** * * * *

OBDB F5                  OPR$K: PUSH     AF
OBDC AF                  XOR      A          ;*** RESET INTERRUPT ***

```

Figure 17. MIDS Software (page 16 of 34).

```

OBDD    32 080F          LD    (INTFLG),A      ;***   FLAG       ***
OBE0    CD 0000*        CALL  MBM$ICLR
OBE3    F1              POP    AF
OBE4    C9              RET

```

```

;*** * * * * * * * * * * * * * * * * * * * *
;*** PRINT MBM BUFFER ON CONSOLE ***
;*** * * * * * * * * * * * * * * * * * * * *

```

```

OBE5    F5              OPR$P: PUSH  AF
OBE6    E5              PUSH  HL

OBE7    21 0810          LD    HL,MBMBUF      ;INIT BUF PNTR
OBEA    CD 0BF6          CALL  P$PG          ;*** PRINT 3 ***
OBED    CD 0BF6          CALL  P$PG          ;*** PAGES ***
OBF0    CD 0BF6          CALL  P$PG          ;*** OF DATA ***

OBF3    E1              POP    HL
OBF4    F1              POP    AF
OBF5    C9              RET

```

```

;      *** PRINT AN MBM PAGE ***

```

```

OBF6    F5              P$PG: PUSH  AF
OBF7    C5              PUSH  BC
OBF8    D5              PUSH  DE

OBF9    06 04              LD    B,4          ;SET NBR OF LINES TO BE PRINTED
OBFB    0E 09              P$C1: LD    C,PRTLN      ;* * * * *
OBFD    11 0100           LD    DE,CRLF        ;* SKIP TO NEXT LINE *
OC00    CD 0005           CALL  CDOS          ;* * * * *
OC03    CD 0C30           CALL  P$PLL        ;PRINT A LONG LINE
OC04    10 F3             DJNZ  P$C1          ;LOOP UNTIL DONE

OC08    3A 0002*          LD    A,(MBM$BMCR+2) ;*** ERROR CORRECTION ***
OC0B    E6 60             AND    RCDBIT+ICDBIT ;***   ENABLED?   ***
OC0D    20 0B             JR     NZ,P$C3      ;YES
OC0F    0E 09             LD    C,PRTLN      ;NO, * * * * *
OC11    11 0100           LD    DE,CRLF        ; * SKIP TO NEXT LINE *
OC14    CD 0005           CALL  CDOS          ; * * * * *
OC17    CD 0C26           CALL  P$PSL        ;PRINT A SHORT LINE

OC1A    0E 09              P$C3: LD    C,PRTLN      ;*** SKIP LINE ***
OC1C    11 0100           LD    DE,CRLF        ;*** BETWEEN ***
OC1F    CD 0005           CALL  CDOS          ;*** PAGES ***

```

Figure 17. MIDS Software (page 17 of 34).

```

0C22 D1 POP DE
0C23 C1 POP BC
0C24 F1 POP AF
0C25 C9 RET

; *** PRINT A LINE OF 4 OR 16 BYTES

0C26 F5 P$PSL: PUSH AF
0C27 C5 PUSH BC
0C28 D5 PUSH DE
0C29 CD 0C4D CALL P$BLK
0C2C D1 POP DE
0C2D C1 POP BC
0C2E F1 POP AF
0C2F C9 RET

0C30 F5 P$PLL: PUSH AF
0C31 C5 PUSH BC
0C32 D5 PUSH DE

0C33 CD 0C4D CALL P$BLK ;PRINT 4 BYTES
0C36 CD 0C4D CALL P$BLK ;PRINT 4 MORE

0C39 0E 02 LD C,PRTCHR ;* * * * *
0C3B 1E 20 LD E,SPACE ;* PRINT 8 *
0C3D CD 0005 CALL CDOS ;* BYTE SPACER *
0C40 CD 0005 CALL CDOS ;* * * * *

0C43 CD 0C4D CALL P$BLK ;PRINT 4 BYTES
0C46 CD 0C4D CALL P$BLK ;PRINT 4 MORE

0C49 D1 POP DE
0C4A C1 POP BC
0C4B F1 POP AF
0C4C C9 RET

0C4D 06 04 P$BLK: LD B,4 ;SET NBR OF BYTES IN BLOCK
0C4F 0E 02 P$C7: LD C,PRTCHR ;* * * * *
0C51 1E 20 LD E,SPACE ;* PRINT SPACER *
0C53 CD 0005 CALL CDOS ;* * * * *
0C56 7E LD A,(HL) ;*** PRINT ***
0C57 CD 0D9B CALL PRT$BYT ;*** A BYTE ***
0C5A 23 INC HL ;BUMP BUF PNTR
0C5B 10 F2 DJNZ P$C7 ;LOOP UNTIL DONE

0C5D CD 0005 CALL CDOS ;END OF BLOCK SPACER
0C60 C9 RET

```

Figure 17. MIDS Software (page 18 of 34).

AD-A118 072

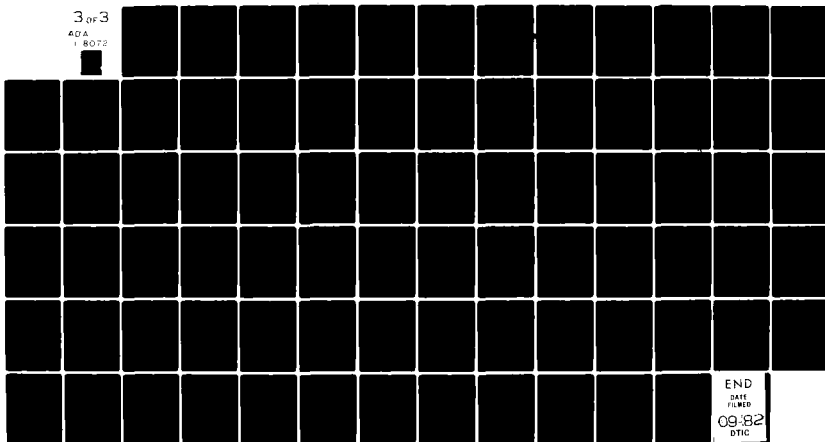
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/6 14/3
AN INFLIGHT RECORDER PROTOTYPE FOR THE INFLIGHT PHYSIOLOGICAL D--ETC(U)
FEB 82 R E MEISNER
AFIT/GCS/EE/82M-5

UNCLASSIFIED

NL

3 of 3

ADA
1-8072



END
DATE
FILMED
09-82
DTIC

```

;*** * * * *
;*** PRINT BMC ADDR REG ON CONSOLE ***
;*** * * * *

```

```

OC61  F5          OPR$Q:  PUSH  AF
OC62  C5          PUSH  BC
OC63  D5          PUSH  DE

OC64  0E 09          LD      C,PRTLN      ;* * * * *
OC66  11 045B        LD      DE,MSGQ      ;* BMC ADDRESS REG = *
OC69  CD 0005        CALL    CDOS         ;* * * * *

OC6C  3E 0E          LD      A,ADRO      ;*** SET BMC PNTR ***
OC6E  D3 29          OUT     (BM$CMD),A   ;*** TO ADDR REG ***

OC70  DB 28          IN      A,(BM$DATA)  ;READ ADDR REG LSB
OC72  47             LD      B,A          ;SAVE LSB FOR LATER PRINT
OC73  DB 28          IN      A,(BM$DATA)  ;READ ADDR REG MSB
OC75  CD 0D9B        CALL    PRT$BYT     ;PRINT MSB
OC78  0E 02          LD      C,PRTCHR    ;* * * * *
OC7A  1E 20          LD      E,SPACE     ;* PRINT A SPACE *
OC7C  CD 0005        CALL    CDOS         ;* * * * *
OC7F  78             LD      A,B          ;RESTORE ADDR LSB
OC80  CD 0D9B        CALL    PRT$BYT     ;PRINT LSB

OC83  D1             POP     DE
OC84  C1             POP     BC
OC85  F1             POP     AF
OC86  C9             RET

```

PAGE

Figure 17. MIDS Software (page 19 of 34).

```

;*** * * * *
;*** READ FIFO ***
;*** * * * *

OC87  F5      OPR$R:  PUSH  AF
OC88  C5      PUSH   BC
OC89  E5      PUSH   HL

OC8A  3E 55      LD    A,FILLER      ;INIT FILL CHAR
OC8C  06 28      LD    B,40D        ;    LOOP CNTR
OC8E  21 0810    LD    HL,MBMBUF    ;    BUF PNTR
OC91  77          LD    (HL),A      ;* * * * *
OC92  23          INC    HL          ;* FILL THE BUF *
OC93  10 FC      DJNZ  $-2          ;* * * * *

OC95  0E 09      LD    C,PRTLN      ;* * * * *
OC97  11 0470    LD    DE,DEMSG     ;* NOTE: 1ST CHAR OF FIFO ... *
OC9A  CD 0005    CALL  CDOS         ;* * * * *

OC9D  3E 00      LD    A,FIFO      ;*** SET BMC PNTR ***
OC9F  D3 29      OUT   (BM$CMD),A  ;*** TO FIFO ***
OCA1  21 0810    LD    HL,MBMBUF    ;INIT INPUT BUF PNTR
OCA4  06 28      LD    B,40D        ;    INPUT COUNT
OCA6  0E 28      LD    C,BM$DATA    ;    FIFO INPUT PORT
OCA8  ED B2      INIR              ;READ 40 BYTES FROM FIFO

OCAA  CD 0D47    CALL  PRT$BMS      ;PRINT BMC STATUS
OCAD  CD 0BE5    CALL  OPR$P        ;PRINT FIFO BUF

OCB0  E1          POP    HL
OCB1  C1          POP    BC
OCB2  F1          POP    AF
OCB3  C9          RET

```

PAGE

Figure 17. MIDS Software (page 20 of 34).

```

;*** * * * *
;*** SET THE BMC REGISTER VALUES ***
;*** * * * *

OCB4  F5          OPR$U:  PUSH  AF
OCB5  C5          PUSH  BC

OCB6  CD 0DC5      CALL  SETBLR      ;SET BLR REG VALUES
OCB9  FE FF        CP    NEG1        ;ERROR?
OCBB  28 1C        JR    Z,U$RT      ;YES
OCBD  CD 0E07      CALL  SETENR      ;SET ENABLE REG VALUE
OCC0  CD 0ED7      CALL  SETADR      ;SET ADDR REG VALUES

;*** COMPUTE BLOCK XFER SIZE ***
;*** BLKSIZ = PGSIZ * $PGS ***

OCC3  0E 40        LD    C,64D      ;INIT MBM PAGE SIZE
OCC5  3A 0002*     LD    A,(MBM$BMC+2) ;GET BMC ENABLE REG VALUE
OCC8  E6 60        AND    RCDBIT+ICDBIT ;HAS ERROR CORRECTION BEEN ENABLED?
OCCA  20 02        JR    NZ,U$C5     ;YES
OCCC  0E 44        LD    C,68D      ;NO, PAGE HAS 4 ADDITIONAL BYTES AVAIL
OCCE  3A 0000*     U$C5: LD    A,(MBM$BMC) ;*** GET NBR OF PAGES ***
OCD1  47          LD    B,A         ;*** TO BE XFERRED ***
OCD2  AF          XOR    A
OCD3  81          U$C6: ADD    A,C   ;ADD ONE MORE PAGE TO BLOCK SIZE
OCD4  10 FD        DJNZ   U$C6
OCD6  32 0000*     LD    (MBM$PSIZ),A ;SET PAGE SIZE TO MATCH BMCR TABLE

OCD9  CD 0CDF      U$RT:  CALL  OPR$V      ;DISPLAY RESULTS OF OPR$U

OCDC  C1          POP    BC
OCDD  F1          POP    AF
OCDE  C9          RET

PAGE

```

Figure 17. MIDS Software (page 21 of 34).

```

;*** * * * *
;*** PRINT BMC REGISTER VALUES ***
;*** * * * *

OCD F5      OPRV: PUSH AF
OCE0 C5      PUSH BC
OCE1 D5      PUSH DE

OCE2 0E 09   LD C,PRTLN      ;* * * * *
OCE4 11 04CE LD DE,BLRMSG    ;* BLR LSB = *
OCE7 CD 0005 CALL CDOS        ;* * * * *
OCEA 3A 0000* LD A,(MBM$BMCR) ;*** PRINT BLR ***
CED CD 0D9B  CALL PRT$BYT     ;*** LSB VALUE ***

OCF0 0E 09   LD C,PRTLN      ;* * * * *
OCF2 11 04E0 LD DE,BLR1MSG   ;* BLR MSB = *
OCF5 CD 0005 CALL CDOS        ;* * * * *
OCFB 3A 0001* LD A,(MBM$BMCR+1) ;*** PRINT BLR ***
OCFB CD 0D9B  CALL PRT$BYT     ;*** MSB VALUE ***

OCFE 0E 09   LD C,PRTLN      ;* * * * *
OD00 11 04F2 LD DE,ENRMSG    ;* ENABLE REG = *
OD03 CD 0005 CALL CDOS        ;* * * * *
OD06 3A 0002* LD A,(MBM$BMCR+2) ;*** PRINT ENR ***
OD09 CD 0D9B  CALL PRT$BYT     ;*** VALUE ***

OD0C 0E 09   LD C,PRTLN      ;* * * * *
OD0E 11 0504 LD DE,ADROMSG   ;* ADDR REG LSB = *
OD11 CD 0005 CALL CDOS        ;* * * * *
OD14 3A 0003* LD A,(MBM$BMCR+3) ;*** PRINT ADDR ***
OD17 CD 0D9B  CALL PRT$BYT     ;*** REG LSB VALUE ***

OD1A 0E 09   LD C,PRTLN      ;* * * * *
OD1C 11 0516 LD DE,ADR1MSG   ;* ADDR REG MSB = *
OD1F CD 0005 CALL CDOS        ;* * * * *
OD22 3A 0004* LD A,(MBM$BMCR+4) ;*** PRINT ADDR ***
OD25 CD 0D9B  CALL PRT$BYT     ;*** REG MSB VALUE ***

OD28 D1      POP DE
OD29 C1      POP BC
OD2A F1      POP AF
OD2B C9      RET

```

PAGE

Figure 17. MIDS Software (page 22 of 34).

```

;*** * * * *
;*** WRITE FIFO ***
;*** * * * *

0D2C  F5      OPR$W:  PUSH  AF
0D2D  C5      PUSH  BC
0D2E  E5      PUSH  HL

0D2F  3E 00      LD    A,FIFO      ;*** SET BMC PNTR ***
0D31  D3 29      OUT   (BM$CMD),A  ;*** TO FIFO ***
0D33  21 0810    LD    HL,MBMBUF  ;INIT INPUT BUF PNTR
0D36  06 28      LD    B,40D      ; INPUT COUNT
0D38  0E 28      LD    C,BM$DATA  ; FIFO INPUT PORT
0D3A  ED B3      OTIR             ;WRITE 40 BYTES TO FIFO

0D3C  E1      POP   HL
0D3D  C1      POP   BC
0D3E  F1      POP   AF
0D3F  C9      RET

;*** * * * *
;*** EXIT ***
;*** * * * *

0D40  ED 78 0978 OPR$X:  LD    SP,(OLDSP)  ;RESTORE OLD STACK
0D44  C3 0000    JP     0           ;RETURN TO CDOS
PAGE

```

Figure 17. MIDS Software (page 23 of 34).

```

;* *****
;*
;* THIS ROUTINE PRINTS THE MBM CONTROLLER STATUS.  IN
;* ADDITION, WARNINGS ARE PRINTED WHEN EITHER BUSY, OP
;* COMPLETE, OR OP FAIL STATUS IS SET.
;*
;* INPUT:  N/A
;*
;* OUTPUT:  APPROPRIATE CONSOLE MSG
;*
;* *****

```

0D47		PRT\$BMS:		
0D47	F5	PUSH	AF	
0D48	C5	PUSH	BC	
0D49	D5	PUSH	DE	
0D4A	CD 0000*	CALL	MBM\$STAT	;READ MBM STATUS
0D4D	CB 7F	BIT	BSYBPS,A	;BUSY BIT SET?
0D4F	28 0A	JR	Z,PRT\$C3	;NO
0D51	0E 09	LD	C,PRTLH	;*****
0D53	11 022E	LD	DE,BSYWRN	;* PRINT BUSY WARNING *
0D56	CD 0005	CALL	CDOS	;*****
0D59	18 1A	JR	PRT\$C7	;IGNORE OTHER BITS
0D5B	CB 77	PRT\$C3: BIT	OPCBPS,A	;OPERATION COMPLETE?
0D5D	28 0A	JR	Z,PRT\$C4	;NO
0D5F	0E 09	LD	C,PRTLH	;*****
0D61	11 023F	LD	DE,OPCMPL	;* PRINT OP COMPLETE MSG *
0D64	CD 0005	CALL	CDOS	;*****
0D67	18 0C	JR	PRT\$C7	;IGNORE OTHER BITS
0D69	CB 4F	PRT\$C4: BIT	OPFBPS,A	;OP FAIL SET?
0D6B	28 0A	JR	Z,PRT\$C7	;NO
0D6D	0E 09	LD	C,PRTLH	;*****
0D6F	11 0257	LD	DE,OPERR	;* PRINT OP FAIL WARNING *
0D72	CD 0005	CALL	CDOS	;*****
0D75	0E 09	PRT\$C7: LD	C,PRTLH	;*****
0D77	11 029C	LD	DE,STATHD	;* PRINT 'STATUS = ' *
0D7A	CD 0005	CALL	CDOS	;*****
0D7D	CD 0D9B	CALL	PRT\$BYT	
0D80	D1	POP	DE	
0D81	C1	POP	BC	

Figure 17. MIDS Software (page 24 of 34).

0D82	F1	POP	AF
0D83	C9	RET	

```

;*****
;*
;* THIS ROUTINE PRINTS THE ADDRESS CONTAINED IN THE HL REG
;* PAIR ON THE CONSOLE, ALONG WITH AN APPROPRIATE MESSAGE.
;*
;* INPUT:  HL - MBM ADDR TO BE PRINTED
;*
;* OUTPUT: MBM ADDR IS PRINTED ON CONSOLE
;*         HL - UNAFFECTED
;*
;*****

```

0D84		PRTSAD:		
0D84	F5	PUSH	AF	
0D85	C5	PUSH	BC	
0D86	D5	PUSH	DE	
0D87	0E 09	LD	C,PRTLN	;*****
0D89	11 04A9	LD	DE,STADDR	;* OPERATION STARTED AT ... *
0D8C	CD 0005	CALL	CDOS	;*****
0D8F	7C	LD	A,H	;*****
0D90	CD 0D9B	CALL	PRT\$BYT	;* PRINT *
0D93	7D	LD	A,L	;* MBM ADDR *
0D94	CD 0D9B	CALL	PRT\$BYT	;*****
0D97	D1	POP	DE	
0D98	C1	POP	BC	
0D99	F1	POP	AF	
0D9A	C9	RET		

PAGE

Figure 17. MIDS Software (page 25 of 34).

```

;*****
;*
;* THIS ROUTINE PRINTS THE HEX VALUE OF THE BYTE IN REG A
;*
;* INPUT:  A - BYTE TO BE PRINTED
;*
;* OUTPUT: DIGIT IS PRINTED ON CONSOLE
;*
;*****

OD9B      PRT$BYT:
OD9B      C5          PUSH    BC
OD9C      D5          PUSH    DE

OD9D      47          LD      B,A          ;SAVE BYTE
OD9E      CB 3F       SRL     A          ;*** SET-UP ***
ODA0      CB 3F       SRL     A          ;*** HIGH ***
ODA2      CB 3F       SRL     A          ;*** ORDER ***
ODA4      CB 3F       SRL     A          ;*** 4 BITS ***
ODA6      CD ODB2     CALL    PRT$DIG
ODA9      78          LD      A,B          ;RESTORE BYTE
ODAA      E6 0F       AND     0FH        ;SET-UP LOW ORDER 4 BITS
ODAC      CD ODB2     CALL    PRT$DIG

ODAF      D1          POP     DE
ODB0      C1          POP     BC
ODB1      C9          RET

;*** PRINT ONE DIGIT ***

ODB2      PRT$DIG:
ODB2      FE 0A       CP      0AH        ;IS HEX DIGIT = 0 - 9?
ODB4      30 04       JR      NC,DIG$C5  ;NO
ODB6      F6 30       OR      30H        ;CONVERT TO ASCII 0 - 9
ODB8      18 04       JR      DIG$PT
ODBA      D6 09       DIG$C5: SUB     9          ;*** CONVERT TO ***
ODBC      F6 40       OR      40H        ;*** ASCII A - F ***
ODBE      0E 02       DIG$PT: LD      C,PRTCHR ;*****
ODC0      5F          LD      E,A          ;* PRINT THE DIGIT *
ODC1      CD 0005     CALL    CDOS        ;*****
ODC4      C9          RET

PAGE

```

Figure 17. MIDS Software (page 26 of 34).

```

;*****
;*
;* THIS ROUTINE SETS THE BMC ADDR REG VALUES LOCATED WITHIN
;* THE MBM DRIVER MODULE.
;*
;* INPUT:  N/A
;*
;* OUTPUT:  BMC BLOCK LENGTH REG VALUES ARE SET
;*          A = -1, IF INVALID INPUT BY USER
;*          X'??' = UNDETERMINED, IF (SEMI-) VALID
;*              INPUT BY USER
;*
;*****

ODC5      SETBLR:
ODC5      C5          PUSH    BC
ODC6      D5          PUSH    DE

ODC7      0E 09      LD      C,PRTLN      ;*** PROMPT USER ***
ODC9      11 02A8    LD      DE,BLKMSG    ;*** FOR NBR OF PAGES ***
ODCC      CD 0005    CALL    CDOS         ;*** PER I/O BLOCK ***

ODCF      0E 0A      LD      C,RDLN      ;*****
ODD1      11 08DC    LD      DE,CONBF     ;* AWAIT RESPONSE *
ODD4      CD 0005    CALL    CDOS         ;*****

ODD7      3A 08DD    LD      A,(CONBF+1)  ;GET NBR OF CHAR READ
ODDA      FE 00      CP      ZERO        ;WAS IT 0? (IMPLIES CARRIAGE RETURN)
ODDC      28 26      JR      Z,SB$RT      ;YES, DO NOT CHANGE BLR VALUES
ODDE      FE 01      CP      ONE         ;WAS IT 1?
ODE0      20 18      JR      NZ,SB$ERR    ;NO

ODE2      3A 08DE    LD      A,(CONBF+2)  ;*** CONVERT CHAR JUST ***
ODE5      CD 0F7D    CALL    A$B$CONV     ;*** READ TO BINARY ***
ODE8      FE FF      CP      NEG1        ;INVALID INPUT?
ODEA      28 0E      JR      Z,SB$ERR     ;YES
ODEC      FE 04      CP      FOUR        ;INPUT <= 3 (CURRENT S/W LIMIT)
ODEE      30 0A      JR      NC,SB$ERR    ;NO

ODF0      32 0000*   LD      (MBM$BMCR),A  ;SET BLR LSB
ODF3      3E 10      LD      A,10H       ;*** SET BLR MSB FOR ***
ODF5      32 0001*   LD      (MBM$BMCR+1),A ;*** 1 BUBBLE XFER ***
ODF8      18 0A      JR      SB$RT

ODFA      0E 09      SB$ERR: LD      C,PRTLN      ;*****
ODFC      11 020D    LD      DE,INPERR    ;* INPUT ERROR *

```

Figure 17. MIDS Software (page 27 of 34).

```

0DFF  CD 0005      CALL  CDOS      ;* * * * *
0E02  3E FF      LD    A,NEG1    ;SET ERROR FLAG

0E04  D1          SB$RT: POP    DE
0E05  C1          POP    BC
0E06  C9          RET

```

```

;* * * * *
;*
;* THIS ROUTINE SETS THE BMC ENABLE REG VALUES LOCATED WITHIN
;* THE MBM DRIVER MODULE.
;*
;* INPUT:  N/A
;*
;* OUTPUT: BMC ENABLE REG VALUES ARE SET
;*
;* * * * *

```

```

0E07  SETENR:
0E07  F5          PUSH    AF
0E08  C5          PUSH    BC
0E09  D5          PUSH    DE
0E0A  E5          PUSH    HL
0E0B  06 00      LD      B,ZERO    ;MBM REINIT NOT REQUIRED FLAG

0E0D  3A 0002*   LD      A,(MBM$BMCR+2) ;*** GET INITIAL ***
0E10  67          LD      H,A      ;*** ENABLE REG VALUE ***
0E11  3A 080F   LD      A,(INTFLG) ;*** INTERRUPT ***
0E14  A7          AND     A        ;*** I/O ENABLED? ***
0E15  CA 0E52   JP      Z,SE$C3    ;NO

0E18  0E 09      SE$C0: LD      C,PRTLN ;* * * * *
0E1A  11 02CC   LD      DE,EN1MSG ;* NORMAL INTERRUPTS? *
0E1D  CD 0005   CALL    CDOS      ;* * * * *
0E20  0E 01      LD      C,RDCHR   ;*** GET USER ***
0E22  CD 0005   CALL    CDOS      ;*** RESPONSE ***

0E25  FE 00      CP      CR        ;USE OLD SETTING?
0E27  28 0C      JR      Z,SE$C1    ;YES
0E29  CB 84      RES     INBPS,H    ;CLEAR NORMAL INT
0E2B  FE 4E      CP      NO        ;DISABLE NORMAL INT?
0E2D  28 06      JR      Z,SE$C1    ;YES
0E2F  FE 59      CP      YES       ;ENABLE NORMAL INT?
0E31  20 E5      JR      NZ,SE$C0   ;NOT SURE, TRY AGAIN

```

Figure 17. MIDS Software (page 28 of 34).

0E33	CB C4	SET	INBPS,H	;SET NORMAL INT'S
0E35	0E 09	SE%C1:	LD C,PRTLN	;*****
0E37	11 02F8		LD DE,EN2MSG	;* ERROR INTERRUPTS? *
0E3A	CD 0005		CALL CDOS	;*****
0E3D	0E 01		LD C,RDCHR	;*** GET USER ***
0E3F	CD 0005		CALL CDOS	;*** RESPONSE ***
0E42	FE 0D	CP	CR	;USE OLD SETTING?
0E44	28 0C	JR	Z,SE%C3	;YES
0E46	CB 8C	RES	IEBPS,H	;CLEAR ERROR INT
0E48	FE 4E	CP	NO	;DISABLE ERROR INT?
0E4A	28 06	JR	Z,SE%C3	;YES
0E4C	FE 59	CP	YES	;ENABLE ERROR INT?
0E4E	20 E5	JR	NZ,SE%C1	;NOT SURE, TRY AGAIN
0E50	CB CC	SET	IEBPS,H	;SET NORMAL INT'S
0E52	0E 09	SE%C3:	LD C,PRTLN	;*****
0E54	11 0323		LD DE,EN3MSG	;* MAX XFER RATE? *
0E57	CD 0005		CALL CDOS	;*****
0E5A	0E 01		LD C,RDCHR	;*** GET USER ***
0E5C	CD 0005		CALL CDOS	;*** RESPONSE ***
0E5F	FE 0D	CP	CR	;USE OLD SETTING?
0E61	28 0C	JR	Z,SE%C5	;YES
0E63	CB 9C	RES	XFRBPS,H	;SET MAX XFER RATE
0E65	FE 59	CP	YES	;MAX XFER RATE?
0E67	28 06	JR	Z,SE%C5	;YES, (0 BIT IMPLIES MAX RATE)
0E69	FE 4E	CP	NO	;MIN XFER RATE?
0E6B	20 E5	JR	NZ,SE%C3	;NOT SURE, TRY AGAIN
0E6D	CB DC	SET	XFRBPS,H	;SET MIN XFER RATE
0E6F	0E 09	SE%C5:	LD C,PRTLN	;*****
0E71	11 034E		LD DE,EN5MSG	;* ENABLE READ CORRECTED? *
0E74	CD 0005		CALL CDOS	;*****
0E77	0E 01		LD C,RDCHR	;*** GET USER ***
0E79	CD 0005		CALL CDOS	;*** RESPONSE ***
0E7C	FE 0D	CP	CR	;USE OLD SETTING?
0E7E	28 14	JR	Z,SE%C6	;YES
0E80	FE 59	CP	YES	;NO, ENABLE RCD?
0E82	20 08	JR	NZ,SE%C53	; NO
0E84	CB EC	SET	RCDDBPS,H	; YES, SET RCD BIT
0E86	CB F8	SET	7,B	; SET MBM REINIT REQUIRED
0E88	CB B4	RES	ICDBPS,H	;CLR ICD--ONLY 1 ERROR CORRECT ALLOWED
0E8A	18 33	JR	SE\$END	

Figure 17. MIDS Software (page 29 of 34).

0E8C	FE 4E	SE\$C53:	CP	NO	;DISABLE RCD?
0E8E	20 DF		JR	NZ,SE\$C5	;NOT SURE, TRY AGAIN
0E90	CB AC		RES	RCDBPS,H	;YES, CLEAR RCD BIT
0E92	CB F8		SET	7,B	;SET MBM REINIT REQUIRED
0E94	CB 6C	SE\$C6:	BIT	RCDBPS,H	;RCD SET?
0E96	28 04		JR	Z,SE\$C61	;NO
0E98	CB B4		RES	ICDBPS,H	;YES, ONLY ONE ERROR CORRECT ALLOWED
0E9A	18 23		JR	SE\$END	
0E9C	0E 09	SE\$C61:	LD	C,PRTLN	*****
0E9E	11 0379		LD	DE,EN6MSG	;* ENABLE INTERNAL CORRECTION? *
0EA1	CD 0005		CALL	CDOS	*****
0EA4	0E 01		LD	C,RDCHR	*** GET USER ***
0EA6	CD 0005		CALL	CDOS	*** RESPONSE ***
0EA9	FE 0D		CP	CR	;USE OLD SETTING?
0EAB	28 12		JR	Z,SE\$END	;YES
0EAD	FE 59		CP	YES	;NO, ENABLE ICD?
0EAF	20 06		JR	NZ,SE\$C63	; NO
0EB1	CB F4		SET	ICDBPS,H	; YES, SET ICD BIT
0EB3	CB F8		SET	7,B	; SET MBM REINIT REQUIRED
0EB5	18 08		JR	SE\$END	
0EB7	FE 4E	SE\$C63:	CP	NO	;DISABLE ICD?
0EB9	20 E1		JR	NZ,SE\$C61	;NOT SURE, TRY AGAIN
0EBB	CB B4		RES	ICDBPS,H	;CLEAR ICD BIT
0EBD	CB F8		SET	7,B	;SET MBM REINIT REQUIRED
0EBF	7C	SE\$END:	LD	A,H	*** SET ENABLE REG VALUE ***
0EC0	32 0002*		LD	(MBM\$BMCR+2),A	*** IN BMC REG TABLE ***
0EC3	CB 78		BIT	7,B	;MUST MBM BE REINIT'ED
0EC5	28 0B		JR	Z,SE\$RT	;NO
0EC7	0E 09		LD	C,PRTLN	;YES, *****
0EC9	11 03A4		LD	DE,REIMSG	; * REINITIALIZING MBM *
0ECC	CD 0005		CALL	CDOS	; *****
0ECF	CD 0A85		CALL	OPR\$1	
0ED2	E1	SE\$RT:	POP	HL	
0ED3	D1		POP	DE	
0ED4	C1		POP	BC	
0ED5	F1		POP	AF	
0ED6	C9		RET		

PAGE

Figure 17. MIDS Software (page 30 of 34).

```

;*****
;*
;* THIS ROUTINE CREATES AN MBM ADDR FROM USER RESPONSES
;* TELLING WHICH BUBBLE AND WHAT PAGE ARE REQUIRED. THIS
;* GENERATED VALUE IS SAVED IN THE BMC ADDR REG SAVE AREA
;* WITHIN THE MBM MODULE.
;*
;* INPUT:  N/A
;*
;* OUTPUT: BMC ADDR REG VALUES ARE SET
;*         A = -1, IF INVALID INPUT BY USER
;*         X'??' = UNDETERMINED, IF (SEMI-) VALID
;*              INPUT BY USER
;*
;*****

OED7      SETADDR:
OED7      C5          PUSH    BC
OED8      D5          PUSH    DE
OED9      E5          PUSH    HL

OEDA      OE 09      LD      C,PRTLW      ;*** PROMPT USER ***
OEDC      11 03E0    LD      DE,MSG4$1    ;*** FOR BUBBLE ***
OEDF      CD 0005    CALL    CDOS         ;*** NUMBER ***

OEE2      OE 0A      LD      C,RDLN      ;*****
OEE4      11 08DC    LD      DE,CONBF     ;* AWAIT RESPONSE *
OEE7      CD 0005    CALL    CDOS         ;*****

OEEA      3A 08DD    LD      A,(CONBF+1)  ;GET NBR OF CHAR READ
OEED      FE 00      CP      ZERO        ;WAS IT 0? (IMPLIES CARRIAGE RETURN)
OEEF      20 08      JR      NZ,SA$C2    ;NO
OEF1      3A 0004$   LD      A,(MBM$BMCR+4) ;*** INIT BUBBLE NBR FOR ***
OEF4      E6 F0      AND      OF0H       ;*** LATER CONCATENATION ***
OEF6      67         LD      H,A         ;*** WITH PAGE NBR ***
OEF7      18 17      JR      SA$C3
OEF9      FE 01      SA$C2: CP      ONE    ;NBR OF CHAR READ = 1?
OEFB      C2 0F67    JP      NZ,SA$ERR   ;NO

OEFE      3A 08DE    LD      A,(CONBF+2)  ;*** CONVERT DIGIT JUST ***
OF01      CD 0F7D    CALL    A$B$CONV    ;*** READ TO BINARY ***
OF04      FE FF      CP      NEG1       ;INVALID INPUT?
OF06      CA 0F67    JP      Z,SA$ERR    ;YES

OF09      67         LD      H,A         ;*****
OF0A      CB 24      SLA      H          ;* SET BUBBLE NBR TO *

```

Figure 17. MIDS Software (page 31 of 34).

```

0F0C    CB 24                SLA    H            ;* MBM ADDR REG FORMAT *
0F0E    CB 24                SLA    H            ;* * * * *

0F10    0E 09                SA$C3: LD    C,PRTLN    ;*** PROMPT USER ***
0F12    11 03FC              LD    DE,MSG4$2    ;*** FOR PAGE ***
0F15    CD 0005              CALL   CDOS        ;*** NUMBER ***

0F18    0E 0A                LD    C,RDLN      ;* * * * *
0F1A    11 08DC              LD    DE,COMBF    ;* AWAIT RESPONSE *
0F1D    CD 0005              CALL   CDOS        ;* * * * *

0F20    3A 08DD              LD    A,(CONBF+1) ;GET NBR OF CHAR READ
0F23    FE 00                CP    ZERO      ;WAS IT 0? (IMPLIES CARRIAGE RETURN)
0F25    20 0B                JR     NZ,SA$C4    ;NO
0F27    3A 0004*             LD    A,(MBM$BMCR+4) ;* * * * *
0F2A    E6 0F                AND    OFH      ;* CONCATENATE BUBBLE NBR *
0F2C    B4                   OR     H          ;* WITH 4 MSB'S OF PAGE NBR *
0F2D    32 0004*             LD    (MBM$BMCR+4),A ;* * * * *
0F30    18 3F                JR     SA$RT
0F32    FE 03                SA$C4: CP    THREE    ;WAS IT THREE?
0F34    20 31                JR     NZ,SA$ERR    ;NO

0F36    3A 08DE              LD    A,(CONBF+2) ;*** CONVERT MOST SIGNIFICANT ***
0F39    CD 0F7D              CALL   A$B$CONV    ;*** DIGIT (MSD) TO BINARY ***
0F3C    FE 08                CP     8          ;INVALID DIGIT INPUT (IE, ) 7)?
0F3E    D2 0F67              JP     NC,SA$ERR    ;YES
0F41    B4                   OR     H          ;*** SET MSD OF ***
0F42    67                   LD     H,A        ;*** MBM PAGE NBR ***

0F43    3A 08DF              LD    A,(CONBF+3) ;*** CONVERT NEXT ***
0F46    CD 0F7D              CALL   A$B$CONV    ;*** DIGIT TO BINARY ***
0F49    FE FF                CP     NEG1       ;INVALID INPUT?
0F4B    28 1A                JR     Z,SA$ERR    ;YES
0F4D    6F                   LD     L,A        ;* * * * *
0F4E    CB 25                SLA    L          ;* SET NEXT *
0F50    CB 25                SLA    L          ;* MSD OF MBM *
0F52    CB 25                SLA    L          ;* PAGE NBR *
0F54    CB 25                SLA    L          ;* * * * *

0F56    3A 08E0              LD    A,(CONBF+4) ;*** CONVERT LSD ***
0F59    CD 0F7D              CALL   A$B$CONV    ;*** TO BINARY ***
0F5C    FE FF                CP     NEG1       ;INVALID INPUT?
0F5E    28 07              JR     Z,SA$ERR    ;YES
0F60    B5                   OR     L          ;*** SET LSD OF ***
0F61    6F                   LD     L,A        ;*** MBM PAGE NBR ***
0F62    22 0003*             LD    (MBM$BMCR+3),HL ;SET BMC ADDR REG VALUES
0F65    18 0A                JR     SA$RT

```

Figure 17. MIDS Software (page 32 of 34).

```

0F67 0E 09      SASERR: LD    C,PRTLN      ;*****
0F69 11 020D    LD    DE,INPERR      ;* PRINT INVALID INPUT MSG *
0F6C CD 0005    CALL   CDOS          ;*****
0F6F 3E FF      LD    A,NEG1        ;SET INVALID INPUT FLAG

0F71 0E 02      SASRT:  LD    C,PRTCHR    ;*****
0F73 11 000A    LD    DE,LF         ;* MAKE OUTPUT PRETTY *
0F76 CD 0005    CALL   CDOS          ;*****
0F79 E1         POP    HL
0F7A D1         POP    DE
0F7B C1         POP    BC
0F7C C9         RET

```

```

;*****
;*
;* THIS ROUTINE CONVERTS THE ASCII CHARACTER FOUND IN REG A
;* TO A BINARY DIGIT.
;*
;* INPUT:  A - ASCII CHAR TO BE CONVERTED
;*
;* OUTPUT: A = CONVERTED DIGIT, IF INPUT WAS VALID
;*          = -1, IF INVALID INPUT BY USER
;*
;*****

```

```

0F7D          A$B$CONV:
0F7D FE 47      CP    'F'+1          ;*** FILTER SOME ***
0F7F 30 09      JR    NC,A$B$CE      ;*** BAD INPUTS ***
0F81 FE 3A      CP    '9'+1          ;*****
0F83 38 02      JR    C,A$B$C5       ;* CONVERT ASCII *
0F85 D6 07      SUB    7             ;* TO HEXIDECIMAL *
0F87 E6 0F      A$B$C5: AND    0FH    ;*****
0F89 C9        RET

0F8A 3E FF      A$B$CE: LD    A,NEG1    ;SET INVALID DIGIT FLAG
0F8C C9        RET

END    START

```

Figure 17. MIDS Software (page 33 of 34).

Macros:

Symbols:

A\$B\$C5	0F87	A\$B\$CE	0F8A	A\$B\$CO	0F7D	A\$RT	0B20
ADRO	000E	ADROMS	0504	ADR1	000F	ADR1MS	0516
BLKMSG	02A8	BLROMS	04CE	BLR1MS	04E0	BM\$CMD	0029
BM\$DAT	0028	BSYBPS	0007	BSYWRN	022E	BUFLEN	00CC
CDOS	0005	CHDERR	01EA	CONBF	08DC	CONTIN	097A
CONV\$2	0BB9	CR	000D	CRLF	0100	DESMG	0470
DIG\$C5	0DBA	DIG\$PT	0DBE	DMA\$PS	0002	EN1MSG	02CC
EN2MSG	02F8	EN3MSG	0323	EN5MSG	034E	EN6MSG	0379
ENRMSG	04F2	FIFO	0000	FILLER	0055	FOUR	0004
GET\$OP	0982	I\$ERR	0BAC	I\$LP5	0BA2	I\$RT	0BB4
ICDBIT	0040	ICDBPS	0006	IEBPS	0001	INBPS	0000
INPERR	020D	INTFLG	080F	IPBPS	0007	LF	000A
MBM\$AB	0B0B*	MBM\$BM	0F63*	MBM\$FF	0B36*	MBM\$IC	0BE1*
MBM\$IN	0A87*	MBM\$IS	0BD7*	MBM\$PS	0CD7*	MBM\$PU	0B3F*
MBM\$RC	0B2D*	MBM\$RE	0AA4*	MBM\$RF	0B02*	MBM\$RS	0AD4*
MBM\$RX	0AE1*	MBM\$RZ	0B24*	MBM\$SR	0B48*	MBM\$ST	0D48*
MBM\$WB	0A7D*	MBM\$WR	0ABD*	MBM\$WS	0B1E*	MBM\$WX	0AEF*
MBM\$WZ	0AF9*	MBM\$WF	0810	MENU	0528	MSG4\$1	03E0
MSG4\$2	03FC	MSGI\$1	041E	MSGI\$2	043D	MSGQ	045B
NEG1	FFFF	NO	004E	04\$RT	0AD9	OLDSP	0978
ONE	0001	OPCBPS	0006	OPCHPL	023F	OPERR	0257
OPFBPS	0005	OPR\$0	0A77	OPR\$1	0A85	OPR\$2	0A8E
OPR\$3	0AB2	OPR\$4	0ACB	OPR\$5	0ADB	OPR\$6	0AE9
OPR\$7	0AF7	OPR\$8	0B00	OPR\$9	0B09	OPR\$A	0B12
OPR\$B	0B22	OPR\$C	0B2B	OPR\$D	0B34	OPR\$E	0B3D
OPR\$F	0B46	OPR\$H	0B4F	OPR\$I	0B5E	OPR\$J	0BD0
OPR\$K	0BDB	OPR\$P	0BE5	OPR\$Q	0C61	OPR\$R	0C87
OPR\$U	0CB4	OPR\$V	0CDF	OPR\$W	0D2C	OPR\$X	0D40
P\$BLK	0C4D	P\$C1	0BFB	P\$C3	0C1A	P\$C7	0C4F
P\$PG	0BF6	P\$PLL	0C30	P\$PSL	0C26	PREMSG	0103
PROMPT	01DD	PRT\$BM	0D47	PRT\$BY	0D9B	PRT\$C3	0D5B
PRT\$C4	0D69	PRT\$C7	0D75	PRT\$DI	0DB2	PRTCHR	0002
PRTEND	0024	PRTLN	0009	PRTSAD	0D84	RCDBIT	0020
RCDBPS	0005	RDCHR	0001	RDLN	000A	REIMSG	03A4
SA\$C2	0EF9	SA\$C3	0F10	SA\$C4	0F32	SA\$ERR	0F67
SA\$RT	0F71	SB\$ERR	0DFA	SB\$RT	0E04	SE\$C0	0E18
SE\$C1	0E35	SE\$C3	0E52	SE\$C5	0E6F	SE\$C53	0E8C
SE\$C6	0E94	SE\$C61	0E9C	SE\$C63	0EB7	SE\$END	0EBF
SE\$RT	0ED2	SEIADR	0ED7	SETBLR	0DC5	SETENR	0E07
SPACE	0020	STACK	0978	STADDR	04A9	START	092E
STATHD	029C	THREE	0003	TWO	0002	U\$C5	0CCE
U\$C6	0CD3	U\$RT	0CD9	WBLBPS	0004	XFRBPS	0003
YES	0059	ZERO	0000				

No Fatal error(s)

Figure 17. MIDS Software (page 34 of 34).

0000'

```

.Z80
CSEG
;TITLE: MBM - MAGNETIC BUBBLE MEMORY DRIVERS
;AUTHOR: CAPT R E MEISNER
;DATE:
;SYSTEM: CROMEMCO Z2D / CDOS 2.36
;SETUP: THIS PROGRAM IS ASSEMBLED AS MBM.REL, FOR LINKING
;       WITH USER PROGRAMS REQUIRING MBM DRIVERS.
;DESCRIPTION: THIS PROGRAM PROVIDES SUBROUTINES FOR DRIVING
;             INTEL 7110 MAGNETIC BUBBLE MEMORIES (MBM) IN BOTH
;             THEIR POLLED AND INTERRUPT I/O CONFIGURATIONS.
;             AVAILABLE SUBROUTINES ARE:
;
;ENTRY  MBM$STAT      ;GET CONTROLLER STATUS
;ENTRY  MBM$ICLR      ;RESET SYSTEM FOR POLLED I/O
;ENTRY  MBM$ISET      ;SET SYSTEM FOR INTERRUPT I/O
;ENTRY  MBM$WBRM      ;WRITE BOOTLOOP REG MASKED
;ENTRY  MBM$INIT      ;MBM INITIALIZATION
;ENTRY  MBM$READ      ;READ 1 PAGE
;ENTRY  MBM$RXBR      ;READ BOOTLOOP REG
;ENTRY  MBM$WRIT      ;WRITE 1 PAGE
;ENTRY  MBM$RSEX      ;READ SEEK
;ENTRY  MBM$WXBR      ;WRITE BOOTLOOP REG
;ENTRY  MBM$WZBL      ;WRITE BOOTLOOP
;ENTRY  MBM$RFSA      ;READ FSA STATI
;ENTRY  MBM$ABRT      ;ABORT
;ENTRY  MBM$SRES      ;SOFTWARE RESET
;ENTRY  MBM$WSEX      ;WRITE SEEK
;ENTRY  MBM$RZBL      ;READ BOOTLOOP
;ENTRY  MBM$RCDT      ;READ CORRECTED DATA
;ENTRY  MBM$FFRE      ;FIFO RESET
;ENTRY  MBM$PURG      ;MBM PURGE
;
;       COMMAND DATA VARIABLES ARE:
;ENTRY  MBM$BMCR      ;BMC REG VALUES
;ENTRY  MBM$PSIZ      ;MBM PAGE SIZE OF XFER
PAGE

```

Figure 18. MBM Software (page 1 of 32).

;OPERATION:

;

;

CONTROLLER (BMC) STATUS REGISTER DEFINITION

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

;

CONTROLLER (BMC) REGISTER DEFINITIONS

REG

NAME

DEFINITION

A

UTILITY

NOT USED BY MBM

B

BLOCK LENGTH (LSB)

C

BLOCK LENGTH (MSB)

BIT 0-10 - NUMBER OF PAGES TO BE X-FERRED

BIT 11 - NOT USED

BIT 12-15 - NUMBER OF FSA CHANNELS

D

ENABLE

BIT 0 - INTERRUPT ENABLE (NORMAL COMPLETION)

BIT 1 - INTERRUPT ENABLE (ERROR)

BIT 2 - DMA ENABLE

BIT 3 - MAX FSA TO BMC XFER RATE

BIT 4 - WRITE BOOTLOOP ENABLE

BIT 5 - ENABLE 'READ CORRECTED DATA (RCD)'

BIT 6 - ENABLE 'INTERNALLY CORRECTED DATA (ICD)'

BIT 7 - ENABLE PARITY INTERRUPT

E

ADDRESS (LSB)

F

ADDRESS (MSB)

BIT 0-10 - STARTING ADDRESS WITHIN EACH MBM

BIT 11-14 - MBM SELECT

BIT 15 - NOT USED

PAGE

Figure 18. MBM Software (page 2 of 32).

```

;***** CONSTANTS *****

0000      ZERO      EQU      0
0001      ONE       EQU      1
000D      CR        EQU      0DH      ;ASCII CARRIAGE RETURN
000A      LF        EQU      0AH      ;ASCII LINE FEED
0038      INT$RST   EQU      38H      ;Z80 INTERRUPT RESTART ADDR (IM=1)
00C3      JP$OPCD   EQU      0C3H     ;OP CODE OF UNCONDITIONAL JUMP
0016      HALFUL    EQU      22D      ;BYTES AVAIL AT HALF FULL INTERRUPT

;CDOS SYSTEM CALL PARAMETERS

0005      CDOS      EQU      0005H     ;CDOS ENTRY POINT
0009      PRTLN     EQU      9         ;PRINT BUFFER LINE ON CONSOLE
0024      PRTEND    EQU      '$'       ;END PRINT BUFFER MARKER

;MBM I/O PORT ASSIGNMENTS

0028      BM$DATA   EQU      28H      ;MBM DATA (I/O)
0029      BM$CMD    EQU      29H      ;MBM COMMAND (OUT ONLY)
0029      BM$STAT   EQU      29H      ;MBM STATUS (IN ONLY)

;REGISTER ADDRESS COUNTER (RAC) ASSIGNMENTS

000B      BLR0      EQU      0BH      ;BLOCK LENGTH REGISTER (LSB)
000C      BLR1      EQU      0CH      ;BLOCK LENGTH REGISTER (MSB)
000D      ENR       EQU      0DH      ;ENABLE REGISTER
000E      ADRO      EQU      0EH      ;ADDRESS REGISTER (LSB)
000F      ADR1      EQU      0FH      ;ADDRESS REGISTER (MSB)
0000      FIFO      EQU      00H      ;FIFO I/O REGISTER

;MBM CONTROLLER (BMC) STATUS BIT POSITIONS

0000      FFRBPS    EQU      0         ;(LSB) - FIFO READY
0002      UNCBPS    EQU      2         ;          UNCORRECTABLE ERROR
0003      CORBPS    EQU      3         ;          CORRECTABLE ERROR
0004      TIMBPS    EQU      4         ;          TIMING ERROR
0005      OPFBPS    EQU      5         ;          OP FAIL
0006      OPCBPS    EQU      6         ;          OP COMPLETE
0007      BSYBPS    EQU      7         ;(MSB) - BUSY

;MBM CONTROLLER ENABLE REG BIT POSITIONS

0000      INBPS     EQU      0         ;(LSB) - INTERRUPT ENABLE (NORMAL)
0001      IEBPS     EQU      1         ;          INTERRUPT ENABLE (ERRORS)
0002      DMABPS    EQU      2         ;          DMA ENABLE

```

Figure 18. MBM Software (page 3 of 32).

```

0003      XFRBPS EQU 3      ; MAX FSA TO BMC XFER RATE
0004      WBLBPS EQU 4      ; ENABLE BOOTLOOP WRITE
0005      RCDBPS EQU 5      ; ENABLE READ CORRECTED DATA
0006      ICDBPS EQU 6      ; ENABLE INTERNALLY CORRECT DATA
0007      IPBPS EQU 7      ;(MSB) - INTERRUPT ENABLE (PARITY)

```

;MBM CONTROLLER ENABLE REG BIT SETTINGS

```

0001      INBIT EQU 01H      ;NORMAL INTERRUPTS
0002      IEBIT EQU 02H      ;INTERRUPT ON ERRORS
0080      IPBIT EQU 80H      ;INTERRUPT ON PARITY ERROR

```

;MBM COMMAND CODES

```

0010      BM$WMBR EQU 10H      ;WRITE BOOTLOOP REGISTER MASKED
0011      BM$INT EQU 11H      ;INITIALIZE
0012      BM$RD EQU 12H      ;READ BUBBLE DATA
0013      BM$WR EQU 13H      ;WRITE BUBBLE DATA
0014      BM$RSK EQU 14H      ;READ SEEK
0015      BM$RBR EQU 15H      ;READ BOOTLOOP REGISTER
0016      BM$WBR EQU 16H      ;WRITE BOOTLOOP REGISTER
0017      BM$WBL EQU 17H      ;WRITE BOOTLOOP
0018      BM$RFSA EQU 18H      ;READ FSA STATUS
0019      BM$ABT EQU 19H      ;ABORT
001A      BM$WSK EQU 1AH      ;WRITE SEEK
001B      BM$RBL EQU 1BH      ;READ BOOTLOOP
001C      BM$RCD EQU 1CH      ;READ CORRECTED DATA
001D      BM$FRE EQU 1DH      ;RESET FIFO
001E      BM$PRG EQU 1EH      ;MBM PURGE
001F      BM$SRE EQU 1FH      ;SOFTWARE RESET
0020      BM$RES EQU 20H      ;RESET STATUS REG AND INTERRUPTS

```

;CONSOLE MESSAGES

```

0000' 20 20 3C 3C      SUPMSG: DB ' <<< COMMAND NOT IMPLEMENTED >>>',PRTEND
0022' 20 20 2A 2A      RDERR: DB ' *** ERROR *** READ PAST END OF PAGE ',PRTEND
0049' 0D 0A 09 3C      ERRMSG: DB CR,LF,' <<< INTERRUPT GENERATED BY ERROR >>>',PRTEND
0071' 0D 0A 09 09      WHON0Z: DB CR,LF,' *** UNDETERMINED ERROR ***',PRTEND
;      *** WHON0Z IS A MSG THAT IS OVER-WRITTEN BY 1 OF ***
;      *** THE 3 FOLLOWING MSGS ONCE ERROR IS DETERMINED ***
0090' 0D 09 09 2A      UNCERR: DB CR,' *** UNCORRECTABLE ERROR ***'
00AE' 0D 0A 24          DB CR,LF,PRTEND
00B1' 0D 09 09 2A      CORERR: DB CR,' *** CORRECTABLE ERROR ***'
00CF' 0D 0A 24          DB CR,LF,PRTEND
00D2' 0D 09 09 2A      TIMERR: DB CR,' *** TIMING ERROR ***'

```

Figure 18. MBM Software (page 4 of 32).

```

00F0' 0D 0A 24          DB      CR,LF,PRTEEND
00F3' 0D 0A          OPCMSG: DB      CR,LF
00F5' 09 3C 3C 3C          DB      '      <<< PREVIOUS OPERATION HAS COMPLETED >>>'
011E' 0D 0A 24          DB      CR,LF,PRTEEND
0121' 09 20 3C 3C          RSTMSG: DB      '      <<< STATUS AND INTERRUPT ARE RESET >>>'
0149' 0D 0A 24          DB      CR,LF,PRTEEND
014C' 20 20 3C 3C          INIMSG: DB      '      <<< SYSTEM INITIALIZED FOR INTERRUPT I/O >>>'
017A' 0D 0A 0A          DB      CR,LF,LF
017D' 43 41 55 54          DB      'CAUTION: IN INTERRUPT MODE, THE ONLY VALID COMMANDS '
01B2' 41 52 45 3A          DB      'ARE:',CR,LF
01B8' 09 09 32 20          DB      '      2 - READ,',CR,LF
01C5' 09 09 33 20          DB      '      3 - WRITE, AND',CR,LF
01D7' 09 09 54 48          DB      '      THOSE GREATER THAN F.',CR,LF
01F0' 09 20 20 57          DB      '      WITH THE EXCEPTION OF 2 AND 3, ALL COMMANDS '
021F' 49 4E 20 54          DB      'IN THE',CR,LF
0227' 09 20 20 52          DB      '      RANGE OF 0 THRU F GIVE UNPREDICTABLE '
024F' 52 45 53 55          DB      'RESULTS.'
0257' 0D 0A 0A 24          DB      CR,LF,LF,PRTEEND
025B' 20 20 3C 3C          REIMSG: DB      '      <<< SYSTEM REINITIALIZED FOR POLLED I/O >>>',PRTEEND
0289' 09 3C 3C 3C          PFRMSG: DB      '      <<< RESETTNG FIFO >>>',CR,LF,PRTEEND

```

***** END CONSTANTS *****

***** VARIABLES *****

```

02A3' 00          RDSIZ: DB      0          ;NBR OF BYTES LEFT TO BE XFERRED
                                           ;DURING READ
02A4' 00          WRSIZ: DB      0          ;NBR OF BYTES LEFT TO BE XFERRED
                                           ;DURING WRITE
02A5'          BUFPTR: DS      2          ;PNTR FOR TRACKING A USERS I/O BUFFER
02A7' 00          INTFLG: DB      0          ;INTERRUPT ENABLED FLAG
02A8'          INTSAV: DS      3          ;SAVE OLD INTERRUPT RESTART ADDR
0005          BRLEN EQU      5          ;LENGTH OF MBM$BMCR
02AB'          MBM$BMCR:          ;BMC REG VALUES (INITIALLY SET AS SPECIFIED BELOW)
02AB' 01          DB      01H          ;BLRO - *** 1 PAGE, 1 CHANNEL ***
02AC' 10          DB      10H          ;BLR1 - ***          XFER          ***
02AD' 08          DB      08H          ;ENR - LOW FREQ XFER
02AE' 00          DB      00H          ;ADRO - *** 1ST PAGE OF ***
02AF' 00          DB      00H          ;ADR1 - *** 1ST BUBBLE ***
02B0'          MBM$PSIZ:
02B0' 44          DB      68D          ;MBM PAGE SIZE IN BYTES (MAX 255)
                                           ;INITIALIZED TO MATCH MBM$BMCR SPECS

```

***** END VARIABLES *****

PAGE

Figure 18. MBM Software (page 5 of 32).

```

;*****
;*
;* THIS ROUTINE GETS THE MBM CONTROLLER STATUS.
;*
;* INPUT:  N/A
;*
;* OUTPUT: A - CONTROLLER STATUS
;*
;*****

02B1' MBM$STAT:
02B1'     IN      A,(BM$STAT)    ;READ CONTROLLER STATUS
02B3'     RET

;*****
;*
;* THIS ROUTINE WRITES THE BUFFER POINTED TO BY HL REG PAIR TO
;* THE SELECTED BOOTLOOP REGISTER(S). NO VALIDATION FOR
;* THE PROPER NUMBER OF 1'S IS REQUIRED SINCE BMC HARDWARE
;* MASKS OFF UNWANTED BITS. (NOTE: THIS ROUTINE IS FOR
;* TESTING ONLY, PRODUCTION ROUTINE NEEDS TO INITIALIZE
;* BLOCK LENGTH AND ADDRESS REGISTERS BEFORE WRITING).
;*
;* INPUT:  HL - PNTR TO FIFO BUF
;*         (HL) - BUFFER OF DATA TO BE WRITTEN
;*
;* OUTPUT: BOOTLOOP REGS ARE SET
;*
;*****

02B4' MBM$WBRM:
02B4'     PUSH   AF
02B5'     PUSH   BC
02B6'     CALL   WAITSTAT        ;WAIT UNTIL BMC AVAIL

;*** INITIALIZE BLR & ADDR REGS

02B9'     LD      B,40D          ;INIT INPUT COUNT
02BB'     LD      C,BM$DATA      ;   FIFO INPUT PORT
02BD'     OTIR                   ;WRITE 40 BYTES TO FIFO

02BF'     LD      A,BM$WMBR      ;***   SEND WRITE   ***
02C1'     OUT     (BM$CMD),A     ;*** BOOTLOOP REG MASKED ***

```

Figure 18. MBM Software (page 6 of 32).

```

02C3'  C1                POP    BC
02C4'  F1                POP    AF
02C5'  C9                RET

```

```

;*****
;*
;* THIS ROUTINE INITIALIZES THE BMC REGISTER TABLE AND THE MBM
;* SYSTEM AS SPECIFIED IN THE BPX72 USERS MANUAL.
;*
;* INPUT:  MBM$BMCR - TABLE OF BMC REG VALUES
;*
;* OUTPUT: MBM$BMCR+3/4 - ADDR REG VALUES ARE UPDATED
;*         MBM PERIPHERAL SYSTEM IS INITIALIZED
;*
;*****

```

```

02C6'  F5                MBM$INIT:
02C6'  F5                PUSH    AF

02C7'  3A 02A7'          LD      A,(INTFLG)    ;*** IS INTERRUPT ***
02CA'  A7                AND      A            ;*** PROCESSING ENABLED? ***
02CB'  C4 048D'          CALL    NZ,INT$INIT   ;YES
02CE'  20 11            JR      NZ,IN$RT      ;YES

02D0'  AF                XOR      A            ;*** SET BMCR ADDR REG ***
02D1'  32 02AE'          LD      (MBM$BMCR+3),A ;*** TO 1ST PAGE OF ***
02D4'  32 02AF'          LD      (MBM$BMCR+4),A ;*** 1ST BUBBLE ***

02D7'  CD 0415'          CALL    WAITSTAT     ;WAIT UNTIL BMC AVAIL
02DA'  CD 03C6'          CALL    SET$BMCR     ;SET BMC REGS

02DD'  3E 11            LD      A,BM$INT      ;*** SEND THE BUBBLE ***
02DF'  D3 29            OUT     (BM$CMD),A    ;*** INITIALIZE COMMAND ***

02E1'  F1                IN$RT: POP    AF
02E2'  C9                RET

```

PAGE

Figure 18. MBM Software (page 7 of 32).

```

;* *****
;*
;* THIS ROUTINE READS FROM 1 TO 3 MBM PAGES INTO A USER
;* DEFINED BUFFER AREA.
;*
;* INPUT:  HL - BEGINNING ADDR OF INPUT BUFFER
;*
;* OUTPUT: (HL) - INPUT BUFFER IS FILLED WITH MBM DATA
;*
;* *****

```

02E3'		MBM\$READ:		
02E3'	F5		PUSH	AF
02E4'	CD 0415'		CALL	WAITSTAT ;WAIT UNTIL BMC AVAIL
02E7'	CD 03C6'		CALL	SET\$BMCR ;SET MBM CONTROLLER REGS
02EA'	3A 02A7'		LD	A,(INTFLG) ;*** IS INTERRUPT ***
02ED'	A7		AND	A ;*** PROCESSING ENABLED? ***
02EE'	C4 04B7'		CALL	NZ,INT\$READ ;YES
02F1'	20 07		JR	NZ,RD\$RT ;YES
02F3'	3E 12		LD	A,BM\$RD ;*** ISSUE ***
02F5'	D3 29		OUT	(BM\$CMD),A ;*** READ ***
02F7'	CD 03EC'		CALL	READ ;READ MBM BLOCK INTO (HL) BUF
02FA'	CD 03DA'	RD\$RT:	CALL	INC\$ADRR ;INCREMENT BMC ADDR REG VALUE
02FD'	F1		POP	AF
02FE'	C9		RET	

PAGE

Figure 18. MBM Software (page 8 of 32).

```

;*****
;*
;* THIS ROUTINE READS THE SELECTED BOOTLOOP REGISTER(S) INTO
;* A FIFO BUFFER. (NOTE: THIS ROUTINE IS FOR TESTING ONLY,
;* PRODUCTION ROUTINE NEEDS TO INITIALIZE BLOCK LENGTH AND
;* ADDRESS REGISTERS BEFORE READING, AND UNSCRAMBLE
;* (DE-INTERLEAVE) THE BOOTLOOP VALUES.
;*
;* INPUT:  HL - PNTR TO BUFFER
;*
;* OUTPUT: (HL) - BUFFER IS FILLED WITH BOOTLOOP DATA
;*
;*****

02FF'      MBM$RXBR:
02FF'      F5          PUSH    AF
0300'      CD 0415'    CALL    WAITSTAT      ;WAIT UNTIL BMC AVAIL

;*** INITIALIZE BLR & ADDR REGS

0303'      3E 15      LD      A,BM$RBR      ;*** SEND READ ***
0305'      D3 29      OUT     (BM$CMD),A     ;*** BOOTLOOP REG ***

0307'      CD 03EC'    CALL    READ          ;READ MBM BLOCK INTO (HL) BUF

030A'      F1          POP     AF
030B'      C9          RET

;*****
;*
;* THIS ROUTINE WRITES 1 TO 3 MBM PAGES FROM A USER DEFINED
;* BUFFER AREA.
;*
;* INPUT:  HL - BEGINNING ADDR OF OUTPUT BUFFER
;*
;* OUTPUT: MBM PAGE(S) WRITTEN
;*
;*****

030C'      MBM$WRIT:
030C'      F5          PUSH    AF
030D'      C5          PUSH    BC
030E'      E5          PUSH    HL

```

Figure 18. MBM Software (page 9 of 32).

030F'	CD 0415'	CALL	WAITSTAT	;WAIT UNTIL BMC AVAIL
0312'	CD 03C6'	CALL	SET\$BMC	;SET MBM CONTROLLER REGS
0315'	3A 02A7'	LD	A,(INTFLG)	*** IS INTERRUPT ***
0318'	A7	AND	A	*** PROCESSING ENABLED? ***
0319'	C4 04D4'	CALL	NZ,INT\$WRIT	;YES
031C'	20 DC	JR	NZ,RD\$RT	;YES
031E'	3E 13	LD	A,BM\$WR	*** SEND WRITE ***
0320'	D3 29	OUT	(BM\$CMD),A	*** COMMAND ***
0322'	CD 041C'	CALL	WATESTRT	*** WAIT UNTIL WRITE ***
0325'	CB 47	WR\$WT1: BIT	FFRBPS,A	*** STARTS AND FIFO ***
0327'	28 FC	JR	Z,WR\$WT1	*** BECOMES AVAILABLE ***
0329'	0E 28	LD	C,BM\$DATA	;SET FIFO OUTPUT PORT
032B'	3A 02B0'	LD	A,(MBM\$PSIZ)	*** SET OUTPUT ***
032E'	47	LD	B,A	*** LENGTH ***
032F'	DB 29	WR\$WT2: IN	A,(BM\$STAT)	;GET BMC STATUS
0331'	CB 7F	BIT	BSYBPS,A	;BUSY?
0333'	28 0B	JR	Z,WR\$RT	;NO
0335'	CB 47	BIT	FFRBPS,A	;ROOM IN FIFO?
0337'	28 F6	JR	Z,WR\$WT2	;NO, THEN WAIT
0339'	ED A3	OUTI		;YES, OUTPUT NEXT BYTE
033B'	00	NOP		
033C'	00	NOP		;GIVE FIFO-READY STATUS TIME TO CHANGE
033D'	00	NOP		
033E'	20 EF	JR	NZ,WR\$WT2	;LOOP UNTIL DONE
0340'	CD 03DA'	WR\$RT: CALL	INC\$ADDR	;INCREMENT BMC ADDR REG VALUE
0343'	E1	POP	HL	
0344'	C1	POP	BC	
0345'	F1	POP	AF	
0346'	C9	RET		

PAGE

Figure 18. MBM Software (page 10 of 32).

```

;*****
;*
;* THIS ROUTINE POSITIONS THE MBM AT A USER SPECIFIED PAGE
;* (RELATIVE TO THE MBM INPUT TRACK).
;*
;* INPUT:  MBM$BMCR+3/4 - PAGE TO BE SELECTED
;*
;* OUTPUT: N/A
;*
;*****

```

```

0347' MBM$RSEX:
0347'     PUSH    AF
0348'     PUSH    HL
0349'     CD 0415'    CALL    WAITSTAT    ;WAIT UNTIL BMC AVAIL

034C'     LD      HL,(MBM$BMCR+3) ;*** DECREMENT BMC ADDR ***
034F'     DEC     HL          ;*** REGISTER VALUE AS ***
0350'     LD      (MBM$BMCR+3),HL ;*** REQUIRED FOR SEEK ***
0353'     CD 03C6'    CALL    SET$BMCR    ;SET MBM CONTROLLER REGS

0356'     LD      A,BM$RSX    ;*** SEND ***
0358'     OUT     (BM$CMD),A  ;*** READ SEEK ***

035A'     INC     HL          ;*** RESET BMC ADDR VALUE ***
035B'     LD      (MBM$BMCR+3),HL ;*** TO USER REQUESTED PAGE ***

035E'     POP     HL
035F'     POP     AF
0360'     RET

```

PAGE

Figure 18. MBM Software (page 11 of 32).

```

;*****
;*
;* THIS ROUTINE WRITES THE BUFFER POINTED TO BY HL REG PAIR TO
;* THE SELECTED BOOTLOOP REGISTER(S). NOTE, THIS ROUTINE
;* REQUIRES VALIDATION OF THE PROPER NUMBER OF 1'S TO BE PUT
;* IN THE BOOTLOOP BEFORE THEY ARE WRITTEN. (NOTE: THIS
;* ROUTINE IS FOR TESTING ONLY, PRODUCTION ROUTINE NEEDS TO
;* INITIALIZE BLOCK LENGTH AND ADDRESS REGISTERS BEFORE
;* WRITING, AND BUBBLE CHANNELS MUST BE INTERLEAVED BIT BY BIT
;* BEFORE THE BOOTLOOP IS WRITTEN).
;*
;* INPUT:  HL - PNTR TO FIFO BUF
;*          (HL) - BUFFER OF DATA TO BE WRITTEN
;*
;* OUTPUT: BOOTLOOP REGS ARE SET
;*
;*****
MBM$WXR:
        PUSH    AF
        PUSH    BC
        CALL    WAITSTAT      ;WAIT UNTIL BMC AVAIL
        ;
;*** INITIALIZE BLR & ADDR REGS ***
        ;
;*** INTERLEAVING ROUTINE GOES HERE ***
        ;
;*** VALIDATION ROUTINE GOES HERE ***
        ;
        LD      B,40D          ;INIT INPUT COUNT
        LD      C,BM$DATA      ;    FIFO INPUT PORT
        OTIR                    ;WRITE 40 BYTES TO FIFO
        ;
        LD      A,BM$WBR        ;*** SEND WRITE ***
        OUT     (BM$CMD),A      ;*** BOOTLOOP REG ***
        ;
0363'    LD 03B9'              CALL    NONSUP      ;COMMAND NOT SUPPORTED
        ;
0366'    C1                   POP     BC
0367'    F1                   POP     AF
0368'    C9                   RET

```

PAGE

Figure 18. MBM Software (page 12 of 32).

```

;* *****
;*
;* THIS AREA RESERVED FOR -- WRITE BOOTLOOP ROUTINE
;*
;* INPUT:  *
;*
;* OUTPUT: *
;*
;* *****
0369' MBM$WZBL:
0369' F5      PUSH    AF
;           CALL    WAITSTAT      ;WAIT UNTIL BMC AVAIL
;
;           LD      A,BM$WBL      ;*** SEND WRITE ***
;           OUT     (BM$CMD),A    ;*** BOOTLOOP ***
;
036A'      CD 0389'
;           CALL    NONSUP        ;COMMAND NOT SUPPORTED
;
036D'      F1
036E'      C9
;           POP     AF
;           RET

;* *****
;*
;* THIS AREA RESERVED FOR -- READ 'FSA STATUS' ROUTINE
;*
;* INPUT:  *
;*
;* OUTPUT: *
;*
;* *****
036F' MBM$RFSA:
036F' F5      PUSH    AF
;           CALL    WAITSTAT      ;WAIT UNTIL BMC AVAIL
;
;           LD      A,BM$FSA      ;*** SEND FSA ***
;           OUT     (BM$CMD),A    ;*** STATUS ***
;
0370'      CD 03B9'
;           CALL    NONSUP        ;COMMAND NOT SUPPORTED
;
0373'      F1
0374'      C9
;           POP     AF
;           RET
PAGE

```

Figure 18. MBM Software (page 13 of 32).

```

;*****
;*
;* THIS ROUTINE TERMINATES THE CURRENTLY EXECUTING COMMAND
;*
;* INPUT:  N/A
;*
;* OUTPUT: N/A
;*
;*****

0375'      MBM$ABRT:
0375'      F5          PUSH    AF

0376'      3E 19      LD      A,BM$ABT      ;*** SEND ***
0378'      D3 29      OUT     (BM$CMD),A    ;*** ABORT ***

037A'      F1          POP     AF
037B'      C9          RET

;*****
;*
;* THIS ROUTINE RESETS THE BMC FIFO AND EACH FSA.
;*
;* INPUT:  N/A
;*
;* OUTPUT: N/A
;*
;*****

037C'      MBM$SRES:
037C'      F5          PUSH    AF

037D'      3E 1F      LD      A,BM$SRE      ;*** SEND S/W ***
037F'      D3 29      OUT     (BM$CMD),A    ;*** RESET ***

0381'      F1          POP     AF
0382'      C9          RET

PAGE

```

Figure 18. MBM Software (page 14 of 32).

```

; * * * * *
;*
;* THIS ROUTINE POSITIONS THE MBM AT A USER SPECIFIED PAGE
;* (RELATIVE TO THE MBM OUTPUT TRACK).
;*
;* INPUT:  MBM$BMCR+3/4 - PAGE TO BE SELECTED
;*
;* OUTPUT:  N/A
;*
; * * * * *
MBM$WSEX:
0383'      F5      PUSH    AF
0384'      E5      PUSH    HL
0385'      CD 0415' CALL    WAITSTAT      ;WAIT UNTIL BMC AVAIL

0388'      2A 02AE' LD      HL,(MBM$BMCR+3) ;*** DECREMENT BMC ADDR ***
038B'      2B      DEC     HL      ;*** REGISTER VALUE AS ***
038C'      22 02AE' LD      (MBM$BMCR+3),HL ;*** REQUIRED FOR SEEK ***
038F'      CD 03C6' CALL    SET$BMCR      ;SET MBM CONTROLLER REGS

0392'      3E 1A    LD      A,BM$WSK      ;*** SEND ***
0394'      D3 29    OUT     (BM$CMD),A      ;*** WRITE SEEK ***

0396'      23      INC     HL      ;*** RESET BMC ADDR VALUE ***
0397'      22 02AE' LD      (MBM$BMCR+3),HL ;*** TO USER REQUESTED PAGE ***

039A'      E1      POP     HL
039B'      F1      POP     AF
039C'      C9      RET

```

PAGE

Figure 18. MBM Software (page 15 of 32).

```

;* *****
;*
;* THIS AREA RESERVED FOR -- READ BOOTLOOP ROUTINE
;*
;* INPUT:  *
;*
;* OUTPUT: *
;*
;* *****
039D' MBM$RZBL:
039D' F5      PUSH    AF
;          CALL    WAITSTAT      ;WAIT UNTIL BMC AVAIL

;          LD      A,BM$RBL      ;*** SEND READ ***
;          OUT     (BM$CMD),A    ;*** BOOTLOOP ***

039E' CD 03B9'      CALL    NONSUP      ;COMMAND NOT SUPPORTED

03A1' F1
03A2' C9      POP     AF
              RET

```

```

;* *****
;*
;* THIS AREA RESERVED FOR -- READ CORRECTED DATA DATA ROUTINE
;*
;* INPUT:  *
;*
;* OUTPUT: *
;*
;* *****
03A3' MBM$RCDT:
03A3' F5      PUSH    AF
;          CALL    WAITSTAT      ;WAIT UNTIL BMC AVAIL

;          LD      A,BM$RCD      ;*** SEND READ ***
;          OUT     (BM$CMD),A    ;*** CORRECTED DATA ***

03A4' CD 03B9'      CALL    NONSUP      ;COMMAND NOT SUPPORTED

03A7' F1
03A8' C9      POP     AF
              RET

```

PAGE

Figure 18. MBM Software (page 16 of 32).

```

;* *****
;*
;* THIS ROUTINE RESETS THE MBM CONTROLLER (BMC) FIFO.
;*
;* INPUT:  N/A
;*
;* OUTPUT: N/A
;*
;* *****

03A9'      MBM$FFRE:
03A9'      F5          PUSH    AF
03AA'      CD 0415'    CALL    WAITSTAT      ;WAIT UNTIL BMC AVAIL

03AD'      3E 1D      LD      A,BM$FRE      ;*** SEND ***
03AF'      D3 29      OUT     (BM$CMD),A     ;*** RESET ***

03B1'      F1          POP     AF

;* *****
;*
;* THIS ROUTINE PURGES MOST OF THE REGISTERS THROUGHOUT THE
;* MBM SYSTEM, INCLUDING SEVERAL IN THE BMC.
;*
;* INPUT:  N/A
;*
;* OUTPUT: N/A
;*
;* *****

03B2'      MBM$PURG:
03B2'      F5          PUSH    AF

03B3'      3E 1E      LD      A,BM$PRG      ;*** SEND ***
03B5'      D3 29      OUT     (BM$CMD),A     ;*** PURGE ***

03B7'      F1          POP     AF
03B8'      C9          RET

PAGE

```

Figure 18. MBM Software (page 17 of 32).

```

;*****
;*
;* THIS ROUTINE PRINTS A WARNING THAT AN OPERATION IS NOT YET
;* IMPLEMENTED.
;*
;* INPUT:  N/A
;*
;* OUTPUT: N/A
;*
;*****

03B9'  C5      NONSUP: PUSH    BC
03BA'  D5      PUSH    DE

03BB'  0E 09      LD      C,PRTLN      ;*** NOTIFY USER ***
03BD'  11 0000'   LD      DE,SUPMSG    ;*** OF COMMAND ***
03C0'  CD 0005      CALL   CDOS        ;*** NON-SUPPORT ***

03C3'  D1      POP     DE
03C4'  C1      POP     BC
03C5'  C9      RET

;*****
;*
;* THIS ROUTINE SETS THE MBM CONTROLLER REGS FROM VALUES
;* STORED IN THE MBM$BMCR TABLE.
;*
;* INPUT:  MBM$BMCR - BMC REG VALUES
;*
;* OUTPUT: MBM CONTROLLER REGS ARE SET
;*
;*****

03C6'      SET$BMCR:
03C6'  F5      PUSH    AF
03C7'  C5      PUSH    BC
03C8'  E5      PUSH    HL

03C9'  3E 08      LD      A,BLRO      ;*** SET BMC POINTER TO ***
03CB'  D3 29      OUT     (BM$CND),A  ;*** BLOCK LENGTH REG ***

03CD'  06 05      LD      B,BRLN      ;SET LENGTH OF MBM$BMCR TABLE
03CF'  0E 28      LD      C,BM$DATA   ; OUTPUT PORT
03D1'  21 02AB'   LD      HL,MBM$BMCR ; OUTPUT BUFFER

```

Figure 18. MBM Software (page 18 of 32).

03D4'	ED B3	OTIR	
03D6'	E1	POP	HL
03D7'	C1	POP	BC
03D8'	F1	POP	AF
03D9'	C9	RET	

```

;*****
;*
;* THIS ROUTINE INCREMENTS THE ADDR REG VALUES STORED IN THE
;* MBM$BMCR TABLE. (A KEY TO THE CODE IS THAT THE PAGE ADDR
;* IN THE BMCR TABLE IS IN THE FLIPPED FORM OF Z-80 ADDRESSES).
;*
;* INPUT:  MBM$BMCR - NBR OF PAGES PER I/O BLOCK
;*          MBM$BMCR+3/4 - BMC ADDR REG VALUES
;*
;* OUTPUT: MBM$BMCR+3/4 - INCREMENTED BY 1
;*
;*****

```

03DA'		INC\$ADDR:	
03DA'	C5	PUSH	BC
03DB'	E5	PUSH	HL
03DC'	ED 4B 02AB'	LD	BC,(MBM\$BMCR) ;*** GET NBR OF PAGES USED ***
03E0'	06 00	LD	B,ZERO ;*** IN PREVIOUS OPERATION ***
03E2'	2A 02AE'	LD	HL,(MBM\$BMCR+3) ;GET ADDR REG VALUES BEFORE OPERATION
03E5'	09	ADD	HL,BC ;*** UPDATE ADDR ***
03E6'	22 02AE'	LD	(MBM\$BMCR+3),HL ;*** REG VALUES ***
03E9'	E1	POP	HL
03EA'	C1	POP	BC
03EB'	C9	RET	

PAGE

Figure 18. MBM Software (page 19 of 32).

```

;* *****
;*
;* THIS ROUTINE USES THE POLLED TRANSFER METHOD TO READ A
;* A BLOCK OF DATA INTO A USER DEFINED AREA.  USER'S BUFFER
;* MUST BE LONG ENOUGH TO HOLD THE REQUESTED MBM BLOCK.
;*
;* INPUT:  HL - ADDR OF BEGINNING OF BUFFER
;*
;* OUTPUT: HL - UNAFFECTED
;*         (HL) - MBM BLOCK POINTED TO BY BMC REGS
;*
;* *****
03EC'  F5          READ:  PUSH  AF
03ED'  C5          PUSH  BC
03EE'  D5          PUSH  DE
03EF'  E5          PUSH  HL

03F0'  CD 041C'    CALL  WATESTRT      ;WAIT FOR BMC TO START READING

03F3'  3A 02B0'    LD     A,(MBM$PSIZ) ;INIT *** MAX PG ***
03F6'  3C          INC     A           ; *** SIZE ***
03F7'  47          LD     B,A         ; *** + 1 ***
03F8'  0E 28       LD     C,BM$DATA  ; INPUT PORT
03FA'  READ$LP:
03FA'  DB 29       IN     A,(BM$STAT) ;GET STATUS
03FC'  CB 7F       BIT    BSYBPS,A   ;BUSY?
03FE'  28 10       JR     Z,READ$RT  ;NO - DONE
0400'  CB 47       BIT    FFRBPS,A   ;YES - DATA AVAIL?
0402'  28 F6       JR     Z,READ$LP  ; NO
0404'  ED A2       INI     ; YES - READ A BYTE
0406'  20 F2       JR     NZ,READ$LP ; LOOP IF PG NOT OVERFLOWED

0408'  0E 09       LD     C,PRTLN    ;*****
040A'  11 0022'    LD     DE,RDERR  ;* ERROR - READ PAST END OF PAGE *
040D'  CD 0005     CALL  CDOS       ;*****

0410'  READ$RT:
0410'  E1          POP     HL
0411'  D1          POP     DE
0412'  C1          POP     BC
0413'  F1          POP     AF
0414'  C9          RET

PAGE

```

Figure 18. MBM Software (page 20 of 32).

```

;* *****
;*
;* THIS ROUTINE MONITORS MBM STATUS UNTIL CONTROLLER BMC
;* BECOMES NOT BUSY.
;*
;* INPUT:  N/A
;*
;* OUTPUT: A - MBM CONTROLLER STATUS
;*
;* *****
0415'      WAITSTAT:
0415'      DB 29          IN      A,(BM$STAT)      ;GET MBM STATUS
0417'      CB 7F          BIT      BSYBPS,A        ;STILL BUSY?
0419'      20 FA          JR      NZ,WAITSTAT      ;YES
041B'      C9            RET

```

```

;* *****
;*
;* THIS ROUTINE MONITORS MBM STATUS UNTIL THE BMC BECOMES BUSY.
;*
;* INPUT:  N/A
;*
;* OUTPUT: N/A
;*
;* *****

```

```

041C'      WATESTRT:
041C'      F5            PUSH     AF

041D'      DB 29      WATELP: IN      A,(BM$STAT)      ;GET BMC STATUS
041F'      CB 7F      BIT      BSYBPS,A        ;HAS READ STARTED YET?
0421'      28 FA      JR      Z,WATELP      ;NO, LOOP UNTIL IT DOES

0423'      F1            POP      AF
0424'      C9            RET

```

PAGE

Figure 18. MBM Software (page 21 of 32).

```

;*****
;*
;* THIS ROUTINE SETS UP THE SYSTEM FOR PROCESSING MBM
;* GENERATED INTERRUPTS, AND REINITIALIZES THE BMC FOR
;* INTERRUPT I/O.
;*
;* INPUT:  MBM$BMC+2 - BMC ENABLE REG VALUE
;*
;* OUTPUT: MBM$BMC+2 - BMC ENABLE REG VALUE WITH NORMAL
;*           INTERRUPTS SET
;*           MBM$BMC+3/4 - BMC ADDR REG SET TO 1ST PAGE OF
;*           1ST BUBBLE
;*           BMC IS INITIALIZED FOR INTERRUPT I/O
;*
;*****
MBM$ISET:
0425'      PUSH    AF
0426'      PUSH    HL
0427'      LD      A,ONE      ;*** SET INTERRUPT ***
0429'      LD      (INTFLG),A ;*** ENABLED FLAG ***

042C'      LD      A,(MBM$BMC+2) ;*** SET NORMAL INTERRUPTS ***
042F'      SET     INBPS,A      ;*** WITHIN THE BMC ***
0431'      LD      (MBM$BMC+2),A ;*** REG VALUE TABLE ***

0434'      CALL    INT$INIT      ;REINITIALIZE MBM

0437'      LD      A,(INT$RST)   ;*****
043A'      LD      (INTSAV),A    ;* SAVE OLD INTERRUPT *
043D'      LD      HL,(INT$RST+1) ;* RESTART OPERATION(S) *
0440'      LD      (INTSAV+1),HL ;*****

0443'      LD      A,JP$OPCD     ;*****
0445'      LD      (INT$RST),A   ;* SET BRANCH TO *
0448'      LD      HL,INT$HNDL   ;* INTERRUPT HANDLER *
044B'      LD      (INT$RST+1),HL ;*****

044E'      XOR     A             ;*****
044F'      LD      (RDSIZ),A     ;* CLEAR READ & WRITE SIZES *
0452'      LD      (WRSIZ),A     ;*****

0455'      POP     HL
0456'      POP     AF
0457'      RET

```

PAGE

Figure 18. MBM Software (page 21 of 32).

```

;*****
;*
;* THIS ROUTINE SETS THE SOFTWARE SYSTEM FOR POLLED MBM I/O,
;* AND REINITIALIZES THE BMC FOR POLLED I/O.
;*
;* INPUT:  MBM$BMC+2 - BMC ENABLE REG VALUE
;*
;* OUTPUT: MBM$BMC+2 - BMC ENABLE REG VALUE WITH ALL
;*           INTERRUPTS TURNED OFF
;*           MBM$BMC+3/4 - BMC ADDR REG SET TO 1ST PAGE OF
;*           1ST BUBBLE
;*           BMC IS INITIALIZED FOR POLLED I/O
;*
;*****

0458'      MBM$ICLR:
0458'      F5          PUSH    AF
0459'      C5          PUSH    BC
045A'      D5          PUSH    DE
045B'      E5          PUSH    HL

045C'      3A 02A7'    LD      A,(INTFLG)      ;*** IS INTERRUPT PROCESSING ***
045F'      A7          AND      A              ;*** ALREADY DISABLED? ***
0460'      28 26       JR      Z,IC$RT        ;YES
0462'      AF          XOR      A              ;NO, *** CLEAR INTERRUPT ***
0463'      32 02A7'    LD      (INTFLG),A      ; *** ENABLED FLAG ***

0466'      F3          DI
0467'      3A 02A8'    LD      A,(INTSAV)      ;*****
046A'      32 0038     LD      (INT$RST),A      ;* RESTORE OLD INTERRUPT *
046D'      2A 02A9'    LD      HL,(INTSAV+1)    ;* RESTART OPERATION(S) *
0470'      22 0039     LD      (INT$RST+1),HL   ;*****

0473'      3A 02AD'    LD      A,(MBM$BMC+2)    ;*** CLEAR ALL ***
0476'      E6 FE       AND      OFFH-INBIT      ;*** INTERRUPTS ***
0478'      E6 82       AND      IEBIT+IPBIT     ;*** WITHIN THE BMC ***
047A'      32 02AD'    LD      (MBM$BMC+2),A    ;*** REG VALUE TABLE ***

047D'      CD 02C6'    CALL    MBM$INIT        ;REINIT MBM SYSTEM
0480'      0E 09       LD      C,PRTLM         ;*****
0482'      11 025B'    LD      DE,REIMSG       ;* SYSTEM REINITIALIZED FOR POLLED I/O *
0485'      CD 0005     CALL    CDOS            ;*****

0488'      E1          IC$RT: POP      HL
0489'      D1          POP      DE
048A'      C1          POP      BC
048B'      F1          POP      AF
048C'      C9          RET

```

Figure 18. MBM Software (page 23 of 32).

```

;*****
;*
;* THIS ROUTINE INITIALIZES THE MBM WHEN THE SYSTEM IS IN
;* ITS INTERRUPT I/O PROCESSING MODE.
;*
;* INPUT:  BMCR - TABLE OF BMC REG VALUES
;*
;* OUTPUT: MBM$BMCR+3/4 - ADDR REG VALUES ARE UPDATED
;*         MBM PERIPHERAL SYSTEM IS INITIALIZED
;*
;*****

048D' INT$INIT:
048D'     F5          PUSH    AF
048E'     C5          PUSH    BC
048F'     D5          PUSH    DE

;NOTE: THE INITIALIZE COMMAND CAUSES RANDOM TOGGLING OF THE DRQ
;      INTERRUPT LINE (THIS IS AN UNDOCUMENTED BUT KNOWN BMC
;      HARDWARE DEFICIENCY). THEREFORE, WHEN INITIALIZING,
;      DISABLE INTERRUPTS UNTIL INITIALIZATION COMPLETES.

0490'     F3          DI
0491'     AF          XOR     A
0492'     32 02AE'    LD      (MBM$BMCR+3),A
0495'     32 02AF'    LD      (MBM$BMCR+4),A
0498'     CD 0415'    CALL    WAITSTAT
049B'     CD 03C6'    CALL    SET$BMCR

;*** SET BMC ADDR REG ***
;*** TO 1ST PAGE OF ***
;*** 1ST BUBBLE ***
;WAIT UNTIL BMC AVAIL
;SET BMC REGS

049E'     3E 11      LD      A,BM$INT
04A0'     D3 29      OUT     (BM$CMD),A
04A2'     0E 09      LD      C,PRTLN
04A4'     11 014C'   LD      DE,INMSG
04A7'     CD 0005    CALL    CDOS
04AA'     CD 0415'   CALL    WAITSTAT

;*** SEND THE BUBBLE ***
;*** INITIALIZE COMMAND ***
;*****
;* SYSTEM INITIALIZED FOR INTERRUPTS *
;*****
;WAIT UNTIL DONE

04AD'     CD 05BD'   CALL    IRESET
04B0'     ED 56      IM      1
04B2'     FB          EI

;RESET INTERRUPTS AND STATUS REG
;SET INTERRUPTS FOR JUMP TO LOC Y'38'

04B3'     D1          POP     DE
04B4'     C1          POP     BC
04B5'     F1          POP     AF
04B6'     C9          RET

```

PAGE

Figure 18. MBM Software (page 24 of 32).

```

;* *****
;*
;* THIS ROUTINE INITIATES AN MBM READ (WITH INTERRUPT
;* PROCESSING) TO A USER DEFINED BUFFER AREA.
;*
;* INPUT:  HL - BEGINNING ADDR OF INPUT BUFFER
;*
;* OUTPUT:  BUFPTR - PNTR TO BEGINNING OF INPUT BUFFER
;*
;* *****

```

04B7'		INT\$READ:		
04B7'	F5		PUSH	AF
04B8'	22 02A5'		LD	(BUFPTR),HL ;INIT BUF PNTR
04BB'	3A 02B0'		LD	A,(MBM\$PSIZ) ;*** SET UP NBR OF BYTES ***
04BE'	32 02A3'		LD	(RDSIZ),A ;*** TO BE XFERRED ***
04C1'	3E 12		LD	A,BM\$RD ;*** ISSUE ***
04C3'	D3 29		OUT	(BM\$CMD),A ;*** READ ***
04C5'	76	IR\$C8:	HALT	;WAIT FOR A 22 BYTE INTERRUPT
04C6'	3A 02A3'		LD	A,(RDSIZ) ;GET REMAINING BYTES TO BE READ
04C9'	FE 16		CP	22D ;ARE LESS THAN 22 BYTES LEFT?
04CB'	30 F8		JR	NC,IR\$C8 ;NO
04CD'	FE 00		CP	ZERO ;ARE EXACTLY 0 BYTES LEFT?
04CF'	28 01		JR	Z,IR\$RT ;YES, OP PROBABLY ALREADY COMPLETE
04D1'	76		HALT	;WAIT FOR OP COMPLETE INT
04D2'	F1	IR\$RT:	POP	AF
04D3'	C9		RET	

PAGE

Figure 18. MBM Software (page 25 of 32).

```

;*****
;*
;* THIS ROUTINE INITIATES AN MBM WRITE (USING INTERRUPT
;* PROCESSING) FROM A USER DEFINED BUFFER AREA.
;*
;* INPUT:  HL - BEGINNING ADDR OF OUTPUT BUFFER
;*
;* OUTPUT:  BUFPTR - PNTR TO BEGINNING OF INPUT BUFFER
;*
;*****
04D4'      INT$WRIT:
04D4'      F5          PUSH    AF
04D5'      C5          PUSH    BC
04D6'      E5          PUSH    HL

04D7'      3E 13      LD      A,BM$WR      ;*** SEND WRITE ***
04D9'      D3 29      OUT     (BM$CMD),A    ;*** COMMAND ***

04DB'      CD 041C'   CALL    WATESTRT    ;*** WAIT UNTIL WRITE ***
04DE'      CB 47      IW$WT1: BIT    FFRBPS,A ;*** STARTS AND FIFO ***
04E0'      28 FC      JR      Z,IW$WT1    ;*** BECOMES AVAILABLE ***

04E2'      0E 28      LD      C,BM$DATA    ;SET FIFO OUTPUT PORT
04E4'      06 28      LD      B,40D        ;      & OUTPUT LENGTH
04E6'      ED B3      OTIR                ;FILL THE FIFO

04E8'      22 02A5'   LD      (BUFPTR),HL  ;INIT BUF PNTR
04EB'      3A 02B0'   LD      A,(BM$PSIZ)  ;*** SET NBR OF ***
04EE'      D6 28      SUB     40D          ;*** BYTES REMAINING ***
04F0'      32 02A4'   LD      (WRSIZ),A    ;*** IN OUTPUT BUF ***

04F3'      76          IW$C8: HALT        ;WAIT FOR 22 BYTE INT
04F4'      3A 02A4'   LD      A,(WRSIZ)    ;GET NBR OF BYTES TO BE WRITTEN
04F7'      FE 00      CP      ZERO        ;WRITE BUFFER EMPTY?
04F9'      20 F8      JR      NZ,IW$C8    ;NO
04FB'      76          HALT                ;YES, WAIT FOR OP COMPLETE INTERRUPT

04FC'      E1          POP     HL
04FD'      C1          POP     BC
04FE'      F1          POP     AF
04FF'      C9          RET

PAGE

```

Figure 18. MBM Software (page 26 of 32).

```

;*****
;*
;* THIS ROUTINE HANDLES MBM INTERRUPTS BY DETERMINING ITS
;* SOURCE AND JUMPING TO APPROPRIATE PROCESSING ROUTINES.
;*
;* INPUT:  RDSIZ - NBR OF BYTES REMAINING TO BE READ
;*         WRSIZ - NBR OF BYTES REMAINING TO BE WRITTEN
;*
;* OUTPUT: RDSIZ & WRSIZ UPDATED (BY INT$?? SUBROUTINES)
;*
;*****
INT$HNDL:
0500'      F5          PUSH    AF
0501'      C5          PUSH    BC
0502'      D5          PUSH    DE

0503'      DB 29      IN      A,(BM$STAT)    ;*** SAVE STATUS ***
0505'      47          LD      B,A          ;*** IN B REG ***

0506'      CB 78      BIT     BSYBPS,B      ;BUSY?
0508'      C2 0558'   JP      NZ,IH$RW      ;YES
050B'      CB 68      BIT     OPFBPS,B      ;OP FAIL?
050D'      CA 0546'   JP      Z,IH$OC       ;NO

0510'      0E 09      LD      C,PRTLN      ;YES, *****
0512'      11 0049'   LD      DE,ERRMSG    ; * ERROR GENERATED INTERRUPT *
0515'      CD 0005     CALL    CDOS        ; *****
0518'      0E 09      LD      C,PRTLN      ;*** UNDETERMINED ERROR ***
051A'      11 0071'   LD      DE,WHONOZ    ;*** (WILL BE OVER-WRITTEN ***
051D'      CD 0005     CALL    CDOS        ;*** ONCE ERROR IS DIAGNOSED) ***

0520'      CB 50      BIT     UNCBPS,B      ;UNCORRECTABLE ERROR?
0522'      28 08      JR      Z,IH$C3      ;NO
0524'      0E 09      LD      C,PRTLN      ;*****
0526'      11 0090'   LD      DE,UNCERR    ;* UNCORRECTABLE ERROR *
0529'      CD 0005     CALL    CDOS        ;*****

052C'      CB 58      IH$C3: BIT     CORBPS,B ;CORRECTABLE ERROR?
052E'      28 08      JR      Z,IH$C4      ;NO
0530'      0E 09      LD      C,PRTLN      ;*****
0532'      11 00B1'   LD      DE,CORERR    ;* CORRECTABLE ERROR *
0535'      CD 0005     CALL    CDOS        ;*****

0538'      CB 60      IH$C4: BIT     TIMBPS,B ;TIMING ERROR?
053A'      28 08      JR      Z,IH$C5      ;NO

```

Figure 18. MBM Software (page 27 of 32).

```

053C' 0E 09          LD      C,PRTLN      ;* * * * *
053E' 11 00D2'       LD      DE,TIMERR    ;* TIMING ERROR *
0541' CD 0005        CALL     CDOS         ;* * * * *

0544'                IH$C5:
;          CALL     MBM$FFRE      ;RESET FIFO
;          CALL     WAITST       ;WAIT FOR RESET COMPLETE
0544' 18 26          JR      IH$DN

0546' 3A 02A3'       IH$OC: LD      A,(RDSIZ) ;*** IS A READ ***
0549' FE 00          CP      ZERO          ;*** PENDING? ***
054B' C4 0575'       CALL     NZ,INT$RD     ;YES, FINISH READING BUF

054E' 0E 09          LD      C,PRTLN      ;* * * * *
0550' 11 00F3'       LD      DE,OPCM$G    ;* OP COMPLETE *
0553' CD 0005        CALL     CDOS         ;* * * * *
0556' 18 14          JR      IH$DN

0558' 3A 02A3'       IH$RW: LD      A,(RDSIZ) ;*** IS A READ ***
055B' FE 00          CP      ZERO          ;*** PENDING? ***
055D' C4 0575'       CALL     NZ,INT$RD     ;YES, READ MORE AND RETURN
0560' 20 0D          JR      NZ,IH$RT      ; TO INTERRUPTED ROUTINE
0562' 3A 02A4'       LD      A,(WRSIZ)    ;*** IS A WRITE ***
0565' FE 00          CP      ZERO          ;*** PENDING? ***
0567' C4 0599'       CALL     NZ,INT$WT     ;YES, WRITE MORE AND RETURN
056A' 18 03          JR      IH$RT        ; TO INTERRUPTED ROUTINE

056C' CD 05BD'       IH$DN: CALL     IRESET ;RESET INTERRUPTS AND STATUS REGS

056F' D1            IH$RT: POP      DE
0570' C1            POP      BC
0571' F1            POP      AF
0572' FB            EI
0573' ED 4D          RETI

PAGE

```

Figure 18. MBM Software (page 28 of 32).

```

;*****
;*
;* THIS ROUTINE SUPPORTS MBM READING WHEN INTERRUPT I/O IS
;* REQUIRED. DATA IS XFERRED FROM THE MBM INTO A USER DEFINED
;* AREA. BLOCK LENGTH CHECKS ARE NOT MADE, SO USER'S BUFFER
;* AREA MUST BE LONG ENOUGH TO HOLD THE REQUESTED MBM BLOCK.
;*
;* INPUT:  BUFPTR - POINTER TO NEXT CHAR IN READ BUFFER
;*
;* OUTPUT: BUFPTR - UPDATED TO NEXT OUTPUT CHAR
;*
;*****

0575'      INT$RD:
0575'      F5          PUSH    AF
0576'      C5          PUSH    BC
0577'      E5          PUSH    HL

0578'      2A 02A5'    LD      HL,(BUFPTR)    ;SET BUF PNTR
0578'      0E 28      LD      C,BM$DATA      ;INPUT PORT
057D'      3A 02A3'    LD      A,(RDSIZ)      ;GET NBR OF BYTES REMAINING IN BUF
0580'      FE 16      CP      HALF          ;LESS THAN HALF OF FIFO BUF LEFT?
0582'      38 07      JR      C,IRD$C2      ;YES
0584'      06 16      LD      B,HALFUL       ;NO, SET HALF FIFO BUF LENGTH
0586'      90          SUB      B           ;DECREASE NBR OF BYTES REMAINING
0587'      ED B2      INIR          ;READ A BLOCK
0589'      18 04      JR      IRD$DN

058B'      47          IRD$C2: LD      B,A      ;SET NBR OF BYTES REMAINING
058C'      AF          XOR      A           ;CLEAR NBR OF BYTES REMAINING IN BUF
058D'      ED B2      INIR          ;READ FINAL BLOCK

058F'      32 02A3'    IRD$DN: LD      (RDSIZ),A    ;SAVE NBR OF BYTES LEFT IN BUF
0592'      22 02A5'    LD      (BUFPTR),HL    ;SAVE PNTR TO NEXT BYTE IN BUF
0595'      E1          POP      HL
0596'      C1          POP      BC
0597'      F1          POP      AF
0598'      C9          RET

PAGE

```

Figure 18. MBM Software (page 29 of 32).

```

;*****
;*
;* THIS ROUTINE SUPPORTS MBM WRITING WHEN INTERRUPT I/O IS
;* REQUIRED. DATA IS XFERRED FROM A USER DEFINED OUTPUT
;* BUFFER TO THE MBM.
;*
;* INPUT:  BUFPTR - POINTER TO NEXT CHAR IN WRITE BUFFER
;*
;* OUTPUT: BUFPTR - UPDATED TO NEXT OUTPUT CHAR
;*
;*****
0599'          INT$WT:
0599'          F5          PUSH    AF
059A'          C5          PUSH    BC
059B'          E5          PUSH    HL

059C'          2A 02A5'    LD      HL,(BUFPTR)    ;SET BUF PNTR
059F'          0E 28      LD      C,BM$DATA      ;OUTPUT PORT
05A1'          3A 02A4'    LD      A,(WRSIZ)      ;GET NBR OF BYTES REMAINING IN BUF
05A4'          FE 16      CP      HALF          ;LESS THAN HALF OF FIFO BUF LEFT?
05A6'          38 07      JR      C,IWR$C2        ;YES
05A8'          06 16      LD      B,HALFUL        ;NO, SET HALF FIFO BUF LENGTH
05AA'          90          SUB     B              ;DECREASE NBR OF BYTES REMAINING
05AB'          ED B3      OTIR                    ;WRITE A BLOCK
05AD'          18 04      JR      IWR$DN

05AF'          47          IWR$C2: LD      B,A      ;SET NBR OF BYTES REMAINING
05B0'          AF          XOR      A              ;CLEAR NBR OF BYTES REMAINING IN BUF
05B1'          ED B3      OTIR                    ;WRITE FINAL BLOCK

05B3'          32 02A4'    IWR$DN: LD      (WRSIZ),A    ;SAVE NBR OF BYTES LEFT IN BUF
05B6'          22 02A5'    LD      (BUFPTR),HL    ;SAVE PNTR TO NEXT BYTE IN BUF
05B9'          E1          POP     HL
05BA'          C1          POP     BC
05BB'          F1          POP     AF
05BC'          C9          RET

PAGE

```

Figure 18. MBM Software (page 30 of 32).

```

;*****
;*
;* THIS ROUTINE CLEARS MBM INTERRUPTS AND CLEARS THE BMC
;* STATUS REG.
;*
;* INPUT:  N/A
;*
;* OUTPUT: BMC STATUS REG = 00
;*         DRQ AND INT INTERRUPT LINES ARE CLEARED
;*
;*****

05BD'      IRESET:
05BD'      F5      PUSH    AF
05BE'      C5      PUSH    BC
05BF'      D5      PUSH    DE

05C0'      3E 20      LD      A,BM$RES      ;*** RESET INTERRUPTS & ***
05C2'      D3 29      OUT     (BM$CMD),A    ;*** CLEAR STATUS REG ***

05C4'      0E 09      LD      C,PRTLN      ;*****
05C6'      11 0121'   LD      DE,RSTMSC    ;* STATUS/INTERRUPT ARE RESET *
05C9'      CD 0005    CALL     CDOS        ;*****

05CC'      DB 29      IRS:   IN      A,(BM$STAT) ;GET BMC STATUS
05CE'      A7          AND      A          ;IS STATUS CLEAR (IMPLIES INT ALSO CLR)
05CF'      20 FB      JR      NZ,IRS      ;NO, WAIT

05D1'      D1          POP      DE
05D2'      C1          POP      BC
05D3'      F1          POP      AF
05D4'      C9          RET

END

```

Figure 18. MBM Software (page 31 of 32).

Macros:

Symbols:

ADRO	000E	ADR1	000F	BLR0	000B	BLR1	000C
BM\$ABT	0019	BM\$CMD	0029	BM\$DAT	0028	BM\$FRE	001D
BM\$INT	0011	BM\$PRG	001E	BM\$RBL	001B	BM\$RBR	0015
BM\$RCD	001C	BM\$RD	0012	BM\$RES	0020	BM\$RFS	0018
BM\$RSK	0014	BM\$SRE	001F	BM\$STA	0029	BM\$WBL	0017
BM\$WBR	0016	BM\$WMB	0010	BM\$WR	0013	BM\$WSX	001A
BRLEN	0005	BSYBPS	0007	BUFPTR	02A5'	CDOS	0005
CORBPS	0003	CORERR	00B1'	CR	000D	DMABPS	0002
ENR	000D	ERRMSG	0049'	FFRBPS	0000	FFRMSG	0289'
FIFO	0000	HALFUL	0016	IC\$RT	0488'	ICDBPS	0006
IEBIT	0002	IEBPS	0001	IH\$C3	052C'	IH\$C4	0538'
IH\$C5	0544'	IH\$DN	056C'	IH\$OC	0546'	IH\$RT	056F'
IH\$RW	0558'	IN\$RT	02E1'	INBIT	0001	INBP3	0000
INC\$AD	03DA'	INMSG	014C'	INT\$HM	0500'	INT\$IN	048D'
INT\$RD	0575'	INT\$RE	04B7'	INT\$RS	0038	INT\$WR	04D4'
INT\$WT	0599'	INTFLG	02A7'	INTSAV	02A8'	IPBIT	0080
IPBPS	0007	IR\$C8	04C5'	IR\$RT	04D2'	IRD\$C2	058B'
IRD\$DN	058F'	IRESET	05BD'	IRS	05CC'	IW\$C8	04F3'
IW\$WT1	04DE'	IWR\$C2	05AF'	IWR\$DN	05B3'	JP\$OPC	00C3
LF	000A	MBM\$AB	0375I'	MBM\$BM	02ABI'	MBM\$FF	03A9I'
MBM\$IC	0458I'	MBM\$IN	02C6I'	MBM\$IS	0425I'	MBM\$PS	02B0I'
MBM\$PU	03B2I'	MBM\$RC	03A3I'	MBM\$RE	02E3I'	MBM\$RF	036FI'
MBM\$RS	0347I'	MBM\$RX	02FFI'	MBM\$RZ	039DI'	MBM\$SR	037CI'
MBM\$ST	02B1I'	MBM\$WB	02B4I'	MBM\$WR	030CI'	MBM\$WS	0383I'
MBM\$WX	0361I'	MBM\$WZ	0369I'	NONSUP	03B9'	ONE	0001
OPCBPS	0006	OPCM\$G	00F3'	OPFBPS	0005	PRTEND	0024
PRTLM	0009	RCDBPS	0005	RD\$RT	02FA'	RDERR	0022'
RDSIZ	02A3'	READ	03EC'	READ\$L	03FA'	READ\$R	0410'
REMSG	025B'	RSTMSG	0121'	SET\$BM	03C6'	SUPMSG	0000'
TIMBPS	0004	TIMERR	00D2'	UNCBPS	0002	UNCERR	0090'
WAITST	0415'	WATELP	041D'	WATEST	041C'	WBLBPS	0004
WHONZ	0071'	WR\$RT	0340'	WR\$WT1	0325'	WR\$WT2	032F'
WRSIZ	02A4'	XFRBPS	0003	ZERO	0000		

No Fatal error(s)

Figure 18. MBM Software (page 32 of 32).

IV. User's Manual

The MBM Interactive Development System (MIDS) described in the following manual is an S-100 based peripheral device used for troubleshooting and verifying operation of Intel 7110 MBM's and their related support IC's. Once a user is familiar with MBM operating characteristics (see Ref 3), the use of MIDS is straightforward. It requires only that the user be able to log onto the host system and initiate execution of the program called MIDS. Software prompts the user for subsequent inputs. In addition, a menu of available operations can be displayed at anytime to assist in input selection.

System Start-up

The following sequence describes how to get started with MIDS.

1. Turn off power.
2. Insure BPK-72 to S-100 interface card is seated in the motherboard.
3. Turn on power.
4. Boot the Operating System.
5. Type "MIDS" on console (ie, start system execution).
6. Console will prompt for additional information.

Command Summary

Once MIDS execution begins, the console displays a help menu and prompts the user to enter an execution command. The help menu lists all valid commands and has the following appearance:

***** MBM COMMAND MENU *****

0 - WRITE B/L REGISTER MASKED	1 - INITIALIZE
2 - READ BUBBLE	3 - WRITE BUBBLE
4 - READ SEEK	5 - READ BOOTLOOP REGISTER
6 - WRITE BOOTLOOP REGISTER	7 - WRITE BOOTLOOP
8 - READ FSA STATUS	9 - ABORT
A - WRITE SEEK	B - READ BOOTLOOP
C - READ CORRECTED DATA	D - RESET FIFO
E - MBM PURGE	F - SOFTWARE RESET
H - DISPLAY COMMAND MENU	I - INITIALIZE MBM BUFFER
J - SET INTERRUPT I/O	K - SET POLLED I/O PROCESSING
P - PRINT MBM BUFFER	Q - READ BMC ADDR REG (PRINT)
R - READ FIFO (AND PRINT)	S - PRINT BMC STATUS
U - SET BMC REG VALUES	V - PRINT BMC REG VALUES
W - WRITE FIFO	X - EXIT TO CDOS

To execute one of the listed commands, the user must enter the single letter appearing to the left of the desired operation title. Commands requiring additional information will prompt the user for it as needed.

Of the 28 operations available on the command menu, the first 16 correspond directly to physical Intel 7110 commands. Therefore, an explanation to commands 0 through F is not reiterated here, but can be found in Appendix E under the BPK-72 Bubble Memory Prototype Kit User's Manual section (Ref 2: 3-10 - 3-13). The remaining 12 commands are used for development support and are explained in the following paragraphs.

H - Display Command Menu. This command lists the menu illustrated above.

I - Initialize MBM Buffer. MIDS software maintains an 204 byte buffer. This is sufficient to hold up to three MBM pages ($3 * 68 = 204$). The Initialize MBM Buffer command provides a way to set the software buffer to a known value before an output operation.

Following initiation of the "I" command the console will prompt the user for an initial value which is put into the first byte of the buffer. Then an increment value, entered after a second prompt, is used to ripple values throughout the buffer. For example, an initial value of 01H and an increment of 01H provides 204 bytes with the following hexadecimal pattern: 01, 02, 03, ... CA, CB, CC.

J - Set Interrupt I/O Processing. This command enables interrupt I/O processing by setting the Enable Register within the BMC to interrupt when an operation completes. Other interrupt conditions can be enabled via the Set BMC Register Values (U) command.

Interrupt I/O is somewhat limited. The interrupt handling routine is set to recognize operation complete and error interrupts. In addition, FIFO half full interrupts are processed only for MBM Read (2) and Write (3) commands. All other interrupts are essentially ignored.

K - Set Polled I/O Processing. Polled I/O is the normal operating configuration for MIDS. The Set Polled I/O Processing (K) command is provided to return MIDS software to

its normal configuration following interrupt I/O processing. In addition to resetting MIDS software, all interrupt enable bits within the BMC are cleared.

P - Print MBM Buffer on Console. This command formats and dumps the hexadecimal byte values found in the software I/O buffer. Two slightly different formats are printed depending on whether error correction is enabled. With error correction only 64 bytes are displayed per MBM page. Without error correction, all 68 bytes per page are displayed.

Q - Read BMC Address Register (and Print). This command reads the BMC Address Register and prints it on the console.

R - Read FIFO (and Print). The BMC contains a 40 byte FIFO as a data buffer between the processor and the bubble device. The "R" command dumps the FIFO to the console.

During the FIFO read, the first byte of data is lost. This loss of data results from software implementation restrictions. To allow for MIDS flexibility, BMC registers must be initialized before each FIFO read. This initialization operation destroys the first byte in the FIFO (Ref 3:3-8).

S - Print BMC Status. This command reads the BMC Status Register and prints it on the console.

U - Set BMC Register Values. Registers within the BMC define operation of the MBM peripheral. The "U" command provides a way to change these register values so that the

BMC can be configured for specific development tasks. (Ref 2:3-2 - 3-7)

Individual register values are set based on responses to console prompts. The first prompt:

NUMBER OF PAGES PER I/O BLOCK =

requests information for setting the Block Length Register.

Answers to the next set of prompts:

ENABLE NORMAL INTERRUPTS? (Y/N/Return)
INTERRUPT ON ERRORS? (Y/N/Return)
MAXIMUM TRANSFER RATE? (Y/N/Return)
READ CORRECTED DATA? (Y/N/Return)
INTERNALLY CORRECT DATA? (Y/N/Return)

are used to generate an Enable Register value. Note that software will not allow interrupts to be enabled unless the system is in interrupt I/O mode (initiated by "J" command). In addition, software allows only one form of error correction to be enabled at any one time. The final prompts:

WHICH BUBBLE?
RECORD NUMBER (3 HEX DIGITS)?

request data for initializing the Address Register.

Some MBM I/O operations update the Address Register to point to the next available MBM page. The "U" command is capable of leaving this and other register values unchanged. Any of the register fields that can be changed by the "U" command can also be left unchanged with a "Return" response.

V - Print BMC Register Values. BMC registers are reset before each MBM operation from values saved in memory. While the "U" command changes these values, the "V" command displays them.

W - Write FIFO. This command dumps the first 40 bytes of the 204 byte software I/O buffer to the BMC FIFO.

X - Exit to CDOS. This command returns execution control to the operating system.

Command Features

Not all commands involve physical access to the MBM peripheral. Following initiation of commands that do, the peripheral status is automatically printed. The status that is displayed may at times present false images of actual peripheral status. This happens because some instructions do not complete before the status is displayed. This is not a fault, but rather a debugging feature of MIDS. This allows the user to observe the results of an operation and to continue processing without having to wait for a valid status which may never come.

Status' that indicate an operation has completed have their most significant bit off, and only one of their next two significant bits on (Ref 2:3-3). On occasions when an unexpected status is displayed, execution of the Print Status (S) command usually provides enough delay so that the expected status is shown. If this request results in another apparently bad status, chances are that an MBM fault exists.

Most MBM commands await completion of previous operations before they start executing. Attempted execution of such commands when the most significant bit of the BMC status (the busy bit) is set, results in a possible infinite loop waiting for the MBM to become available. So, before

entering a command, the user must insure that the MBM status is not busy. One way to accomplish this is via the Abort (9) command.

MBM Initialization

The following sequence of commands insures that the MBM peripheral is set up to properly process user requests. First, an MBM Abort (9) command is sent to terminate any currently executing command and to clear BMC status. The status returned should be either 40H or 41H. After obtaining either one of these status' the MBM Initialization (1) command should be executed. Again the final status should be either 40H or 41H. Any other status, for either command, indicates problems that must be solved before other commands in the range of 0 through F can be executed.

Interrupt Processing

An interrupt processing capability is available with MIDS only to prove that MBM interrupt facilities work as claimed by the manufacturer. The primary advantage of using interrupts, concurrent processing of tasks, is not supported by MIDS. Following initiation of an MBM command, a wait loop is entered until all interrupts related to the requested operation are processed. Consequently, each command executes to completion before another is started.

Errors

User errors fall into two categories. One type is detected by MIDS software, while the other is found by the operating system. Errors caught by MIDS software cause an error message to be printed, execution of the current command to cease, and return to the MIDS command entry level. At the command entry level the user can retry the erroneous command, or try a different command. Errors caught by MIDS are:

INVALID COMMAND - requested operation does not match those available on the command menu;

INVALID INPUT - additional data requested during a command is invalid; some invalid inputs do not cause an error message, but instead, cause the original question to be asked again.

Errors found by the operating system do not have the same gracious effect as those errors found by MIDS. Operating system errors cause an error message to print and control to pass back to the operating system level. The user may then reexecute MIDS or, in extreme cases, reboot CDOS. The most common way to get an operating system error is to request an MBM operation that is not supported by interrupt processing, while MIDS is in its interrupt mode. See command "J" for a discussion of valid interrupt operations.

Appendix D

IFPDAS IR Debugging Tool

Contents

I.	Introduction	237
II.	User Instructions	238
	Monitoring	238
	Single Step	238
	IR Reset	239
	Memory/Peripheral Read	239
	RAM/Peripheral Write	240
III.	Hardware	241
	Schematic Diagrams	241
	IR Bus Buffers	241
	Bus Monitor	244
	IR Reset	247
	Single Step	248
	Input/Output	251
	IC Map	253

IFPDAS IR Debugging Tool

I. Introduction

The IFPDAS IR described in this thesis is a prototype. Because of this, it requires tools for software development. One such tool is the IFPDAS Inflight Recorder Debugging Tool (RDT). The RDT is a hardware front panel for the IR processor. It does not contain a monitor program or any other software, but does give programmers a way to trace software execution.

The RDT is designed so as not to affect IR operation. The only impact of RDT design on the IR is bus loading. As explained later, the RDT presents single P2CMOS loads to many of the pins on the IR busses. The addition of these single loads is transparent to IR operation.

No IR hardware changes are required to accommodate the RDT. This fact, coupled with bus loading transparency, means that the IR will operate identically with or without the RDT. Thus, hardware changes do not have to be factored into operating predictions whenever the IR is detached from the RDT.

II. User Instructions

The RDT is a hardware front panel for the IR processor.

Capabilities that the RDT provides are:

1. monitoring address and data busses,
2. single stepping through programs,
3. resetting the IR processor,
4. reading a byte from memory or a peripheral, and
5. writing a byte to RAM or a peripheral.

Another capability that users do not explicitly see is the one for unimpeded operation of the IR. The IR can run independent of the RDT in two ways. One is with the interface cable between the IR and RDT detached. Another way is to put the RDT in "RUN" (SW106) mode with all other debugging functions disabled. A benefit of this method is that the hexadecimal displays will monitor program execution and provide feedback on its operation.

Monitoring

Monitoring activity takes place during program execution. Programs execute in one of two modes, full speed or single step. During both modes, hexadecimal numbers displayed on the front panel reflect the address of the currently executing instruction.

Single Step

The combined use of switched SW106 and SW107 allow users to execute an IR program with breaks between instructions. To enable single step operation, SW106 is switched to "S/S".

As soon as this happens and the current instruction completes execution, the IR processor halts to await a step command. The momentary switch, SW107, transmits this command when depressed. Each time SW107 is toggled one IR instruction is executed. When SW106 is in its "RUN" position, SW107 is disabled.

IR Reset

Reset action takes place regardless of other RDT switch settings. Whenever SW112 is depressed, the IR processor is forced to restart program execution at hexadecimal location 0000H. This is the same address where program execution begins upon power up. Because power up automatically causes an IR Reset, performing a reset through the RDT is not necessary to start program execution.

Memory/Peripheral Read

When used together, SW108, SW109, SW110, and SW111 provide the IR with a memory and peripheral input capability. To perform a read, SW110 is set to "RD". Switches SW108 and SW109 determine the input source and enables SW111, the read/write strobe. When the strobe is toggled, the byte at the address shown on the hexadecimal display is latched into the data display. The action of the read strobe is disabled whenever both SW108 and SW109 are in their "NOP" positions.

After choosing to perform an I/O operation (SW108 = MEM or SW109 = PER) and before toggling the read strobe, the address display can be changed to a user defined value.

Individual digits are incremented by depressing the switch directly below the displays. Note that peripheral addresses occupy only one byte, and must be entered in either the two high-order or the two low-order hexadecimal digits.

RAM/Peripheral Write

The RDT write operation dumps the information shown in the data display to the memory or peripheral address shown in the address display. Write operations work similar to the read operation described above. With SW110 set to "WR", SW108 and SW109 determine the type of output to be performed, while SW111 determines when the operation will occur. One obvious difference between the read and write operations is that the data display must be initialized before the write strobe is toggled. Another difference is that a memory write operation is restricted to the RAM address space. Memory read operations can also access EEPROM addresses.

III. Hardware

Schematic Diagrams

Instead of having one large schematic diagram, RDT hardware is described using smaller, functionally grouped diagrams. When combined as one, the individual diagrams completely define the RDT. The rule that binds the diagrams is signal naming conventions. From one diagram to the next, common signal paths have identical names.

Most control signals found in the following schematics are prefixed with either an "O" or an "I". An "O" prefix indicates that the signal originates from within the RDT hardware and is "Output" to the IR bus. Signals "Input" from the IR bus are preceded with an "I". Signals with no prefix are generated and used internal to the RDT. Control signals may also have a postfix of "*" to indicate that they are an active low signal.

Another standard feature of RDT hardware is that all switches are debounced. The debouncing circuit is implemented in every case by a data flip-flop (FF) with preset and clear inputs. Its theory of operation is presented in the discussion of the IR Reset function.

IR Bus Buffers. The interface between the IR bus and the RDT is fully buffered. Figure 19 shows that all signals - with the exception of OWAIT*, OBREQ*, OPS*, and ORESET-IN* - are connected via P2CMOS buffers. So, RDT inputs present

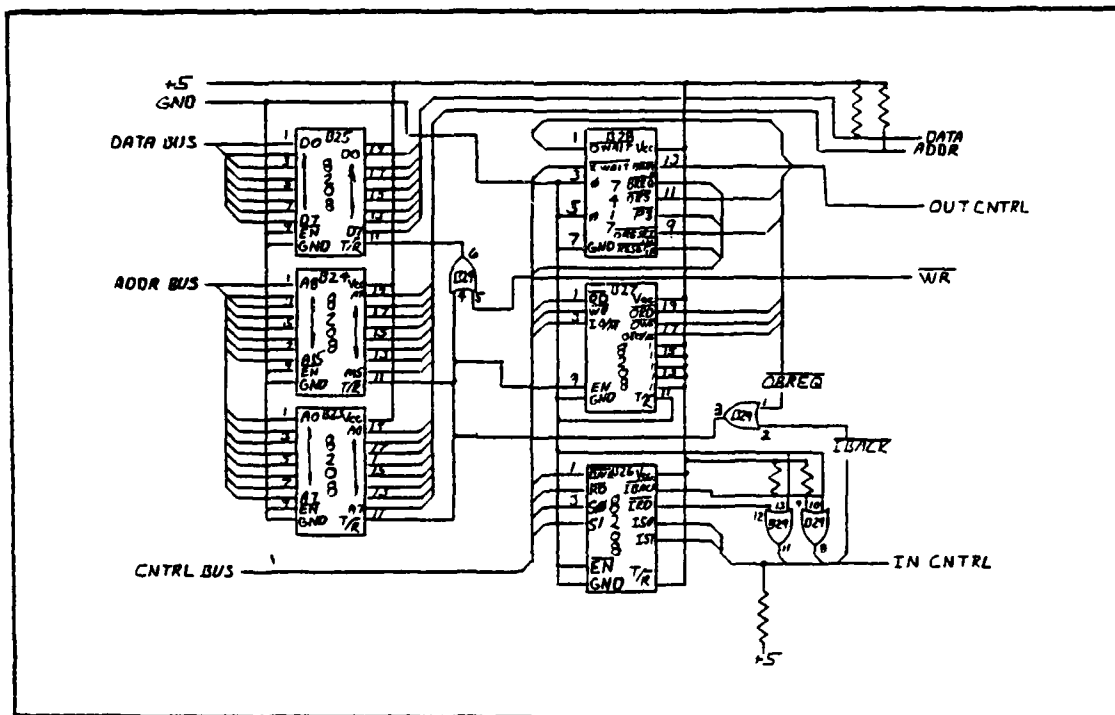


Figure 19. RDT I/O Buffers.

only single P2CMOS loads to the IR bus; and outputs have the same drive capacity as components of the IR. The three remaining signals are output to the IR control bus through open-collector gates.

IR signals required by the RDT fall into three categories: bidirectional, input and output. Placement within a category depends upon when and how individual signals are enabled through a buffer. Data and address busses, which provide both input and output for the RDT, are bidirectional. Control bus signals S0, S1, RD*, and BACK* are sources of input. Output signals are the XWAIT*, BREQ*, PS*, RESET-IN*, RD*, WR*, and IO/M* control lines.

While RD* appears both as an input and an output signal, it is not considered bidirectional because it is buffered by B26 as an input, and by B27 as an output.

Bidirectional lines are buffered by B23, B24, and B25. These 82PC08, Bidirectional Transceivers, operate continuously with their direction of transmission determined by the RDT function being performed. When the RDT is in a monitor or single step mode, all three transceivers act as input buffers. In the memory/peripheral write mode, they are output buffers. However, when reading memory or a peripheral, B25 is an input buffer and B23 and B24 are output buffers. The two OR gates in the upper part of Figure 19 provide direction control logic for these three transceivers.

IR lines categorized as input signals are buffered by B23. The input buffer is hardwired to transfer data from the IR to the RDT continuously. The two OR gates fed directly from B26 are used as a second level of input to increase the fan-out of the P2CMOS IC's for driving the LSTTL circuitry of the RDT.

IC's B27 and B28 are output buffers. To preclude bus contention problems, ORD*, OWR*, and OIO/M* use the tri-state feature of the 82PC08. During operations where the RDT does not require control of IR resources, the output buffers are disabled. Neither monitor nor single step operations need control over the IR to accomplish their tasks. However, I/O operations must use the IR buses. Once an I/O operation gains control of IR resources, $BREQ* + BACK* = 0$, the 82PC08

is enabled and signals generated by the RDT are sent to the IR.

The four output signals which do not pass through Bidirectional Transceivers - OWAIT*, OBREQ*, OPS*, and ORESET-AN* - are interfaced to the IR via 7417 open-collector buffers. The reason open-collector buffers are required is that corresponding signals on the IR control bus are held normally high through pull-up resistors. To drive these lines low, open-collector gates are used.

Bus Monitor. Hexidecimal displays are provided for monitoring the IR data and address busses. Toggle switches and counters add the capability for initializing these busses whenever the RDT is in an I/O operation mode.

Before discussing construction of the monitors, an understanding of the differing functional requirements between the data and address monitors is useful. While programs execute, the RDT is in a monitoring mode. That is, the address monitor reflects the addresss of the currently executing instruction and the data monitor is blank. In its I/O mode the RDT gains control of the IR busses from the NSC800. Regardless of whether an input or an output operation is being performed, the value in the address monitor is gated to the IR address bus. Similarly, the data monitor is gated to the IR data bus, but only during an output operation. During input the data monitor reflects the value found on the data bus.

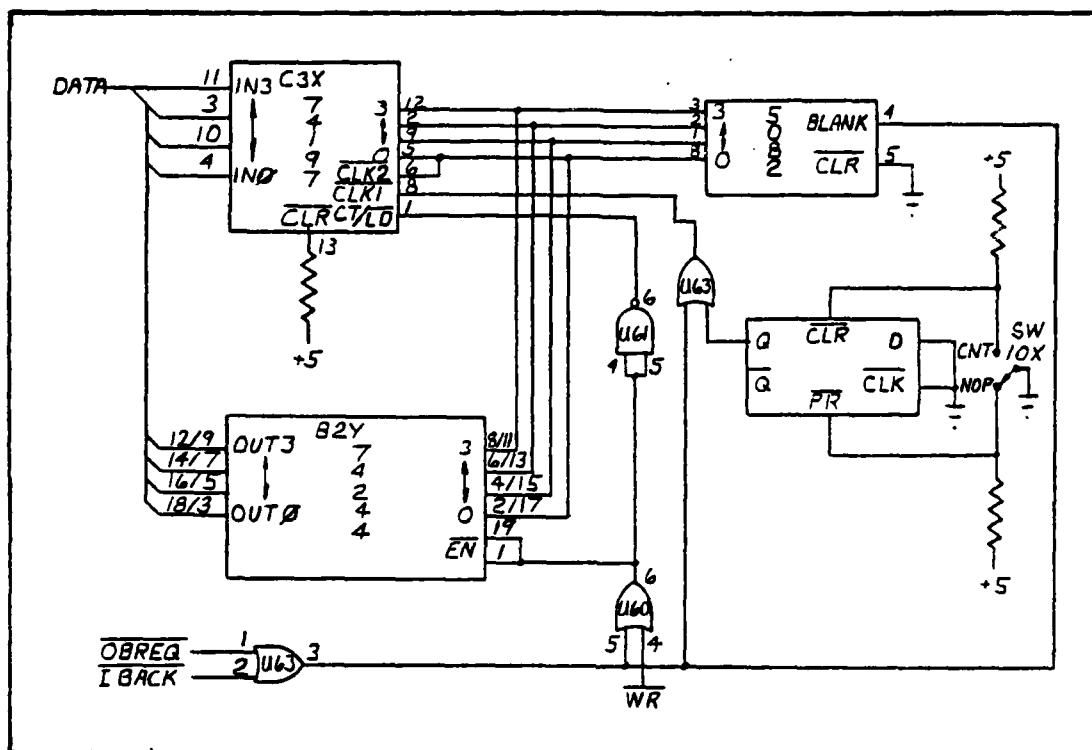


Figure 20. Data Bus Monitor.

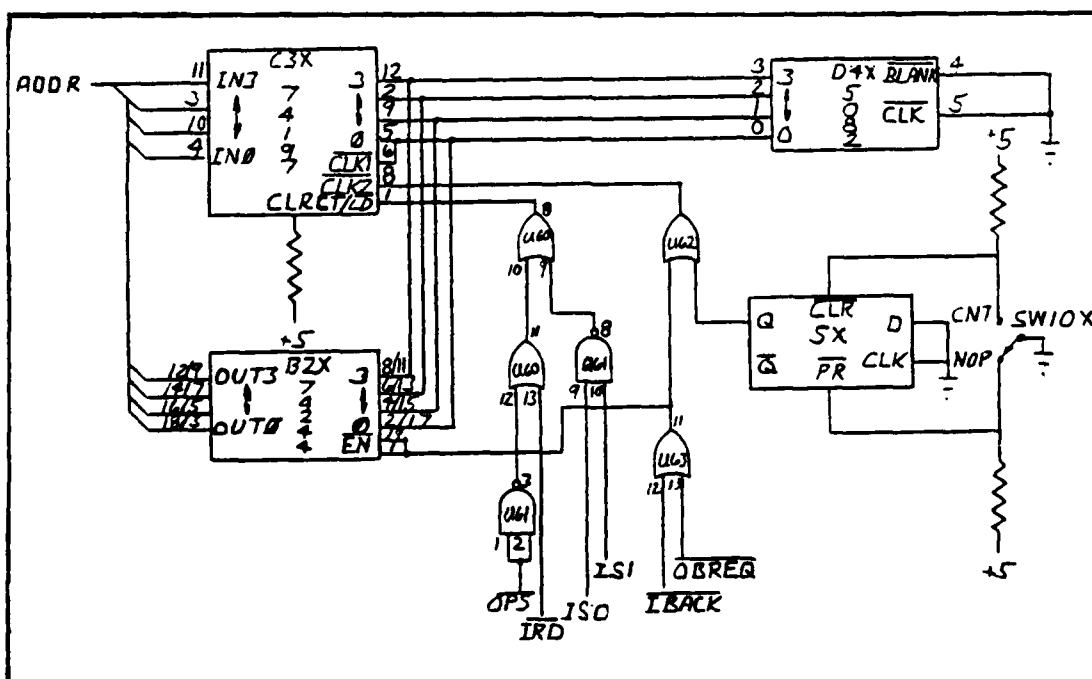


Figure 21. Address Bus Monitor.

Figures 20 and 21 show circuits for monitoring and initializing four bits of a bus. In both diagrams IN0 represents the least significant bit of the four bit group. B2X, C3X, D4X, SW10X, and the OR gate connected to CLK1 of C3X are reproduced twice for the data monitor and four times for the address monitor. This covers all 24 bits of the data and address busses. Other logic gates shown in the figures determine when particular components are enabled.

Figure 20 shows the circuit for monitoring/initializing the data bus. During program execution, the logical OR of OBREQ* and IBACK* is always one. Consequently three significant actions occur. One is that the hexadecimal display, D4X, is blanked. Another is that the toggling action of SW10X is blocked from C3X. The third is that the output buffer, B2X, is disabled. When the RDT is in I/O mode, OBREQ* and IBACK* are zero. The result is that D4X is no longer blanked and SW10X increments the C3X counter. Combined with a write request, WR* = 0, OBREQ* and IBACK* also enables C3X to be incremented and allows its output to pass to the data bus. During read operations WR* = 1 and again B2X is disabled. However, C3X is enabled in its latched mode, passing information from the data bus to D4X.

Figure 21 shows the circuit for monitoring/initializing the address bus. When the RDT is in its program execution mode, C3X acts as a latched buffer, passing appropriate information to and blocking undesirable bus activity from D4X. During execution of an instruction, the address and

data busses change several times. Consequently, control signals determine the proper time for latching information into C3X. The desired information is available when IS0 and IS1 are both high, indicating an operation code fetch cycle, and RD* is low (Ref 24:4-13). Under these conditions CT/LD* equals zero and bus information is latched into the counter.

When the RDT is in its I/O mode OBREQ*, IBACK*, IS0, and IS1 are low, and the CT/LD* pin of C3X is high. This disables additional information from latching into the counter from IR busses, and allows the IR address bus to be initialized. Initialization involves incrementing C3X to a desired value using SW10X. The OR gate connected between SW10X and C3X stops count pulses from reaching C3X unless the RDT has control. So, even though C3X is usually count enabled (CT/LD*=1), count clock pulses (CLK1*) are blocked from C3X unless the RDT is in an I/O mode.

IR Reset. The NSC800 and its peripheral controllers are reset whenever the RESET-IN* pin of the CPU is grounded. Figure 22 is a schematic of the circuit used to ground RESET-IN*. The diagram consists entirely of a switch debouncer.

A data FF with preset and clear inputs works well for switch debouncing. With the CLK input tied low, data inputs to the FF are disabled and output is dependent on only the preset and clear inputs. At any time only one of either the preset or clear inputs is low. The output of the FF reflects the switch position. When the switch is changed, voltage

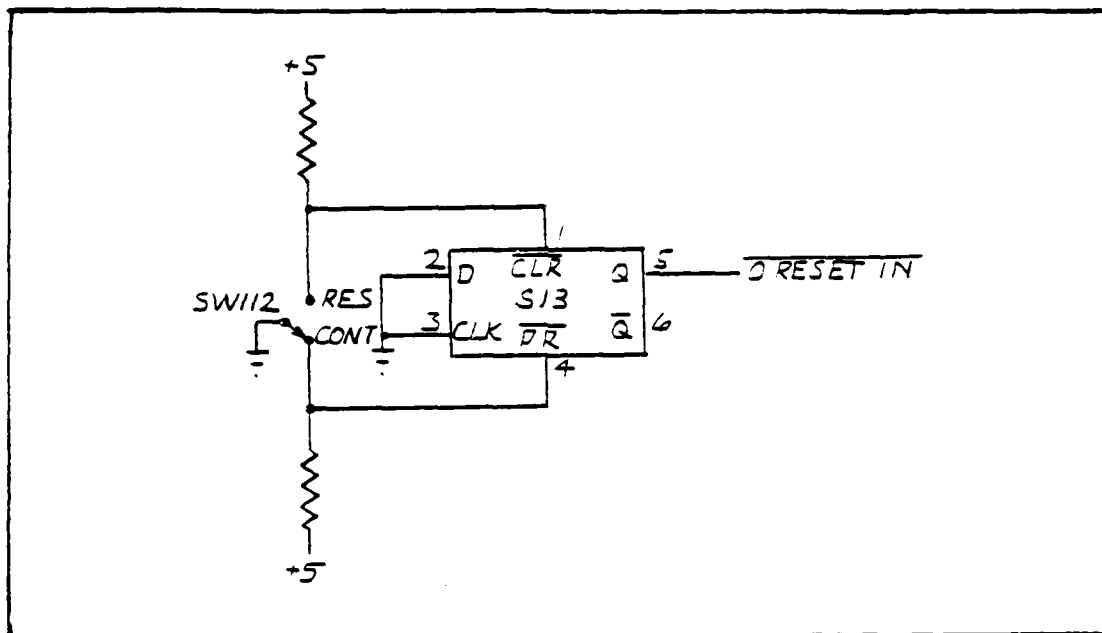


Figure 22. IR Reset Function.

spikes appear as the switch disconnects from one terminal and as it connects to the other. These two causes of spikes are mutually exclusive. So, the FF reflects switch positioning without intermittent voltage spikes.

Single Step. The power save feature of the NSC800 allows implementation of a single step function. During the last clock cycle of each instruction, the PS* pin of the NSC800 is sampled; and when found in a low state, program execution is suspended. The NSC800 Microprocessor Family Handbook suggests a way of using this feature to control a single step function. (Ref 24:4-23)

In general, single stepping works by holding PS* low until time for a step. Then PS* is set high, allowing program execution to continue. Before the current

instruction completes, the RD* strobe from the operation code fetch cycle clears PS* and again execution is suspended. The result is that only one instruction is executed every time PS* is toggled high.

Figure 23 shows the circuit used for implementing single stepping within the RDT. The circuit effectively works as outlined above. However, RDT complexity requires that enhancements be made to tailor single step functioning.

The first enhancement provides a switch to allow a choice between normal program execution and single step execution. In its "RUN" position, the switch provides a high input to two OR gates. This effectively blocks single step actions by maintaining OPS* and OWAIT* high. In its "S/S" position, a low signal is input to the blocking gates, allowing step toggling to control OPS* and OWAIT*.

The requirement for a wait state to be generated externally from the IR results from the interaction of the ALE pulse generated by the NSC800 and the wait state generation circuitry. ALE is held high whenever the NSC800 is in a power save mode, PS* = 0 (Ref 24:4-23). But wait states are valid for only one machine cycle after ALE goes high. The facts that a single step operation extends across many machine cycles while PS* = 0, and the first CPU operation performed after PS* goes high is an operation code fetch from EEPROM, require that an external wait state be generated.

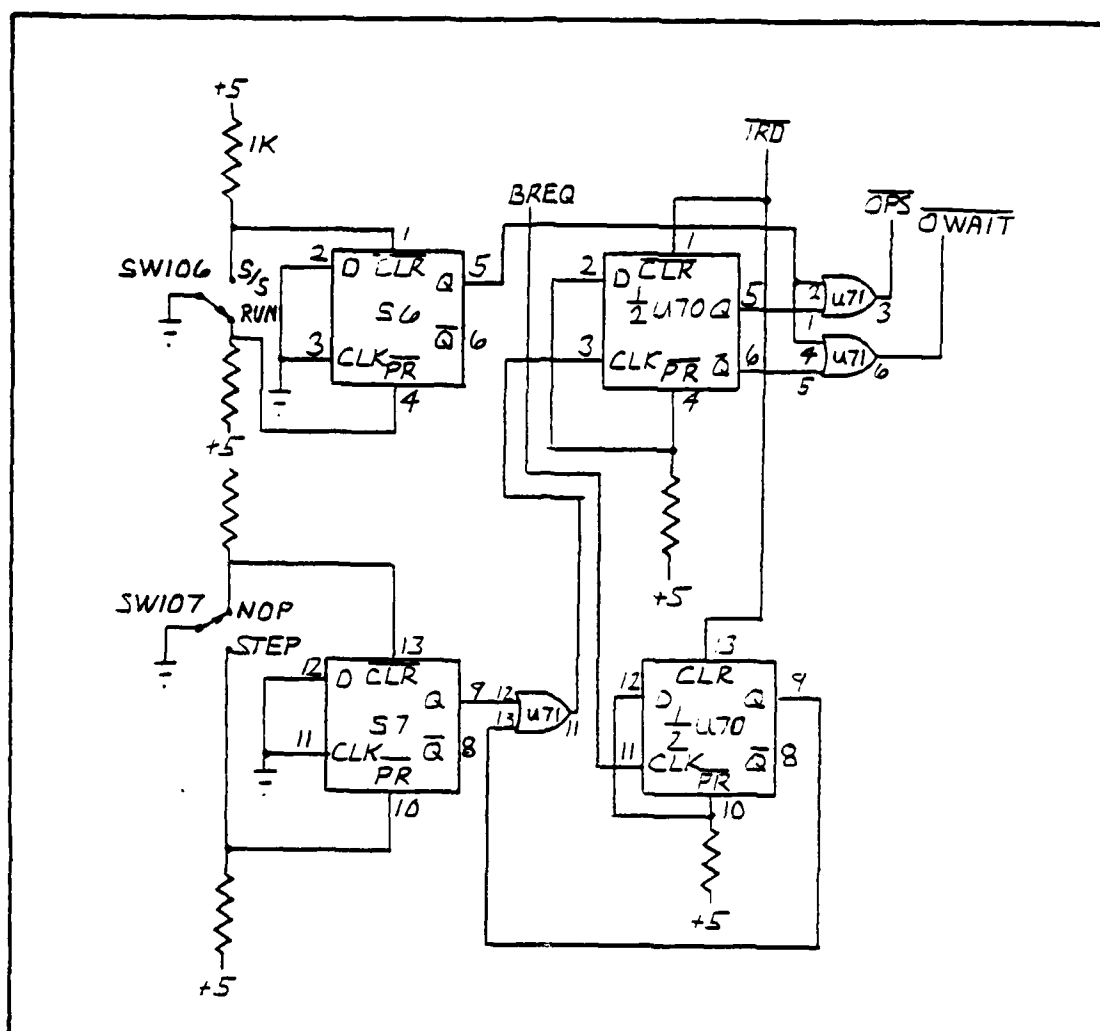


Figure 23. Single Step Function.

Another enhancement involves RDT requests for control of the IR bus structure. Since the NSC800 only samples BREQ* during the last clock cycle of an instruction (Ref 24:4-12), at least one instruction must execute before bus control is relinquished. The lower right-hand FF of Figure 23 is the component which insures at least one is executed. Upon activation of BREQ, a one is latched into this FF. The one

then passes through an OR gate, causing the upper right-hand FF to latch a one onto OPS* and a zero onto OWAIT*. Both latches are reset by the RD* strobe which originates during an operation code fetch cycle. While this action insures one instruction is executed, all is in vain if SW6 is set to "RUN". Either way, an instruction is executed, allowing OBREQ* to be recognized.

Input/Output. I/O operations can be performed on both memory and peripheral devices. Setting either SW108 or SW109 selects a type of I/O device and enables RDT I/O. SW110 and SW111 determine the type of I/O operation and when it will be performed. Figure 24 shows the I/O portion of RDT circuitry. For discussion, Figure 24 is divided at output pin 6 of U81. This splits the diagram into a bus requesting circuit and an I/O strobe generating circuit.

Before an I/O operation can proceed, the RDT must gain control of the IR busses. The first step in getting control is to request it by setting OBREQ* low. When SW108 = "MEM" or SW109 = "PER", one of the switch debouncers will cause the pin 10 of U82 to change from its normally high output state. A low output from U82 is used as the bus request signal - OBREQ*. The IR processor recognizes that OBREQ* = 0 before fetching another instruction, and responds by setting IBACK* low. This response indicates that the RDT has control of the IR busses and causes output pin 6 of U81 to go low. This low output enables the I/O strobe generating portion of the the diagram.

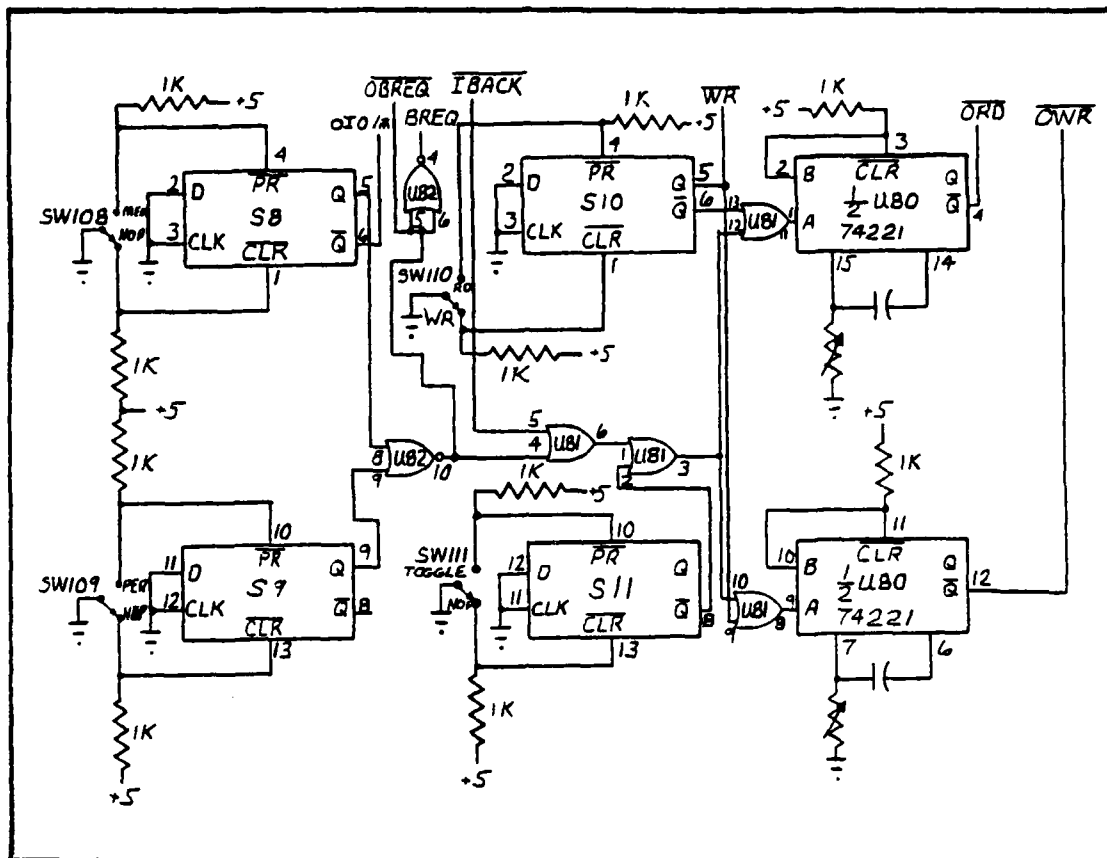


Figure 24. Memory/Peripheral I/O Circuit.

SW110 determines whether a read or a write will be performed by allowing toggle pulses to reach an appropriate 74221, one-shot. Once enabled by the OBREQ*/IBACK* sequence, pulses from the SW111 momentary switch are applied through these enable gates to the falling edge triggers of one-shots. Outputs from the one-shots are pulses of known width that are used for the ORD* and OWR* strobes. The width of each strobe is determined by the I/O circuit with the longest pulse requirements. EEPROM's, with a typical access time of 500 nanoseconds (Ref 12), require the longest read strobe of any

memory or peripheral circuit. Allowing for possible atypical operation, the ORD* generating one-shot is tuned to 600 nanoseconds. The OWR* strobe width is set at 200 nanoseconds. This time is governed by the NSC810 I/O port (Ref 24:A-27), the slowest device that can be written to by the RDT.

IC Map

In general, IC's are grouped by the RDT function they support. Figure 25 illustrates the relative position of IC groups as they appear on the RDT wirewrap card. In addition, naming conventions used in previous schematic diagrams help identify IC functions. Letter prefixes and their meaning are:

- B = Buffer,
- BC = Bus Connector,
- C = Counter,
- D = Display,
- R = Resistor Pack,
- S = Switch Debouncers,
- SW = Switch, and
- U = Individual Operations.

The "U" group is further broken down so that

- U6 = Display/Initialize Operation,
- U7 = Single Step, and
- U8 = I/O Operation.

Table XVI is a more definitive list of the IC functions used in the RDT.

IC sockets on the RDT wirewrap card do not contain the prefixes described above. Instead, only the number following the letter prefix is found on the sockets. Numbering

consistency provides the correlation between the schematic diagrams and the wirewrap sockets.

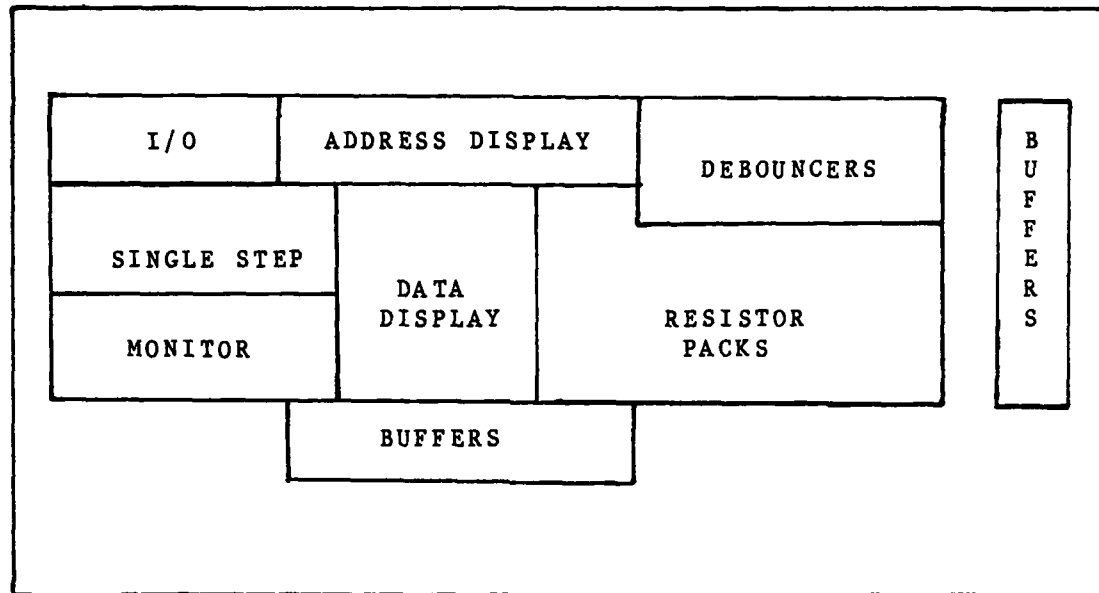


Figure 25. RDT IC Functional Groupings.

TABLE XVI
RDT IC Listing

Device Type	Functional Designation	Schematic Reference
5082	Hexidecimal LED Display	D40-D45
7400	Quad 2-input NAND Gates	U61,U81
7402	Quad 2-input NOR Gates	U60,U82
7417	Hex Open-Collector Buffer	B28
7432	Quad 2-input OR Gates	B29,U62, U63,U71
7474	Dual D-type Flip-Flops	S1-S13,U70
74197	Presettable Binary Counter	C30-C35
74221	Dual Monostable Multivibrator	U80
74244	Octal Tri-State Buffers	B20-B22
82PC08	Bidirectional Transceiver	B23-B27
	1K x 8 Resistor Pack	R90,R92, R94,R96-R99

Appendix E

Manufacturers' Data Sheets

This appendix contains manufacturers' data sheets for the IC components used in the IR prototype. However, they are not published with the thesis. Instead, they are on file at AFIT/EN, Wright-Patterson AFB, OH, 45423.

VITA

Robert Eugene Meisner was born on 10 September 1952 at the Carlisle Barracks, Pennsylvania. Being a member of a military family he attended many schools before graduating from high school in Olla, Louisiana. Continuing his education in Louisiana, he earned a Bachelor of Science degree in Computer Science in May 1974. Upon graduation, he recieved a commision in the US Army through the ROTC program. While in the Army he held positions as an AUTODIN terminals programmer, a company executive officer, and a battalion supply staff officer. In August 1977, he recieved an interservice transfer to the USAF and was assigned to Hq SAC. He spent his entire tour as a computer systems analyst, supporting SIOP production before being accepted to AFIT. He entered the AFIT, School of Engineering in June 1980.

Permanent Address: 29 Halsey Drive

Marietta, GA 30062

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/82M-5	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN INFLIGHT RECORDER PROTOTYPE FOR THE INFLIGHT PHYSIOLOGICAL DATA ACQUISITION SYSTEM III		5. TYPE OF REPORT & PERIOD COVERED MS THESIS
7. AUTHOR(s) Robert E. Meisner, Captain, USAF		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson Air Force Base, Ohio 45433		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS School of Aerospace Medicine Crew Systems Division (SAM/VNB) Brooks AFB, Texas 78235		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE February 1982
		13. NUMBER OF PAGES 271
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES APPROVED FOR PUBLIC RELEASE: LAW AFR 190-17 <i>Lynn E. Wolaiver</i> LYNN E. WOLAIVER Dean for Research and Professional Development AIR FORCE INSTITUTE OF TECHNOLOGY WRIGHT-PATTERSON AFB, OHIO 45433 25 JUL 1982		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Inflight Physiological Data Acquisition System (IFPDAS) Complementary Metal-Oxide Semiconductor Electrically Erasable Programmable Read-Only Memory Magnetic Bubble Memory Remote Data Acquisition Microprocessor Microcomputer Analog to Digital Conversion		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) See reverse		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

Block 20.

A prototype for the Inflight Recorder component of the Inflight Physiological Data Acquisition System was built. The Inflight Recorder is a remote data acquisition computer for sampling physiological data. Characteristics of the recorder's design were solid-state, microprocessor controlled, expandability, 16 sensor inputs, and 122 samples per second. Demonstration of battery operation for four hours and unobstructive size characteristics awaits further testing.

Following a hardware requirements analysis, the prototype was built using Complementary Metal Oxide Semiconductor (CMOS) integrated circuits. Components featured in the design were a CMOS microprocessor; Electrically Erasable Programmable Read Only Memories (EEPROM); a monolithic, 16 channel, analog to digital converter; and Magnetic Bubble Memories (MBM).

In addition to building the IR prototype, several development tools were constructed. One was a EEPROM Programmer. Another was an MBM Interactive Development System. A third was a hardware front panel for debugging IR software. User's manuals for these tools appear in appendices to the thesis.

B

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

ATE
LMED
9-8