IMPLEMENTATION OF A ZILOG Z-80 BASE REALIZATION LIBRARY FOR THE COMPUTER SYSTEMS DESIGN ENVIRONMENT (U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA T J SMITH MAR 84
THESIS

IMPLEMENTATION OF A ZILOG Z-80 BASE REALIZATION LIBRARY FOR THE COMPUTER SYSTEMS DESIGN ENVIRONMENT

by

Theodore John Smith, Jr.

March 1984

Thesis Advisor: Alan A. Ross

Approved for Public Release; Distribution Unlimited
# Implementation of a Zilog Z-80 Base Realization Library for the Computer Systems Design Environment

**Authors:** Theodore John Smith, Jr.

**Performing Organization Name and Address:** Naval Postgraduate School, Monterey, California 93943

**Controlling Office Name and Address:** Naval Postgraduate School, Monterey, California 93943

**Report Date:** March, 1984

**Number of Pages:** 106

**Security Classification:** UNCLASSIFIED

**Distribution Statement:** Approved for Public Release; Distribution Unlimited

**Supplementary Notes:**

**Key Words:** Computer aided design, CSDL, Z-80, realization, primitive monitor

**Abstract:** This thesis develops a Zilog-80 based realization library for use in the automated design of microprocessor based control systems. The library is designed around Standard Bus boards. This bus is supported by a number of manufacturers for use in breadboard construction, through a number of standard cards. The library of primitives developed implement the construction used in Computer System Design Language (CSDL) for (Continued)
ABSTRACT (Continued)

the Z-80 cpu. CSDL is a high level language that allows the specifications of tasks and procedures, and their time constraints. Use of the design system and this library can enable Z-80 based prototype controllers to be quickly designed and built.
Implementation of a Zilog Z-80 Base Realization Library for the Computer Systems Design Environment

by

Theodore John Smith Jr.
Major, United States Army
B.S., Loyola University, 1970

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1984

Author: Theodore John Smith Jr.
Approved by: Alan A. Ross
Thesis Advisor

Daniel R. Delle
Second Reader

Richard A. Els
Chairman, Department of Administrative Science

Kenneth T. Hannah
Dean of Information and Policy Sciences
This thesis develops a Zilog-80 based realization library for use in the automated design of microprocessor based control systems. The library is designed around Standard Bus boards. This bus is supported by a number of manufacturers for use in breadboard construction, through a number of standard cards. The library of primitives developed implement the constructions used in Computer System Design Language (CSDL) for the Z-80 cpu. CSDL is a high level language that allows the specifications of tasks and procedures, and their time constraints. Use of the design system and this library can enable Z-80 based prototype controllers to be quickly designed and built.
# TABLE OF CONTENTS

I. INTRODUCTION ................................................. 9

II. BACKGROUND .................................................. 13

III. DESIGN ........................................................ 19
   A. THE CURRENT DESIGN SYSTEM .............................. 19
   B. CURRENT PROBLEMS WITH THE INTEL 8080 LIBRARY ... 21
   C. METHODOLOGY .............................................. 23

IV. IMPLEMENTATION .............................................. 25
   A. MONITOR ................................................... 25
   B. ARITHMETIC ............................................... 29
   C. CONTROL STRUCTURES .................................... 31
   D. INTERRUPTS ............................................... 33
   E. INPUT/OUTPUT DEVICES ................................... 36
      1. Onboard Cpu Three Channel Counter Timer Chip .... 37
      2. 8 Bit Analog To Digital Conversion Board ........... 38
      3. 8 Bit Digital To Analog Conversion Board .......... 39
      4. 64 Port 8 Bit Standard Bus To Digital I/O Bus ...... 39
      5. Keyboard/Display Card ................................ 40
      6. Dual UART Board ........................................ 41
   F. TESTING .................................................... 41
8. FORMATTING OF PRIMITIVES

1. Pollock Fixit
2. Ross's Newcsdl
3. Walden's Data Base Input

V. RESULTS AND CONCLUSIONS

APPENDIX A - PRIMITIVE TITLE INDEX
APPENDIX B - GLOBAL VARIABLE LISTING
APPENDIX C - ZILOG-80 REALIZATION LISTING
LIST OF REFERENCES
INITIAL DISTRIBUTION LIST
LIST OF FIGURES

1. Cost Versus Production by Language 11
2. Current Ross Controller Design System 20
3. Monitor for Z-80 Realization 27
ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Lieutenant Colonel Alan Ross, and second reader, Professor Daniel Dolk, for their assistance and support in this thesis. I thank my wife Celeste and Peter for their support and faith in me.
I. INTRODUCTION

During the past ten years there has been a microelectronics revolution in which ever increasing numbers of functions have been put on a single chip. This has translated into a shift in system costs from capital to labor. Hardware is no longer the dominant factor in the cost of a computer system. The rapid drop in the price of general purpose computer chips has also caused a shift in their application. Because of their low cost many of these chips are replacing specialized control hardware.

In the past hardware was extremely expensive. Even the design of a simple controller required a large number of components. Many of these simple logic components cost as much as the single chip general purpose controller today. Because hardware was so expensive, controllers were developed to use the minimum amount of hardware possible, at the expense of a great deal of labor. This took the form of engineers trying to minimize the number of gates or logic functions used to implement the controller. The microelectronic industry has now managed to put a functional computer on a chip for the same cost as a few gates ten years ago.
Two examples of control applications are speed control of a power plant generator and starting control of a gas turbine engine. Control of power plant generators has been done in the past by mechanical controllers. The first example shows the displacement of the mechanical controllers because of the high cost mechanical systems. Even though the mechanical controller was expensive, the degree of control was imprecise. Here the driving factor has been the desire to increase the amount of control possible as well as reduce the cost of the controller. The starting of gas turbine engines has been for the most part manual, because of the large number of malfunctions that can occur during starting. Recently, mechanical and digital starting systems have been developed to start gas turbine engines. The second example shows a new application for a controller, primarily based on the reduced cost of digital component.

Unfortunately the very low cost of the microprocessor cpu is a small part of the total system cost. Currently a microprocessor can cost less than four dollars [Ref. 1: p.536], but the cost of a programmer can exceed $12 per hour [Ref. 2: p.90]. Unless the system being controlled is very costly or the volume produced high, the cost of designing the digital system is not affordable. This cost can also be exacerbated by the choice of programming language. Figure 1 illustrates the costs of programming in a high order language versus assembly language. Note that implicit in this graph is a belief that assembly code will be more
efficient at run time. Current optimizing compilers are becoming more efficient and are approaching the efficiency of assembly code. This has the effect of decreasing the slope of the higher order language total cost line and moving the intersection of the two line to higher production levels. The implication is that except for the most demanding of high production applications, the coding should be done in a high level language.

The second factor affecting the cost of programming is the complexity of the system to be controlled. As the complexity of the control program becomes larger than a single individual can intellectually handle, the program must be subdivided and designed by a team. Currently, hardware is chosen early in the design process and the software is then written for that hardware. Early design errors can cause all subsequent programming to be redone as well as the possibility of having to select different hardware.

![Graph of Cost Versus Production by language](image)

**Cost Versus Production by language**

Figure 1 [Ref. 3: p.432]
The final factor is the speed at which the control must be performed. Some control functions must be accomplished within a critical length of time. This factor competes directly with the previous two factors, cost and complexity. There are two methods to determine if a program is fast enough: program the controller and time the response of the system based on an input or calculate the basic execution time of the instructions in the program. In the first case, exhaustive testing must be used to establish the maximum runtime of the program. The second method is more exact, but is labor intensive. In both cases, the program must be present before the timing can be determined. If the program is too slow it must be rewritten or possibly the hardware must be changed in order to sufficiently increase the system speed.

These problems in design have caused the computer community as well as the controller community to increase the use of simulations and other tools to minimize the risk of these errors. Some have stressed the hardware design, others have stressed the software design, and others have increased the use of models. As costs have decreased, a final group has attempted to design the whole system, including both hardware and software. In this last case we see increased use of prototype development in computer design. In the past this was not done because of the cost of developing the prototype. It is becoming more feasible now because the size and complexity of the final system
makes errors in design too costly to be routinely repaired after the systems are fielded.

This thesis will investigate and develop a library of realizations based on the Zilog Z-80 cpu to support the design system developed by ROSS [Ref. 4: pp.7-8]. Since Ross’s original work only contained a single library, this second library will provide the option of building a realization with more than one library. It will be possible to test the design system’s capabilities to choose a realization library based on a problem statement. The Z-80 hardware will be provided by the Standard Bus system of prototyping boards, to make it possible to quickly design and reconfigure prototypes. With the completion of these goals it will be possible to construct a working controller using Ross’s design system and finally verify the accuracy of this design concept.

II. BACKGROUND

Computer aided design has been an evolving process over the past 20 years, intended to reduce the labor required to engineer a product. This process can be described by tracing several different threads. The first thread to be followed will be the design of controller systems. This implies a ready pool of predesigned hardware. The second
area to be looked at will be the design of hardware.

Finally, the design of the complete system will be examined.

Matelan proposed that computer aided design be applied to the design of real time controllers, by adding the use of realization libraries of standard components. He presented a methodology for defining the timing constraints driving a real time controller. This consists of having pairs of contingencies and their associated tasks. By dividing the total problem into paired contingencies and tasks, the individual pairs can be reordered to meet the overall timing constraints.[Ref. 5: pp.17-20]

Ross implemented Matelan's ideas for timing analysis and added the potential for background tasks [Ref. 6: pp.15-22]. The processor chosen for the realization library was the INTEL 8080. This particular processor was chosen because of its availability and low cost. In the course of his thesis, Ross also corrected Matelan's example problem [Ref. 7: p.77]. This came about while trying to debug Ross's timing analyzer. The timing analyzer was correct and Matelan's example was wrong, because Matelan had scheduled a potential second occurrence of a contingency task pair within the time needed for the execution of the first contingency pair. This highlights the need for some automated means of assisting the designer in avoiding similar errors. Ross provided sample executions of the program with accompanying paper hardware realization, however, no actual hardware was built or tested.
Pollock attempted to build an actual hardware realization in the form of a fuel controller for a car. Pollock never did attain this goal in the course of writing his thesis. He did add a variety of additional hardware to the INTEL 8080 realization library to include a floating point chip for floating point operations and transcendental functions. He criticized Ross's FORTRAN implementation of the timing analyzer and functional mapper. He suggested that all of the present programs be scrapped and work begin anew retaining Matelan's theoretical foundations. Much of this criticism appears to be the result of the difficulty in adding additional primitives. This is caused in large part by the column reads performed by the FORTRAN functional mapper on the primitive listing. The structure of the primitive listing itself has pointers that refer to parts of the listing by a relative line displacement that makes changing the primitives tedious. [Ref. 6: p.34]

Manwaring argues for a similar system, independent of Ross and Matelan. His ideas on libraries of implementations for different processors are equivalent to Ross's, but he does not include the selection of the processor by the design system. As in Ross's system he argues for a design system that chooses hardware and the software to run the system, by means of a high level language to state the problem. He also does not consider the analysis of timing in his design system. But he does concede that the compiled code may run too slowly and portions of the software may
have to be manually optimized to meet time constraints. Additionally, his proposed design system does not manipulate the timing problem and realizations to the extent that Ross's does. He proposes generating a compiler error if the desired primitive doesn't exist in a particular processor's library, rather than trying a different hardware realization. He does argue that the rapid generation of programs can make digital controllers available for more applications. [Ref. 9: pp.431-435]

Biehl, also independent of others, presents the system LODE-MIR to design controllers using microprocessors. The input is in the form of a state table and flow diagram. LODE-MIR process the state table and flow diagram to obtain an intermediate language. The intermediate language is then manipulated to optimize the following: the sequence in which the condition variables are tested, the assignment of conditional inputs, the control of outputs to ports and the number of jump instructions. This system is in contrast to Ross's in that there are no time constraints, the system merely attempts to make the controller as fast as possible. The ultimate machine code is just a direct translation of the intermediate code to the target controller code, with no guarantee of a specific response. [Ref. 10: pp.328-333]

Finally, Sherlock studied the problem of making entries of the problem into the Problem Statement Analyzer easier. After a lengthy review of human factors engineering, a program was written to make the input of a problem
formulated in Computer System Design Language into Ross’s
design system easier. [Ref. 11: pp.15-16]

The second thread in the description of automated design
tools is the design of the hardware itself. Chu has been
working in the area of microcomputer design since 1965. He
states that the following need to be described in the
language: identification of the selected LSI, MSI, and SSI
chips, a plan for the interconnection of these chips and a
description of the internal structures and sequences of
these chips. He also describes three levels of increasing
detail in the design process: functional, ideal timing, and
real time levels. He proposes that, given a functional
description, a computer system could be designed and
simulated prior to the actual construction of the system.
The system would need a large data base of chip
identifications and interconnections to function. He
proposes that manufacturers provide disk packs of their chip
descriptions in addition to the data books that they now
provide. [Ref. 12: pp.45-51]

Heath, Carroll and Cwik describe a modified version of
Chu’s Computer Design Language that was running at Auburn
University. Two new declarations were added to allow the
easy implementation of buses and front panel lights. They
conclude that the following can easily be tested using
simulation: basic system organization, function of some
microcode, timing problems, limitations on input, and
throughput rate. [Ref. 13: pp.93-108]
Hartenstein and Von Puttkamer describe the language KARL and its associated graphical description ABL. KARL is a Pascal-like language for allowing simulations of a processor at the register transfer level. It is an improvement over CDL in its original form in that it allows the integration of a graphical description of the design through the use of ABL, a block diagram language. Once inputted, the design can be simulated for correctness and finally the design system can output detailed layout drawing and mask specifications. [Ref. 14: pp.155-160]

The third and final thread is the design of both the hardware and the software. A feasibility study was conducted on the design of an integrated design facility by the U.S. Air Force at the Rome Air Development Center. The initial study was performed by Sperry Univac. The concept included a design facility where total system design alternatives could be emulated for the purpose of providing and evaluating designs prior to actual development. Two important conclusions of the study were: the system could be used for requirements formulation as well as hardware and software design specifications and the present system was inadequate for the analytic determination of operating performance. Further, the study concluded that additional tools should be added to assess operating performance, even though the tools would only provide approximate answers. [Ref. 15: pp.19-20, 253-258]
Having reviewed the current progress in automated controller design, further study on the problems associated with the design system developed by Ross will be done. This is done to limit the scope of the problem to a manageable size.

III. DESIGN

A. THE CURRENT DESIGN SYSTEM

The area of study for this thesis is the design system established by Ross. In this system the hardware is selected from predesigned components. The major goal of the system is the rapid design of prototype controllers. The system is not intended to produce a final packaged production design. It will produce a breadboard controller capable of verifying the feasibility of the controller, its desired characteristics, and timing.

The current design system is shown graphically in figure 2. The library of primitives currently contains only an Intel 8080 realization. The implications of this are that if the controller cannot be designed using an Intel 8080 processor, then the system will indicate that the controller is not possible.
The purpose of this research is to add a Zilog Z-80 based realization to that library. Construction of this second library will give the design system an alternative method of realizing a controller. This will also enable further testing to be done on the design system itself.

Another reason for adding the Z-80 realization library is to allow the actual construction of a test controller from the design system. A thesis currently being conducted by Riley will use the design system and the primitives developed in this thesis to construct a generic gas turbine starting controller [Ref. 16]. Since an actual controller will be constructed using the primitives in this thesis, provisions have been added to the primitives to allow ease of debugging and testing.
So far there has been no complete exercise of the design system. Pollock attempted to build a fuel controller for a car but no hardware was built. Heilstedt studied the problem of using the design system to construct digital filters, but also did not build any hardware. Many of his problems related to converting the program to run on a VAX 11/780. A prime purpose of this thesis is to get a working system that can be used for a complete design test in future projects. With this as a goal, the Standard Prolog breadboard system was chosen as a source of hardware. This particular system is available for testing and offers potential for quick prototype assembly.

B. CURRENT PROBLEMS WITH THE INTEL 8080 LIBRARY

The current realization library, based on an Intel 8080, has its origin in Ross's original thesis in 1978. As mentioned before, the library has been modified and expanded by Pollock and Heilstedt, with specific projects in mind. In the six years since the library's inception the cost of hardware has continued to decrease and its power increase. The change in orientation can be seen in the types of primitives put in the realization. Pollock added a great deal of hardware, including a floating point processor, to the 8080 library. The Zilog Z-80 was chosen as a newer chip that could add a higher performance library to the design system. It was also chosen because of its popularity and similarity to the Intel 8080.
The floating point processor added to the Intel 8080 library confuses some of the design issues in using the design system. By adding the floating point chip the cost of the system is greatly increased. Because of their infrequent use, floating point chips do not have a production volume that has allowed the general purpose processing chips to decrease in price. The use of the floating point chip will be eliminated in the Z-80 library. An alternative to this in a hierarchy of processors is a high performance library being designed around the Intel 8088 by Cletal [Ref. 17]. Because of the hardware instructions available, this chip may offer similar performance with reduced cost.

The current Intel 8080 library does not treat negative numbers correctly. The comparison, multiply and division primitives all were written only with positive integers in mind. The Z-80 library will incorporate code that also provides correct operations with 2's complement arithmetic.

The monitor was initially designed to be used without a stack. Because of that the monitor, when calling a routine, used an intermediate table to determine where the task was located. Later Pollock added a stack to the primitive listing, but the monitor structure was never completely modified to take advantage of it.

Finally, there is no protection from the propagation of errors. Should an underflow or overflow occur, the sign of the result will be incorrect and no action is taken to
minimize the effect. In terms of control, it is possible that the item controlled will be directed to perform the exact opposite action from what is required. As an example, if a positive number opens a valve and a negative number closes the valve, then an overflow will cause the valve to be closed at a time when the control program is trying to open the valve wider.

C. METHODOLOGY

To capitalize on the effort that has gone into the Intel 8080 library, each of the primitives will be reviewed. If the primitive is still required, then the initial draft of the Z-80 primitive will be a direct translation of the 8080 code to Z-80 assembly code. Speed improvements will then be attempted using the additional instructions available to the Z-80. If there is a tradeoff to be made in speed or code size, then speed will be chosen. This will eliminate the use of the jump relative instruction, since it is slower but more compact than the jump absolute instruction.

All loop and control structures in each primitive will initially be made with labels to facilitate writing the primitive. Then each primitive will be incrementally tested using a debugger. When the primitive works correctly, the labels will be replaced with relative assembly jumps to insure portability in the actual use of the primitive. Each primitive will then be tested again with a debugger to insure that the primitive still functions correctly. In
testing a primitive all paths through the code will be exercised. To do this numbers from a representative class will be chosen and entered through the debugger.

The floating point processor used in the Intel 8080 library will not be used in the Z80 library. The floating point arithmetic operations will be implemented in software. To insure compatibility with other computers, the IEEE single precision floating point standard will be used as an interface form for external information. Keeping with the spirit of the standard and the intent of the control system, overflows will be given a value of infinity. Though not discussed in the floating point standard, conversions of infinity or numbers larger than the range of an integer value will be converted to the largest integer value representable. This in an attempt to minimize the propagation of opposite control responses.

The completed primitives will be aggregated into a single library file and then processed into a format acceptable to Ross's Design system by use of Pollock's formatting utility. This will allow the primitives to be individually tested. Being able to add a few primitives at a time and then have them formatted by Pollock's formatter program can isolate any problems introduced by the newly added primitives. Using Pollock's formatting program enables a primitive library to use Ross's design program directly, without having to make tedious computations of various pointers. Walden's data base storage [Ref. 18] of
the primitives will not be tested by this library, in order
to minimize the amount of change to the design system and
because it does not allow the entry of code from a file.

Appendix A lists the names of the primitives and a brief
description of their purpose. This list has been compiled
to reduce the volume of data that must be scanned to
understand the coverage of the library primitives. It
should be noted that in contrast to the Intel 8080 library,
many of the hardware primitives are eliminated in favor of
software primitives. Further, the aggregation of the
hardware primitive is now at the board level, rather than
the component level.

IV. IMPLEMENTATION

A. MONITOR

The original monitor structure used by Ross and Pollock
consists of an infinite loop of contingency task pairs. The
order of the pairs is determined by Ross's timing analyzer
based on the timing constraints of the various tasks.
Control is transferred by the use of an intermediate table
which contains the actual address of the task to be
executed. When a task is completed, control is then
returned to the task loop. A task counter is incremented
and the next task is executed via the intermediate table.
When all tasks in the loop have been executed once, the task
counter is reset to point to the first task in the loop and
the process repeats itself. This structure has its source in the original library developed by Ross which did not have a stack. The lack of a stack prohibited the use of calls and returns for subroutines. Later a stack was added and subroutines used to transfer control to the contingency test, but the monitor was not changed.

The Z80 monitor, shown in Figure 3, consists of a main loop that has a single entry and no exit. Switches cause various submonitors to be executed from the main monitor loop. This was done to minimize testing of conditions that are mutually exclusive. Timing analysis is done on the submonitors plus the time to execute the main monitor loop with all switches false except one. All tasks, conditions and procedures are executed as subroutine calls from the polling loop in the submonitor. Initialization is handled as a submonitor without any timing constraint. The initialization will be executed first and then the initialization switch set to false to preclude initializations from being executed on subsequent iterations of the main monitor loop. Since all submonitors return to the top of the main monitor loop, the top of the main monitor loop resets the stack pointer to eliminate any pending operations in the submonitor loop. This also allows error handling to be executed by jumping to the top of the main monitor.
The library supports testing the monitor prior to building hardware and programming eroms by the use of two switches set in GLOBALS.DAT. These switches are the DEBUG
and NOROM switches. Three modes are allowed: Standard Board
with loading rom, program in an ALTOS CP/M computer I/O
through standard BDOS calls, and Standard Board without
loading rom. The default mode is for both of the switches
to be false. This allows loading of programs for test into
the ram memory of the standard board system, using an ALTOS
computer. The BDOS jump location is also defined in
GLOBALS.DAT, to allow changes for computer with a BDOS jump
location other than 5.

By setting the DEBUUG switch to true to activate
conditional assembly of test components, the conditional
assembly relocates the beginning of the monitor to 100h. In
the standard board system this area would normally have the
loading monitor. That monitor has provisions for offsetting
the interrupt locations to 4000h. To keep the rest of the
program in the same relative position during the debug mode,
a jump instruction is put at 100h to jump to 4000h. This
will cause all code to be identical above 4000h to the
breadboard system. This allows the use of a CP/M-based
machine to test the software prior to hardware construction.
The top of ram, where the stack would normally be located,
is moved down to the top of the temporary program area in
CP/M, to location 32767. Inputs that would normally be an
I/O request to a input board will be transformed into BDOS
calls for input at the keyboard. Outputs will be handled as
a BDOS request for display to the console. In both cases
the BDOS call includes a description of the board the input
or output would be going to as well as the value of the respective input or output.

In contrast to the test monitor the working monitor will have its initialization assembly begin at 4000h. This is the normal starting point for the NPS Prolog loading rom. This location was chosen to allow ease in starting the monitor program.

If a rom is actually burned and the nps loading rom is removed, then the conditional assembly must start at 66h. If a non-maskable interrupt, (generated by a reset button), is issued to the system, it will begin executing code at 66h. This permits the user of the system to use the reset button as a start button as well as a trouble button. The ROM is banked to start at 0000h, the locations 0000 to 0065h, normally unused. These locations will have no operation codes placed there. Any maskable interrupt can then be accidentally triggered without crashing the system. The maskable interrupt would merely jump to 38h, and do NOP’s until it reached the initialization routine for the monitor.

B. ARITHMETIC

The format for the single byte integer operations used by the 8080 realizations has been retained in the 180 realization. It is a two’s complement arithmetic. The add and subtraction are essentially the same as in the Intel
library. The multiplication and division have been changed, since negative numbers were not correctly implemented. In addition, the option of checking the single byte operations has been added by checking for an overflow after performing the indicated operation. If an overflow occurs, the result will be set to the largest representable positive or negative number based on the sign of the result.

The format for the double byte integer operations is the same as the Intel 8080. The format is stored with the least significant byte first in memory. By using this format, the double byte operations in the Z80 instruction set can be used directly. This particular hardware instruction is used because it takes less time to fetch two bytes with one instruction, than fetching a single byte at a time and using two instructions. Once again the addition and subtraction are the same as the Intel 8080, since the same machine instruction is used for the operation. The multiplication and the division have been changed to treat negative numbers correctly. The check for overflow is done in a similar manner as the check done on the single precision arithmetic.

The format for the floating point operations is a departure from those done in the Intel 8080 library, since a floating point chip is not used in the Z80 library. The original approach was to do floating point using the IEEE standard single precision format. Further study of this approach has shown it to be impractical for two reasons. The first reason is the Z80 is a two's complement machine
and the IEEE format requires sign magnitude operations. The second reason is the packing of the IEEE format across byte boundaries. To use the IEEE format would require converting the sign magnitude number to two's complement and shifting the number to a usable format. This would have to be done before and after every arithmetic operation. To eliminate the overhead of the transformation, a primitive has been made to convert the format used in the realization to and from the IEEE format. The cost to implement the change has been the use of an additional byte of storage. The format used for the Z80 floating point is:

Exponent sign mantissa

39....32 31 30.....0

The exponent is an eight bit number as in the IEEE standard, however it does not have an offset. Instead it is represented as a two's complement number. The mantissa is represented as a four byte two's complement number. The mantissa is fetched and stored two bytes at a time, so the position of bytes 1, 2, 3, 4 are stored as 2, 1, 4, 3. The additional byte is necessary to preserve the accuracy for rounding to the IEEE format. The leading 1 is expressed, rather than being encoded as in the IEEE format.

C. CONTROL STRUCTURES

The control structures can be divided into two categories, selection in straight line code and subroutines. The distinction is made because of the manipulation that
Ross's timing analyzer does with the order that conditions are tested and tasks selected. In the first category four constructs are supported: IF, While, JUMP-ON-TRUE, and JUMP-ON-FALSE. The IF and the WHILE support the CSDL language constructs directly, that is, there are primitives in two parts that will directly perform either of the two selections. They are used by placing the first portion of the primitive before the conditionally executed code and the second part of the primitive directly following the code. In the case of the IF, if the condition is not true, then a jump is done to the code immediately following the second portion of the if primitive. The JUMP-ON-TRUE, and JUMP-ON-FALSE support the CSDE compiler with more primitive operations for compilation into higher constructs. In all of these primitives, selection around straight line code is involved, that is, these primitives will cause a section of code to be included based on a condition, but will not directly support a construct such as an "else".

The second general category contains the control structures effecting subroutines. They are TABENT, PROC and two versions of EXITPROC. The purpose of these control structures is to allow the timing analyzer to manipulate the order that conditions/tasks are called to guarantee the maximum timing. To manipulate the timing the timing analyzer will change the order in which condition/tasks are polled as well as duplicating some condition/tasks to insure that they are polled often enough to ensure the timing
guarantee. This manipulation is done by making successive entries using the primitive TABENT, which will cause a entry of a condition/task in the polling loop. The primitive TABENT uses subroutines in two ways. First it uses an unconditional call to a subroutine to evaluate a condition. After returning from the conditional evaluation, a second subroutine call is made to the task based on the results of the condition. This is a change from the method that Ross used in constructing his polling loop. He chose to minimize the amount of ram required at the expense of increasing the number of jumps required to execute a contingency/task pair. The primitive combines the functions of Ross's primitives TABENT and TABACCP2 into a single primitive. The increase in memory will only become a problem when there are great differences in the timing requirement of different contingency/tasks. The great difference will cause the timing analyzer to make multiple entries of the contingency/task with the shortest time constraint. PROC marks the beginning of a subroutine. EXITPROC provides both a conditional and an unconditional return.

D. Interrupts

The use of interrupts was not implemented in the 8080 library. Interrupts can offer the advantage of faster response to a particular contingency, but they add additional execution time to the contingency/task pairs in the form of interrupt overhead. Since the very nature of
the design system is guaranteed maximum timing, the faster response to a particular task is not a major issue as long as the overall timing constraints are met.

The non-maskable interrupt is used by the Z-80 library as a means to start the controller. This is implemented by starting the definition of the variables, input/output/ports, hardware initializations and stack initialization at 66h, the non-maskable interrupt location in the Z-80. From here the execution will take the program into a continuous polling loop of the contingency task pairs.

A problem arises in trying to control tasks where speed is computed by the design system. Currently, the design system guarantees that a task will be tested within a certain time constraint, but it can be tested earlier. This can occur when all the conditions are false and no tasks are performed. In this case the only time required to execute the polling loop is the time to do the actual test of conditions. Within the structure of the primitives there is no way to make them run in a fixed amount of time, since the arithmetic operations take a different amount of time depending on the particular input values. Also the polling loop's overall timing changes depends on how many of the tasks are executed. The only guarantee the system gives is that the time the monitor will take to execute all the tasks and tests of conditions will be less than some maximum. An external reference is needed to insure a fixed amount of
time has elapsed. In the case of the 8080 library a clock
was added to the system to give an external reference. To
minimize the cost and keep the system simple the Z-80
library uses the CTC chip. The particular standard board
used contains a counter timer chip with three channels as
well as the Z80A and provisions for adding on-board memory.
One of the channels in the CTC is used to generate a
maskable interrupt in a fixed period of time. This
interrupt will be handled by incrementing a word in memory
that represents the current time. The details of the
implementation of the clock interrupt primitive are in
appendix C. Brief conditions will be recorded by using
another channel on the CTC to count the occurrences
independent of the operation of the cpu. This is implemented
through another primitive that connects the external signal
directly to the third channel of the CTC chip. A second
primitive is used to read the counter in the channel and
reset the counter to zero.

This particular clock primitive has an interrupt rate of
a millisecond. It is included for use, but has a major
disadvantage in overhead. The interrupt service routine
requires 100 clock cycles to implement. This equates to
2.5% of the available monitor time. The interrupt rate can
be reduced to every 10 milliseconds for an overhead of .25%.
This in turn makes the polling loop longer to accommodate
the accuracy of the clock. Because the exact point of
interrupt cannot be determined prior to execution, all of
the times computed for the contingency/tasks must have the interrupt service time of the clock added to their maximum time. Several executions of the interrupt are required to built up some accuracy in the computation of the stored time in 10's of milliseconds. This would be adequate for events that are very slow (in the realm of seconds), but inadequate for those taking fractions of a second.

E. INPUT/OUTPUT DEVICES

The selection of input or output software primitives has an additional effect of adding an associated hardware primitive. The hardware primitive in turn require some initialization prior to operation. The hardware initialization is done before any user defined initialization in order to hide the details from the user. Since the user is not required to consciously select all the I/O references in the very beginning of his program, an I/O board can be added with any I/O reference. A method to add the required initialization is needed that could be placed anywhere without effecting the runtime execution of primitives. The compilations of the hardware and software is currently done in a single pass. This also requires that the actual initialization code be added to the output code file at the time of the request. A linked list was used to allow the random placement of any required hardware initializations. This does have an undesirable effect on timing. To keep other segments of code from executing the
initialization, a unconditional jump is used before the initialization. Since it isn't known when this is executed by the monitor as part of a condition, the hardware adds the time for an unconditional jump anytime an initialization is required.

1. **Onboard Cou Three Channel Counter Timer Chip**

   The solution to the interrupt overhead dilemma was to eliminate the interrupt driven clock. The second clock primitive uses two of the CTC chip's channels. One is used as in the interrupt case to create a one millisecond clock. The second is used in lieu of a memory location and an interrupt service routine. The second channel's clock input is tied to the output of the one millisecond channel. The capacity of the channel is 65,536 milliseconds or just about a minute. This short time period is a problem for some applications that may require more than a minute of running time, so a 25 millisecond clock primitive was added. The construction was the same as the 1 millisecond clock except for the variable loaded into the channel 0 counter. The other service primitives remain the same, except that the magnitude returned represents a different elapsed time.

   There are two additional primitives related to this clock: one to read the time and one to reset the clock to 0. This leaves one channel in the CTC on the cpu board for other purposes.
The remaining channel is used to count fleeting external events that potentially take fractions of a monitor loop execution. That channel is initialized and read/reset via another primitive. Since the CTC chip was included with the cpu board and contains only 3 channels, the primitives will allow a single counting channel and a clock or three counting channels. No provisions were made in the library to add additional counter timer chips nor to multiplex the channel over several inputs.

2. 8 Bit Analog To Digital Conversion Board

Input of analog voltages was done by an analog-to-digital primitive using a MOSTEK MDX-A/D8 board. The accuracy of this board is limited to 8 bits. The primitive will allow up to 32 analog signals, using two separate hardware boards. Each of the requested signals are counted using a global variable NATODE. This cannot be used directly since each board has a single port address. To decode the address a second variable NATODP is used to indicate the actual address used. Up to 16 signals can be multiplexed to that port. For signals less than a board’s capacity, the NATODE variable was used to compute the proper pin for the input signal. For signals greater than one board, 16 was subtracted from the NATODE and that number was used to assign the input signal to the second board. To keep from changing the actual value of NATODE, a scratch variable is changed and that value is printed in place of NATODE. The hardware configurations of the board are
controlled by the hardware primitive if a board is required, however, the analog signal assignment to that board was done by the software primitive. This is in contrast to the other primitives where all the signals were assigned pin by only the hardware primitive. This was done in the software listing, because of the multiple signals assigned to a single board.

3. 8 Bit Digital To Analog Conversion Board

Output of analog signals is planned through a digital to analog standard bus board. The configuration of the board is analogous to that of the analog to digital board. The signal capacity is also planned to be the same.

4. 64 Port 8 Bit Standard Bus To Digital I/O Bus

This board was intended to be used to turn devices on and off. However, the MOSTEK MDX-DIOB1 board was found to be merely an interface from the Standard Bus to the Digital I/O bus. The board was designed to multiplex 64 channel through a single standard board card. The board has the capability to send or receive 8 bit wide signals to a MOSTEK digital I/O board. This is the capability that was desired when the board was purchased. However, the digital I/O board that actually controlled external inputs was not purchased. To retain some of its intended capability, turning leds on, the address lines were wired to turn on lamps when a particular addresses was outputted. This is just an intermediate primitive to make use of the board until the digital I/O board can be purchased. Because the
light is only illuminated when the address is strobed, this is not an adequate primitive for a working system. To keep the light illuminated long enough for a person to see the lamp, the address is strobed for several seconds, making the execution time for this primitive long.

5. **Keyboard/Display Card**

This board was added to the realization library to provide limited front panel access to the controller. This particular board has a programmable key pad from input and light emitting diodes and segment alphanumeric output capability. Because of the many functions on the card, each function is provided as a separate primitive. The card is include only once and is initialized in the hardware primitive. The two rocker switches are tested using primitive S.ROCKER. The board also contains eight light emitting diodes that are controlled using primitive S.OUTLED. In the case of S.OUTLED the individual LED is turned on or off based on a boolean value. The Keyboard can be defined using S.INKEY. Pressing the key associated with a boolean flag causes the flag to be complemented. All three of these primitives assign the key or light in the software primitive. None of the primitives associated with the keyboard/display card will add any additional boards, but will issue an error message if more lights or keys are requested than are available on a single board. The last primitive associate with this card is S.OUTDIGIT. This primitive will print a message to the alphanumeric digits on
the card in the form of a scrolling banner. The code for this primitive was provided with the board. It has a distinct penalty in that it uses 500 bytes.

6. **Dual UART Board**

This board was provided along with a bootstrap rom on the cpu board as a method to load programs into the controller. This provided the means of testing the boards program using a ram board. The final controller does not require this board unless specifically determined by the application. No primitive is required to use this board, but its installation is required if the NPS Bootstrap loading prom is used. That prom assumes that the terminal is connected to the A port and that another Altos computer is connected to the B port of the board. In final use the program would be contained in a rom or a prom, making the loading unnecessary.

F. **TESTING**

Testing of the primitives has been done in three phases. First, the primitives have been functionally tested individually in the course of writing them. They have all passed an assembly and have been executed individually using a debugger to insure that the outputs are appropriate. The code associate with an initialization of a board has been adapted from the manufacturers examples and has been assembled. It has not been run on the Prolog System to
check its accuracy. The primitives have been tested individually, but have not been tested for interactions.

Because the controllers written with these primitives can be potentially quite complex, two additional methods of testing the controller's program are provided. The first is the debug switch. This is a global variable that allows a conditional assembly of the primitives. The purpose of the switch is to allow the program to be run on a microcomputer that has a z-80 processor and the CPM operating system. The switch causes all outputs to be sent to the console and all inputs to be requested from the keyboard via a message to the console. To implement these features the additional primitives wrtbin and messout are included. Their purposes are binary output and message output respectively. The second method is the use of the serial input/output board and bootstrap rom to load the program from a microcomputer into the controller. This permits the controller to be run and controlled from the microcomputer. This also permits the controller to be tested with its I/O boards for proper operation prior to loading the program into a prom.

8. FORMATTING OF PRIMITIVES

The format of the primitives is driven by several factors. The most obvious is the instruction set of the cpu, and the next is the form of the arguments. The form of the arguments used in the Z-80 library were typically rs1t, arg1, or rs1t, arg1, arg2. This is the same form as the
Intel 8080 library. In the construction of the primitives all results are stored at the conclusion of a primitive. In the case of a complex evaluation this might cause a value to be stored in memory only to be fetched by the next primitive. This is very costly in terms of execution time. During the construction of the primitives care was taken to insure that all results are in the registers that initially contain argument2. This could allow an auxiliary set of primitives to be developed in the future to take advantage of the chaining of arithmetic operations. The format of the title line and the reserve words are defined in Ross’s thesis and have not changed. [Ref. 19: pp. 79-85]

Since the primitives were tested individually on a CPM machine prior to placing them in the library, several stages of collating and compromises in editing speed were made to insure integrity of the primitive. All the code and text associated with the primitive was kept in a single file by primitive. To debug the primitive a header file and trailer file was added to allow the proper assembly of each single primitive. These three files were combined into the actual test file that was assembled and debugged. All the text that was not part of the code for the primitive but was necessary for the operation of the primitive in the design system was commented out in column one. Data on the execution time of each line of code was kept as a comment after the code. It is in the form of ;?a ??t ?b comment. The “a” was the memory cycle time of the instruction. The
"t" was the machine cycle time of the instruction and the
"b" was the number of ROM bytes used by the instruction.

The code format used by the design system and the z80
assembler can cause conflicts. In order to insure the
integrity of a primitive, the entire primitive was stored in
a single file. All the text outside of the markers begin
stext and endtext is used by the design system and is not
compatible with the assembler. To keep this text from
causing assembly errors, this text was commented out using
the ";". The primitives when written did not have blanks in
the first five columns. Ross's program NEWCSDL requires
that the library have a line number on each line. This can
be done by the program FIXIT however the program does not
append the program line to a line number, but rather changes
the first five columns to an appropriate line index. The
primitive had to be reformatted by adding five blanks to the
beginning of the line before sending it to the VAX. They
were run through a program that removed the ";" if it
existed in the first column and stuffed five spaces in front
of all lines. The primitives were then appended together
and sent by modes to the VAX. The VAX uses a carriage
return to indicate the end of a line. When sending a file
the terminal program sends a carriage return and a line
feed, giving each line an extra linefeed. After receipt of
the file, all the line feeds added by the terminal program
had to be edited out. At this point the file is in a format
acceptable by FIXIT.
1. **Pollock FixIt**

Pollock expressed difficulty in trying to use Ross’s primitive format and he wrote a FORTRAN program to correct the placement of indices used in the title line (Ref. 20: pp. 23-24). The problems originate in the structure of the primitive file as designed by Ross. Each library file is required to have an alphabetic index of all primitive title lines. Implicit in this is the requirement to number all the lines. Ross’s format for the numbering was in the format "vxxxx", where x represents the number of the line. Within each title line there are four additional pointers, first INCL, first CALC, first line of primitive and last line of primitive. All of these pointers make any change extremely difficult. Even simple changes require running this program since the VAX editor creates a variable length record file and Ross’s program requires a fixed length record file.

Pollock’s program FIXIT was written to minimize the effect of the format requirement imposed by Ross’s program. The program was written for a Cyber computer. The program was transported here via tape, however the program did not run due to differences in machines and program errors. The program is now corrected and working on a VAX 11/780. Much of the program’s problems were in the differences in I/O and word size in the two machines. The program is now set up to take a file of primitives named “innname.dat”. It produces a file “outname.dat” of correctly formatted primitives that
includes a sorted title directory at the beginning of the file with pointers to the individual primitives. Without this utility, change to the realization would be very tedious.

2. Ross's Newcsdl

When running NEWCSDL there are options to produce a trace by subroutine of the program execution. This is a very necessary utility, since the program is very complex and sensitive to input format. The detailed trace produced by NEWCSDL is very necessary because of the size of the program. This became painfully obvious while trying to get FIXIT to give an acceptable input file for the primitives. The error messages produced by NEWCSDL were incorrect due to a format error. The carriage control has been corrected, and the messages are now intelligible. In using NEWCSDL trace option in full mode, every subroutine called is printed to a log file along with the movement of key data. The carriage control that was a problem in the output of messages was also the principal problem in the input of the primitives. The carriage control character kept moving the data of the input file over by one character, causing the data to be in the wrong column. The requirement for fixed column reads in this program precludes any editing of a primitive file without running it through FIXIT now named FORMAT. The name was changed to minimize the confusion between the various versions of the program. FORMAT, in addition to correcting any pointer inconsistencies also
converts the file from a variable format to a fixed 80 character record format, which is the required input format for NEWCSDL.

In writing the primitives, the desirability of a modulo operator in the calc function became evident. The addition of multiple boards makes this desirable in assigning pinouts on the additional boards. Currently the number of ports by type requested is accumulated as a variable in GLOBALS.DAT. Ross calcit subroutine was examined to determine if this was feasible. Currently, the only primitives that could include multiple boards are the 8 bit analog to digital primitive and the 8 bit digital to analog primitive. Because of this limited number of boards requiring a modulo function, a simple scratch variable and subtraction of any values greater than the number of ports on the board was used instead. If the number of replicated board grows, then it may be worthwhile in the future to change NEWCSDL to include this function.

Two additional files used by NEWCSDL are potentially affected by new libraries. The first file MONITOR.DAT contains primitives that are used all the time by NEWCSDL and are included regardless of the application. Because the intermediate table containing the order of the contingency tasks has been eliminated, one primitive TAACP2 has been eliminated and the code in NEWCSDL needs to be modified to eliminate the reference to that primitive. The second file, GLOBALS.DAT, contains the names of global
variables used by the primitive library. Because of the change in orientation of the library from individual components to boards, many globals were no longer needed. A list of the global variables used with this library and their application is given in appendix B.

3. Walden's Data Base Input

Walden designed a database system to eliminate the need for the various files used in NEWCSDL. No code was available for test at the time of the implementation of this thesis, so no attempt was made to try inputting any of the primitive library into a database. The method of input into the data base specified by Walden was to type all data in at a terminal. This is a reasonable method for the title lines and some of the smaller files. It does not appear to be a feasible method for entering a primitive library because of the size of the data. A data base eliminates the need to use the FORMAT program to correct the pointers and eliminates some of the overhead on CALC, and INCLUDE. However it would require typing all of the primitive's assembly code that has been previously debugged, with the attendant errors and duplication of work. To be really usable, the data base needs a method of getting the assembly code associated with a primitive from its file used for testing and debugging. By doing this the code could be transferred with a minimal amount of error. The code would still be available for separate assembly and debugging by the primitive's author. [Ref. 21]
V. RESULTS AND CONCLUSIONS

Preliminary results of Riley's thesis indicate that the controller can be built using the design system. [Ref. 22] This particular controller does not overly tax the design system or the primitive library, but does provide a functional test.

In the course of constructing the library, integration of the various tools became the major problem. In constructing any library an assembler and debugger are necessary. The Z-80 has a number of assembler programs, however, they do not run on a VAX11/780. The transport of the assembly file to the mainframe after construction can be tedious. My choice was using a 300 baud modem. This was not much of a problem when the primitives were few in number, but became a real logistics problem as the library neared completion.

The implementation of an interrupt driven controller has turned out not to be desirable because of the additional overhead. This was tried in the form of an interrupt driven clock for the primitive library. The clock was using about 2.5% of the monitor's available time. This required no selection of competing interrupts, since there was only the single interrupt in this case. The overhead was also the minimal possible, since only the AF and HL registers were
saved. If all the primitives had the potential for interrupt, then the AF, BC, DE, HL, IX, AF', BC', DE', and HL' would also have to be saved. For comparison, saving all the registers would have increased the overhead to 7.5%. This still would not include the time necessary to select which contingency/task pair should be executed. At this point the interrupt driven monitor was abandoned.

Further primitives can be added to the Z-80 library to increase the number of arithmetic and control functions available, but this may be time poorly spent. In building a good design system a suite of libraries is necessary. The Z-80 is the low end of the library in terms of performance and cost. Because the costs of processors continue to fall, it may be desirable to include the most extensive of libraries only in higher performance chips.
APPENDIX A

PRIMITIVE TITLE INDEX

This appendix was created from the index of the primitive title lines. Where there are multiple primitives with the same name, but different precisions, only one name is listed. The function of the primitive is then briefly described, and its limitations.

<table>
<thead>
<tr>
<th>NAME</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>h.atod</td>
<td>Include a 8 bit analog to digital conversion board</td>
</tr>
<tr>
<td>h.cardcage</td>
<td>Include a 8 slot card cage and power supply in the primitive listing.</td>
</tr>
<tr>
<td>h.clock</td>
<td>Detail the connections of the counter timer chip on the cpu board to produce a 1 millisecond or a 25 millisecond clock out of channels 0 and 1.</td>
</tr>
<tr>
<td>h.dtoa</td>
<td>Include the hardware for a 8 bit digital to analog conversion board</td>
</tr>
<tr>
<td>h.inout</td>
<td>Include a 8 bit 64 channel standard bus to digital i/o board with wiring to illuminate up to eight lights using the address lines on the card</td>
</tr>
<tr>
<td>h.memory</td>
<td>Hardware primitive to include a 16k ram board based on total ram and rom requirements. This particular board is a battery backup board and can also be used as a quasi rom if a write inhibit switch is on. The board can be disabled in 4k segments, making that particular segment nonwriteable by the cpu.</td>
</tr>
<tr>
<td>h.processor</td>
<td>Hardware primitive to include a Zilog Z-80a with a 4mhz clock and a three channel counter timer chip on a single board. Included also is a bootstrap rom using the first 4k of address space.</td>
</tr>
<tr>
<td>h.tcardcage</td>
<td>This is a primitive that is invoked when a card slot is requested. It checks that the number of slots requested does not exceed the number available.</td>
</tr>
<tr>
<td>s.add</td>
<td>Comes in a variety of forms to support byte, two byte and floating point addition. It provides no error checking on the result of the</td>
</tr>
</tbody>
</table>
computation.

s.addck Comes in a variety of forms to support byte, two byte and floating point addition. It provides error checking and will not allow the result to have an inappropriate sign. If an overflow is made then, it will put the largest possible value in the result.

s.and Perform a logical and

s.assign Performs an assignment operation.

s.assigncons Assigns a constant value to a previously defined variable. This does not reserve space for the variable, only puts a specific value in the variable.

s.atod Primitive to perform a analog to digital conversion

s.blockcons Primitive to mark the beginning of a submonitor block

s.blockend Primitive to mark the end of a submonitor block

s.blockexit Primitive to leave a submonitor and reset all pending operations

s.blockstart Primitive to cause a submonitor to be executed

s.clockcons Primitive to create an interrupt driven clock

s.clockcons Primitive to create a clock using counter timer chip channels 0 and 1. No interrupt is involved. The accuracy of the time tick is once every millisecond. The time is accumulated in channel 1 as a 16 bit down counter. The time can be read by using s.rdttime. This primitive will latch the current time to an output buffer and subsequently input the latched time.

s.clockcon25 Primitive to create a clock using counter timer chip channels 0 and 1. No interrupt is involved. The accuracy of the time tick is once every 25 milliseconds. The time is accumulated in channel 1 as a 16 bit down counter. The time can be read by using s.rdttime. This primitive will latch the current time to an output buffer and subsequently input the latched time.

s.cold Primitive cause the system to do a cold boot.
That is reinitialize all hardware and perform any user directed initializations.

s.cons Define a constant. That is put a value in the rom position of memory. Comes in version for byte, two byte and floating point constants

s.div Comes in version for byte, two byte and floating point divisions.

s.dtoa Software primitive to perform an 8 bit digital to analog conversion.

s.end Primitive that must be last. Indicated the end of program and includes some necessary pointers for hardware initialization.

s.eq Performs comparison of byte or word values for the condition of equality and outputs boolean result

s.every Forces execution of monitor every time by making condition always true.

s.exitproc Marks the end of a procedure that is executed as a subroutine. When used with a conditional call, this primitive will first set a boolean variable with the same name as the procedure to false prior to executing a return. If it is used in conjunction with an unconditional procedure then it will simple execute a return.

s.float Converts a two byte variable to a floating point variable.

s.forend Marks end of a FOR construction

s.forstart Creates a FOR variable date from lower to upper.

s.fptoieee Converts floating point format in controller to IEEE single precision format.

s.ge Performs comparison of byte or word values for the condition of greater than or equal and outputs boolean result

s.gt Performs comparison of byte or word values for the condition of greater than and outputs boolean result

s.ifcons Marks to to an if construction
s.ifend Marks the end of an if construction

s.initalcons Marks the beginning of user defined initialization requirements

s.initalend Marks the end of user defined initialization requirements. Implied in the end is setting the flag associated with the user defined initialization to false and jumping to the top of the main monitor loop.

s.jmpf Causes a jump to a location if a variable is false

s.jmpt Causes a jump to a location if a variable is true

s.le Performs comparison of byte or word values for the condition of less than or equal and outputs boolean result

s.loc Marks a portion of a program with a label.

s.lt Performs comparison of byte or word values for the condition of less than and outputs boolean result

s.main This construction is always required. It initializes some basic pointers and creates a stack.

s.messout Sends a message to the output device. It is used only in the debug mode.

s.monitor Creates the top of the main polling loop and reinitializes the stack pointer

s.mult Comes in a variety of forms to support byte, two byte and floating point multiplication. It provides no error checking on the result of the computation.

s.ne Performs comparison of byte or word values for the condition of not equal and outputs boolean result

s.not Performs a logical not.

s.or Performs a logical or

s.out Output a signal to a port.

s.perform Primitive to invoke a procedure subroutine call.
s.proc  Primitive to mark the beginning of a conditional
or unconditional subroutine call. It is used in
conjunction with s.exitproc to mark the end of
the subroutine.

s.rdtim  Reads the time from the CTC chip on the cpu
board. Time is represented in either
milliseconds or 25 milliseconds depending on
which clock construction has been used. In both
cases the actual number read is a two byte
number. The clock is a down counter, so that
the elapsed time goes from a negative number to a
positive number to zero and then repeats
starting with a negative number.

s.start  Forces monitor execution provided for
compatibility with older library. Not needed
since monitor starts at nonmaskable interrupt
location.

s.sub    Comes in a variety of forms to support byte, two
byte and floating point subtraction. It
provides no error checking on the result of the
computation.

s.subck  Comes in a variety of forms to support byte, two
byte and floating point subtraction. It
provides error checking and will not allow the
result to have an inappropriate sign. If an
overflow is made then, it will put the largest
possible value in the result.

s.tabend Marks the end of the main monitor polling loop.

s.tabent Put an entry in a polling loop. It is used with
both the main and submonitor loops.

s.var    Defines a variable in ram

s.warm   Performs a warm boot. That is jump to the top
of the main polling loop, reinitialize the
stack, and perform any user defined
initializations.

s.whend  Mark the end of a while construction

s.whileco Mark the beginning of a while construction.
Performs the test of a condition, if false jumps
to the next instruction past the location marked by whend.

s.wrtbin Used in the debug mode to output a location to
the screen in binary.

s.xor 

Perform an exclusive or.

s.tabaccp2 Is a dummy primitive to provide compatibility with the 8080 realization listing. In the 8080 listing tabent is divided into two primitives tabent and tabaccp2. This requires less memory if there are extremes in the time requirements of different contingencies. By combining the two primitives an intermediate table is eliminated and the execution time is increased by eliminating two unconditional jumps.
APPENDIX B
GLOBAL VARIABLE LISTING

This appendix lists exactly the file GLOBALS.DAT under the column name. Included in the listing are the initial values of the variable. Since the variables are limited to six characters, the purpose and limitations are not readily apparent. The purpose of the variable and its use are described in the next column.

<table>
<thead>
<tr>
<th>NAME</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>arnd</td>
<td>Is a pointer in the form of &quot;@&lt;arnd&gt;&quot;. It purpose is to allow sections of code to be placed anywhere. This allows initializations to be placed with a primitive. In this way the primitive can be executed and the initialization is jumped around. The only way to access the initialization is to use the inlk chain.</td>
</tr>
<tr>
<td>chips</td>
<td>Is a count of the number of integrated circuits that have been added.</td>
</tr>
<tr>
<td>debug</td>
<td>Is a flag that will cause the primitives to be conditionally assembled to produce input and output through standard BDOB calls. The default is to use the normal input and output boards. To use a cpm microcomputer to test the primitive, debug must be set to 1. This can be done as an exit to the globals.dat file.</td>
</tr>
<tr>
<td>initlk</td>
<td>Is a variable that is used in a pointer chain for the initialization of hardware. The actual form is @i&lt;initlk&gt;. This for the first link would appear as @i0, the second @i1.</td>
</tr>
<tr>
<td>keybrd</td>
<td>Is a boolean variable indicating if the 7303 keyboard/display board has been requested by any primitive. This primitive was necessary to eliminate the potential of several copies of the board being requested by different primitives. There are several primitives because of the number of separate functions that are included on the board.</td>
</tr>
<tr>
<td>natode</td>
<td>Is a variable enumerating the number of analog to digital ports that have been requested. The hardware primitive will</td>
</tr>
</tbody>
</table>
natodp0. Is a variable containing the address of the I/O port. It is initially 0. If more than 16 atod ports are requested then the address is changed to 4.

ndtoas0. Is a variable enumerating the number of digital to analog ports that have been requested. The hardware primitive will indicate a failure if the number is greater than 32.

ndtoap0. Is a variable containing the address of the I/O port. Currently, not dtoa boards are available for the system, so the exact address has not been determined.

ninout0. Is the number of inout ports that have been request for the mostek adx-diobi board. An error is produced if the number requested exceeds 64.

nkey 0. Is a variable indicating the number of push button keys that have been requested.

nled -1. Is the number of light emitting diodes that have been requested. The maximum is eight. An error will be generated if more than eight are requested. The lights are number zero thru seven.

nogdt 0. Is the number of digits of the alphanumeric display that are requested.

norom 0. Is a flag indicating the absence of the loading rom. If the controller is to be used without external support, this flag must be set to 1. This will cause the conditional assembly to start at 66d and will allow the normal use of interrupts. If the controller is being simulated on a cpm computer, this flag should be 0. The normal assembly has a jump to 4000h at 100h to accommodate the use of cpa. This does not effect the use of the loading rom, since the location 100h can't be altered by the loader.

nrockr0. Is the number of rocker switches that have been requested for the 7303 keyboard/display board. Currently this is limited to two switches. Requesting more than two will
cause an error.

**ramptr0.** Keeps a count on the number of bytes of storage that have been requested in a RAM area. This counter starts at zero, but will be moved to the top of the address space and decremented as memory is requested for variables and stack space.

**romptr0.** Keeps a count on the number of bytes of storage that have been requested in a ROM area. This counter starts at 0 and is incremented as instructions are added.

**scrtch0.** Is a scratch variable. It is used in various ways by different primitives to compute intermediate results.

**slot 0.** Indicate the slot in the card cage that a board will use. An error will be generated if more than eight cards are used.
APPENDIX C

ZILOG-80 REALIZATION LISTING

This appendix contains the listing of the library as it is required to be formatted. The first line of the library contains the title of the library along with the cpu's clock period and memory speed in quarter microseconds. This is followed by the alphabetical index of all the title lines. The individual primitives are listed after the index. The format of the individual primitive repeats the title line, has comments describing the primitive, then actual code and is unbounded until another title lines is encountered. This makes it possible to have multiple segments of code interspersed with calc statements. Not shown because it is an extra line, is the requirement to a line after the last line of the last primitive. If this is not included, the last line of the last primitive will not be included, since the FORMAT program is looking for a last dummy primitive to mark the end of the library.

The library is listed vertically to allow the fully 80 columns to be displayed.
v0107endtext
v0108calc romptr=romptr+11
v0109calc (result, arg1, arg2:0, 8, 8, 8, 8: 45, 118, 34, 29, 0, 109, 138)
v0110com primitive to perform comparison between 2 8-bit numbers
v0111com list=result, argument 1, argument 2 : stor, time, ext, c, 1, addr
v0112begin stext
v0113id a, (arg2): 4m 13t 3b if arg2 lt arg1 then result=ffh
v0114id b, a: 1m 4t 1b b=arg2
v0115id a, (arg1): 4m 13t 3b c=arg1
v0116id c, a: 1m 4t 1b set sign flag of arg1
v0117and a: 4t 1b
v0118jp p, $+00dn: 3m 10t 3b jump if arg1 is positive
v0119id a, b: 1m 4t 1b arg1 =
v0120and a, c: 1m 4t 1b set sign flag of arg2
v0121id b, c: 1m 4t 1b arg2 . swap. arg1
v0122jp m, $+01hn: 3m 10t 3b arg2 = - arg1 = - comp backwards
v0123id a, m: 2m 7t 3b arg2 = + arg1 = - false
v0124jp $+013h: 2m 12t 3b
v0125id a, b: 1m 4t 1b
v0126and a, c: 1m 4t 1b set sign flag of arg2
v0127id a, c: 1m 4t 1b restores arg1 to accumulator
v0128jp p, $+007h: 3m 10t 3b arg2 = + arg1 = +
v0129id a, 1111111111b: 2m 7t 2b arg2 = - arg1 = + true
v0130jp $+009h: 3m 12t 2b result false arg2 >= arg1
v0131cp b: 1m 4t 1b
v0132id a, 000000000b: 2m 7t 2b result true arg2 lt arg1
v0133id a, 000000000b: 2m 7t 2b result true arg2 lt arg1
v0134id (result), a: 1m 4t 1b
v0135endtext
v0136calc romptr=romptr+19
v0137com primitive to generate a 1 millisecond clock. the value of
v0138com current time is stored in the channel 1 counter of the cpu
v0139com ctc chip. to access this particular time rdtms must be used
v0140com to read/rst the time use rdtmte primitive.
v0141com note when running this does not have any over head in terms
v0142com of slowing the monitor since no interrupts are involved
v0143com the clock is limited to about one minute of elapsed time
v0144com channel 0 will be used for the clock at factory installed
v0145com address of fo
v0146com channel 1 is used to store current time at a factory installed
v0147com address of fi
v0148incl h.clock
v0149begin stext
v0150jp ee<argn> ; 3m 10t 3b
v0151<initial>:
v0152endtext
v0153calc initial=initial+1
v0154begin stext
v0155ld a, 00110100b: 2m 7t 1b counter 0 + load lab then mb+ mode2+ bcd
v0156out (0f3h), a ; 3m 11t 2b set mode control
v01601d a, 00
v0161out (0f0h), a
v0162d a, 20h
v0163out (0f0h), a
v0164d a, 01110001b
v0165out (0f3h), a
v0166d a, 00
v0167out (0f1h), a
v0168out (0f1h), a
v0170ip el<ini1k>

v0178e<arnd> nop

v0172endtext
v0173calc arnd=arnd+1

v0174calc romptr=romptr+27

v0175a. and

(rslt, arg1, arg2:0, 1, 2, 4, 8, 16, 32, 64, 128)

v0176com primitive to perform logical and

v0177com list=rslt, argument 1, argument 2 : stor, time, ext, c1, addr

v0178begin stext
v0179d a, (<arg1>), 4m 13t 3b

rslt = arg1 .and. arg2

v0180d b, a

v0181d b, (<arg2>), 4m 13t 3b

v0182and b

v0183d (<rslt>), a, 4m 13t 3b

v0184endtext
v0185calc romptr=romptr+11

v0186a. add

(rslt, arg1, arg2:0, 16, 32, 64, 128)

v0187com primitive to add arg1 and arg2 and store in rslt

v0188com list=rslt, arg1, arg2: precision, length, time, ext, c1, addr

v0189begin stext
v0190id hl, (<arg1>), 6m

load arg1 in hl pair

v0191id bc, (<arg2>), 6m

load arg2 in bc pair

v0192add hl, bc

v0193d (<rslt>), hl, 6m

save result

v0194endtext
v0195calc romptr=romptr+13

v0196s. convb16

(rslt, arg1:0, 16, 32, 64, 128)

v0197com routine to convert a 8 bit to 16 bit number

v0198begin stext
v0199id hl, 0

load hl pair

v0200id a, (<arg1>), 4m

v0201id l, a

v0202add a, 80h

v0203ip nc, +5

v0204id h, 0fh

v0205id (<rslt>), hl, 16m

v0206endtext
v0207calc romptr=romptr+17

v0208s. float

(rslt, arg1:0, 32, 64, 16, 46, 528, 143, 59, 0, 208, 267)

v0209com primitive to convert an 16 bit number to floating point

v0210com normalization to 2's complement form exp mantissa

v0211com exponent is also in 2's complement form.

v0212com mantissa is normalized to sign magnitude
v0213com list=rst, argl: precisions: s, t, e, c, i, addr
v0214begin stext
v0215d hl, (<argl>) : 8m 20t 4b load argl
v0216exx : 1m 4t 1b prime registers
v02171d hl, 0 : 3m 10t 3b zero byte 3 and 4
v02181d c, 15 : 2m 7t 2b put 15 in c’ for largest exponent
v0219exx : 1m 4t 1b return to main registers
v0220id a, 0 : 3m 8t 2b put zero in accumulator -- zero rslt?
void 21cph : 1m 4t 1b is mask zero
void 2222j r nz, $+01h : 2m 7t 2b quit test of rslt if non zero
v02226exx : 1m 4t 1b prime registers
v0226cp h : 1m 4t 1b is byte 2 zero
v0227j r nz, $+00ah : 2m 7t 2b quit test of rslt if non zero
v0228cp l : 1m 4t 1b is byte 3 zero
v0229j r nz, $+00ah : 2m 7t 2b quit test of rslt and flip reg non zero
v0230id c, a : 1m 4t 1b put 0 in exponent result is zero
v0231exx : 1m 4t 1b main registers arg2 is 0
v0232j r $+03h : 3m 12t 2b result is zero store
v0233exx : 1m 4t 1b main registers
v0234id m, h : 1m 4t 1b test sign of result
v0235and a : 1m 4t 1b set sign bit
v0236j p m, $+01h : 3m 10t 3b if neg goto neg normalization
v0237bit 6, h : 2m 8t 2b test me bit sl. xxxx pos normal
v0238j p nz, $+027h : 3m 10t 3b if 1 in mab then store result
v0239exx : 1m 4t 1b prime registers
v0240cp c : 1m 4t 1b is exponent zero?
v0241j p z, $+02h : 3m 10t 3b if exponent is 0 then stop
v0242dec c : 1m 4t 1b decrement exponent in c’
v0243add hl, hl : 1m 11t 1b shift rslt low word
v0244exx : 1m 4t 1b main registers
v0245rl i : 1m 2t 8t 2b shift rslt high word
v0246rl h : 1m 2t 8t 2b
v0247j p $-01h : 3m 10t 3b check to see if normalized
v0248j p 6, h : 2m 8t 2b test me bit s0..xxxx neg normal
v0249j p z, $+03h : 3m 10t 3b if 0 in mab then store result
v0250exx : 1m 4t 1b prime registers
v0251cp c : 1m 4t 1b is exponent zero?
v0252j p z, $+00dh : 3m 10t 3b if exponent is 0 then stop
v0253dec c : 1m 4t 1b decrement exponent in c’
v0254add hl, hl : 1m 11t 1b shift rslt low word
v0255exx : 1m 4t 1b main registers
v0256rl i : 1m 2t 8t 2b shift rslt high word
v0257rl h : 1m 2t 8t 2b
v0258j p $-01h : 3m 10t 3b check to see if normalized
v0259exx : 1m 4t 1b main registers
v02601d (<rslt+>), hl : 5m 16t 3b save rslt me word
v0261exx : 1m 4t 1b prime registers
v02621d (<rslt+>), hl : 5m 16t 3b save rslt is word
v0263id a, c : 1m 4t 1b get exponent of rslt
v0264id (<rslt+>), a : 4m 13t 3b save rslt exponent
v0265exx : 1m 4t 1b main registers
v0266andtext
v0267calc romptr=romptr+34
v0268s.xor (rslt, arg1, arg2:0,8,0,8,0,8;11,14,10,0,268,278)
v0268com primitive to perform logical exclusive or
v0270com list=result,argument 1, argument 2; t, stor,time,ext,c1,address
v0271begin stext
v0272id a,(<arg1>);4m 13t 3b rslt = arg1.xor. arg2
v0273id b, a;4m 1t 3b
v0274id a,(<arg2>);4m 13t 3b
v0275xor b;4m 1t 3b
v0276id (<rslt>), a;3m 13t 3b
v0277endtext
v0278calc romptr=romptr+11
v0279s.rdtime (rslt:0,16,10,58,16,16,0,279,295)
v0280com primitive to read a clock generated by h.clock.
v0281com current time is stored in the channel 1 counter of the cpu
v0282com ctc chip, to access this particular time rdtime must be used
v0283com the clock is limited to about one minute of elapsed time
v0284com channel 0 will be used for the clock at factory installed
v0285com address of f0
v0286com channel 1 is used to store current time at a factory installed
v0287com address of f1
v0288begin stext
v0289id a,01000001b;2m 7t 1b counter 1 + latch + mode2+ hex
v0290out (0f3h), a;3m 1t 2b set mode control to latch channel 1
v0291in 1,(0f1h);3m 12t 2b get lsb
v0292in h,(0f1h);3m 12t 2b get msb
v0293id (<rslt>), h;5m 16t 3b load time to <rslt>
v0294endtext
v0295calc romptr=romptr+10
v0296s.outled (arg1:0,0,24,100,25,10,0,268,331)
v0297com primitive to add output routine for up to 8 light emitting
v0298com diodes on the 7303 keyboard/display board.
v0299com keybrd is a boolean flag indicating if board has previously been
v0300com called.
v0301com nled is the number of the outlined that have been requested.
v0302com arg1 is the boolean that is tested to determine if the led should
v0303com be lighted.
v0304incl h.keystandyh(;;)
v0305if nled .ne.-1 skip 6
v0306calc romptr=romptr-1
v0307begin stext
v0308org <romptr>
v0309outled defb 0 ; set status of all lights off
v0310org <romptr>
v0311endtext
v0312calc nled=nled+1
v0313if nled .lt. 8 skip 4
v0314begin stext
v0315you have requested more than 8 leds on the keyboard display
v0316the board is limited to a maximum of 8
v0317endtext
v0318begin stext
v03721d a, (arg2) +4 13t 3b c=arg2
v03731d c, a +1m 4t 1b
v0374and a +1m 4t 1b
v0375jp p,$006h +3m 10t 3b jump if arg2 is positive
v03761d a, b +1m 4t 1b arg2 = -
v0377and a +1m 4t 1b set sign flag of arg1
v03781d b, c +1m 4t 1b arg1.swap.arg2
v0379jp m,$+01h +3m 10t 3b arg1 = -arg2 = -comp backwards
v03801d a, 0 +2m 7t 2b arg1 = +arg2 = -false
v0381jr $+016h +3m 12t 2b
v03821d a, b +1m 4t 1b
v0383and a +1m 4t 1b set sign flag of arg1
v03841d a, c +1m 4t 1b restore arg2 to accumulator
v0385jp p,$007h +3m 10t 3b arg1 = +arg2 = +
v03861d a, $11111111b, 2m 7t 2b arg1 = -arg2 = +true
v0387jr $+00ch +3m 12t 2b
v0388cp b +1m 4t 1b
v0389id a, 00000000b, 2m 7t 2b result false arg1 >=arg2
v03901d z, $7 +3m 10t 3b
v0391jp m, $+4 +3m 10t 3b
v0392cp 1 +1m 4t 1b result true arg1 lt arg2
v0393 jr EXIT, a +4m 13t 3b
v0394endtext
v0395calc romptr=romptr+19
v0396a.ne (silt,arg1,arg2:0,8,0,8,0,8:15,72,20,13,0,396,409)

v0397com primitive to perform comparison between 2 8-bit numbers
v0398com list= result, argument 1, argument 2 :stor, time, ext, c, 1, addr
v0399begin stext
v04001d a, (arg1) +4m 13t 3b if arg1 arg2 then silt=ffh
v04011d b, a +1m 4t 1b
v04021d a, (arg2) +4m 13t 3b
v0403cp b +1m 4t 1b
v0404id a, 0 +2m 8t 1b
v0405jr z, $+003h +3m 13t 2b result not equal
v0406cp l +1m 4t 1b result equal
v04071d (silt), a +4m 13t 3b
v0408endtext
v0409calc romptr=romptr+15
v0410a.ge (silt, arg1, arg2:0,8,0,8,0,8:42,108,31,28,0,410,438)

v0411com primitive to perform comparison between 2 8-bit numbers
v0412com list= result, argument 1, argument 2 :stor, time, ext, c, 1, addr
v0413begin stext
v04141d a, (arg2) +4m 13t 3b if arg2 le arg1 then silt=ffh
v04151d b, a +1m 4t 1b b=arg2
v04161d a, (arg1) +4m 13t 3b
v04171d c, a +1m 4t 1b c=arg1
v0418and a +1m 4t 1b set sign flag of arg1
v0419jp p,$00dh +3m 10t 3b jump if arg1 is positive
v04201d a, b +1m 4t 1b arg1 = -
v0421and a +1m 4t 1b set sign flag of arg2
v04221d b, c +1m 4t 1b arg2 .swap. arg1
v0423jp m, $+01h +3m 10t 3b arg2 = -arg1 = -comp backwards
v04241d a, 0 +2m 7t 2b arg2 = +arg1 = -false
v0425jr $+013h ;3m 12t 2b
v0426ld a, b ;1m 4t 1b
v0427.and a ;1m 4t 1b
v0428ld a, c ;1m 4t 1b
v0429ai p, $+007h ;3m 10t 3b
v0430i a, $+007h ;3m 10t 2b
v0431jr $+007h ;3m 12t 2b
v0432cp b ;1m 4t 1b
v0433ld a, $+007h ;3m 10t 3b
v0434qp p, $+4 ;3m 10t 3b
v0435sph <rlat>, a ;1m 13t 3b
v0436ld <rlat>, a ;1m 13t 3b
v0437endtext
v0438calc romptr=romptr+42
v0439s.rockr {arg1:0,0;8;13,53,14,7,6,439,461}
v0440com primitive to add a test of the rocker switches on the 7303
v0441com keyboard/display board, it is limited to two switches.
v0442com an error is generated if more than two switches are requested.
v0443com nrockr is the number of rocker switches that have been requested.
v0444com arg1 is a boolean value based on the value of the rocker switch.
v0445incl n.keydisplay(:)

v0446calc nrockr=nrockr + 1
v0447if nrockr .lt. 3 skip 4
v0448begin text
v0449you have requested more than 2 rocker switcher on the keyboard/display.
v0450the board is limited to a maximum of 2
v0451endtext
v0452calc scratch= 5 + nrockr
v0453begin text
v0454 in a,(000h) ;3m 11t 2b
v0455 bit <scratch>, a ;1m 8t 2b
v0456 ld a, 0 ;2m 7t 2b
v0457 jr nc, $+4 ;3m 10t 3b
v0458 cpl ;1m 4t 1b
v0459 ld <(arg1)>, a ;3m 13t 3b
v0460endtext
v0461calc romptr=romptr+13
v0462s.forand (index,lab,elab:0,8,0,255:7,27,8,9,0,462,471)
v0463com primitive to end a for loop
v0464com list=index,start label,end label
v0465com max loop count 255
v0466begin text
v0467ld a,<(index)> ;4m 13t 3b
v0468inc a ;1m 4t 1b
v0469jp <lab> ;3m 10t 3b
v0470endtext
v0471calc romptr=romptr+7
v0472s.le (rlat,arg1,arg2:0,8,0,8,0,8:42,108,31,28,0,472,500)
v0473com primitive to perform comparison between 2 8-bit numbers
v0474com list=result,argument 1,argument 2 : stor,time,ext,c1,addr
v0475begin text
v0476ld a,<(arg1)>, ;4m 13t 3b
v0477le a, ;1m 4t 1b
v0478ld b, a ;1m 4t 1b
b=arg1
v04781d a, (<arg2>) ; 4m 13t 3b c=arg2
v04791d c, a ; 1m 4t 1b set sign flag of arg2
v0480add a, ; 1m 4t 1b jump if arg2 is positive
v0481jp p, $+00dh ; 3m 10t 3b arg2 = -
v0482ld a, b ; 1m 4t 1b set sign flag of arg1
v0483add a, ; 1m 4t 1b arg1 .swap arg2
v0484ld b, c ; 1m 4t 1b arg1 .swap arg2
v0485jp m, $+011h ; 3m 10t 3b arg1 = - arg2 = - comp backwards
v0486ld a, d, 0 ; 2m 7t 2b arg1 = + arg2 = - false
v0487jr $+013h ; 3m 12t 2b
v0488ld a, b ; 1m 4t 1b set sign flag of arg1
v0489add a, c ; 1m 4t 1b restore arg2 to accumulator
v0491jp p, $+007h ; 3m 10t 3b arg1 = + arg2 = +
v0492ld a, 1111111b ; 2m 7t 2b arg1 = - arg2 = + true
v0493jr $+009h ; 3m 12t 2b
v0494cp b, ; 1m 4t 1b result false arg1 >= arg2
v0496jp p, $+4 ; 3m 10t 3b result true arg1 < arg2
v0497cpil ; 1m 4t 1b result true arg1 < arg2
v0498endtext
v0500calc romptr=romptr+42
v0501s.addcd (rslt, arg1, arg2:0,8,0,8,0,8:23,80,24,14,0,501,515)
v0502com primitive to add arg1 and arg2 and store in rslt
v0503com list=rslt, arg1, arg2:precisions:a,t,e,c,i,addr
v0504begin text
v0505ld a, (<arg1>) ; 13t 4m 3b store arg1 in accumulator
v0506ld h1, <arg2> ; 10t 3m 3b have h1 point to arg2 byte
v0507add a, (h1) ; 7t 2m 1b add accumulator with arg2
v0508jp po, $+13 ; 3m 10t 3b if no overflow store result
v0509ld c, $+8 ; 3m 10t 3b if carry the maximize minus rslt
v0510ld a, 01111111b ; 2m 7t 2b put in largest positive value
v0511jp $+5 ; 3m 10t 3b
v0512ld a, 10000000b ; 2m 7t 2b put in largest negative value
v0513ld ((rslt)), a ; 13t 4m 3b save result of add in rslt
v0514endtext
v0515calc romptr=romptr+23
v0516s.sub (rslt, arg1, arg2:0,8,0,8,0,8:10,43,13,9,0,516,525)
v0517com primitive to subtract arg2 from arg1 and store in rslt
v0518com list=rslt, arg1, arg2:precisions:a,t,e,c,i,a
v0519begin text
v0520ld a, (<arg1>) ; 4m 13t 3b load arg1 in accumulator
v0521ld h1, <arg2> ; 3m 10t 3b point h1 to arg2
v0522sub (h1) ; 2m 7t 1b arg1 - arg2
v0523ld ((rslt)), a ; 4m 13t 3b save result
v0524endtext
v0525calc romptr=romptr+10
v0526s.add (rslt, arg1, arg2:0,8,0,8,0,8:10,43,13,9,0,526,535)
v0527com primitive to add arg1 and arg2 and store in rslt
v0528com list=rslt, arg1, arg2:precisions:a,t,e,c,i,addr
v0529begin text
v0530ld a, (<arg1>) ; 13t 4m 3b store arg1 in accumulator
v0531d hl, <arg2> ; 10t 3m 3b have hl point to arg2 byte
v0532add a, (hl) ; 7t 2m 1b add accumulator with arg2
v0533ld (<rlt>), a ; 13t 4m 3b save result of add in rlt
v0534endtext
v0535calc romptr=romptr+10
v0536s. it (rlt, argl, arg2: 0, 8, 0, 16, 0, 16: 46, 131, 38, 26, 0, 536, 562)
v0537com primitive to perform comparison between 2 16-bit numbers
v0538com li=x=rlt, argument 1, argument 2 : stor , time , ext , c , 1 , addr
v0539begin text
v0540ld de, (<argl>) ; 3m 20t 4b if arg1 lt arg2 then rlt=ffh de=<arg1
v0541ld hl, (<arg2>) ; 5m 16t 3b
v0542ld a, h ; 31m 4t 1b
v0543and a ; 1m 4t 1b set sign flag of arg2
v0544jp p, $+13 ; 3m 10t 3b jump if arg2 is positive
v0545ld a, d ; 1m 4t 1b arg2 =

v0546and e ; 1m 4t 1b set sign flag of arg1
v0547jp m, $+18 ; 3m 10t 3b arg1 = - arg2 = - comp backwards
v0548ld a, 0 ; 2m 7t 2b arg1 = + arg2 = - false
v0549jp $+24 ; 3m 10t 3b
v0550ld a, c ; 1m 4t 1b
v0551and a ; 1m 4t 1b set sign flag of arg1
v0552jp p, $+8 ; 3m 10t 3b arg1 = + arg2 = +

v0553ld a, 1111111b ; 2m 7t 2b arg1 = - arg2 = + true
v0554jp $+14 ; 3m 10t 3b
v0555bchl, d,e ; 4m 15t 2b result false arg1 >= arg2
v0556ld a, 00000000b ; 2m 7t 2b result false arg1 >= arg2
v0557jp x, $+7 ; 3m 10t 3b
v0558jp m, $+4 ; 3m 10t 3b
v0559ctp ; 1m 4t 1b result true arg1 lt arg2
v0560ld (<rlt>), a ; 4m 13t 3b
v0561endtext

v0562calc romptr=romptr+48
v0563s. forstart (index, low, upr, slab, elab: 0, 8, 0, 8, 0, 8, 255: 13, 50, 15, 11, 0, 563, 574)
v0564com primitive to set up a loop with constant bounds
v0565com index, low, upr, slab, elab, start label, end label
v0566com max loop count 255
v0567begin text
v0568ld a, <lwr> ; 2m 7t 2b lower bound of counter
v0569ld (<index>, a ; 4m 13t 3b initialize index to lower
v0570slab1d a, (<index>); 4m 13t 3b get value of index at top of loop
v0571cp upr ; 2m 7t 2b compare to upper limit
v0572jp z, (<elab>)+3 ; 3m 10t 3b jump out of loop on index<upr
v0573endtext
v0574calc romptr=romptr+13
v0575s. ge (rlt, argl, arg2: 0, 8, 0, 16, 0, 16: 45, 118, 34, 26, 0, 575, 601)
v0576com primitive to perform comparison between 2 16-bit numbers
v0577com li=x=rlt, argument 1, argument 2 : stor , time , ext , c , 1 , addr
v0578begin text
v0579ld de, (<arg2>) ; 3m 20t 4b if arg2 largel then rlt=ffh de=<arg2>
v0580ld hl, (<arg1>) ; 5m 16t 3b
v0581ld a, h ; 1m 4t 1b
v0582and a ; 1m 4t 1b set sign flag of arg1
v0583jp p, $+13 ; 3m 10t 3b jump if arg1 is positive
If

so Go 1 ;

M3N - S N - 0 NS

04 z -\r

VO 40*

4.

4.

4.

61 -.

4...

.4

04 z -\r

Ne

4.

6~~1

1.

*1N.

Ils6 ii

1

4.4

2

U 4 Ne

4

L 111

4.

L.

4.

a

4.

f.4

01.0

4' -la

aa

aa

G0

0MM~~~ -WM

4.

a ii- i w os is

a g

4.

f.4

01.0

4' -la

a

a

em

A.0

em

. -4-4

L--maa

em

em

A

4

*0

flm

Goli do l.

vC

50"C

0o1

Cow1 +

CL

a~

Co

a a

a

a

*0

Ans

Goi
d0 1.

vC

m

50

'C

0o1

Cow1 +

CL

a~

Co

a a

a

a

*0

flm

Goli do l.
v0637jp a, $+4 ;3m 10t 3b
v063aepl
v0639ld (=<tcl>), a, ;1m 4t 1b
v0640endtext
v0641calc romptr=romptr+19
v0642ag.ai
v0643com primitive to perform comparison between 2 16-bit numbers
v0644com list=result,argument 1, argument 2 ::::store, time, ext, c, i, addr
v0645begin text
v06461d de,(=<tcl>); 6m 20t 4b if arg2 lt arg1 then resl=ffh de=<tcl>
v06471dlh,(=<tcl>);5m 16t 3b
v06481d a, h; 1m 4t 1b
v0649and a ;1m 4t 1b set sign flag of arg1
v0650jp p,$+13 ;3m 10t 3b jump if arg1 is positive
v0651ld a, d ;1m 4t 1b arg1 =
v0652and a ;1m 4t 1b set sign flag of arg2
v0653jp m,$+18 ;3m 10t 3b arg2 = - arg1 = - comp backwards
v06541d a, 0 ;2m 7t 2b arg2 = + arg1 = - false
v0655jp $+24 ;3m 10t 3b
v06561d a, d ;1m 4t 1b
v0657and a ;1m 4t 1b set sign flag of arg2
v0658jp p,$+8 ;3m 10t 3b arg2 = + arg1 = +
v06591d a,11111111b;2m 7t 2b arg2 = - arg1 = + true
v0660jp $+14 ;3m 10t 3b
v0661bc h,de ;1m 4t 1b
v06621d a,00000000b;2m 7t 2b result false arg2 gt arg1
v0663jp z, $+7 ;3m 10t 3b
v0664jp m, $+4 ;3m 10t 3b
v0665cpl ;1m 4t 1b
v0666ld (=<tcl>), a ;4m 13t 3b
v0667endtext
v0668calc romptr=romptr+19
v0669calc a0 (i) ;27, 14, 4, 17, 12, 669, 704
v0670com primitive to generate a 25 millisecond clock. the value of
v0671com current time is stored in the channel 1 counter of the cpu
v0672com ctc chip. to access this particular time rdtm must be used
v0673com to read/rest the time use rdtmtime primitive.
v0674com note when running this does not have any over head in terms
v0675com of slowing the monitor since no interrupts are involved
v0676com the clock is limited to about one minute of elapsed time
v0677com channel 0 will be used for the clock at factory installed
v0678com address of f0
v0679com channel 1 is used to store current time at a factory installed
v0680com address of f1
v0681incl h.clock
v0682begin text
v0683jp eq<and> ;3m 10t 3b
v0684<initk>
v0685endtext
v0686calc initk=initk+1
v0687begin text
v06881d a,00110101b;2m 7t 1b counter 0 + load lab then mb+ mode2+ hex
v0689out (ff3h),a ;3m 11t 2b set mode control
v06901d a,050h ; 12m 7t 1b lab of 50000 decimal or c350 hex
v06891out (0f0h),a ; 3m 11t 2b load counter time channel 0
v06892d a,0c3h ; 2m 7t 1b lab of 2000 bcd
v06893out (0f0h),a ; 3m 11t 2b load counter time channel 0
v06894d a,01110001b ; 2m 7t 1b counter 1 + load lab than mb+ mode2+ hex
v06895out (0f3h),a ; 3m 11t 2b set mode control
v06896d a,00 ; 1m 7t 1b lab of ffff+1 hex
v06897out (0f1h),a ; 3m 11t 2b load counter time channel 1
v06898d a,00h ; 1m 7t 1b lab of ffff+1 hex
v06899out (0f1h),a ; 3m 11t 2b load counter time channel 1
v07001p el initik> ; 3m 10t 3b
v07010a<arnd>:nop ; 1m 4t 3b isolate the initialization
v07020endtext
v07030calc arnd<arnd+1
v07040calc romptr=romptr+27
v07050a<arnd=main (12,14,12,5,3,705,764)
v07060calc primitive to define controller setup and initialization
v07070calc list = empty : empty : storage, time, ext, calc, incl, addr
v07080inc1 h.processor (1)
v07090incl h.cardcage (1)
v07100calc romptr=36767
v07110calc romptr=102
v07120calc romptr=16384
v07130calc romptr=romptr-1
v07140calc romptr=romptr-32
v07150calc the rom pointer is set to start at 102 or 66h corresponds to reset
v07160calc location for the z-80. this will be the cold boot point for the
v07170calc controller. all initializations will be done immediately after
v07180calc this point. following the initializations will be the top of the
v07190calc polling loop for the task contingency pairs.
v07200calc to allow the use of a debug prom developed at the naval
v07210calc postgraduate school electrical engineering department the
v07220calc starting location is changed to 4096 to allow a the system
v07230calc to auto boot and to allow loading of memory from another
v07240calc computer via the dual uart card. the loading prom inhibits the
v07250calc use of the reset location because of the location of the code and
v07260calc the interrupt locations used in a debugger for the prolog system
v07270begin stext
v07280 2ilog z-80 based system
v07290
v07300
v07310; 1dsec
v07320; 1dsec
v07330; 1dsec
v07340
v07350
v07360; z80
v07370asag
v07380endtext
v07390calc debug .eq. 0 skip 4
v07400begin stext
v07410calc 100h
v07420calc start;jp 4000h ; start execution at 100h then jump to 4000h offset
v0743endtext
v0744begin text
v0745org <ramptr>
; ram pointer is pointing to top of memory - stack
v0746begin var: defb 0
; ram switch for software initializations
v0747estak: defb 32
; 32b define stack area
v0748org <ramptr>
; begin code after reserved interrupt area
v0749secold id sp, estak+32
; 3m 10t 3b initialize stack pointer
v0750ld a, 0
; 2m 7t 2b
v0751ld (initvar), a
; 3m 13t 3b set initialize to false
v0752di
v0753endtext
v0754if debug .eq. 1 skip 3
v0755begin text
v0756bi pl,<initkl>
; 3m 10t 3b do hardware initializations
v0757endtext
v0758if debug .eq. 0 skip 5
v0759begin text
v0760jp espvar
; if in debug can't do any initializations
v0761 ; therefore jump to espvar to get main loop started
v0762endtext
v0763incl s; wrtbin
v0764calc romptr=romptr+12
v0765s.inkey
; (argl:0,8:11,47,14,11,0,765,778)
v0766com primitive to add input routine for keyboard entry using the
v0767com 7303 keyboard/display board.
v0768com keyboard is a boolean flag indicating if board has previously been
v0769com called.
v0770com nkey is the number of the inkeys that have been requested.
v0771com argl is the boolean that is toggled by the keyboard. pressing
v0772com the appropriate key will cause the value of the boolean to be
v0773com complemented.
v0774begin text
v0775endtext
v0776calc romptr=romptr+11
v0777s.outdigit (argl:0,8:11,47,14,0,0,777,789)
v0778com primitive to add output routines to display a message to the
v0779com alphanumeric display on the 7303 keyboard/display board.
v0780com keybd is a boolean flag indicating if board has previously been
v0781com called.
v0782com nodigit is a boolean flag indicating that the outdigit routines have
v0783com been previously added. they will be called as a subroutine to
v0784com display the output. there is no limit to the number of different
v0785com messages that can be displayed.
v0786com argl is the boolean that is tested to determine if the led should
v0787com be lighted.
v0788begin text
v0789endtext
v0790s.consfp (nam, val, :0,32:5,0,8,0,780,788)
v0791com primitive to define data for floating point number
v0792com list=dataname, value=value, prec, stor, time, ext, c, i, addr
v0793begin text
v0794<nam>: defb <val> ; define a one byte integer exponent
v0795 defw <mant> ; define msw of mantissa
v0796          defm <man2> ;define law of mantissa
v0797endtext
v0798calc romptr=romptr+5
v0799s.addc    (resl, rarg, arg2:0, 16, 0, 16, 0, 16, 33, 144, 39, 18, 0, 799, 817)
v0800com primitive to add arg1 and arg2 and store in resl
v0801begin stmt
v08021d hl, (<arg>); 6m  20t 4b  load arg1 in hl pair
v08031d bc, (<arg>); 6m  20t 4b  load arg2 in bc pair
v08041d a, i;    1m  4t  1b
v0805add a, c;    1m  4t  1b  add lab
v08061d l, a;    1m  4t  1b
v08071d a, h;    1m  4t  1b
v0808adc a, b;   1m  4t  1b  add mb
v08091d h, a;    1m  4t  1b
v08101dp po, $+15; 3m  10t 3b  if no overflow store result
v08111p c, $+9;   3m  10t 3b  if carry the maximize minus reslt
v08121d hi, 7fffh; 3m  10t 3b  put in largest positive value
v08131p $-6;     3m  10t 3b
v08141d hl, 8000h; 3m  10t 3b  put in largest negative value
v08151d ((resl)),h; 6m  20t 4b  save result
v0816endtext
v0817calc romptr=romptr+33
v0818s.assigncons(var, data: 0, 0, 0, 0, 5, 20, 6, 7, 0, 818, 825)
v0819com primitive to assign a value of constant to a variable
v0820com list=var, data=var=var-prec, data=prec=stor, time, ext, calc, incl, addr
v0821begin stmt
v08221d a, <data> ; 2m  7t  2b  assign <data>
v08231d <var>, a ; 4m  13t 3b  to <var>
v0824endtext
v0825calc romptr=romptr+5
v0826s.assign (var, data: 0, 0, 0, 8, 6, 26, 8, 7, 0, 826, 833)
v0827com primitive to assign a value of one variable to another variable
v0828com list=var, data=var=var-prec, data=prec=stor, time, ext, calc, incl, addr
v0829begin stmt
v08301d a, <data> ; 4m  13t 3b  assign <data>
v08311d <var>, a ; 4m  13t 3b  to <var>
v0832endtext
v0833calc romptr=romptr+6
v0834s.assign (var, data: 0, 16, 0, 16, 7, 36, 11, 7, 0, 834, 841)
v0835com primitive to assign a value of one variable to another variable
v0836com list=var, data=var=var-prec, data=prec=stor, time, ext, calc, incl, addr
v0837begin stmt
v08381d hl, <data>; 6m  20t 4b  assign <data>
v08391d <var>, hl ; 5m  18t 3b  to <var>
v0840endtext
v0841calc romptr=romptr+7
v0842s.proc (nam: 1, 4, 1, 7, 0, 842, 849)
v0843com primitive to define procedure entry point
v0844com list=proc-name: empty: stor, time, ext, calc, incl, addr
v0845begin stmt
v0846procedure <nam>
v0847s<nam>;  nop ; 1m  4t  1b  entry point for <nam>
v0848endtext
v0849calc romptr=romptr+1
v0850calc (name=val, :0,8,1,0,0,6,0,850,856)
v0851com primitive to define data
v0852com list=data-name, value=value-prec, stor, time, ext, c, i, addr
v0853begin stext
v0854<name>: defb <val> ; reserve one byte for data
v0855endtext
v0856calc romptr=romptr+1
v0857com primitive to add one entry to monitor table
v0858#com list= func-name, task name:: empty:: t, e, c, i, address
v0859begin stext
v0860call @<func> :5m 17t 3b test for contingency <func>
v0861id a, @<func> :4m 13t 3b get contingency result
v0862call z, @<task> :5m 17t 3b if true execute task
v0863endtext
v0864calc romptr=romptr+10
v0865calc tabaccp2 (::, 0,0,868,879)

v0866com this is a dummy primitive to allow compatibility with the 8080
v0867com library. The functions that would be performed in this primitive
v0868com are all located in stabent. this has the effect of eliminating
v0869com intermediate table and increasing execution speed. If there are
v0870com wide variations in contingency/task speeds more memory will be
v0871com than in the 8080 primitive. Note: .main is also changed because
v0872com the elimination of the intermediate table
v0873#com list= func-name, task name:: empty:: t, e, c, i, address
v0874begin stext
v0875 : this space is deliberately void. this is a dummy primitive.
v0876endtext
v0877calc romptr=romptr+10
v0878var (name::0,8,0,0,0,3,0,880,888)

v0879com primitive to define storage for 8 bit variable integer or logical
v0880com list=data-name, value=value-prec, stor, time, ext, c, i, addr
v0881calc romptr=romptr-1
v0882begin stext
v0883org <romptr> :8 bit variable <name> in ram
v0884<name>: defb 0 ;0m 0t 1b
v0885org <romptr>

v0886endtext
v0887perform (name::3,17,5,6,0,889,895)

v0888com primitive to invoke a procedure
v0889com list=proc-name:: empty:: stor, time, ext, calc, incl, addr
v0890begin stext
v0891call @<name> ;5m 17t 3b perform procedure <name>

v0892endtext
v0893calc romptr=romptr+3
v0894com primitive to subtract arg2 from arg1 and store answer in rslt
v0895com list=rslt, arg1, arg2, 0,16,0,16,0,16,33,134,39,19,0,896,915
v0896com primitive to subtract arg2 from arg1 and store answer in rslt
v0897com list=rslt, arg1, arg2, precious:: t, e, c, i, addr

v0898begin stext
v0899id hi, (<arg1>) :6m 20t 4b load arg1 in hi pair
v0899id bc, (<arg2>) :6m 20t 4b load arg2 in bc pair
v0021d a, l  ;im 4t  1b  subtract lab
v0025d a, b  ;im 4t  1b  subtract mb
v0002d c  ;im 4t  1b
v0041d a, i  ;im 4t  1b
v0051d a, h  ;im 4t  1b
v0061d a, c  ;im 4t  1b
v0007d h, n  ;im 4t  1b
v0083j p, 0  ;im 8t  1b
v0004j p, 8  ;im 8t  1b
v0010d h, 7  ;im 10t  1b
v0011j p, 6  ;im 10t  1b
v0012d h, 9  ;im 10t  1b
v0013d h, 80  ;im 10t  1b
v0014endtext
v015calc romptr=romptr+33
v016s.add (rlt, arg1, arg2: 0, 32, 0, 32, 0, 32, 205, 286, 597, 126, 0, 916, 1042)
...
v1061f <cont>.eq.0 skip 4
v1062begin stax
v1063ld a, 0 ;2m 7t 2b reset contingency <cont>
v1064ld (<cont>).a ;4m 13t 3b following completed task
v1065endtext
v1066begin stax
v1067ret ;3m 10t 1b return to monitor,exit <nam>
v1068endtext
v1069calc romptr=romptr+6
v1070s.com primitive to define data for 16 bit integer
v1071com list=data-name,value:value-prec,stor,time,ext,c,1 addrs
v1072begin stax
v1073<name>: defw <val> ;define a two byte integer
v1074endtext
v1075calc romptr=romptr+2
v1076s.var (name:0,16:0,0,0,3,0,1077,1085)
v1077com primitive to define storage for 16 bit variable integer
v1078com list=data-name,value:value-prec,stor,time,ext,c,1 addrs
v1079calc romptr=romptr - 2
v1080begin stax
v1081org <romptr> ;16 bit variable <name> in ram
v1082<name>: defw 0 ;0m 0t 2b
v1083org <romptr>
1084endtext
v1085fptoleee (relt,arg):0,32,0,32,105,558,140,67,0,1086,1153)
v1086com primitive to convert the floating point to the ieee standard
v1087com format exp,byte2,sign,byte1,byte4,byte3 to sign exp,magnitude
v1088com list=relt,arg:=precisions:s,t,a,c,1,addr
v1089begin stax
v1090ld a,(<arg>) ;4m 13t 3b get exponent of arg2
v1091ld d a, 000h ;2m 7t 2b put in offset
v1092ld c, e ;1m 4t 1b save exponent in c'
1093ld a, 000h ;2m 7t 2b mask for sign
v1094sand d ;1m 4t 1b get sign of relt
v1095ld b, 6 ;2m 7t 2b number of shifts
v1096ld de,(<arg>+1);5m 16t 4b first word
v1097ld h1,(<arg>+3);5m 16t 3b second word
v1098ld m,$+010h ;3m 10t 3b number is minus
1099ld r1 d ;2m 8t 2b rotate mantissa 6 places
v1100ld r1 e ;2m 8t 2b to the right
v1101ld r1 h ;2m 8t 2b
1102ld r1 l ;2m 8t 2b
v1103ld d,$-008h ;3m2 13t 2b put exp in front of mantissa
v1104ld d, c ;1m 4t 1b put exp into leading
1105ld d, c ;1m 4t 1b
v1106ld r1 d ;2m 8t 2b of mantissa
1107ld r1 e ;2m 8t 2b sign is positive
1108ld r1 h ;2m 8t 2b
v1109ld r1 l ;2m 8t 2b
1110ld $+031h ;3m 10t 3b store relt
1111ld a, l ;1m 4t 1b change to sign magnitude
v1112ld l, a ;1m 4t 1b from 2's complement
v11141d a, h ;lm 4t 1b
v1115cpl ;lm 4t 1b
v11161d h, a ;lm 4t 1b
v11171d a, e ;lm 4t 1b
v1118cpl ;lm 4t 1b
v11191d e, a ;lm 4t 1b
v11201d e, d ;lm 4t 1b
v1121cpl ;lm 4t 1b
v11221d d, a ;lm 4t 1b
v11231nc l ;lm 4t 1b
v11241d h, e ;lm 4t 1b
v1126adc e, o ;lm 7t 2b
v11261d h, e ;lm 4t 1b
v11271d e, a ;lm 4t 1b
v1128adc e, o ;lm 7t 2b
v11291d e, a ;lm 4t 1b
v11301d d, a ;lm 4t 1b
v11321d d, a ;lm 4t 1b
v1133rd 1 l 2a 8t 2b
to the right
v1134rr e ;lm 8t 2b
v1135rr h ;lm 8t 2b
v1136rr l ;lm 8t 2b
v1137dznz $000h 3m2 13t 8 2b
v11381d d, c ;lm 4t 1b
v1139rd 1 l 2a 8t 2b
v1140rr e ;lm 8t 2b
v11411rr d, h ;lm 8t 2b
v1142rr l ;lm 8t 2b
v1143set 7, d ;lm 8t 2b
v11441d e, d ;lm 4t 1b
v11451d (<rel>), a 3m 13t 3b
v11461d a, e ;lm 4t 1b
v11471d (<rel>+1), a 3m 13t 3b
v11481d a, h ;lm 4t 1b
v11491d (<rel>+2), a;3m 13t 3b
v11501d l, e ;lm 4t 1b
v11511d (<rel>+3), a;3m 13t 3b
v1152endtext
v1153cali romptr=romptr+105
v1154a. loc (loc :11,4,1,6,6,1154,1160)
v1155com primitive to define a label (location)
v1156com list=label-name :empty: storage, time, ext, calc, incl, addr
v1157begin stext
v1158<loc> ; define location <loc>
v1159endtext
v1160cali romptr=romptr+1
v1161a.end (1:3,10,3,8,10,1161,1171)
v1162com primitive to end software listing and complete implementation
v1163com list=empty:empty:stor, time, ext, calc, incl, addr
v1164begin stext
v11651initlk:jp espwr ;3m 10l 3b initialization of hardware is complete
v1166start top of main monitor loop
; end of software listing ready for assembly

vi167end

vi168endtext

vi168calc romptr=romptr+3

vi170com put in memory needed for implementation in ram and rom

vi171incl h.memory (;

vi172a.every (num: 7,34,9,10,0,1172,1182)

vi173com primitive to define dummy function for every-period statement

vi174com list=proc-nam rempty: storage, time, ext, calc, incl, addr

vi175begin text

vi176dummy procedure for every-period type contingency

vi177<name>: nop ;3m 4t 1b dummy function entry point

vi178 id a,1 ;2m 7t 2b force function value

vi179 id (num),a,3m 13t 3b to true value (1)

vi180 ret ;3m 10t 1b return to monitor

vi181endtext

vi182calc romptr=romptr+7

vi183fun (rslt, arg1, arg2:0,32,0,32,0,32:206,2269,597,126,0,1183,1309)

vi184com primitive to subtract 2 floating point numbers and store rslt

vi185com form is rslt le arg1 - arg2

vi186com format exp,byte1,byte2,byte3,byte4

vi187com mantissa is in form e1.e2....e11.e12 or 10.e2....e11.e12

vi188com s is sign of number

vi189com list=rslt, arg1: precision: s, t, e, c, i, addr

vi190begin text

vi191id d,e,(<arg2>+1):6m 20t 4b first word

vi192id h1,(<arg2>+1):5m 16t 3b first word

vi193id d,e,(<arg2>+1):6m 20t 4b second word

vi194id h1,(<arg2>+1):5m 16t 3b second word

vi195id d,e,(<arg2>+1):4m 13t 3b get exponent of arg1

vi196id b,a ;1m 4t 1b save exponent in b'

vi197id d,e,(<arg1>);4m 13t 3b get exponent of arg2

vi198id c,a ;1m 4t 1b save exponent in c'

vi199id d,e,(<arg2>);4m 4t 1b main registers

vi200id d,e,(<arg2>);5m 20t 4b mask for sign

vi201id d,e,(<arg2>);5m 16t 3b sign of arg1

vi202id d,e,(<arg2>);4m 13t 3b save sign of arg1 main

vi203id d,e,(<arg2>);4m 13t 3b mask for sign

vi204id d,e,(<arg2>);4m 13t 3b sign of arg2

vi205id c,a ;1m 4t 1b save sign of arg2 main

vi206id d,e,(<arg2>);4m 13t 3b prime registers

vi207id d,e,(<arg2>);4m 13t 3b put arg2 exponent in a

vi208id a,b ;1m 4t 1b subtract exponent arg1

vi209id c,a ;1m 4t 1b main registers

vi210id c,a ;1m 4t 1b subtract arg1 exponent

vi211id d,e,(<arg1>);4m 13t 3b main registers

vi212id z,$+04h ;3m 10t 3b exponents are equal no shift required

vi213id m,$+024h ;3m 10t 3b arg1 gt arg2 exponent

vi214id m,$+025h ;3m 10t 3b is shift it 31

vi215id a,31 ;2m 3t 7t 2b limit to 31 shifts

vi216id a,7 ;2m 3t 7t 2b shift msb arg2 right retain sign

vi217id l ;2m 3t 7t 2b rotate byte2 arg2

vi218id c,a ;1m 4t 1b prime registers

vi219id c,a ;1m 4t 2b rotate byte3 arg2
v1220rr 1 ;2m 8t 2b  rotate byte4 arg2
v1221jp nc,$+005h;3m 10t 3b  stickey bit
v1222set 0, 1;2m 8t 2b  set least significant bit
v1223exx ;1m 4t 1b  return to main registers
v1224dec a ;1m 4t 1b
v1225jp nz,$-010h ;3m 10t 3b  check to see if exp are aligned
v1226ld c, b;1m 4t 1b  sign of result will be same as arg1
v1227exx ;1m 4t 1b  prime registers
v1228ld c, b;1m 4t 1b  exponent for result is in c'
v1229exx ;1m 4t 1b  main registers
v1230jp $+01dh ;3m 10t 3b  justified continue with add
v1231cp -31 ;2m 7t 2b  is shifts lt -31
v1232jp p,$+005h ;3m 10t 3b  is shifts it 31
v1233ld a, -31 ;2m 7t 2b  limit to 31 shifts
v1234ar a d;2m 8t 2b  shift mb argl right keep sign
v1235rr e;2m 8t 2b  rotate byte2 arg1
v1236exx ;1m 4t 1b  prime registers
v1237rr rr d;2m 8t 2b  rotate byte3 arg1
v1238rr e;2m 8t 2b  rotate byte4 arg1
v1239jp nz,$+005h ;3m 10t 3b  stickey bit
v1240set 0, e;2m 8t 2b  set least significant bit
v1241exx ;1m 4t 1b  main registers
v1242inc a;1m 4t 1b  add one to negative number till 0
v1243jp nz,$-010h ;3m 10t 3b
v1244exx ;1m 4t 1b  prime register begin subtract
v1245and e;1m 4t 1b  clear carry
v1246bc h1, de;1m 15t 2b  subtract 3 and 4 bytes of mantissa
v1247exx ;1m 4t 1b  main registers
v1248ld a, l;1m 4t 1b  prepare to add byte2
v1249bc e a;1m 4t 1b  sub byte2
v1250ld l, d;1m 4t 1b  save byte2
v1251ld a, h;1m 4t 1b  prepare to add byte1
v1252bc a, d;1m 4t 1b  sub byte1
v1253ld h, a;1m 4t 1b  save byte1 of result
v1254jp pg, $+006h ;3m 12t 1b  correct for overflow
v1255jp $+00ah ;3m 10t 3b  no overflow
v1256rr h;2m 8t 2b  shift mb of mantissa of rslt right
v1257rr l;2m 8t 2b  rotate byte2
v1258exx ;1m 4t 1b  prime registers
v1259rr h;2m 8t 2b  rotate byte3
v1260rr l;2m 8t 2b  rotate byte4
v1261inc c;1m 4t 1b  correct exponent in c'
v1262exx ;1m 4t 1b  return to main registers
v1263ld a, 0;2m 8t 2b  put zero in accumulator -- zero rslt?
v1264cp h;1m 4t 1b  is mb zero
v1265jr nz,$+010h ;2m 7t 2b  quit test of rslt if non zero
v1266cp l;1m 4t 1b  is byte2 zero
v1267jr nz,$+00dh;2m 7t 2b  quit test of rslt if non zero
v1268exx ;1m 4t 1b  prime registers
v1269cp h;1m 4t 1b  is byte 2 zero
v1270jr nz,$+008h;2m 7t 2b  quit test of rslt and flip reg non zero
v1271cp l;1m 4t 1b  is byte 3 zero
v1272jr nz,$+005h;2m 7t 2b  quit test of rslt and flip reg non zero
vi326endtext
vi327calc romptr=romptr+8
vi328as a monitor ( ; 7.27,7,10.0,1328,1338)
vi329com primitive to define p2 monitor as controller supervisor
vi330com list = empty:empty: storage,time,ext,calc,int,addr
vi331begin stat
vi332*monitor section*
vi333as espvr:id a.(@initvar);3m 1st 3b mark top of the polling loop and test
vi334; to see if the initializations have been
vi335and a ; 1m 4t 1b done. if not do so
vi336jp z,@init;3m 1st 3b
vi337endtext
vi338calc romptr=romptr+7
vi339as tabend ( ; 3,10,3,6,0,1339,1345)
vi340com subroutine to define and of monitor table
vi341com list= empty:empty:a,t,e,c,i,addr)
vi342begin stat
vi343jp espvr; go to the top of the polling loop of monitor table
vi344endtext
vi345calc romptr=romptr+3
vi346as varfp (name;0,16,0,0,0,3,0,1346,1356)
vi347com primitive to define storage for a floating point number
vi348com list= data-name,value: value: prec: stor:time: ext: c: i: addr)
vi349calc romptr=romptr - 9
vi350begin stat
vi351org <romptr> ;fp variable <name> in ram
vi352as <name> ; defw 0 ;0m 0t 2b
vi353defw 0 ; 2b
vi354defb 0 ; 1b
vi355org <romptr> ;
vi356endtext
vi357as .fmx (srt,arg1,arg2:0,32,0,32,0,32,204,2263,597,0,0,1357,1519)
vi358com primitive to multiply two floating point numbers and
vi359com store the result in srt
vi360com srt = arg1 * arg2
vi361com format exp:byte1,byte2,byte3,byte4
vi362com mantissa is in form .xxxx
vi363com s is sign of number
vi364com is two's complement number
vi365com list= srt,arg1:presentions:st.e,c,i:addr
vi366begin stat
vi367d b, (arg1>1);6m 20t 4b first word
vi368h1,0 ;1m 10t 3b zero hi for result
vi369e 1m 4t 1b prime registers
vi369d b, (arg1>3);6m 20t 4b second word
vi370h1,0 ;1m 10t 3b zero hi for result
vi371d b, (arg1<1);6m 20t 4b get arg1 exponent
vi372id lx,<arg2>;4m 14t 4b put address of arg2 in lx
vi373add s, (lx+0);5m 19t 3b results exponent
vi374p pe,$227 ;3m 12t 3b if exponent is out of range fix
vi375id c, a ;1m 4t 1b save exponent in c'
vi376id c, a ;1m 4t 1b main registers
vi377id s, (lx+3);5m 19t 3b load byte 4 of mantissa
vi378d b, 8 ;2m 7t 2b prepare for mult 4 byte
v1379arl a ; 2m 8t 2b shift multiplier
v1380jp nc, $8+8 ; 3m 10t 3b if no 1 don’t add
v1381eex x ; 1m 4t 1b prime register beginning of add
v1382add hl, de ; 3m 11t 1b add 3 and 4 bytes of mantissa
v1383eex x ; 1m 4t 1b main registers
v1384adc hl, de ; 4m 15t 2b add bytes 1 and 2
v1385 rr h ; 2m 8t 2b shift mab of mantissa of result right
v1386rr l ; 2m 8t 2b rotate byte2
v1387eex x ; 1m 4t 1b prime registers
v1388rr h ; 2m 8t 2b rotate byte3
v1389rr l ; 2m 8t 2b rotate byte4
v13890eex x ; 1m 4t 1b return to main registers
v1390l djn $-20 ; 3m 2 13t 2b
v1391ld a, (1x+4) ; 5m 19t 3b load byte 4 of mantissa
v1392ld b, 8 ; 2m 7t 2b prepare for mult 4 byte
v1393ld r l ; 2m 8t 2b shift multiplier
v1394ld bc, $8+8 ; 3m 10t 3b if no 1 don’t add
v1395eex x ; 1m 4t 1b prime register beginning of add
v1396add hl, de ; 3m 11t 1b add 3 and 4 bytes of mantissa
v1397eex x ; 1m 4t 1b main registers
v1398adc hl, de ; 4m 15t 2b add bytes 1 and 2
v1399rr h ; 2m 8t 2b shift mab of mantissa of result right
v1400rr l ; 2m 8t 2b rotate byte2
v1402eex x ; 1m 4t 1b prime registers
v1403rr h ; 2m 8t 2b rotate byte3
v1404rr l ; 2m 8t 2b rotate byte4
v1405eex x ; 1m 4t 1b return to main registers
v1406djn $-20 ; 3m 2 13t 2b
v1407ld a, (1x+1) ; 5m 19t 3b load byte 4 of mantissa
v1408ld b, 8 ; 2m 7t 2b prepare for mult 4 byte
v1409rl rl ; 2m 8t 2b shift multiplier
v1410ld bc, $8+8 ; 3m 10t 3b if no 1 don’t add
v1411eex x ; 1m 4t 1b prime register beginning of add
v1412add hl, de ; 3m 11t 1b add 3 and 4 bytes of mantissa
v1413eex x ; 1m 4t 1b main registers
v1414adc hl, de ; 4m 15t 2b add bytes 1 and 2
v1415 rr h ; 2m 8t 2b shift mab of mantissa of result right
v1416rr l ; 2m 8t 2b rotate byte2
v1417eex x ; 1m 4t 1b prime registers
v1418rr h ; 2m 8t 2b rotate byte3
v1419rr l ; 2m 8t 2b rotate byte4
v1420eex x ; 1m 4t 1b return to main registers
v1421djn $-20 ; 3m 2 13t 2b
v1422ld a, (1x+2) ; 5m 19t 3b load byte 4 of mantissa
v1423ld b, 8 ; 2m 7t 2b prepare for mult 4 byte
v1424rl rl ; 2m 8t 2b shift multiplier
v1425ld bc, $8+8 ; 3m 10t 3b if no 1 don’t add
v1426eex x ; 1m 4t 1b prime register beginning of add
v1427add hl, de ; 3m 11t 1b add 3 and 4 bytes of mantissa
v1428eex x ; 1m 4t 1b main registers
v1429adc hl, de ; 4m 15t 2b add bytes 1 and 2
v1430 rr h ; 2m 8t 2b shift mab of mantissa of result right
v1431rr l ; 2m 8t 2b rotate byte2
v1538 id hl,$+00eh  ;3b
v1539ldir  ;2b
v1540ld b, 2  ;2b number of bytes for output
v1541ld c, <natodp>  ;2b atodp = 0 port to be loaded
v1542otir  ;2b
v1543jp $+008h  ;3b
v1544defb 0cfh  ;1b
v1545defb 080h  ;1b
v1546defb 007h  ;1b
v1547defb 04fh  ;1b
v1548defb 007h  ;1b
v1549endtext
v1550calc initlk=initlk+1
v1551begin stext
v1552jp @<initlk>  ;3b jump to next hardware initialization
v1553nopl  ;1b end of initialization for first 8 bit a to d board
v1554endtext
v1555if natode .lt. 16 skip 4
v1556if natode .gt. 16 skip 3
v1557incl h.atod 11
v1558calc natodp = 4
v1559com the second sd board is ported at address 000<natodp>
v1560begin stext
v1561jp $+01ch  ;3b start of initialization for first 8 bit a to d board
v1562id b, 3  ;2b number of bytes to output
v1563id c, <natodp>  ;2b atodp = 4 port to be loaded
v1564id hl,$+00ch  ;3b
v1565otir  ;2b
v1566ld b, 2  ;2b number of bytes for output
v1567ld c, <natodp>  ;2b atodp = 4 port to be loaded
v1568otir  ;2b
v1569jp $+008h  ;3b
v1570defb 0cfh  ;1b
v1571defb 080h  ;1b
v1572defb 007h  ;1b
v1573defb 04fh  ;1b
v1574defb 007h  ;1b
v1575endtext
v1576calc initlk=initlk+1
v1577begin stext
v1578jp @<initlk>  ;3b
v1579nopl  ;1b end of initialization for second 8 bit a to d board
v1580endtext
v1581com list=source: input-lines, ., condas, ,. storage, time, ext, calc, incl, addr
v1582if natode .lt. 16 skip 1
v1583calc scrtct = natode - 16
v1584if debug .eq. 0 skip 11
v1585begin stext
v1586ld de, $+0013h
v1587ld c, 9
v1588call <bdos>
v1589ld c, 1 ;bdos call requesting a character from the console
v1590call <bdos>
v1591ld (<signam>), a ; save the results of the input
v1592jp $+021h
v1593 defm "requesting input for <signam>"

v1594 n0p ; keep message out of straight line execution
v1595endtext
v1596f debug.eq.1 skip 13

v1597begin stext
v1598ld a,<scratch> ; 3m 1t 3b channel to be selected for input
v1599out(<netodp>),a ; 3m 1t 2b clear control
v1600or 060h ; 1m 4t 1b set start conv, addr latch
v1601out(<netodp>),a ; 3m 1t 2b issue a/d control
v1602in a,(<netodp>); 3m 1t 2b read status
v1603bit r,r ; 2m 8t 2b check done bit
v1604jr z, $-4 ; 2m 7t 2b loop till done
v1605converstion time is 138 microseconds = one full execution of
v1606done polling loop
v1607in a,((netodp)>2); 3m 1t 2b read a/d data
v1608ld (<signam>),a ; 3m 1t 3b save results of input in <signam>

v1609endtext
v1610calc romptr = romptr + 43
v1611s.blockexit (switch:0,8:5,20,5,8,0,1611,1619)

v1612com primitive to set the switch associated with a submonitor to
v1613com false and thus cause the submonitor to exit to the main
v1614com monitor at the bottom of the submonitor loop

v1615begin stext
v1616ld a, 0 ; 2m 7t 2b set switch to false
v1617ld (<switch>),a ; 3m 13t 3b set switch to false

v1618endtext
v1619calc romptr=romptr+5
v1620s.blockcons (submon:0,255;1,4,1,9,0,1620,1629)

v1621com primitive to mark the beginning of a submonitor, it is used
v1622com along with the blockend to mark the end of a block to mark the
v1623com end of a submonitor block and the blockexit to orderly leave
v1624com block after all code is tested in the block.

v1625com blockstart will test and jump out of main monitor to submonitor

v1626begin stext
v1627submon>: n0p ; 1m 4t 1b mark top of submonitor

v1628endtext
v1629calc romptr=romptr+1
v1630s.blockstart(switch,submon:0,8,0,255;7,27,7,9,0,1630,1639)

v1631com primitive to cause the variable switching on the submonitor to
v1632com be tested and if true the submonitor to be executed

v1633begin stext
v1634ld a, (<switch>); 3m 13t 3b get switch value
v1635and a ; 1m 4t 1b set flags
v1636jp nz <submon> ; 3m 10t 3b if true execute the

v1637; the submonitor

v1638endtext
v1639calc romptr=romptr+7
v1640s.mult (rst, arg1,arg2:0,16,0,16,0,16,39,1105,289,22,0,1640,1662)

v1641com multiply 2 16 bit numbers and get 16 bit result

v1642begin stext
v1643ld de,(arg1>>8); 6e 20t 4b put arg1 in de
v16641d bc,(<arg2>);6m 20t 4b load arg2
v16651d a, b 2m 4t 1b split arg2 to aandc
v16661d hl, 0 ;3m 10t 3b clear rslt
v16671d b, 15d ;2m 7t 2b set counter to 7bits
v1668rrs 1m 4t 1b
v1669rr c 2m 8t 2b
v1650jpc nc, $+4 ;3m 10t 3b
v1651add hl, de ;3m 11t 1b
v1652le a 2m 8t 2b
v1653rl d 2m 8t 2b
v1654djnz $-00bh ;3m 13t 2b +7 +2m 8t on last time
v1655rru 1m 4t 1b
v1656rr c 2m 8t 2b
v1657jpc nc, $+8 ;3m 10t 3b
v1658and a 1m 4t 1b
v1659bc hl, de ;4m 15t 2b
v16601d (<rslt>, hl);5m 16t 3b save result
v1661endtext
v1662calc romptr=romptr+39
v1663es.out (signam,direct:0,8,14,45,12,6,10,1887)
v1664com primitive to allow binary output on any of 8 channels
v1665com signam is the name of an 8 bit variable for output
v1666com ninout is the number of 8 bit in/out channels that have been
v1667com requested, this primitive is current setup to allow only
v1668com one input/output board.
v1669calc ninout = ninout + 1
v16701f ninout .le 6 skip 4
v1671begin text
v1672you have requested more than 6 in/out channels on the mdx-dio1b
v1673board. currently the primitive is set up to light only 6 lights
v1674endtext
v1675if ninout ,gt, 1 skip 1
v1676incl h,ninout (;;)
v1677calc scrch= 2+*(ninout-1)+128
v1678begin text
v16791d a, (<signam>);3m 13t 3b find out if switch is on
v1680cpp 1 2m 7t 2b
v1681jp nz , ee<arr> ;3m 10t 3b if switch is off don't output address
v1682@<arr>:out (<scrch>), a ;3m 11t 2b light the lamp
v1683jp al<arr> ;3m 10t 3b keep light on forever
v1684@<arr>nop 1m 4t 1b don't light the lamp
v1685endtext
v1686calc arr=arr+1
v1687calc romptr = romptr + 14
v1688smult (rst, arg1, arg2;0,8,0,8,0,16;33,521,137,21,0,1689,1709)
v1689com multiply 2 8 bit number and get 16 bit result
v1690begin text
v16911d a, (<arg1>);3m 13t 3b put arg1 in a
v16921d a, a 1m 4t 1b
v16931d a, (<arg2>);2m 7t 2b load arg2
v16941d hl, 0 ;3m 10t 3b clear rslt
v16951d d, h 1m 4t 1b clear d for shifts
v16961d b, 7 ;2m 7t 2b set counter to 7bits
v1697rrc  ;1m 4t 1b
v1698jp nc,$+4  ;3m 10t 3b
v1699add hi, de  ;3m 11t 1b
v1700lea e     ;2m 8t 2b
v1701rl d      ;2m 8t 3b
v1702jnz $-9    ;3m 13t 2b *7 +2m 8t on last time
v1703rrc  ;1m 4t 1b
v1704jp nc,$+6  ;3m 10t 3b
v1705sand a    ;1m 4t 1b
v1706sbc hi, de ;1m 15t 2b
v1707ld (<rslt>),nl;5m 16t 3b  save result
v1708endtext
v1709calc romptr=romptr+33
v1710e mul  ;(rslt, arg1, arg2:0,8,0,8,0,8:34,522,137,22,0,1710,1732)
v1711com binary multiplication primitive
v1712begin text
v1713ld e,(<arg1>);3m 13t 3b  put arg1 in e
v1714ld e,a;1m 4t 1b
v1715ld a,(<arg2>);2m 7t 2b  load arg2
v1716ld hi,0;3m 10t 3b
v1717ld d,h;1m 4t 1b  clear rslt
v1718ld b,7;2m 7t 2b  set counter to 7 bits
v1719rrc  ;1m 4t 1b
v1720jp nc,$+4  ;3m 10t 3b
v1721add hi,de;3m 11t 2b
v1722lea e;1m 8t 2b
v1723rl d;2m 8t 2b
v1724jnz $-9 ;3m 13t 2b *7 +2m 8t on last time
v1725rrc  ;1m 4t 1b
v1726jp nc,$+6  ;3m 10t 3b
v1727and a;1m 4t 1b
v1728sbc hi,de;4m 15t 2b
v1729ld a,1;1m 4t 1b  truncate result to 8 bits
v1730ld (<rslt>),a;4m 13t 3b  save result
v1731endtext
v1732calc romptr=romptr+34
v1733calc (switch,submon:0,8,0,265,10,37,10,9,0,1733,1742)
v1734com primitive to mark the end of a submonitor, it will test switch
v1735com if switch is false it will return to the supervisor.

v1736begin text
v1737ld e,(<switch>);3m 13t 3b  get switch
v1738and a;1m 4t 1b  set flags
v1739jp z, espvar;3m 10t 3b  if false goto main monitor
v1740jp <submon> ;3m 10t 3b  else jump to top of submonitor
v1741endtext
v1742calc romptr=romptr+10
v1743calc (3:10,3,6,0,1743,1749)

v1744com primitive to cause the system to warm boot. That is to reinitialize
v1745com the stack and restart the main monitor loop.

v1746begin text
v1747jp espvar;3m 10t 3b
v1748endtext
v1749calc romptr=romptr+3
v1750 s fcns (arg1, fmark: 0, 8, 0, 255; 7, 27, 7, 9, 0, 1750, 1759)
v1751 com primitive to generate an if-then statement. It is used with
v1752 com the construction if and to mark the end of the executable
v1753 com statement.
v1754 begin stext
v1755 id a, (<arg1>); 3m 1st 3b get result of logical expression
v1756 and a; 1m 4t 1b
v1757 z p, <ifmark>; 3m 10t 3b
v1758 endtext
v1759 calc romptr=romptr+7
v1760 =cold (::3, 10, 3, 7, 0, 1760, 1767)
v1761 com primitive to cause the system to cold boot. That is to reinitialize
v1762 com all I/O ports and all variables that have a hard assignment as
v1763 com well as resetting the stack.
v1764 begin stext
v1765 p ecold ; 3m 10t 3b
v1766 endtext
v1767 calc romptr=romptr+3
v1768 dtos (signam: 0, 8, 43, 641, 23, 10, 12, 1768, 1825)
v1769 com primitive to convert a digital signal to an analog signal
v1770 com signam is the name of the signal to be converted
v1771 com initlk is a counter indicating the last link to jump to in any
v1772 com hardware required initialization.
 v1773 com ndtos is the number of 8 bit d to a ports that have been requested
v1774 com this is used to allow more than one channel per d to a port.
 v1775 com this primitive is currently set up to include 2 d to a boards
v1776 com for a total of 32 possible channels.
 v1777 com endrd is a scratch link for miscellaneous jumps
v1778 calc ndtos = ndtos + 1
v1779 f ndtos .gt. 1 skip 3
v1780 incl h. dtao (::)
v1781 calc dtos = 0
v1782 com the first a to d board is ported at address 000<dtos>
 v1783 begin stext
v1784 p around ; 3b start of initialization for first 8 bit a to d board
v1785 intlk = intlk + 1; 2b number of bytes to output
v1786 endtext
v1787 calc intlk = intlk + 1
v1788 begin stext
v1789 p endlk ; 3b jump to next hardware initialization
v1790 around: nol; 1b end of initialization for first 8 bit a to d board
v1791 endtext
v1792 if ndtos .lt. 16 skip 4
v1793 f ndtos .gt. 16 skip 3
v1794 incl h. dtao (::)
v1795 calc dtos = 4
v1796 com the second dtao board is ported at address 000<dtos>
 v1797 begin stext
v1798 dtos around2 ; 3b start of initialization for second 8 bit a to d board
v1799 intlk = intlk + 1; 2b number of bytes to output
v1800 endtext
v1801 calc intlk = intlk + 1
v1802 begin stext
v1803 jap @<initial> ; 3b jump to next hardware initialization
v1804 arround2: nop ; b end of initialization for first 8 bit a to d board
v1805 andtext
v1806 if ndtose .1t. 16 skip 1
v1807 calc scrch = ndtose - 16
v1808 if debug .eq. 0 skip 11
v1809 begin stext
v1810 Id ds, @<arnd>
v1811 id c, 9
v1812 call <bdos>
v1813 andtext
v1814 id a, (@<signam>)
v1815 call @<rtbin
v1816 jap @<arnd>
V1817 defm "output from d to a channel <signam> 1as"
V1818 @<arnd> ; nop ; keep message out of straight line execution
v1819 endtext
v1820 calc arnd = arnd + 1
v1821 if debug .eq. 1 skip 13
v1822 begin stext
v1823 output for board
v1824 andtext
v1825 calc romptr = romptr + 4
v1826 calc clockons (1: 43, 14, 4, 7, 0, 1826, 1870)
V1827 com primitive to generate 1 millisecond clock. The value of
V1828 com current time is stored in atime. time will be stored as a
V1829 com 16 bits positive integer variable. com channel 0 will be used for the clock at factory installed
V1830 com address of f0
V1831 com interrupt has an overhead of 32m 113t every millisecond
V1832 com ramptr=ramptr-2
V1833 begin stext
V1834 org 1038; offset interrupt location
V1835 push af
V1836 push hl
V1837 push af
V1838 push hl
V1839 out (0F4h), a
V1840 out (0F4h), a
V1841 out (0F4h), a
V1842 out (0F4h), a
V1843 inc (hl)
V1844 pop hl
V1845 pop af
V1846 ret 1
V1847 org <ramptr>
V1848 etime; defw 0 ; set beginning of time to 0
V1849 org <romptr>
V1850 jap @<arnd> ; 3m 10 t 3b
V1851 @<initial>:
V1852 andtext
V1853 calc initial=initial+1
V1854 begin stext
V1855 df ; im 4t 1b protect system from interrupts
v18581m 1: 12m 8t 2b set interrupt to mode 1
v18571d 1, 00110100b: 2m 7t 1b counter 0 + load 1sb then mb + mode2+ bcd
v1856out (DF9h), a; 3m 11t 2b set mode control
v18551d a, 00; 2m 7t 1b load of 2000 bcd
v1854out (DF8h), a; 3m 11t 2b load counter time channel 0
v18531d a, 20h; 2m 7t 1b load of 2000 bcd
v1852out (DF9h), a; 3m 11t 2b load counter time channel 0
v18513d a, 00000010b; 2m 7t 1b set interrupt on channel 0
v1854out (DF4h), a; 3m 11t 2b
v18551d 1, 4t 1b enable interrupts
v1854jp ei<initlk> 3m 10t 3b
v1857<endtext>
v1858calc arnd=arnd+1
v1859calc romptr=romptr+43
v1871s. ifend (ifmark: 0, 255: 1, 4, 1, 7, 0, 1871, 1878)
v1872com primitive to mark the end of the condition to be executed in
v1873com an if-then statement.
v1874begin stext
v1875<ifmark> nop 1m 4t 3b end of statements to be executed
v1876 ;
if condition was true in condition was true
v1877<endtext>
v1879calc romptr=romptr+1
v1879s. initialcom (1, 1, 4, 1, 5, 0, 1879, 1884)
v1880com primitive to mark the beginning of the things to be initialized.
v1881begin stext
v1882initial; nop 1m 4t 1b mark top of initialization
v1883<endtext>
v1884calc romptr=romptr+1
v1885s. initialend (1, 8, 30, 8, 9, 0, 1885, 1894)
v1886com primitive to mark the end of the things to be initialized.
v1887com also set initvar flag to true so that initialization is done
v1888com only once.
v1889begin stext
v18901d a, 1 12m 7t 2b
v18911d (ainitvar), a; 3m 13t 3b set initvar to true
v1892jp esper 3m 10t 3b execute main monitor
v1893<endtext>
v1894calc romptr=romptr+8
v1895s. when (wend, whotop: 0, 255: 3, 10, 3, 7, 0, 1995, 1992)
v1896com primitive to generate the end in a while do statement.
v1897com it marks the end of the statement that are to be executed
v1898com if the condition is true.
v1899begin stext
v1900<whend> jp <whotop> 1m 10t 3b
v1901<endtext>
v1902calc romptr=romptr+3
v1903s. messout (arg1: 16; , 0, 0, 1903, 1917)
v1904com primitive to print a message to the screen when the debug switch
v1905com is set to true
v1906begin stext
v1907messout:id de, string
v19081d c, 0
v1809call <bdos>
v1810ld de, argl
v1811call <bdos>
v1812ld de string
v1813call <bdos>

; and return to the calling program
v1814string; def "output for $"
v1815stringl; def "$"
v1817endtext
v1818whilecon (argl,whend,whetop:0,8,0,255:7,27,7,0,0,1928,1927)
v1819com primitive to generate an while do statement. It is used with
v1820com the construction to mark the end of the executable
v1821com statements.

v1822begin stext
v1823whiletop:ld a,(argl);3m 13t 3b get result of logical expression
v1824and a; 3m 4t 1d
v1825jp z, <whend>2;3m 10t 3b
v1826endtext

v1827calc remptr=romptr+7
v1828es.wrtbin (....,0,0,1928,1978)
v1829com primitive to write the contents of a memory location to the
v1830com terminal using a standard 05h call. This routine is used and
v1831com included in code only when the debug switch is set to true.
v1832com the value to be output is given in reg a. No time constraints are
v1833com given for this primitive since it is used only for debug purposes.
v1834com the code has been copied from a class handout in ca3201 taught
v1835com by a. reas

v1836begin stext
v1837es.wrtbin:push bc ; to save the registers, they are pushed onto
v1838push de ; the stack
v1839push hi
v1840push af
v1841ld h,$8 ; the h reg will count the number of chars
v1842ld b,00000000b ; the l reg will hold a mask to select
v1843 ; the bit to be transmitted
v1844ld b,a ; save the number to be transmitted in the b reg
v1845ld a,l ; get the bit select mask
v1846rrca ; rotate the mask right one position to the next
v1847ld b,1 ; bit and then save a copy back in the l reg.

v1848and b ; and the mask and the number together, to set
v1849 ; the zero flag to represent the bit to be sent
v1850ld x,'O' ; prepare to send the character 'O'

v1851jp z,5 ; if the bit was zero, send 'O'

v1852ld x,'1' ; if the jump test fails, change the char to '1'

v1853note: we set the flags with the and inst.

v1854and tested with the jp z inst. the ld does

v1855; not affect flags.

v1856ld c,2 ; any call to the cp/m bdos expects a function
v1857; number in reg c. function 2 will write the
v1858; code in the a reg to the terminal.

v1859push hi ; unfortunately, cp/m will erase the cpu registers.

v1860push bc ; so the ones we care about must be saved
v1861call <bdos> ; this is the call to the cp/m routine
v1982 pop bc ; restore the registers
v1983 pop hl
v1984 dec h ; count down one more bit
v1985 jp nz,s-15h; if we have not reached zero, go back to
v1986 ; send another bit of the number.
v1987 ld e,0ch ; if the number is done, send a
v1988 ld c,2 ; carriage return
v1989 call <bdos> ; and
v1990 ld e,0ch ; a line feed
v1991 ld c,2 ; to the
v1992 call <bdos> ; terminal.
v1993 pop af ; now restore the
v1994 pop hl ; registers which
v1995 pop de ; were used by
v1996 pop bc ; write
v1997 ret ; and return to the calling program
v1998 endtext
v1999s.ne (rlt, arg1, arg2:0,16,0,16,0,16,0,16,0,16,0,16,0,16,0,16,0,16,0,16)
v1999com primitive to perform comparison between 2 16-bit numbers
v1999com llt=rlt,argument 1, argument 2:rlt,time,ext,c,f,addr
v19992begin stext
v19993ld de,(<arg1>); 0m 20t 4b if arg1 arg2 then rlt=ffh de=<arg1>
v19994ld h!,(<arg2>); 5m 1t 3b
v19995abc h!,de; 4m 1t 2b
v19996ld a,0 ; 2m 8t 1b
v19997jr z, 5+3 ; 3m 1t 2b result equal
v19998cpi \\ 1m 4t 1b result not equal
v19999ld (<rlt>),a; 3m 1t 3b
v1999endtext
v1999calc romptr=romptr+16
v19992s.start (1:2,3,3,0,1992,1998)
v19993com 8080 library required a primitive to start execution
v19994com this is included for compatibility but is not required
v19995begin stext
v19996jp espwr ; 3m 10t 3b jump to top of main monitor loop
v19997endtext
v1999calc romptr=romptr+3
v1999h.inout (1:2,3,3,0,1999,2023)
v2000com primitive to include a mostek mdz-dib01 board
v2001com 8 bit 64 channel digital input/output board
v2002com ninout is the number of 8 bit ports that have been requested
v2003com board is set up for use with the a 4 mhz system
v2004calc slot = slot + 1
v2005incl h.tcardcage (1)

v2006begin htext
v2007 put mostek mdz-dib01 board in slot <slot>
v2008 connect the following jumper pins
v2009 1-2 on j3 a15 = 0
v2010 3-4 on j3 a14 = 0
v2011 5-6 on j3 a13 = 0
v2012 7-8 on j3 a12 = 0
v2013 9-10 on j3 a11 = 0
v2014 11-12 on j3 a10 = 0
v2015 13-14 on j3 a9 = 0
v2016 15-16 on j3 a8 = 0
v2017 17-18 on j3 a7 = 0
v2018 5-6 on j4 multiple does not used
v2019 disconnect the following jumper pins
v2020 18-20 on j3 a6 = 1 set high
v2021 3-4 on j4 limits signal length to 25 feet no wait states
v2022 address space 0000000000111111
v2023endtext
v2024h. start (++;552, 30, 15, 7, 8, 2024, 2084)
v2025com primitive to include a molex md3-a/d8 board
v2026com analog to digital conversion module
v2027com notate 16 the number of 8 bit a to d ports that have been requested
v2028com board is set up for use with the 8 4 mhz system
v2029com set jumper j3-1 to j3-2 for 4 mhz clock
v2030if notate .ne. 1 skip 13
v2031calc slot = slot + 1
v2032incl h.tcardcage (;;)
v2033begin htext
v2034 put first molex md3-a/d8 board in slot <slot>
v2035 connect the following j3 jumper pin
v2036 1-2
v2037 disconnect the following jumper pins
v2038 j6
v2039 j7
v2040 j4
v2041 connect the following j5 jumper pins
v2042 1-2 a7 = 0
v2043 3-4 a8 = 0
v2044 5-6 a9 = 0
v2045 7-8 a4 = 0
v2046 9-10 a3 = 0
v2047 11-12 a2 = 0
v2048 address space 00000000
v2049endtext
v2050if notate .ne. 17 skip 14
v2051calc slot = slot + 1
v2052incl h.tcardcage (;;)
v2053begin htext
v2054 put second molex md3-a/d8 board in slot <slot>
v2055 connect the following j5 jumper pins
v2056 1-2 a7 = 0
v2057 3-4 a8 = 0
v2058 5-6 a9 = 0
v2059 7-8 a4 = 0
v2060 9-10 a3 = 0
v2061 remove jumper on
v2062 11-12 a2 = 1
v2063 address space 00000000
v2064endtext
v2065h. processor (;;, 3, 2065, 2089)
v2066com primitive to include z=80 cpu board 4 mhz
v2067calc slot = slot + 1
v2001incl h.tcardcage
v2002begin htext
v2003 put z-80 cpu board in slot <slot>
v2004 memx high
v2005 set jumpers in the following pattern
v2006 jumper pattern
v2007 w2 010
v2008 w3 001
v2009 w4 010
v2010 w5 1
v2011 w6 001
v2012 w7 01
v2013 w8 110
v2014 w9 111
v2015 w10 1
v2016 w12 101010
v2017 w13 10
v2018 w14 10
v2019 w15 01
v2020 note numbering is from left to right and from top to bottom.
v2021 address space 0000-7fff
v2022endtext
v2023 include h.tcardcage
v2024com primitive to include card cage and power supply for controller
v2025begin htext
v2026 connect powersupply to card cage
v2027 endtext
v2028h.memory
v2029com primitive to include required memory
v2030calc slot = slot + 1
v2031incl h.tcardcage
v2032if romptr .lt. ramptr skip 5
v2033begin htext
v2034 the program space and the variable space have collided
v2035 you do not have enough memory to execute your program
v2036 your memory is limited to 16k
v2037endtext
v2038begin htext
v2039 put 16k memory board in slot <slot>
v2040 set jumpers in the following pattern
v2041 jumper pattern
v2042 w1 1111111
v2043 w2 10
v2044 w3 0
v2045 w4 01
v2046 w5 1
v2047 address range for card is 4000-7fff
v2048endtext
v2049h.tcardcage
v2050com primitive to limit the number of slots in card cage to 8
v2051if slot .le. 8 skip 4
v2052begin htext
v2053 you have exceeded the maximum number of allowable slots in the
v2121  card cage. it is limited to 8.
v2122endtext
v2123hdtoa  (:=552,30,15,4,5,2123,2137)
v2124comm digital to analog conversion module
v2125comm ndtoae is the number of 8 bit d to a ports that have been requested
v2126if ndtoae.map.1 skip 5
v2127calc slot = slot + 1
v2128incl h.cardcage (;

v2129begin htext
v2130  put first dtoa board in slot <slot>
v2131endtext
v2132if ndtoae.map.17 skip 5
v2133calc slot = slot + 1
v2134incl h.cardcage (;

v2135begin htext
v2136  put second second dtoa board in slot <slot>
v2137endtext
v2138a.div  (rel, arg1, arg2:0,8,0,8,0,8:58,504,129,41,0,2138,2179)
v2139comm routine to divide arg1 by arg2 and store in rel
v2140comm taken from zaks p 137
v2141begin text
v21421d a, (<arg1>) ;m4 t13 b3 get dividend
v2143and a ;m4 4t 1b
v2144id h,0 ;m4 7t 2b
v2145jp p, 97 ;m3 10t 3b
v2146cpl ;m1 4t 1b
v2147inc a ;m1 4t 1b
v2148id h,080h ;m2 7t 2b
v2149id a, a ;m1 t4 b1
v2150id a, (<arg2>) ;m4 t13 b3 get divisor
v2151and a ;m4 4t 1b
v2152jp p, 90bh ;m3 10t 3b
v2153cpl ;m4 4t 1b
v2154inc a ;m4 4t 1b
v2155id c, a ;m4 4t 1b
v2156id a, 080h ;m2 7t 2b
v2157xor h ;m1 4t 1b
v2158id h, a ;m1 4t 1b
v2159id a, c ;m1 4t 1b
v2160id c, a ;m1 t4 b1
v2161xor a ;m1 t4 b1 clear accumulator
v2162id b, 8 ;m2 t7 b2 set loop counter
v2163rl e ;m2 t8 b2 rotate
v2164rla ;m1 t4 b1
v2165sub c ;m1 t4 b1 trial subtract
v2166jr nc, 9+3 ;m3 t12 b2 subtract ok
v2167add a, c ;m1 t4 b1 restore accum,set cy
v2168jp nz 9-7 ;m3 t13 b2 m2 t8 on last loop
v2169id b, a ;m1 t4 b1 put remainder in b
v2170id a, e ;m1 t4 b1 get quotient
v2171rla ;m1 t4 b1 shift in last result bit
v2172cpl ;m1 t4 b1 complement bits
v2173bit 7, h ;m2 8t 2b
v2174jp z, $+5
v2178cp
v2176inc a
v2177id (\(<\text{rlt}\>) , a ; #4 t13 b3 store quotient in rlt
v2178endtext
v2178csic romptr=romptr+55
v2180x.div (\(<\text{rlt}\>, \arg1, \arg2: 0, 16, 0, 16, 0, 16, 78, 1458, 374, 57, 0, 2180, 2237)\)
\(\text{v2181com primitive to divide arg1 by arg2 and store in rlt}\)
\(\text{v2182com list=rlt, arg1, arg2; precisions=s,t,e,c,i,addr}\)
\(\text{v2183begin test}\)
\(\text{v2184ld d, h, (\(<\text{arg1}\>) ; 5m 10t 3b load arg1 in hl pair}\)
v2185bit 7, h
\(\text{v2186ld b, 0 ; 2m 7t 2b}\)
v2187jp z, $+12
\(\text{v2188ld a, h ; 1m 4t 1b}\)
v2189cp
\(\text{v2190ld h, a ; 1m 4t 1b}\)
v2191ld e, l
\(\text{v2192cp}\)
v2193ld l, e
\(\text{v2194inc hl}\)
v2195ld b, 080h
\(\text{v2196ld de, (\(<\text{arg2}\>) ; 6m 20t 4b load arg2 in bc pair}\)
v2197bit 7, d
\(\text{v2198ld a, 0 ; 2m 7t 2b}\)
v2199jp z, $+12
\(\text{v2200ld a, d ; 1m 4t 1b}\)
v2201cp
\(\text{v2202ld d, a ; 1m 4t 1b}\)
v2203ld a, e
\(\text{v2204cp}\)
v2205ld e, a
\(\text{v2206inc de}\)
v2207ld a, 080h
\(\text{v2208xor b}\)
v2209es af, af'
\(\text{v2210ld c, l ; 1m 4t 1b}\)
v2211ld a, h
\(\text{v2212ld b, 16d ; 2m 7t 2b}\)
v2213ld h, 0
\(\text{v2214ld c}\)
v2215rl
\(\text{v2216adc hl, hl ; 3m 11t 1b}\)
v2217adc hl, de
\(\text{v2218jr nc, $+3 ; 3m 12t 2b}\)
v2219add hl, de
\(\text{v2220ccf ; 1m 4t 1b}\)
v2221jd mx $-11
\(\text{v2222jr c ; 1m 4t 1b}\)
v2223rl
\(\text{v2224ld h, a ; 1m 4t 1b}\)
v2225ld l, c
\(\text{v2226es af, af'} ; 1m 4t 1b\) restore sign of rlt
v2227 jp p, $+10
v2228 id e, h
v2229 clp1
v2220 id h, a
v2221 id e, l
v2222 clpl
v2231 id l, a
v2232 inc hl
v2233 id ($<slot>), hl; 8m
v2234 inc hl
v2235 inc hl
v2236 inc hl
v2237 calc romptr=romptr+78
v2238 clock (00h, 00h)

v2239 com primitive to create an clock in the ctc chip of the z80 cpu board

v2240 begin htext
v2241 connect j1-20 to j1-12 this connects channel 0 output to channel 1
v2242 input clock

v2243 endtext
v2244 h.keyboard (68h, 68h)

v2245 com primitive to add the 7303 keyboard/display card. this primitive
v2246 is called by outled, inkey or outalp. since the primitive has
v2247 the capability to do all three functions with accompanying software
v2248 the software is separated and the card will be included only once
v2249 if keyboard: eq. 1 skip 14
v2250 calc keyboard = 1
v2251 calc slot = slot + 1
v2252 inc h, tcardcage

v2253 begin htext
v2254 put first prolog std 7303 keyboard/display card in slot <slot>
v2255 connect the following jumper pins
v2256 x6
v2257 y4
v2258 x0
v2259 z1
v2260 disconnect the following jumper pins
v2261 all others
v2262 address space 11000000, 11000001

v2263 endtext
v2264 uart (00h, 00h)

v2265 begin htext
v2266 this is a dummy primitive to remind you to put in the dual uart card
v2267 if you wish to use the nps loading rom, the require setting are as
v2268 follows.

v2269 set jumpers in the following pattern
v2270 jumper pattern
v2271 s1 01
v2272 s2 01
v2273 s3 10
v2274 s4 0001
v2275 s5 00001000
v2276 address space e0 thru e7

v2277 endtext
v2278 com this has to be the last line stupid or the machine will bomb!
LIST OF REFERENCES


7. Ibid.


<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Initial Distribution List</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>2</td>
<td>Defense Technical Information Center</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cameron Station</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alexandria, Virginia 22314</td>
</tr>
<tr>
<td>2.</td>
<td>2</td>
<td>Library, Code 0142</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>3.</td>
<td>4</td>
<td>LTC Alan Ross, Code 52RB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>4.</td>
<td>1</td>
<td>Prof. Herschel H. Loomis, Code 62LM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, California 93943</td>
</tr>
<tr>
<td>5.</td>
<td>1</td>
<td>MAJ Theodore J. Smith, Jr.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>USA Computer Systems Acquisition Agency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200 Stoval</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alexandria, VA 22331</td>
</tr>
</tbody>
</table>
END FILMED 11-84 DTIC