USING THE CONTROL SYSTEM DESIGN ENVIRONMENT IN THE DESIGN OF A DATA LINK. (U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA F S FOX JUN 84
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963-A
NAVAL POSTGRADUATE SCHOOL
Monterey, California

Using the Control System Design Environment in the Design of a Data Link Receiver Unit for the Coast Guard HH-65A Helicopter

by

F. Sutter Fox

June 1984

Thesis Advisor: Alan A. Ross

Approved for public release; distribution unlimited
Using the Control System Design Environment in the Design of a Data Link Receiver Unit for the Coast Guard HH-65A Helicopter

F. Sutter Fox

Naval Postgraduate School
Monterey, California 93943

Naval Postgraduate School
Monterey, California 93943

June 1984

165

Approved for public release; distribution unlimited

Computer-aided design, Data Link Receiver, Control System Design Environment, Microprocessor Controller Device

This thesis is an attempt to prove the value of the Control System Design Environment by designing a shipboard or ground-based data link receiver to communicate with the data link installed in the Coast Guard HH-65A helicopter. The Control System Design Environment was intended to allow a designer to use a high-level language to describe the required inputs and outputs of a system. This high-level language, the Control System Design Language (CSDL) is translated into a list of primitives by a Pascal (Continued)
program, CSDL.PAS. The primitive list is then compiled into assembly language by a FORTRAN program, NEWCSDL.FOR. The final output includes the hardware and software lists to build a controller that meets the designer's specifications. This particular project includes a project design much more ambitious than any previously attempted in the Control System Design Environment.
Using the Control System Design Environment in the Design of a Data Link Receiver Unit for the Coast Guard HH-65A Helicopter

by

F. Sutter Fox
Lieutenant Commander, United States Coast Guard
B.S., United States Coast Guard Academy, 1971
M.B.A., Roosevelt University, 1977

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1984

Author: F. Sutter Fox

Approved by: Alan A. Ross
Thesis Advisor

William Menne
Second Reader

Chairman, Department of Computer Science

Dean of Information and Policy Sciences
ABSTRACT

This thesis is an attempt to prove the value of the Control System Design Environment by designing a shipboard- or ground-based data link receiver to communicate with the data link installed in the Coast Guard HH-65A helicopter. The Control System Design Environment was intended to allow a designer to use a high-level language to describe the required inputs and outputs of a system. This high-level language, the Control System Design Language (CSDL), is translated into a list of primitives by a Pascal program, CSDL.PAS. The primitive list is then compiled into assembly language by a FORTRAN program, NEWCSDL.FOR. The final output includes the hardware and software lists to build a controller that meets the designer's specifications. This particular project includes a project design much more ambitious than any previously attempted in the Control System Design Environment.
# TABLE OF CONTENTS

I. INTRODUCTION ...................................................... 7
II. DATA LINK RECEIVER SPECIFICATIONS ......................... 11
III. CSDL DESIGN ..................................................... 30
IV. IMPLEMENTATION .................................................. 42
V. CONCLUSIONS AND RECOMMENDATIONS ......................... 51

LIST OF REFERENCES .................................................. 54
APPENDIX A-CSDL PROGRAM ......................................... 56
APPENDIX B-CSDL PROGRAM (REVISED) ............................. 63
APPENDIX C-CSDL.PAS OUTPUT (PRIMITIVE LISTING) ............. 70
APPENDIX D-NEWCSDL.FOR OUTPUT LISTING (SOFTWARE) ....... 82
APPENDIX E-NEWCSDL.FOR OUTPUT LISTING (HARDWARE) ....... 131

INITIAL DISTRIBUTION LIST .......................................... 165
ACKNOWLEDGEMENTS

I wish to acknowledge the help and advice of my thesis advisor, Lieutenant Colonel Alan A. Ross, U.S. Air Force, and my second reader, Captain Bradford D. Mercer, U.S. Air Force. My special thanks and appreciation go to my wife, Bonnie, my son, Darryl, and my daughter, Laurel. Their concern, patience, and support were invaluable to me in my time at the Naval Postgraduate School.
I. INTRODUCTION

The design of electronic equipment, including microprocessor-controlled equipment, has traditionally been a time and money consuming proposition. The design must be worked out manually and paper-tested, changes and improvements made, and more paper-testing performed. When it appears that the design is feasible, one or more prototypes are built and tested. Building prototypes is expensive because they are labor-intensive and fail to benefit from economies of scale. The use of computer aided design (CAD) has become more prevalent in many design applications in recent years because of these reasons. One such design aid is the Control System Design Environment proposed by Matelan [Ref. 1] and implemented by Ross [Ref. 2].

The Control System Design Environment makes use of the Control System Design Language (CSDL). This high-level language provides the user with a method to describe the inputs and outputs of a controller and specify time constraints for completion of the required tasks. A translator program takes the CSDL problem statement written by the user, tests the syntax, and then generates symbol and variable tables. It also translates the CSDL statements...
into a format called the primitives list with the associated parameters and selection criteria. The primitives are used as macro calls to the realization libraries. These libraries are based on families of microprocessors. The original library built by Ross consisted solely of the Intel 8080 family. Recent additions to the realization libraries include the Zilog Z80 by Smith [Ref. 3] and the Intel 8086 by Cetel [Ref. 4].

A family of microprocessors is chosen for the implementation by the designer and noted in the CSDL description. A solution is attempted and if it fails, another family is chosen and another solution is tried. If all families fail, the failure is reported to the user. When there is a success, software is generated to support the hardware, and monitor code is output for the overall control of the system. The automation of these functions makes it possible to rapidly and inexpensively design, build, and test prototypes. The ability to describe the functional specifications of a control process in a high-level language and let the CSDE provide output in form of hardware and software design can greatly simplify the work involved and thus lower the cost of producing working prototypes.

The application of the Control System Design Environment (CSDE) to the design of hardware and software for controller applications has been explored by a number of
researchers since Ross first designed CSDL. Some of those who have contributed to CSDE include Carson, Cetel, Heilstedt, Pollock, Riley, Sherlock, Smith, Walden, and Woffinden. Their accomplishments and contributions are all recorded in their respective theses. [Refs.5-13]

The goal of this thesis is to attempt a validation of the Control System Design Environment. This will be accomplished by using CSDE to design a microprocessor-based data link receiver for the data link to be installed in the Coast Guard HH-65A helicopter. Since CSDE was designed to produce process controllers, the production of a data link receiver will demonstrate the flexibility of the Control System Design Environment to handle additional and more complex types of problem descriptions beyond those considered in the original design of CSDE.

This project is a departure both in size and scope from any previous attempt at using the CSDE system. Several researchers have used CSDE to design controllers. Pollock used CSDE to design a fuel injection system for an automobile in 1981 [Ref. 14]. Heilstedt designed digital filters using CSDE in 1983 [Ref. 15]. The latest CSDE design is an automatic start sequencer for a jet engine performed by Riley in 1984 [Ref. 16]. The design of a microprocessor-based data link receiver is a much more ambitious application than any of these previous works. It requires the movement of strings of data throughout the system while
watching for keyboard input from an operator. The goal of the project is concerned with the use and abilities of the CSDE and not with producing a working prototype of a unit that will function according to Coast Guard requirements. The data link receiver will be a subset of and not a complete implementation of the Coast Guard requirements and specifications as outlined in the next chapter.
II. DATA LINK RECEIVER SPECIFICATIONS

Development of systems, including computer systems, can be costly due to the time and effort required to design, build, and test prototypes. The use of the Control System Design Environment can dramatically reduce the time and effort involved in designing a microprocessor system and in producing the associated software. The cost of a system is spread over the number of items produced, and in general, prototypes are produced in small, and thus expensive, quantities. The automation of the design of hardware and software promises to greatly reduce the cost of the design and prototyping portions of new systems development, especially those systems that will be produced in small numbers.

An example of a system currently under development was chosen for an implementation under CSDE for this thesis. The U.S. Coast Guard is presently acquiring a new helicopter, the HH-65A. One of the features of the aircraft avionics suite is a data link transceiver which will automatically send and receive flight information data. Unfortunately, there is no compatible transceiver available for use aboard Coast Guard cutters or at air stations. The potential contributions to a wide range of Coast Guard
missions, not to mention the safety of flight ramifications, make automated communications between Coast Guard operational units via data link extremely desirable.

It is virtually impossible today for the U.S. Coast Guard to conduct truly covert law enforcement operations with helicopters deployed aboard cutters or based at air stations. Safety procedures call for a number of emitters to be employed for the duration of the flight. A properly equipped smuggling operation can gain considerable intelligence against the Coast Guard, particularly from voice communications. Voice communications are notoriously susceptible to monitoring with any of a number of relatively inexpensive scanners available in the electronics market. Making such communications protected or secure can deny the smuggler the information contained in the transmissions, but it cannot conceal Coast Guard presence during the critical preliminary search. A system is needed that will allow the passing of safety and other important data between a helicopter and its cutter or air station and, at the same time, lend a higher degree of covertness to the operation. Since the new HH-65A helicopter is being built with a data link capability installed as part of its avionics package, there exists an opportunity to conduct covert law enforcement operations with cutters or air stations. Since the data link can send position and operations information automatically using preselected time periods, the pilots are
relieved of one more duty that distracts from the mission accomplishment.

From the pilots' point of view, a data link capability means that their attention need not be diverted from the normal scan of aircraft instruments, the airspace around the helicopter, and the water or ground over which they are searching. The onboard computer system does many of the navigation functions automatically and, with the installation of data link capability, can make the required operations reports to the controlling Coast Guard unit. On a typical mission the aircrew must monitor the UHF and/or HF radios for communications with their controlling unit, the VHF-AM radio for normal communications with FAA facilities and other aircraft, and the VHF-FM radio for communications with vessels. Thus, the pilots must monitor up to four different radios simultaneously while communicating with other members of the aircrew over the internal communications system (ICS). These communications requirements tax the concentration of the aircrew and contributes to their fatigue. The data link can relieve the crew of one duty while enhancing the security of the flight operation.

The Coast Guard Office of Operations sent a memorandum to the Office of Research and Development in June 1983 requesting development of a shipboard version of the data link. The performance standards and specifications listed in
that request have been used as a basis for the functional specifications for the CSDE implementation of this thesis project.

The performance standards and criteria outlined in the request for support specified a "shipboard version of the data link built into the HH-65A helicopter". This language does not reflect the possible use of the data link at a Coast Guard Air Station. This thesis will assume that the design of shipboard equipment will be more than capable of working ashore as well as at sea.

It would be possible to use a commercially available microcomputer for this project. Writing the assembly language software to drive that system would not be too difficult. This approach, however, would provide a software engineering problem without adding anything new to the knowledge base of computer-aided design. It is far more enlightening to attempt the project through the use of CSDE in order to reduce design costs for new systems.

The goal of this project is concerned with the use and abilities of the CSDE and not with producing a working prototype of a data link receiver unit that will function according to Coast Guard requirements. The data link receiver designed using CSDE will be a subset of and not a complete implementation of the specifications listed by the Office of Operations. Because of this, there will be no
effort to fully meet the performance requirements specified in the request for support.

The following is a listing of the requirements and performance standards as stated in the request and describes how each will be addressed in this project.

The data link receiver system:

1. Must operate on all frequencies (selectable) from HF to UHF (30.000-399.975 MHz).
   Comment: This is outside the scope of the thesis work and will be assumed to have been met. The point of this thesis is not the solution of interfacing problems with the radio transmitters.

2. Must be compatible with the AN/ARC-182 transceiver on the HH-65A helicopter.
   Comment: This is outside the scope of the thesis work for the same reasons as cited in the paragraph above and will be assumed to have been met.

3. Must be compatible with data link system presently installed in the HH-65A. This system, manufactured by Collins, operates at a 300 baud rate with data burst.
Comment: This is central to the design but little information was available for use in this thesis. The details will be addressed later in this section.

4. Must be of the smallest size and weight practicable for installation in CIC/CSC on all flight-deck equipped cutters, up to 200 feet from transceiver and antenna. A remote readout for the pilothouse is extremely desirable.

Comment: The small size and weight should follow from an efficient design. The installation aboard cutters will be assumed as will the solution regarding the distance between transceiver and antenna. The remote readout requirement will not be addressed. It is expected that the addition of a remote readout will be relatively simple when the system design is complete.

5. Must be capable of automatically tracking and polling at least three aircraft in sequence at selective time intervals from 5-30 minutes, and must be capable of manually polling an aircraft data link.

Comment: These requirements will be met in full.
6. Must be capable of providing an automatic response to an aircraft interrogation. This response would give the cutter's position by the best electronic navigational aids available and would provide for data verification, as in the HH-65A system.

Comment: The acknowledgement of a message receipt and the reply with a position will be met. Provisions for manual input of navigation information for use with a stationary receiving unit (an air station) will be included. This will also mean that manual input will be possible aboard a cutter if the electronic navigational aids become unusable. The data verification will not be included for reasons stated later.

Attempts to acquire the actual protocols for the communications and the technical specifications for the data link equipment installed in the HH-65A helicopters failed. Coast Guard sources could not provide the necessary information. The Collins Government Avionics Division of Rockwell International Corporation, makers of the HH-65A data link, did not respond to requests for the information. The protocols, message formats, and other specifications used in this thesis are estimates of what and how the data
link system should work and not the result of any proprietary information.

The data link receiver is configured as shown in Figure 1. There are four inputs: from the radio interface unit, the electronic navigation devices interface unit, the clock, and the keyboard. There are four outputs: to the radio interface unit, the video display unit, the printer, and the clock. All inputs to and outputs from the system are digital signals using ASCII code.

There is a pattern of levels of abstraction in this view of the project. At the center is the data link receiver processor designed for this thesis. This processor is responsible for the proper routing of messages to and from memory, calling menus from ROM and sending them to the video display terminal, updating positions, and performing tasks in response to input from the keyboard. It is assumed that there is a radio interface unit that receives and processes all signals. If a message is addressed for this particular Coast Guard unit, then the radio interface unit receives the message, checks the correctness of the message, and sets a flag to tell the data link receiver controller that a message awaits.

The electronic navigation devices are an abstraction for the actual machines that compute the receiver unit's position. These devices may include LORAN-C, OMEGA, navigation satellites, or any other navigation instruments.
DATA LINK RECEIVER

Figure 1
that may be polled by the navigation interface unit. These devices pass their position information to the navigation interface which sets a flag to let the main processor know that a new position has been computed. The system operator may override the automatic polling of the interface unit by the system processor in the event of a malfunction in the navigation instruments. In the case of a shore unit, there is no need for a navigation interface. The operator would manually enter the position of the air station and set the system to prevent the interface being polled in the case where the receiving unit is a shore unit and thus in a constant position. The same method could be used for a ship at anchor when most navigation devices are normally secured.

It is essential that the ship’s position be sent as often as necessary to keep the helicopter’s computer updated.

The third input comes from a clock. The clock is used for time-stamping the positions computed by the navigation devices. This information is passed as part of the acknowledgement messages sent in response to a helicopter’s message and also when polling helicopters. The radio interface unit is assumed to have direct access to the clock to obtain the time. The output to the clock from the processor is for setting the date and time and is entered by the operator through the keyboard.

All commands to the system are entered using a keyboard. Menus are provided on the video display unit for
the operator's assistance. Most inputs through the keyboard consist of single keystrokes for the selection of operations as listed in the menu currently displayed. There are several cases where more than one keystroke is required. When an aircraft is logged into or out of the system, the clock is set, or the position of the receiving unit is manually inserted, the operator must enter the appropriate number of characters.

The video display unit may consist of some sort of smart terminal or it may be an interface unit between the main data link processor and the VDT. In either case, the data link receiver's processor need only send certain codes to the video display port. These codes trigger the appropriate actions by which a driver in the video display interface causes the menus to be displayed on the screen. Messages from the helicopters are displayed in a similar manner. This is another advantage of the levels of abstraction. The technology, architecture, and implementation of the device is invisible to the data link receiver processor.

The printer receives its commands and data in a manner similar to that of the video display terminal. It does not receive as much text as the VDT because the menus are not printed. It exists in the system mainly to provide hard copy output of the message traffic during a mission. Should the hard copy not be required or desired, the operator has
only to secure the power switch of the printer. This makes the overall system simpler because there is no need to include an on/off function for the printer in the system.

This approach of levels of abstractions is a reasonable one for several reasons. It would be difficult for CSDE to design a controller that would perform all the requirements of this controller and still be able to meet the required time constraints for each task. The overall system must monitor the airwaves, receive a message, check it for validity, and store it properly not to mention all the other functions required to drive the video display unit, monitor the keyboard for input, and other such tasks.

One criticism of CSDE has been that there is no way to force a design with two CPUs [Ref.17]. CSDE will design a system with two processors when one cannot perform all the functions set forth by the designer within the required time constraints. Nor does CSDE presently allow for more than two CPUs. There are, however, ways around these limitations. The central part of the receiver is designed with the assumption of several "smart" interfaces. These other interface units may also be designed using CSDE as long as the user is careful to specify compatible links between the units. The result is a number of processors integrated into one system much as an operating system or a communications network may be viewed as consisting of layers.
The radio interface will receive messages sent to the particular receiving unit. The message will be tested for completeness and correctness. If the message passes this test, it is held in a buffer and a status flag set. The processor checks the status bit when it polls the radio receiver and if it is set, it reads the message into memory in a serial fashion and passes it to the video display unit and the printer.

The radios, navigation devices, and the keyboard are polled according to the time constraints in the CSDL contingency list. The number of polls of a device per time period vary with the immediacy of the input. For instance, the keyboard will be polled more often than the navigation device. The system must respond to keyboard inputs fast or the operator may become frustrated or think the system is locked up. The position of the unit doesn’t change at a fast rate so the position need only be updated every minute or so.

The system will handle at least three helicopters simultaneously. A maximum of ten aircraft may be logged in at any given time. The tail numbers of the helicopters must be logged into the system memory manually by the system operator. The system will receive and process messages from an aircraft not logged in as long as the receiving unit is specifically addressed in the message. It will not,
however, poll any helicopter that is not logged into the system.

The polling of aircraft is done either automatically or manually as selected by the operator. If the polling is on automatic, the operator must select a polling time period from 1 to 30 minutes in duration. The normal reporting period for a Coast Guard helicopter is every 15 minutes. The performance standards and criteria for the system specified periods between five and thirty minutes. A one-minute interval was added for closer monitoring of a helicopter during an in-flight emergency or a critical period during a search and rescue case or law enforcement action.

Messages from the helicopters are fixed in format. The messages are received and held in a buffer by the radio interface. While the message is read into memory, it is printed on the video display terminal and on the printer. Messages are acknowledged by the radio interface by sending the position of the receiving unit and the time the position was calculated back to the helicopter. The position is read from the navigation interface by the processor and stored in memory. It is called from memory when needed and passed serially to the radio interface.

A garbled message from an aircraft is not acknowledged. When a message is not acknowledged, the helicopter's data link processor will resend the message
after waiting a random period of time. The message will be sent again and again with random wait times between transmissions until acknowledged. After being sent a certain number of times without acknowledgement, the pilots are notified by a message printed on their VDTs.

If two or more helicopters send messages simultaneously, none will be acknowledged. This contention is not serious since the helicopters will resend their messages after waiting a random period. The messages are short (less than 64 bytes sent at 300 baud) and most cutters operate with only one helicopter at a time. An exception are the five polar icebreakers that normally carry two helicopters. Operations from air stations present the highest probability of contention since they might have two or more helicopters airborne on missions at the same time. The short duration of the messages coupled with the few aircraft generally under the control of one unit plus the random timing for resending a message creates a situation where the contention is self-correcting.

The messages are stored in the same packed form as they are received. When the messages are processed for display, they are filled out with the necessary descriptive enhancements. The overlays or templates for this purpose reside in ROM and are inserted by the video display unit as the messages are displayed on the VDT. The system has the ability to store the last ten messages received in memory.
The protocol of the data link transmissions include the preamble, control data, and information. The preamble and control data sections are used by the radio interface. The information section alone is passed to the processor by the radio interface. The information section of the message follows the format below. The number of characters in each field are shown in parentheses.

1. Helicopter number. (From)
   
   4 digits (1409)

2. Ship/ground station identifier. (To)
   
   4 digits (7184)

3. Date and time. (DTG for message numbering)
   
   12 characters (311545ZMAY84)

4. Position time. (This is the time the position was calculated. The time may be local or Zulu, depending on local doctrine).
   
   4 digits (1543)

5. Message type. (Message types include normal position reports, poll response, etc., or mission type such as fisheries patrol, drug interdiction,
search and rescue, etc., and can declare an aircraft emergency.

1 character (3)

6. Position. (Latitude/Longitude. Format is degrees, minutes, and tenths of minutes followed by N or S for latitude, E or W for longitude; i.e., Lat = ddmmtN, Long = ddmmtW)

13 characters (36429N088321W)

7. Ground speed (in knots).

3 digits (105)

8. Track (in degrees true).

3 digits (220)


4 digits (1200)

10. Wind direction (in degrees true).

3 digits (345)

11. Wind velocity (in knots).

3 digits (022)
12. Altitude (in hundreds of feet).
   2 digits (12)

13. CRC (cyclic redundancy check. This is used in the radio interface unit but is not read into the processor’s memory).

   Each field of the message will be complete, that is, each field will use its full number of characters. If the information for a particular field is unavailable for some reason (the failure of an instrument such as the omnidirectional airspeed indicator or lack of an updated position because of loss of LORAN lock on) the field will be filled with blanks (20H). Fields that are short will use blanks (20H) or zeroes (00H) as appropriate to pad the data field.

   The format for the examples given above looks like this:

   31 34 30 39 37 31 38 34 33 31 35 34 35 5A 4D 41 59 38
   34 31 35 34 33 33 36 34 32 39 4E 30 38 38 33 32 31 57
   31 30 35 32 32 30 31 32 30 30 33 34 35 30 32 32 31 32

   These numbers are hexadecimal representations of the ASCII characters for the numbers and letters used in the examples.
Keyboard inputs provide the operator's control over the system. The input system is simplified to the point where single keystrokes are all that are necessary to invoke different functions. When a keyboard input is detected, the appropriate menu is placed on the video display unit. The operator uses single keystrokes to select and invoke the different system functions. The menus are contained in ROM and are written to the VDT by the video display unit interface. The data link receiver processor sends the appropriate code to the video display unit interface to call the menus to the screen.
III. CSDL DESIGN

The Control System Design Environment was first proposed by Matelan [Ref. 18] as a method of simplifying the design of process controllers. Ross implemented the Control System Design Language (CSDL) as part of this environment [Ref. 19]. The designer of a controller system describes the inputs and outputs needed and the required response times for different functions. Using this high-level language makes the design of the controller much simpler than it would be without the use of computer-aided design tools.

Some of the syntax rules of CSDL should be mentioned. CSDL programs must be in upper case characters only. Most statements end in a semicolon. The names of functions end with a colon while task names end with a semicolon. The final end statement has no punctuation. Variable names cannot exceed ten characters. Because they are later truncated to six characters, the user should insure that the first six characters in a name are unique. A complete listing of the formal syntax of CSDL is available in Carson’s thesis [Ref. 20].

There are five sections to CSDL programs. The sections are Identification, Environment, Contingency List,
and Procedures. The Identification section is simply the name of the designer, the date, and the project name. It is intended to identify the program and does not have any effect when the program is run. It appears in this format:

IDENTIFICATION

DESIGNER : "SUTTER FOX"
DATE : "05-31-84"
PROJECT : "COAST GUARD DATA LINK RECEIVER"

The second section is the design criteria. This is the portion of the program where the designer can choose the primitive list and processor family for CSDL to use. At present there are three realization libraries. They are based on the 8080, Z80, and 8086 microprocessor families. The format for the design criteria section is:

DESIGN CRITERIA
METRIC FIRST;
VOLUMES 1;
MONITORS 1;

METRIC FIRST refers to the first realization that meets the timing requirements of the system. Alternatively, the designer may prescribe cost or power as the minimum requirements for the system. VOLUMES refers to the realization libraries in the order they are numbered. Since only one volume was resident when this project was translated, the first (and only) volume was designated. As with the realization volume, there is only one monitor volume for the realization volume, and thus the first (and only) monitor is selected.
The design variables are declared in the environment section. The system for the data link receiver produced under CSDL requires a greater number of variables than any earlier attempt at a CSDE controller generation. This will be discussed in more detail later in this chapter. There are three types of variables in a CSDL program. Input variables are values that are sensed by the controller from the outside of the controller processor. The output variables send values outside the controller. The declaration of these variables includes the number of lines necessary between the controller and the outside world and the type of technology desired for their design. The third type of variable is the arithmetic variable. Arithmetic variables used in computations within the controller itself. The declarations made in the environment section are analogous to declarations made in a block structured language such as Pascal, Ada, or PL/I. The format of the environment section is:

ENVIRONMENT

INPUT : KEYFLG,1,TTL; KEYCHAR,8,TTL;
       MANPOS,8,TTL; END INPUT;

OUTPUT : MENU,8,TTL; POLL,8,TTL;
          MSGVDT,8,TTL; MSBRCVD,1,TTL;
          END OUTPUT;

ARITHMETIC : KEVINMAIN,8; MINTAC,8;
             NEXTMSG,8; NEXTAC,8; COUNT,8;
             END ARITHMETIC;
The procedures section is much the same as found in block structured languages. This section contains the high-level descriptions of the arithmetic and data manipulations required to make the system work. There are two types of blocks within the procedures section: functions and tasks. The functions and tasks are always coupled in what are referred to as contingency/task pairs. Each function is evaluated in its turn as set forth in the contingency list. If it is true, its associated task is performed. The requirement that each function have one and only one task creates some problems with programming with CSDL that will be discussed later. An example of a function and a task follows.

FUNCTION KEYINMAIN:
  BINARY, 1;
  SENSE (KEYFLAG);
  IF KEYFLAG = 1 THEN KEYINMAIN := 1; END IF;
END KEYINMAIN;

TASK KBINPMAIN:
  MENU := 0; ISSUE (MENU);
  SENSE (KEYCHAR);
  IF KEYCHAR = 1 THEN MINTAC := 1; END IF;
  IF KEYCHAR = 2 THEN MMSGDSPLY := 1; END IF;
  IF KEYCHAR = 3 THEN MLOCATION := 1; END IF;
  IF KEYCHAR = 4 THEN MCLOCKSET := 1; END IF;
  IF KEYCHAR = 5 THEN MLOGINOUT := 1; END IF;
  KEYINMAIN := 0;
END KEYINMAIN;

FUNCTION KEYINMAIN is called according to the time constraints set forth in the contingency list as described in the next paragraph. The keyboard active status flag is checked by SENSE (KEYFLG). If the flag is set, then the
variable KEYINMAIN is set. Thus, the function is true and the associated task KBINPMAIN is performed. This task calls for the video display interface to put the appropriate menu on the VDT with the statements MENU:=0 and ISSUE (MENU). The menu presents the five selections available to the operator and waits for input. Upon pressing a number key from one to five, the appropriate variable is set to one. Note that the variable is the same name as a function in the CSDL program. When that particular function is tested and is found to be true, its associated task will be performed. The final line sets the function associated with the task to zero so that it will not be performed again until set. It is important in this data link receiver project to insure that no more than one function be set at any given time in order to preserve the flow of program control.

The fifth section is the contingency list. In this section the designer lists the contingencies (functions) that occur and the time constraints for performing the associated tasks. This is where CSDL differs greatly from languages that execute in a linear manner such as FORTRAN, BASIC, or Pascal. The timing requirements may be such that some functions are tested several times before another certain function is tested at all. Some procedures (tasks) may be performed every designated time period. The execution of functions and procedures are dependent upon the timing requirements the designer delineates in this section.
of a CSDL program. An example for the contingency section is:

CONTINGENCY LIST
WHEN KEVINMAIN :100 MS DO KBINPMAIN;
WHEN MINTAC :100 MS DO INTAC;
WHEN SMANUAL :100 MS DO MANUAL;
WHEN SNAUTO :100 MS DO AUTO;
WHEN TPOLL :100 MS DO POLLAUTO;
WHEN MLOCATION :100 MS DO LOCATION;
WHEN TMLOCATION :100 MS DO MANLOC;
WHEN POSCH :100 MS DO POSUPDATE;
WHEN MMGDSPLY :100 MS DO MSGDSPLY;
WHEN MCLOCKSET :100 MS DO CLOCKSET;
WHEN MLOGINOUT :100 MS DO LOGINOUT;
WHEN TLOGIN :100 MS DO LOGIN;
WHEN TLOGOUT :100 MS DO LOGOUT;
WHEN MSGIN :100 MS DO MSGSTORE;

Writing code in CSDL is not as easy or convenient as in many high-level languages. There are several factors that can make it a frustrating experience for those who are used to the constructs available in languages such as Pascal, PL/I, and even BASIC. The reader should not judge CSDL too harshly. It should be remembered that the CSDL language was designed for simpler controllers than the one attempted in this thesis.

There are no comments in CSDL other than those enclosed in quotes in the Identification Section. It is widely accepted that commenting within programs makes it easier to maintain those programs. Since most candidates for a CSDE implementation are relatively simple, and since most CSDL programs will be written and implemented in a fairly short time, this may not be a very big problem.
The data link receiver project would normally be a candidate for an interrupt-driven system. As it is presently implemented, CSDL has no ability to design systems that use interrupts.

Since all contingencies and tasks in CSDL must be in a one to one ratio, all the functions and tasks have been placed together in the listing for this data link receiver CSDL program. This makes it clearer for the reader (not to mention the programmer) when perusing the code. There are many menus in this system and the functions and tasks have been named to reflect the fact that some menus are called from other menus. The main menu presents five choices. If, for instance, the operator selects "Interrogate Aircraft", FUNCTION MINTAC is set to 1 (true). FUNCTION MINTAC is paired with TASK INTAC. Functions generally have more letters in their names than their tasks since functions generally have a prefix added to the name of the associated task. TASK INTAC calls up a menu wherein the operator may select a return to the main menu without any function being carried out, or the operator may elect to interrogate the aircraft manually or automatically. If the selection is to interrogate automatically, FUNCTION SMAUTO is set true. TASK AUTO is paired with FUNCTION SMAUTO, and by now the pattern may be clear to the reader. The name of function that is set to true by a selection made under the main menu is prefixed with an M. A menu called from a task that is
associated with "M" function has a function prefixed with "SM" for secondary menu. When there is another submenu, the function name will be prefixed with "TM" for tertiary menu. Tasks have names similar to their associated functions but without the prefixes.

One construct that would be most welcome in CSDL is the CASE statement. This would allow ease of programming when one out of several possible paths would be chosen. The lack of a CASE construct requires one to write multiple IF statements which does not make for the most elegant programming. Several of the procedures in this data link receiver project have ten IF statements where some other languages would be able to express the same function in one or two lines. This does take up some space in memory but when compiled it requires less space and may be insignificant in terms of overhead. The real problem that is evident is that there are many more variables than necessary in most other high-level languages. CSDL does not allow for subscripting variables which leads the designer of a system to writing many more lines of code and having to name each variable instead of using subscripts. For example, ten messages are required to be in memory at any given time. This requires ten different variable names for these ten messages. A pointer must keep track of the next message block available for use and it would be quite simple to use subscripted variables for this purpose. Since this
is not possible, a series of nearly identical IF statements must be traversed when locating the next block. The same problem occurs when printing the messages. The code must explicitly name each of the ten variables. It would require less memory for the program to be able to refer to these variables with subscripts and would also make for clearer code. The complexity could be simplified by adding new primitives that would allow for the constructs using subscripted variables. Examples of tasks with multiple statements that could be handled with a CASE statement include KBINPMAIN, INTAC, and AUTO. Nearly every task in this program could benefit from the use of subscripted variables.

The original design approach was to divide the memory for the ten messages according to the number of aircraft logged into the system. Two message memories were to be reserved for messages from helicopters not logged in. Thus, there would have been a maximum of eight aircraft tracked where each would have one message available for immediate recall. This would be the worst case scenario and also highly unlikely. If only one helicopter was logged in, the system could maintain the last eight messages for that one helicopter. Since CSDL does not provide a capability for indexing variables, it would be difficult to provide such dynamic allocation of the ten message memory areas.
It is highly likely that there will be three or less helicopters logged into one system at any given time. It is less likely that there would be four or more aircraft logged in simultaneously. Standard safety procedures call for a helicopter to maintain a radio guard at all times and send an "operations normal" and position report every fifteen minutes. If there were three helicopters logged into one system that would mean 12 reports in an hour. This means that the two earliest reports would be overwritten by newer messages at the end of an hour but there would be at least three messages remaining for each aircraft. Even if there were ten aircraft using the system, there would be at least one message in memory for each aircraft. This would be the last position sent and would be used as a datum for search and rescue procedures should communications be lost with the aircraft. Position reports include heading, track, ground speed, and other environmental data that would provide excellent search planning information. Since manual polling of the helicopters is provided, it would be possible to manually poll one or more helicopters enough times so that there would not be any messages in the memory from a particular helicopter. There are two backups in this case. First, the printer should be enabled so that there would be hard copy of all the messages. The second is the multi-track audio tapes that monitor telephone and radio traffic at air stations and aboard the flight-deck equipped
cutters. Should it be required in an emergency, the tapes could be run back as far as necessary and then replayed to feed the raw radio signals to the radio interface device.

Reading messages in from the electronic navigation device interface created a problem when coding the program. Each message is 56 bytes long and the input to the processor is 8 bits wide. In order to read in the message, the 56 bytes must be read into the processor and sent to memory, the video display unit, and the printer in a serial fashion. There was no construct in CSDL that would read in a string of characters of this length. This function is one that is basic to the operation of this system. String handling can be added by writing a new primitive that would read in the 56 bytes or any other number by overloading the SENSE (input) statement in CSDL. Without this addition to the language, there could be no CSDL implementation that would satisfy the requirements of the data link receiver project. A similar primitive can be specified to write a message out to the VDT and printer when the operator selects that function. These primitives can be general enough that they may be used for different sizes of strings.

When the CSDL program is completed, it is translated by Carson’s CSDL.PAS program. This program, written in Pascal, takes the high-level CSDL program and translates it into a primitive list for the controller. The primitive
list is used in the next step in the control system design environment process.
IV. IMPLEMENTATION

The next step in the process from the high-level description to the hardware and software listings is to take the output of the CSDL translator and feed it through Ross' FORTRAN program, NEWCSDL.FOR.

The main task performed by NEWCSDL.FOR is to map the primitive list compiled by CSDL.PAS to the selected realization library. The output includes a listing of the hardware to implement the controller and the assembly language software to run it. Two files, the primitive list and another containing information about the contingency list, are used by the Optimizer Module in NEWCSDL.FOR to set up a formatted application table and an index to the selected realization volume. The Functional Mapper constructs the Realization Timing Table and determines if the realization is feasible. A monitor sequence is added and actual values are substituted for dummy parameters and an output listing is generated. If the Timing Analyzer fails to find a feasible single processor realization, the contingency/task list is partitioned and a dual processor realization is generated under the control of the Optimizer. For greater detail, see Ross' doctoral dissertation [Ref. 21].
The project at this point had progressed from the CSDL description through translation by Carson’s CSDL.PAS program. This was the second attempt to use CSDL.PAS on a project. The first use was for a test of CSDL.PAS conducted by Carson by running Riley’s jet engine start sequencer controller through the translator.

There were some problems getting the CSDL description for the data link receiver project through the translator. The data link receiver required so many variable names that it exceeded the size limits set in CSDL.PAS. One problem concerned a CASE statement in the CSDL.PAS program where 20 possible cases existed. This problem was resolved by increasing the allowed number of cases to 95. There were other errors generated in translation that were quite frustrating. One problem was overcome when it was discovered that CSDL.PAS would generate errors whenever a tabkey had been used to produce spacing within the CSDL description. All tabkey spacing was removed and replaced with spaces generated using the spacebar. CSDL.PAS should be modified to allow for spaces generated by either the space bar or the tabkey.

More problems were discovered when trying to run the primitive list generated by CSDL.PAS through NEWCSDL.FOR. FORTRAN is notorious for the inflexibility of program inputs, which is a carryover from the days of card input.
NEWCSDL.FOR is no exception. Title lines for functions and procedures in the primitive files have the format:

```
1 6 23
p xxx t.generated for: procedure name
```

where xxx represents the line number of the primitive and t stands for title line. The t.generated for primitive marks the beginning of a new procedure. The procedure names in the title lines of the primitive list as generated by CSDL.PAS were in column 22. When NEWCSDL.FOR is run with the procedure name in the wrong column, everything between the title lines is ignored and error messages are generated for each title line in the file. Each title line in the primitive list had to be reformatted to meet the requirements of NEWCSDL.FOR.

Another problem encountered in the translation was that the s.ni primitive should have appeared as:

```
p xxx s.ni (:::)
```

The s stands for a software primitive. An h stands for a hardware primitive. The colons inside the parentheses are required because they are used to separate variables, parameters, and attributes. They were not in the primitive list generated by CSDL.PAS. The colons were added using the text editor. A similar error was discovered in the s.main primitive.

The entire primitive list must be in lower case letters except for the function and task names. They may be
in upper or lower case as long as they are consistent with the case used in the contingency list file. Although CSDL.PAS allows variable names of up to ten characters in length, NEWCSDL.FOR has a maximum of six characters. The designer must insure that no two variables have the same six first characters.

While the formatting errors and the lack of colons in the proper places created some unnecessary work, the biggest problem at this point was the incompatibility of the primitive list and the Z80 realization library. For instance, the function of one basic primitive is to sense a value on a particular input line to the processor. This primitive has the form:

\[ p \text{xxx s.sensecond (keychar:8)} \]

where sensecond stands for sense condition, keychar is the name of some variable, and eight refers to the arithmetic precision of the variable. It was discovered that the Z80 realization library did not contain any realization of this primitive.

The Z80 library was designed by Smith at the same time Riley was working on the jet engine start controller. This was before Carson's CSDL.PAS program was available. Riley had to translate the CSDL listing into a primitive list by hand. Since he chose to implement his project using the Z80 library, he and Smith worked closely together. Prolog equipment was used for the project implementation and so
Smith geared the Z80 library toward that end. Thus, the Z80 library was not as general as needed to produce controllers using other hardware. In particular, the I/O primitives were designed specifically to match the Prolog hardware. The s.sense and s.issue primitives were not needed for the Prolog implementation and thus were not included in the Z80 realization library.

At this point it was decided to shift the emphasis from using the Z80 realization library to the 8080 and 8086 realization libraries. There were two reasons for this. First, the designer of the 8086 library, Cetel, was still available to make adjustments to the library. Second, the 8086 library closely followed the example of the original 8080 library built by Ross. If the primitive list could be adjusted to run under NEWCSDL.FOR, then two realizations of the data link receiver could be produced. This would help standardize the realization libraries to where any library could be used with the primitive list output by CSDL.PAS and further processed by NEWCSDL.FOR.

Other problems remained stemming from the incompatibility of the primitive list produced by CSDL.PAS and the realization libraries. CSDL.PAS produced other primitives that did not exist in the realization libraries. These primitives included s.inputport and s.outputport. Inputport and outputport are both primitives that remain to be added to the realization libraries.
The s.forcons and s.forend primitives in the 8080 and 8086 libraries are not in the same format. These primitives mark the beginning and end of for-next loops. There are two variables for the upper and lower values of the loop. NEWCSDL.FOR expects actual numbers but CSDL.PAS produces variable names instead.

The s.exitproc primitive marks the end of a procedure in the primitive list. NEWCSDL.FOR was designed to use the contingency name in the parameter list to reset the value of the contingency to zero after the task was executed. The CSDL program written for this project included a statement at the end of each task explicitly resetting the contingency. Ross decided that the realization libraries and NEWCSDL.FOR would be changed to adopt this latter method of resetting the contingency.

There are several different primitives with the same names in a realization library. This is to allow for different precisions of arithmetic manipulations. NEWCSDL.FOR performs a binary search to find a primitive name. When it finds the primitive, it searches up the realization library index to find the first instance of the primitive name. NEWCSDL.FOR then works down through the index to find the first instance of the primitive that will satisfy the precision required. For instance, s.var and
s.cons primitives in the 8086 realization library had the format:

```
s.var (nam, val: 0, 8 :...etc)
s.var (nam, val: 0, 16 :...etc)
s.var (nam, val: 0, 24 :...etc)
```

where the 0, n referred to variables with zero to n bits of precision. It was discovered that NEWCSDL.FOR was choosing the greatest precision available every time. This was corrected by changing 8086 realization library to the format:

```
s.var (nam, val: 0, 8 :...etc)
s.var (nam, val: 9, 16 :...etc)
s.var (nam, val: 17, 24 :...etc)
```

The correct precision is now selected for these particular primitives but the entire 8086 library must be examined for other instances of this precision error.

NEWCSDL.FOR requires a listing of the contingency/task pairs in a file named IADEFL.DAT as one of its inputs. CSDL.PAS creates such a file but it is not in the required format. A new line for the system must be added as the first line in the file. The other columns must be corrected to the format as set forth in Ross' doctoral dissertation.

The multiplication primitive, s.mult, was present in the 8086 library as s.mul. The two precisions of multiply were renamed s.mult to conform with the standard. The s.mult primitive in the 8080 library had been changed by Polluck from a strictly software implementation to one that called an arithmetic chip to do a hardware multiply. The
chip had been removed from the library at some point before this project was started. Either the chip or the software multiplication routine must be restored for the 8080 library to correctly handle multiplication.

The size of the program also created problems in NEWCSDL.FOR. One of the stacks in the Formatter Module proved to be too small for the data link receiver project. The stack overflowed before the completion of the realization. The program was adjusted by Ross to allow for a greater stack size. The output of NEWCSDL.FOR for the 8086 library is in Appendices D and E.

At the time of writing this thesis, the following corrections and alterations to the primitive file must be made for NEWCSDL.FOR to properly process it. The two lines with
t.generated for: SYSTEM ************
must be corrected to start the word system in column 23.
The second line,
s.MAIN (::)
must be changed so that the word main is in lower case letters. All lines with s.inputport or s.outputport must be removed from the program altogether until those primitives are added to the realization libraries.

Even with these changes, some errors were still produced. To avoid problems with the s.in, s.ni, s.forcons, s.forend, and s.exitproc primitives, the original CSDL
program was rewritten to remove for-next loops and timed blocks. These primitives must be standardized before they may be used without concern for any errors they may produce. The revised CSDL program is in Appendix B.

The two realization libraries now produce the software and hardware listings to implement the data link receiver project but not without errors. One prominent mistake is that each time an input is sensed, another chip is added to the hardware listing. There should only be one I/O chip produced for a particular input or output. At the time of this writing, CSDL.PAS and NEWCSDL.FOR are being patched to correct some of the problems discovered while designing the data link receiver.
V. CONCLUSIONS AND RECOMMENDATIONS

The Control System Design Environment holds great promise as a tool for simplifying the work of designers and reducing the expense of producing controllers. This ambitious data link receiver project has shown that the CSDE has greater application than may have been realized by those who did the early work on it. This project has shown that it may be possible to design a system in modules and use CSDE to design each each of the modules independently of the others as long as the interfaces are compatible.

This project has also demonstrated the necessity for compatibility among the different programs within the Control System Design Environment. Since the output of one program is the input of another, there must be a conscious effort to standardize the interfaces.

Heilstedt has recommended that NEWCSDL.FOR be rewritten in a newer language than FORTRAN [Ref. 22]. While it is true that other languages may be easier to maintain than FORTRAN, there are other items in the Control System Design Environment that should be addressed first for a better return on the investment of the time and effort that would be required. Since VMS on the VAX computer allows a file produced under one language to be used as input to
another language, there is no need to rush a reprogramming of NEWCSDL.FOR. A rewrite of NEWCSDL.FOR would make the input less column-dependent. A more critical problem is the incompatibility among the different realization libraries. There should be a standardization of primitive names and their associated functions. Without this standardization, the mapping from primitive lists to the realization libraries will continue to be a hit or miss proposition.

One of the major objectives of the Control System Design Environment is that much of the work can be automated to make designing, prototyping, testing, and implementation of controllers faster and less expensive. The lack of standardization requires the intervention of the designer to make the transitions between the various elements of the system.

Along with the standardization of the realization libraries, CSDL.PAS should be updated to incorporate the standards. This program is a real boon to the designer since it removes the tedious work of translating the CSDL program into the primitive list. As previously discussed, implementing some other high level constructs in CSDL.PAS would be an enhancement of the value of the program for the system user. These constructs should include CASE statements and the use of subscripted variables.

The Control System Design Environment has great promise and could be a lucrative product when it is improved.
to provide an automatic transition from CSDL description to the hardware and software listings. As presently implemented, it requires too much effort while moving through the different segments of the system. A great deal of work remains to be done, especially the testing of the interfaces between the different sections and of the realization libraries. It seems that the work will be well worth it.
LIST OF REFERENCES


APPENDIX A
CSDL PROGRAM

IDENTIFICATION
DESIGNER: "S.J. TTER FOX"
DATE: "05-31-84"
PROJECT: "COAST GUARD DATA LINK RECEIVER"

DESIGN CRITERIA
METRIC FIRST;
VOLUMES 1;
MONITORS 1;

ENVIRONMENT

INPUT: KEYFLG,1,TTL; KEYCHAR,R,TTL; MANPOS,8,TTL;
NEWPOS,1,TTL; POSITION,R,TTL; MSGREADY,1,TTL;
MESSAGE,R,TTL; ACNUM,R,TTL;
END INPUT;

OUTPUT: MENU,B,TTL; POLL,8,TTL; MSGVDT,B,TTL;
MSGRCVD,1,TTL;
END OUTPUT;

ARITHMETIC: KEYINMAIN,8; MINTAC,8; MSGDSPA,8;
MLOCATION,8; MCLOCKSET,8; MLOGINOUT,8; SMMANUAL,8;
SMAUTO,8; AC0,8; AC1,8; AC2,8; AC3,8; AC4,8; AC5,8;
AC6,8; AC7,8; AC8,8; AC9,8; INTPERIOD,8; MSG0,8;
MSG1,8; MSG2,8; MSG3,8; MSG4,8; MSG5,8; MSG6,8;
MSG7,8; MSG8,8; MSG9,8; TLOCATION,8; TLOGIN,1;
TLOGOUT,1; NEXTMSG,8; NEXTAC,9; TPOLL,1; COUNT,8;
END ARITHMETIC;

PROCEDURES

FUNCTION KEYINMAIN:
BINARY,1;
SENSE (KEYFLG);
IF KEYFLG=1 THEN KEYINMAIN:=1; END IF;
END KEYINMAIN;

TASK KBINPMMAIN:
MENU := 0; ISSUE (MENU);
SENSE (KEYCHAR);
IF KEYCHAR = 1 THEN MINTAC := 1; END IF;
IF KEYCHAR = 2 THEN MSGDSPLY := 1; END IF;
IF KEYCHAR = 3 THEN MLOCATION := 1; END IF;
IF KEYCHAR = 4 THEN MLOCKSET := 1; END IF;
IF KEYCHAR = 5 THEN MLOGINOUT := 1; END IF;
KEYINMAIN := 0;
END KBINPMAIN;

FUNCTION MINTAC:
    BINARY, 1;
    SENSE (KEYCHAR);
    END MINTAC;

TASK INTAC:
    MENU := 1; ISSUE (MENU);
    SENSE (KEYCHAR);
    IF KEYCHAR = 0 THEN KEYINMAIN := 1; END IF;
    IF KEYCHAR = 1 THEN SMMANUAL := 1; END IF;
    IF KEYCHAR = 2 THEN SMAUTO := 1; END IF;
    MINTAC := 0;
END INTAC;

FUNCTION SMMANUAL:
    BINARY, 1;
    SENSE (KEYCHAR);
    END SMMANUAL;

TASK MANUAL:
    IF AC0 /= 0 THEN POLL := 0; ISSUE (POLL); END IF;
    IF AC1 /= 0 THEN POLL := 1; ISSUE (POLL); END IF;
    IF AC2 /= 0 THEN POLL := 2; ISSUE (POLL); END IF;
    IF AC3 /= 0 THEN POLL := 3; ISSUE (POLL); END IF;
    IF AC4 /= 0 THEN POLL := 4; ISSUE (POLL); END IF;
    IF AC5 /= 0 THEN POLL := 5; ISSUE (POLL); END IF;
    IF AC6 /= 0 THEN POLL := 6; ISSUE (POLL); END IF;
    IF AC7 /= 0 THEN POLL := 7; ISSUE (POLL); END IF;
    IF AC8 /= 0 THEN POLL := 8; ISSUE (POLL); END IF;
    IF AC9 /= 0 THEN POLL := 9; ISSUE (POLL); END IF;
    SMMANUAL := 0;
END MANUAL;

FUNCTION SMAUTO:
    BINARY, 1;
    SENSE (KEYCHAR);
    END SMAUTO;
TASK AUTO:

    MENU:=2; ISSUE(MENU);
    SENSE (KEYCHAR);
    IF KEYCHAR=0 THEN KEVMAIN := 1 ; END IF;
    IF KEYCHAR=1 THEN INTPERIOD := 30 ; END IF;
    IF KEYCHAR=2 THEN INTPERIOD := 20 ; END IF;
    IF KEYCHAR=3 THEN INTPERIOD := 15 ; END IF;
    IF KEYCHAR=4 THEN INTPERIOD := 10 ; END IF;
    IF KEYCHAR=5 THEN INTPERIOD := 5 ; END IF;
    IF KEYCHAR=6 THEN INTPERIOD := 1 ; END IF;
    SNAUTO:=0;
    TPOLL:=1;
END AUTO;

FUNCTION TPOLL:

    BINARY,1;
    IF INTPERIOD=30 THEN IN 30 4 00 DO TPOLL:=1; END IN;
    IF INTPERIOD=20 THEN IN 20 4 00 DO TPOLL:=1; END IN;
    IF INTPERIOD=15 THEN IN 15 4 00 DO TPOLL:=1; END IN;
    IF INTPERIOD=10 THEN IN 10 4 00 DO TPOLL:=1; END IN;
    IF INTPERIOD= 5 THEN IN  5 4 00 DO TPOLL:=1; END IN;
    IF INTPERIOD= 1 THEN IN  1 4 00 DO TPOLL:=1; END IN;
END TPOLL;

TASK POLLAUTO:

    IF AC0/=0 THEN POLL:=0; ISSUE (POLL); END IF;
    IF AC1/=0 THEN POLL:=1; ISSUE (POLL); END IF;
    IF AC2/=0 THEN POLL:=2; ISSUE (POLL); END IF;
    IF AC3/=0 THEN POLL:=3; ISSUE (POLL); END IF;
    IF AC4/=0 THEN POLL:=4; ISSUE (POLL); END IF;
    IF AC5/=0 THEN POLL:=5; ISSUE (POLL); END IF;
    IF AC6/=0 THEN POLL:=6; ISSUE (POLL); END IF;
    IF AC7/=0 THEN POLL:=7; ISSUE (POLL); END IF;
    IF AC8/=0 THEN POLL:=8; ISSUE (POLL); END IF;
    IF AC9/=0 THEN POLL:=9; ISSUE (POLL); END IF;
    TPOLL:=0;
END POLLAUTO;

FUNCTION MMMSGDISPLY:

    BINARY,1;
    SENSE (KEYCHAR);
EVD MSGDSPLY;

TASK MSGDSPLY;
  MSGVDT:=MSG0; ISSUE(MSGVDT);
  MSGVDT:=MSG1; ISSUE(MSGVDT);
  MSGVDT:=MSG2; ISSUE(MSGVDT);
  MSGVDT:=MSG3; ISSUE(MSGVDT);
  MSGVDT:=MSG4; ISSUE(MSGVDT);
  MSGVDT:=MSG5; ISSUE(MSGVDT);
  MSGVDT:=MSG6; ISSUE(MSGVDT);
  MSGVDT:=MSG7; ISSUE(MSGVDT);
  MSGVDT:=MSG8; ISSUE(MSGVDT);
  MSGVDT:=MSG9; ISSUE(MSGVDT);
  MSGDSPLY:=0;
EVD MSGDSPLY;

FJNCTION MLOCATION:
  BINARY,1;
  SENSE (KEYCHAR);
EVD MLOCATION;

TASK LOCATION;
  MENU:=3; ISSUE (MENU);
  SENSE (KEYCHAR);
  IF KEYCHAR=0 THEN KEVINMAIN:=1; END IF;
  IF KEYCHAR=1 THEN NEWPOS:=1; END IF;
  IF KEYCHAR=2 THEN NEWPOS:=0; TMLOCATION:=1; END IF;
  MLOCATION:=0;
EVD LOCATION;

FJNCTION TMLOCATION:
  BINARY,1;
  SENSE (KEYCHAR);
EVD TMLOCATION;

TASK MANLOC;
  SENSE (MANPJS);
  POSITION:=MANPOS;
  TMLOCATION:=0;
EVD MANLOC;

FJNCTION MCLOCKSET:
  BINARY,1;
  SENSE (KEYCHAR);
EVD MCLOCKSET;

TASK CLOCKSET;
  MENU:=4; ISSUE (MENU);
SENSE (KEYCHAR);
MCLOCKSET:=0;
END CLOCKSET;

FUNCTION 4LOGINOUT:
    BINARY,1;
    SENSE (KEYCHAR);
END 4LOGINOUT;

TASK LOGINOUT:
    MENU:=5; ISSUE (MENU);
    SENSE (KEYCHAR);
    IF KEYCHAR=0 THEN KEYINMAIN:=1; END IF;
    IF KEYCHAR=1 THEN TLOGIN:=1; END IF;
    IF KEYCHAR=2 THEN TLOGOUT:=1; END IF;
    MLOGINOUT:=0;
END LOGINOUT;

FUNCTION TLOGIN:
    BINARY,1;
    SENSE (KEYCHAR);
END TLOGIN;

TASK LOGIN:
    ACNUM:=0;
    FOR COUNT FROM 1 TO 4 DO
        SENSE (KEYCHAR);
        ACNUM:= (ACNUM*10)+KEYCHAR;
    END FOR;
    IF NEXTAC=0 AND AC0=0 THEN AC0:=ACNUM; END IF;
    IF NEXTAC=1 AND AC1=0 THEN AC1:=ACNUM; END IF;
    IF NEXTAC=2 AND AC2=0 THEN AC2:=ACNUM; END IF;
    IF NEXTAC=3 AND AC3=0 THEN AC3:=ACNUM; END IF;
    IF NEXTAC=4 AND AC4=0 THEN AC4:=ACNUM; END IF;
    IF NEXTAC=5 AND AC5=0 THEN AC5:=ACNUM; END IF;
    IF NEXTAC=6 AND AC6=0 THEN AC6:=ACNUM; END IF;
    IF NEXTAC=7 AND AC7=0 THEN AC7:=ACNUM; END IF;
    IF NEXTAC=8 AND AC8=0 THEN AC8:=ACNUM; END IF;
    IF NEXTAC=9 AND AC9=0 THEN AC9:=ACNUM; END IF;
    NEXTAC:=NEXTAC+1;
    IF NEXTAC<10 THEN NEXTAC:=0; END IF;
    TLOGIN:=0;
END LOGIN;

FUNCTION TLOGOUT:
    BINARY,1;
    SENSE (KEYCHAR);
END TLOGOUT;
TASK LOGOUT;
    ACNUM := 0;
    FOR COUNT FROM 1 TO 4:4 DO
        SENSE (KEYCHAR);
        ACNUM := (ACNUM*10) + KEYCHAR;
    END FOR;
    IF AC0 = ACNUM THEN AC0 := 0; END IF;
    IF AC1 = ACNUM THEN AC0 := 1; END IF;
    IF AC2 = ACNUM THEN AC0 := 2; END IF;
    IF AC3 = ACNUM THEN AC0 := 3; END IF;
    IF AC4 = ACNUM THEN AC0 := 4; END IF;
    IF AC5 = ACNUM THEN AC0 := 5; END IF;
    IF AC6 = ACNUM THEN AC0 := 6; END IF;
    IF AC7 = ACNUM THEN AC0 := 7; END IF;
    IF AC8 = ACNUM THEN AC0 := 8; END IF;
    IF AC9 = ACNUM THEN AC0 := 9; END IF;
    TLOGOUT := 0;
END LOGOUT;

FUNCTION POSCH:
    BINARY, 1;
    SENSE (NEWPOS);
    IF NEWPOS = 1 THEN POSCH := 1; END IF;
END POSCH;

TASK POSUPDATE:
    SENSE (POSITION);
    POSCH := 0;
END POSUPDATE;

FUNCTION MSGIN:
    BINARY, 1;
    SENSE (MSGREADY);
    IF MSGREADY = 1 THEN MSGIN := 1; END IF;
END MSGIN;

TASK MSGSTORE:
    SENSE (MESSAGE);
    ISSUE (MSGRCVD);
    IF NEXTMSG = 0 THEN AC0 := MESSAGE; END IF;
    IF NEXTMSG = 1 THEN AC1 := MESSAGE; END IF;
    IF NEXTMSG = 2 THEN AC2 := MESSAGE; END IF;
    IF NEXTMSG = 3 THEN AC3 := MESSAGE; END IF;
    IF NEXTMSG = 4 THEN AC4 := MESSAGE; END IF;
    IF NEXTMSG = 5 THEN AC5 := MESSAGE; END IF;
    IF NEXTMSG = 6 THEN AC6 := MESSAGE; END IF;
    IF NEXTMSG = 7 THEN AC7 := MESSAGE; END IF;
    IF NEXTMSG = 8 THEN AC8 := MESSAGE; END IF;
    IF NEXTMSG = 9 THEN AC9 := MESSAGE; END IF;
NEXTMSG := NEXTMSG + 1;
IF NEXTMSG = 10 THEN NEXTMSG := 0; END IF;
END MSGSTORE;

CONTINGENCY LIST
WHEN KEYINMAIN : 100 MS DO KBINPMAIN;
WHEN qintac : 100 MS DO INTAC;
WHEN SMUmanual : 100 MS DO MANUAL;
WHEN SMAuto : 100 MS DO AUTO;
WHEN TPOLL : 100 MS DO POLLAUTO;
WHEN CLOCATION : 100 MS DO LOCATION;
WHEN TMLOCATION : 100 MS DO MANLOC;
WHEN POSCH : 1000 MS DO POSUPDATE;
WHEN WMMSGDSPLY : 100 MS DO MSGDSPLY;
WHEN WCLOCKSET : 100 MS DO CLOCKSET;
WHEN WLOGINOUT : 100 MS DO LOGINOUT;
WHEN TLOGIN : 100 MS DO LOGIN;
WHEN TLOGOUT : 100 MS DO LOGOUT;
WHEN MSGIV : 100 MS DO MSGSTORE;
END
APPENDIX B

CSDL PROGRAM
(REVISIED)

IDENTIFICATION
DESIGNER: "SJITTER FOX"
DATE: "05-31-84"
PROJECT: "COAST GUARD DATA LINK RECEIVER"

DESIGN CRITERIA
METRIC FIRST;
VOLUMES 1;
MONITORS 1;

ENVIRONMENT
INPUT: KEYFLG, 1, TTL; KEYCHAR, 8, TTL; MANPOS, 8, TTL;
NEWPOS, 1, TTL; POSITION, 8, TTL; MSGREADY, 1, TTL;
MESSAGE, 8, TTL; ACNUM, 8, TTL;

END INPUT;

OUTPUT: MENU, 8, TTL; POLL, 8, TTL; MSGVDT, 8, TTL;
MSGRCVD, 1, TTL;

END OUTPUT;

ARITHMETIC: KEYINMAIN, 8; MINVAC, 8; MSGDSPLY, 8;
LOCATION, 8; MCLOCKSET, 8; MLOGINOUT, 8; SMANUAL, 8;
SMAUTO, 8; AC0, 8; AC1, 8; AC2, 8; AC3, 8; AC4, 8; AC5, 8;
AC6, 8; AC7, 8; AC8, 8; AC9, 8; INPERIOD, 8; MSG0, 8;
MSG1, 8; MSG2, 8; MSG3, 8; MSG4, 8; MSG5, 8; MSG6, 8;
MSG7, 8; MSG8, 8; MSG9, 8; TLOCATION, 8; TLOGIN, 1;
TLOGOUT, 1; NEXTMSG, 8; NEXTAC, 8; TPOLL, 1; COUNT, 8;
CLOCK, 8;

END ARITHMETIC;

PROCEDURES

FUNCTION KEYINMAIN:
BINARY, 1;
SENSE (KEYFLG);
IF KEYFLG=1 THEN KEYINMAIN:=1; END IF;
END KEYINMAIN;

TASK KBINPMAIN;
    MENU:0; ISSUE (MENU);
    SENSE (KEYCH); 
    IF KEYCHAR=1 THEN MINTAC :=1; END IF;
    IF KEYCHAR=2 THEN MSGDISPLY:=1; END IF;
    IF KEYCHAR=3 THEN MLOCATION:=1; END IF;
    IF KEYCHAR=4 THEN MCLOCKSET:=1; END IF;
    IF KEYCHAR=5 THEN MLOGINOUT:=1; END IF;
    KEYINMAIN:=0;
END KBINPMAIN;

FUNCTION MINTAC:
    BINARY,1;
    SENSE(KEYCHAR);
END MINTAC;

TASK INTAC;
    MENU:=1; ISSUE (MENJ);
    SENSE (KEYCH); 
    IF KEYCHAR=0 THEN KEYINMAIN:=1; END IF;
    IF KEYCHAR=1 THEN SMMANUAL :=1; END IF;
    IF KEYCHAR=2 THEN SMAUTO :=1; END IF;
    MINTAC:=0;
END INTAC;

FUNCTION SMMANUAL:
    BINARY,1;
    SENSE(KEYCHAR);
END SMMANUAL;

TASK MANUAL;
    IF AC0/=0 THEN POLL:=0; ISSUE (POLL); END IF;
    IF AC1/=0 THEN POLL:=1; ISSUE (POLL); END IF;
    IF AC2/=0 THEN POLL:=2; ISSUE (POLL); END IF;
    IF AC3/=0 THEN POLL:=3; ISSUE (POLL); END IF;
    IF AC4/=0 THEN POLL:=4; ISSUE (POLL); END IF;
    IF AC5/=0 THEN POLL:=5; ISSUE (POLL); END IF;
    IF AC6/=0 THEN POLL:=6; ISSUE (POLL); END IF;
    IF AC7/=0 THEN POLL:=7; ISSUE (POLL); END IF;
    IF AC8/=0 THEN POLL:=8; ISSUE (POLL); END IF;
    IF AC9/=0 THEN POLL:=9; ISSUE (POLL); END IF;
    SMMANUAL:=0;
END MANUAL;
FUNCTION SMAUTO:
    BINARY, 1;
    SENSE (KEYCHAR);
END SMAUTO;

TASK AUTO:
    MENU:=2; ISSUE(MENU);
    SENSE (KEYCHAR);
    IF KEYCHAR=0 THEN KEYINMAIN := 1; END IF;
    IF KEYCHAR=1 THEN INTPERIOD := 30 ; END IF;
    IF KEYCHAR=2 THEN INTPERIOD := 20 ; END IF;
    IF KEYCHAR=3 THEN INTPERIOD := 15 ; END IF;
    IF KEYCHAR=4 THEN INTPERIOD := 10 ; END IF;
    IF KEYCHAR=5 THEN INTPERIOD := 5 ; END IF;
    IF KEYCHAR=6 THEN INTPERIOD := 1 ; END IF;
    SENSE (CLOCK);
    INTTIME := CLOCK;
    SMAUTO:=O;
    TPOLL:=1;
END AUTO;

FUNCTION TPOLL:
    BINARY, 1;
    SENSE (CLOCK);
    IF CLOCK-INTTIME > INTPERIOD THEN TPOLL:=1; END IF;
END TPOLL;

TASK POLLAUTO:
    IF AC0/=0 THEN POLL:=0; ISSUE (POLL); END IF;
    IF AC1/=0 THEN POLL:=1; ISSUE (POLL); END IF;
    IF AC2/=0 THEN POLL:=2; ISSUE (POLL); END IF;
    IF AC3/=0 THEN POLL:=3; ISSUE (POLL); END IF;
    IF AC4/=0 THEN POLL:=4; ISSUE (POLL); END IF;
    IF AC5/=0 THEN POLL:=5; ISSUE (POLL); END IF;
    IF AC6/=0 THEN POLL:=6; ISSUE (POLL); END IF;
    IF AC7/=0 THEN POLL:=7; ISSUE (POLL); END IF;
    IF AC8/=0 THEN POLL:=8; ISSUE (POLL); END IF;
    IF AC9/=0 THEN POLL:=9; ISSUE (POLL); END IF;
    TPOLL:=0;
END POLLAUTO;

FUNCTION MMSGDSPLY:
    BINARY, 1;
    SENSE (KEYCHAR);
END MMSGDSPLY;

TASK MSGDSPLY:
FUNCTION MLOGINOUT:
    BINARY,1;
    SENSE (KEYCHAR);
END MLOGINOUT;

TASK LOGINOUT:
    MENU:=S; ISSUE (MENU);
    SENSE (KEYCHAR);
    IF KEYCHAR=0 THEN KEYINMAIN:=1; END IF;
    IF KEYCHAR=1 THEN TLOGIN:=1; END IF;
    IF KEYCHAR=2 THEN TLOGOUT:=1; END IF;
    MLOGINOUT:=0;
END LOGINOUT;

FUNCTION TLOGIN:
    BINARY,1;
    SENSE (KEYCHAR);
END TLOGIN;

TASK LOGIN:
    ACNUM:=0;
    SENSE (ACNUM);
    IF NEXTAC=0 AND AC0=0 THEN AC0:=ACNUM; END IF;
    IF NEXTAC=1 AND AC1=0 THEN AC1:=ACNUM; END IF;
    IF NEXTAC=2 AND AC2=0 THEN AC2:=ACNUM; END IF;
    IF NEXTAC=3 AND AC3=0 THEN AC3:=ACNUM; END IF;
    IF NEXTAC=4 AND AC4=0 THEN AC4:=ACNUM; END IF;
    IF NEXTAC=5 AND AC5=0 THEN AC5:=ACNUM; END IF;
    IF NEXTAC=6 AND AC6=0 THEN AC6:=ACNUM; END IF;
    IF NEXTAC=7 AND AC7=0 THEN AC7:=ACNUM; END IF;
    IF NEXTAC=8 AND AC8=0 THEN AC8:=ACNUM; END IF;
    IF NEXTAC=9 AND AC9=0 THEN AC9:=ACNUM; END IF;
    NEXTAC:=NEXTAC+1;
    IF NEXTAC=10 THEN NEXTAC:=0; END IF;
    TLOGIN:=0;
END LOGIN;

FUNCTION TLOGOUT:
    BINARY,1;
    SENSE (KEYCHAR);
END TLOGOUT;

TASK LOGOUT:
    ACNUM:=0;
    SENSE (ACNUM);
    IF AC0=ACNUM THEN AC0:=0; END IF;
    IF AC1=ACNUM THEN AC1:=1; END IF;
    IF AC2=ACNUM THEN AC2:=2; END IF;

IF AC3=ACNUM THEN ACO:=3; END IF;
IF AC4=ACNUM THEN ACO:=4; END IF;
IF AC5=ACNUM THEN ACO:=5; END IF;
IF AC6=ACNUM THEN ACO:=6; END IF;
IF AC7=ACNUM THEN ACO:=7; END IF;
IF AC8=ACNUM THEN ACO:=8; END IF;
IF AC9=ACNUM THEN ACO:=9; END IF;
TLOGOUT:=0;
END LOGOUT;

FUNCTION POSCH:
  BINARY, 1;
  SENSE (NEWPOS);
  IF NEWPOS=1 THEN POSCH:=1; END IF;
END POSCH;

TASK POSUPDATE:
  SENSE (POSITION);
  POSCH:=0;
END POSUPDATE;

FUNCTION MSGIN:
  BINARY, 1;
  SENSE (MSGREADY);
  IF MSGREADY=1 THEN MSGIN:=1; END IF;
END MSGIN;

TASK MSGSTORE:
  SENSE (MESSAGE);
  ISSUE (MSGRCVD);
  IF NEXTMSG=0 THEN ACO:=MESSAGE; END IF;
  IF NEXTMSG=1 THEN AC1:=MESSAGE; END IF;
  IF NEXTMSG=2 THEN AC2:=MESSAGE; END IF;
  IF NEXTMSG=3 THEN AC3:=MESSAGE; END IF;
  IF NEXTMSG=4 THEN AC4:=MESSAGE; END IF;
  IF NEXTMSG=5 THEN AC5:=MESSAGE; END IF;
  IF NEXTMSG=6 THEN AC6:=MESSAGE; END IF;
  IF NEXTMSG=7 THEN AC7:=MESSAGE; END IF;
  IF NEXTMSG=8 THEN AC8:=MESSAGE; END IF;
  IF NEXTMSG=9 THEN AC9:=MESSAGE; END IF;
  NEXTMSG:=NEXTMSG+1;
  IF NEXTMSG=10 THEN NEXTMSG:=0; END IF;
END MSGSTORE;

CONTINGENCY LIST
  WHEN KEYINMAIN = 100 MSG DO KBINPMAIN:

68
WHEN MINTAC : 100 MS DO INTAC;
WHEN SMMANUAL : 100 MS DO MANUAL;
WHEN SMAUTO : 100 MS DO AUTO;
WHEN TPOOL : 100 MS DO POLLAUTO;
WHEN MLOCATION : 100 MS DO LOCATION;
WHEN TMLOCATION : 100 MS DO MANLOC;
WHEN POSC4 : 1000 MS DO POSUPDATE;
WHEN MMGSDispLY : 100 MS DO MSGDispLY;
WHEN MCLOCKSET : 100 MS DO CLOCKSET;
WHEN MLOGINOUT : 100 MS DO LOGINOUT;
WHEN TLOGIN : 100 MS DO LOGIN;
WHEN TLOGOUT : 100 MS DO LOGOUT;
WHEN MSGIN : 100 MS DO MSGSTORE;

END
APPENDIX C

CSOL.PAS OUTPUT
PRIMITIVE LISTING

P 1t.generated for: SYSTEM
P 2s.main (::*
P 16s.var (KEYINMAIN:8,0)
P 17s.var (MINTAC:8,0)
P 18s.var (MMSGDSPLY:8,0)
P 19s.var (MLOCATION:8,0)
P 20s.var (MCLOCKSET:8,0)
P 21s.var (MLOGING OUT:8,0)
P 22s.var (SWMANUAL:8,0)
P 23s.var (SWAUTO:8,0)
P 24s.var (AC0:8,0)
P 25s.var (AC1:8,0)
P 26s.var (AC2:8,0)
P 27s.var (AC3:8,0)
P 28s.var (AC4:8,0)
P 29s.var (AC5:8,0)
P 30s.var (AC6:8,0)
P 31s.var (AC7:8,0)
P 32s.var (AC8:8,0)
P 33s.var (AC9:8,0)
P 34s.var (ITPERIOD:8,0)
P 35s.var (MSG0:8,0)
P 36s.var (MSG1:8,0)
P 37s.var (MSG2:8,0)
P 38s.var (MSG3:8,0)
P 39s.var (MSG4:8,0)
P 40s.var (MSG5:8,0)
P 41s.var (MSG6:8,0)
P 42s.var (MSG7:8,0)
P 43s.var (MSG8:8,0)
P 44s.var (MSG9:8,0)
P 45s.var (TLOCATION:8,0)
P 46s.var (TLOGIN:1,0)
P 47s.var (TLOGOUT:1,0)
P 48s.var (NEXTMSG:8,0)
P 49s.var (NEXTAC:8,0)
P 50s.var (TPO LL:1,0)
P 51s.var (COUN T:8,0)
P 52s.var (CLOCK:8,0)
P 53t.generated for: KEYINMAIN

***************

P 54s.roc (KEYINMAIN:)
P 55s .sensecond (KEYFLG:1)
P 56s.eq (@T01,KEYFLG,AC0:8,1,8)
**CASE MAIN**

P 57s.jmof  (AT01,AT01:8)
P 58s.assign (KEYINMAIN,AC01:1,8)
P 59s.loc  (AT01:)
P 60s.exitproc (KEYINMAIN,KEYINMAIN:)
P 61t.generated for: KBINPMAIN

***************
P 62s.proc  (KBINPMAIN:)
P 63s.assign (MENU,AC02:8,8)
P 64s.issuevent (MENU:8)
P 65s.sensecond (KEYCHAR:8)
P 66s.eq  (AT01,KEYCHAR,AC01:8,8,8)
P 67s.jmof  (AT01,AT02:8)
P 68s.assign (MINTAC,AC01:8,8)
P 69s.loc  (AT02:)
P 70s.eq  (AT01,KEYCHAR,AC03:8,8,8)
P 71s.jmof  (AT01,AT03:8)
P 72s.assign (MMSGDSPLY,AC01:8,8)
P 73s.loc  (AT03:)
P 74s.eq  (AT01,KEYCHAR,AC04:8,8,8)
P 75s.jmof  (AT01,AT04:8)
P 76s.assign (MLLOCATION,AC01:8,8)
P 77s.loc  (AT04:)
P 78s.eq  (AT01,KEYCHAR,AC05:8,8,8)
P 79s.jmof  (AT01,AT05:8)
P 80s.assign (MCLOCKSET,AC01:8,8)
P 81s.loc  (AT05:)
P 82s.eq  (AT01,KEYCHAR,AC06:8,8,8)
P 83s.jmof  (AT01,AT06:8)
P 84s.assign (MLOGINOUT,AC01:8,8)
P 85s.loc  (AT06:)
P 86s.assign (KEYINMAIN,AC02:1,8)
P 87s.exitproc (KBINPMAIN,KBINPMAIN:)
P 88t.generated for:MINTAC

***************
P 89s.proc  (MINTAC:)
P 90s.sensecond (KEYCHAR:8)
P 91s.exitproc (MINTAC,MINTAC:)
P 92t.generated for: INTAC

***************
P 93s.proc  (INTAC:)
P 94s.assign (MENU,AC01:8,8)
P 95s.issuevent (MENU:8)
P 96s.sensecond (KEYCHAR:8)
P 97s.eq  (AT01,KEYCHAR,AC02:8,8,8)
P 98s.jmof  (AT01,AT07:8)
P 99s.assign (KEYINMAIN,AC01:1,8)
P 100s.loc  (AT07:)
P 101s.eq  (AT01,KEYCHAR,AC01:8,8,8)
P 102s.jmof  (AT01,AT08:8)
P 103s.assign (SMMANUAL,AC01:8,8)
P 104s.loc  (AT08:)

71
P 105s.eq (AT01,KEYCHAR,AC03:8,8,8)
P 106s.jmof (AT01,309:8)
P 107s.assign (SMAUTO,AC01:8,8)
P 108s.loc (309:)
P 109s.assign (MINTAC,AC02:1,8)
P 110s.exitproc (INTAC,INTAC:)
P 111t.generated for: SMMANUAL

***************************
P 112s.proc (SMMANUAL:)
P 113s.elsecond (KEYCHAR:8)
P 114s.exitproc (SMMANUAL,SMMANUAL:)
P 115t.generated for: MANUAL

***************************
P 116s.proc (MANUAL:)
P 117s.ne (AT01,AC0,AC02:8,8,8)
P 118s.jmof (AT01,310:8)
P 119s.assign (POLL,AC02:8,8)
P 120s.issuevent (POLL:8)
P 121s.loc (@10:)
P 122s.ne (AT01,AC1,AC02:8,8,8)
P 123s.jmof (AT01,311:8)
P 124s.assign (POLL,AC01:8,8)
P 125s.issuevent (POLL:8)
P 126s.loc (@11:)
P 127s.ne (AT01,AC2,AC02:8,8,8)
P 128s.jmof (AT01,312:8)
P 129s.assign (POLL,AC03:8,8)
P 130s.issuevent (POLL:8)
P 131s.loc (@12:)
P 132s.ne (AT01,AC3,AC02:8,8,8)
P 133s.jmof (AT01,313:8)
P 134s.assign (POLL,AC04:8,8)
P 135s.issuevent (POLL:8)
P 136s.loc (@13:)
P 137s.ne (AT01,AC4,AC02:9,8,9)
P 138s.jmof (AT01,314:8)
P 139s.assign (POLL,AC05:8,8)
P 140s.issuevent (POLL:8)
P 141s.loc (@14:)
P 142s.ne (AT01,AC5,AC02:8,8,8)
P 143s.jmof (AT01,315:8)
P 144s.assign (POLL,AC06:8,8)
P 145s.issuevent (POLL:8)
P 146s.loc (@15:)
P 147s.ne (AT01,AC6,AC02:8,8,8)
P 148s.jmof (AT01,316:8)
P 149s.assign (POLL,AC07:8,8)
P 150s.issuevent (POLL:8)
P 151s.loc (@16:)
P 152s.ne (AT01,AC7,AC02:8,8,8)
P 153s.jmof (AT01,317:8)
P 203s.jmof (@TO1,AC0:8,8)
P 204s.assign (INTPERIOD,AC0:8,8)
P 205s.loc (@26:)
P 206s.assign (INTTIME,CLOCK:8,8)
P 207s.assign (SMAUTO,AC0:2:8,8)
P 208s.assign (TPOLL,AC0:1:8)
P 209s.exitroc (AJTO,AUTO:)
P 210t.generated for: TPOLL

*********************
P 211s.exitroc (TPOLL:)
P 212s.sub (@TO1,CLOCK,INTTIME:8,8,8)
P 213s.qt (@TO1,AT01,INTPERIOD:8,8,8)
P 214s.jmof (@TO1,AT27:8)
P 215s.assign (TPOLL,AC0:1:8)
P 216s.loc (@27:)
P 217s.exitroc (TPOLL,TPOLL:)
P 218t.generated for: POLLAUTO

*********************
P 219s.exitroc (POLLAUTO:)
P 220s.ne (@TO1,AC0,AC02:8,8,8)
P 221s.jmof (@TO1,AT28:8)
P 222s.assign (POL,AC02:8,8)
P 223s.issuevent (POL:8)
P 224s.loc (@28:)
P 225s.ne (@TO1,ATC01,AC02:8,8,8)
P 226s.jmof (@TO1,AT29:8)
P 227s.assign (POL,AC01:8,8)
P 228s.issuevent (POL:8)
P 229s.loc (@29:)
P 230s.ne (@TO1,AC2,AC02:8,8,8)
P 231s.jmof (@TO1,AT30:8)
P 232s.assign (POL,AC0:3:8,8)
P 233s.issuevent (POL:8)
P 234s.loc (@30:)
P 235s.ne (@TO1,AC3,AC02:8,8,8)
P 236s.jmof (@TO1,AT31:8)
P 237s.assign (POL,AC0:4:8,8)
P 238s.issuevent (POL:8)
P 239s.loc (@31:)
P 240s.ne (@TO1,AC4,AC02:8,8,8)
P 241s.jmof (@TO1,AT32:8)
P 242s.assign (POL,AC0:5:8,8)
P 243s.issuevent (POL:8)
P 244s.loc (@32:)
P 245s.ne (@TO1,AC5,AC02:8,8,8)
P 246s.jmof (@TO1,AT33:8)
P 247s.assign (POL,AC0:6:8,8)
P 248s.issuevent (POL:8)
P 249s.loc (@33:)
P 250s.ne (@TO1,AC6,AC02:8,8,8)
P 251s.jmof (@TO1,AT34:8)
**P 252s.assign (POLL,AC07:8,8)**
**P 253s.issuevent (POLL:8)**
**P 254s.loc (@34:)**
**P 255s.ne (@T01,AC7,AC02:8,8,8)**
**P 256s.jmof (@T01,AC5:8)**
**P 257s.assign (POLL,AC08:8)**
**P 258s.issuevent (POLL:8)**
**P 259s.loc (@35:)**
**P 260s.ne (@T01,AC8,AC02:8,8,8)**
**P 261s.jmof (@T01,AC6:8)**
**P 262s.assign (POLL,AC09:8)**
**P 263s.issuevent (POLL:8)**
**P 264s.loc (@36:)**
**P 265s.ne (@T01,AC9,AC02:8,8,8)**
**P 266s.jmof (@T01,AC7:8)**
**P 267s.assign (POLL,AC10:8)**
**P 268s.issuevent (POLL:8)**
**P 269s.loc (@37:)**
**P 270s.assign (TPOLL,AC02:1,8)**
**P 271s.exitproc (POLLAUTO,POLLAUTO:)**
**P 272t.generated for: MSGDSPLY**

***************************************************************************
**P 273s.oroc (MSGDSPLY:**
**P 274s.gensecond (KEYCHAR:8)**
**P 275s.exitproc (MSGDSPLY,MSGDSPLY:**
**P 276t.generated for: MSGDSPLY**

***************************************************************************
**P 277s.oroc (MSGDSPLY:**
**P 278s.assign (MSGVDT,MSG0:8,8)**
**P 279s.issuevent (MSGVDT:8)**
**P 280s.assign (MSGVDT,MSG1:8,8)**
**P 281s.issuevent (MSGVDT:8)**
**P 282s.assign (MSGVDT,MSG2:8,8)**
**P 283s.issuevent (MSGVDT:8)**
**P 284s.assign (MSGVDT,MSG3:8,8)**
**P 285s.issuevent (MSGVDT:8)**
**P 286s.assign (MSGVDT,MSG4:8,8)**
**P 287s.issuevent (MSGVDT:8)**
**P 288s.assign (MSGVDT,MSG5:8,8)**
**P 289s.issuevent (MSGVDT:8)**
**P 290s.assign (MSGVDT,MSG6:8,8)**
**P 291s.issuevent (MSGVDT:8)**
**P 292s.assign (MSGVDT,MSG7:8,8)**
**P 293s.issuevent (MSGVDT:8)**
**P 294s.assign (MSGVDT,MSG8:8,8)**
**P 295s.issuevent (MSGVDT:8)**
**P 296s.assign (MSGVDT,MSG9:8,8)**
**P 297s.issuevent (MSGVDT:8)**
**P 298s.assign (MSGDSPLY,AC02:1,8)**
**P 299s.exitproc (MSGDSPLY,MSGDSPLY:**
**P 300t.generated for: MLJCATION**
**********

P 301s.roc (MLOCATION:)
P 302s.sensecond (KEYCHAR:8)
P 303s.exitproc (MLOCATION,MLOCATION:)
P 304t.generated for: LOCATION

**********

P 305s.roc (LOCATION:)
P 306s.assign (MENU,AC04:8,8)
P 307s.issuemvent (MENU:8)
P 308s.sensecond (KEYCHAR:8)
P 309s.eq (@T01,KEYCHAR,AC02:8,8,8)
P 310s.jmpf (@T01,AC38:8)
P 311s.assign (KEYINMAIN,AC01:1,8)
P 312s.loc (@38:8)
P 313s.eq (@T01,KEYCHAR,AC01:8,8,8)
P 314s.jmpf (@T01,AC39:8)
P 315s.assign (NEWPOS,AC01:1,8)
P 316s.loc (@39:8)
P 317s.eq (@T01,KEYCHAR,AC03:8,8,8)
P 318s.jmpf (@T01,AC40:8)
P 319s.assign (NEWPOS,AC02:1,8)
P 320s.assign (TMLOCATION,AC01:8,8)
P 321s.loc (@40:8)
P 322s.assign (MLOCATION,AC02:1,8)
P 323s.exitproc (LOCATION,LOCATION:)
P 324t.generated for: TMLOCATION

**********

P 325s.roc (TMLOCATION:)
P 326s.sensecond (KEYCHAR:8)
P 327s.exitproc (TMLOCATION,TMLOCATION:)
P 328t.generated for: MANLOC

**********

P 329s.roc (MANLOC:)
P 330s.sensecond (MANPOS:8)
P 331s.assign (POSITION,MANPOS:8,8)
P 332s.assign (TMLOCATION,AC02:1,8)
P 333s.exitproc (MANLOC,MANLOC:)
P 334t.generated for: MCLOCKSET

**********

P 335s.roc (MCLOCKSET:)
P 336s.sensecond (KEYCHAR:8)
P 337s.exitproc (MCLOCKSET,MCLOCKSET:)
P 338t.generated for: CLOCKSET

**********

P 339s.roc (CLOCKSET:)
P 340s.assign (MENU,AC05:8,8)
P 341s.issuemvent (MENU:8)
P 342s.sensecond (KEYCHAR:8)
P 343s.assign (MCLOCKSET,AC02:1,8)
P 344s.exitproc (CLOCKSET,CLOCKSET:)
P 345t.generated for: MLOGINOUT

76
*******
P 346s.oroc (MLOGINOUT:)
P 347s.sensecond (KEYCHAR:8)
P 348s.exitroroc (MLOGINOUT,MLOGINOUT:)
P 349t.generated for: LOGINDUT
*******
P 350s.oroc (MLOGINOUT:)
P 351s.assign (MENU:@C06:8,8)
P 352s.issuevent (MENU:8)
P 353s.sensecond (KEYCHAR:8)
P 354s.eq (@T01,KEYCHAR,@C02:8,8,8)
P 355s.jmof (@T01,841:8)
P 356s.assign (KEYINMAIN,@C01:1,8)
P 357s.loc (@41:)
P 358s.eq (@T01,KEYCHAR,@C01:8,8,8)
P 359s.jmof (@T01,842:8)
P 360s.assign (TLOGIN,@C01:1,8)
P 361s.loc (@42:)
P 362s.eq (@T01,KEYCHAR,@C03:8,8,8)
P 363s.jmof (@T01,843:8)
P 364s.assign (TLOGINOUT,@C01:1,8)
P 365s.loc (@43:)
P 366s.assign (MLOGINOUT,@C02:1,8)
P 367s.exitroroc (MLOGINOUT,MLOGINOUT:)
P 368t.generated for: TLOGIN
*******
P 369s.oroc (TLOGIN:)
P 370s.sensecond (KEYCHAR:8)
P 371s.exitroroc (TLOGIN,TLOGIN:)
P 372t.generated for: LOGIN
*******
P 373s.oroc (LOGINDUT:)
P 374s.assign (ACNUM,@C02:8,8)
P 375s.sensecond (ACNUM:8)
P 376s.eq (@T01,NEXTAC,@C02:9,8,8)
P 377s.eq (@T02,AC0,@C02:8,8,8)
P 378s.and (@T01,8T01,8T02:8,8,8)
P 379s.jmof (@T01,844:8)
P 380s.assign (AC0,ACNUM:8,8)
P 381s.loc (@44:)
P 382s.eq (@T01,NEXTAC,AC0:8,8,8)
P 383s.eq (@T02,AC1,AC02:8,8,8)
P 384s.and (@T01,8T01,8T02:8,8,8)
P 385s.jmof (@T01,845:8)
P 386s.assign (AC1,ACNUM:8,8)
P 387s.loc (@45:)
P 388s.eq (@T01,NEXTAC,AC03:8,8,8)
P 389s.eq (@T02,AC2,AC02:8,8,8)
P 390s.and (@T01,8T01,8T02:8,8,8)
P 391s.jmof (@T01,846:8)
P 392s.assign (AC2,ACNUM:8,8)
P 393s.loc (@46:)
P 394s.eq (@T01,NEXTAC,AC04:8,8,8)
P 395s.eq (@T02,AC3,AC02:8,8,8)
P 396s.and (@T01,A01,A02:8,8,8)
P 397s.jmpf (@T01,A47:8)
P 398s.assign (AC3,ACNUM:8,8)
P 399s.loc (@47:)
P 400s.eq (@T01,NEXTAC,AC05:8,8,8)
P 401s.eq (@T02,AC4,AC02:8,8,8)
P 402s.and (@T01,A01,A02:8,8,8)
P 403s.jmpf (@T01,A48:8)
P 404s.assign (AC4,ACNUM:8,8)
P 405s.loc (@48:)
P 406s.eq (@T01,NEXTAC,AC06:8,8,8)
P 407s.eq (@T02,AC5,AC02:8,8,8)
P 408s.and (@T01,A01,A02:8,8,8)
P 409s.jmpf (@T01,A49:8)
P 410s.assign (AC5,ACNUM:8,8)
P 411s.loc (@49:)
P 412s.eq (@T01,NEXTAC,AC07:8,8,8)
P 413s.eq (@T02,AC6,AC02:8,8,8)
P 414s.and (@T01,A01,A02:8,8,8)
P 415s.jmpf (@T01,A50:8)
P 416s.assign (AC6,ACNUM:8,8)
P 417s.loc (@50:)
P 418s.eq (@T01,NEXTAC,AC08:8,8,8)
P 419s.eq (@T02,AC7,AC02:8,8,8)
P 420s.and (@T01,A01,A02:8,8,8)
P 421s.jmpf (@T01,A51:8)
P 422s.assign (AC7,ACNUM:8,8)
P 423s.loc (@51:)
P 424s.eq (@T01,NEXTAC,AC09:8,8,8)
P 425s.eq (@T02,AC8,AC02:8,8,8)
P 426s.and (@T01,A01,A02:8,8,8)
P 427s.jmpf (@T01,A52:8)
P 428s.assign (AC8,ACNUM:8,8)
P 429s.loc (@52:)
P 430s.eq (@T01,NEXTAC,AC10:8,8,8)
P 431s.eq (@T02,AC9,AC02:8,8,8)
P 432s.and (@T01,A01,A02:8,8,8)
P 433s.jmpf (@T01,A53:8)
P 434s.assign (AC9,ACNUM:8,8)
P 435s.loc (@53:)
P 436s.add (@T01,NEXTAC,AC01:8,8,8)
P 437s.assign (NEXTAC,A01:8,8)
P 438s.eq (@T01,NEXTAC,AC14:8,8,8)
P 439s.jmpf (@T01,A54:8)
P 440s.assign (NEXTAC,AC02:8,8)
P 441s.loc (@54:)
P 442s.assign (TL0GIN,AC02:1,8)
P 443s.exitproc (LOGIN,LOGIN:)

78
P 444t.generated for: TLOGOUT
*****************************
P 445t.proc (TLOGOUT:)
P 446t.sensecond (KEYCHAR:8)
P 447t.exitproc (TLOGOUT,TLOGOUT:)
P 448t.generated for: LOGOUT
*****************************
P 449t.proc (LOGOUT:)
P 450t.assign (ACNUM,AC02:8,8)
P 451t.sensecond (ACNUM:8)
P 452t.eq (@T01,AC0,ACNUM:8,8,8)
P 453t.jmpf (@T01,AC55:8)
P 454t.assign (AC0,AC02:8,8)
P 455t.loc (@AC55:)
P 456t.eq (@T01,AC1,ACNUM:8,8,8)
P 457t.jmpf (@T01,AC56:8)
P 458t.assign (AC0,AC01:8,8)
P 459t.loc (@AC56:)
P 460t.eq (@T01,AC2,ACNUM:8,8,8)
P 461t.jmpf (@T01,AC57:8)
P 462t.assign (AC0,AC03:9,8)
P 463t.loc (@AC57:)
P 464t.eq (@T01,AC3,ACNUM:8,8,8)
P 465t.jmpf (@T01,AC58:8)
P 466t.assign (AC0,AC04:8,8)
P 467t.loc (@AC58:)
P 468t.eq (@T01,AC4,ACNUM:8,8,8)
P 469t.jmpf (@T01,AC59:8)
P 470t.assign (AC0,AC05:8,8)
P 471t.loc (@AC59:)
P 472t.eq (@T01,AC5,ACNUM:8,8,8)
P 473t.jmpf (@T01,AC60:8)
P 474t.assign (AC0,AC06:8,8)
P 475t.loc (@AC60:)
P 476t.eq (@T01,AC6,ACNUM:8,8,8)
P 477t.jmpf (@T01,AC61:8)
P 478t.assign (AC0,AC07:8,8)
P 479t.loc (@AC61:)
P 480t.eq (@T01,AC7,ACNUM:8,8,8)
P 481t.jmpf (@T01,AC62:8)
P 482t.assign (AC0,AC08:8,8)
P 483t.loc (@AC62:)
P 484t.eq (@T01,AC8,ACNUM:8,8,8)
P 485t.jmpf (@T01,AC63:8)
P 486t.assign (AC0,AC09:8,8)
P 487t.loc (@AC63:)
P 488t.eq (@T01,AC9,ACNUM:8,8,8)
P 489t.jmpf (@T01,AC64:8)
P 490t.assign (AC0,AC10:8,8)
P 491t.loc (@AC64:)
P 492t.assign (TLOGOUT,AC02:1,8)
P 540s.jmpf (@T01, @T2:8)
P 541s.assign (AC5, MESSAGE:8, 8)
P 542s.loc (@T2:8)
P 543s.eq (@T01, NEXTMSG, AC07:9, 8, 8)
P 544s.jmpf (@T01, @T7:8)
P 545s.assign (AC6, MESSAGE:8, 8)
P 546s.loc (@T7:8)
P 547s.eq (@T01, NEXTMSG, AC08:8, 8, 8)
P 548s.jmpf (@T01, @T7:8)
P 549s.assign (AC7, MESSAGE:8, 8)
P 550s.loc (@T7:8)
P 551s.eq (@T01, NEXTMSG, AC09:8, 8, 8)
P 552s.jmpf (@T01, @T7:8)
P 553s.assign (AC8, MESSAGE:8, 8)
P 554s.loc (@T7:8)
P 555s.eq (@T01, NEXTMSG, AC10:8, 8, 8)
P 556s.jmpf (@T01, @T7:8)
P 557s.assign (AC9, MESSAGE:8, 8)
P 558s.loc (@T7:8)
P 559s.add (@T01, NEXTMSG, AC01:8, 8, 8)
P 560s.assign (NEXTMSG, @T01:8, 8)
P 561s.eq (@T01, NEXTMSG, AC12:8, 8, 8)
P 562s.jmpf (@T01, @T7:8)
P 563s.assign (NEXTMSG, AC02:8, 8)
P 564s.loc (@T7:8)
P 565s.exitproc (MSGSTORE, MSGSTORE:)
P 566s.generated for: SYSTEM ****************************
P 567s.cons (AC01, 1:8)
P 568s.cons (AC02, 0:8)
P 569s.cons (AC03, 2:8)
P 570s.cons (AC04, 3:8)
P 571s.cons (AC05, 4:8)
P 572s.cons (AC06, 5:8)
P 573s.cons (AC07, 6:8)
P 574s.cons (AC08, 7:8)
P 575s.cons (AC09, 8:8)
P 576s.cons (AC10, 9:8)
P 577s.cons (AC11, 10:8)
P 578s.cons (AC12, 20:8)
P 579s.cons (AC13, 15:8)
P 580s.cons (AC14, 10:8)
P 581s.var (@T01:8)
P 582s.var (@T02:8)
APPENDIX D

NEWCSDL.FOR OUTPUT
LISTING (SOFTWARE)

;---------------------------------
; - intel 8086 realization -
;---------------------------------

sys14 equ 0D000H
sys13 equ 0C000H
sys12 equ 0B000H
sys11 equ 0A000H
sys10 equ 9000H
sys9 equ 8000H
sys8 equ 7000H
sys7 equ 6000H
sys6 equ 5000H
sys5 equ 4000H
sys4 equ 3000H
sys3 equ 2000H
sys2 equ 1000H

; this routine allows for a 1K stack. a stack that
grows larger than
; this will overflow into the data segment. to allow a
larger stack
; the ramptr global primitive in s.main must be set to
the required
; value. this method overlaps 64K of stack segment and
64K of data
; segment.

; org 0000H
mov DS,DX
mov DX,03FFH
mov SS,DX
mov 0F000H
jmp 1024

; set data segment
; set stack segment
; set data segment
; jump to low
address of highest

org ****** ;64K block
jmp 0E000H

from address pointer
to bottom of 2nd

block

high 64K

org 1024

;define 8-bit storage
org 1023

;8 bit variable KEYINM

in ram

KEYINM: do 0
org 983046

;from address pointer

;8 bit variable MINTAC

in ram

MINTAC: do 0
org 983046

;from address pointer

;8 bit variable MMSGDS

in ram

MMSGDS: do 0
org 983046

;from address pointer

;8 bit variable MLOCAT

in ram

MLOCAT: do 0
org 983046

;from address pointer

;8 bit variable MCLOCK

in ram

MCLOCK: do 0
org 983046

;from address pointer

;8 bit variable MLOGIN

in ram

MLOGIN: do 0
org 983046

;from address pointer

;8 bit variable SMANU

in ram

SMANU: do 0
org 983046

;from address pointer

;8 bit variable SMAUTO

in ram

SMAUTO: do 0
org 983046

;from address pointer

;8 bit variable ACO in

ram

ACO: db 0
; define 8-bit storage
org 983046
"from address pointer"
org 1023
"8 bit variable AC1 in ram"

ram
AC1: db 0
"from address pointer"
org 983046
"8 bit variable AC2 in ram"
org 1023

ram
AC2: db 0
"from address pointer"
org 983046
"8 bit variable AC3 in ram"
org 1023

ram
AC3: db 0
"from address pointer"
org 983046
"8 bit variable AC4 in ram"
org 1023

ram
AC4: db 0
"from address pointer"
org 983046
"8 bit variable AC5 in ram"
org 1023

ram
AC5: db 0
"from address pointer"
org 983046
"8 bit variable AC6 in ram"
org 1023

ram
AC6: db 0
"from address pointer"
org 983046
"8 bit variable AC7 in ram"
org 1023

ram
AC7: db 0
"from address pointer"
org 983046
"8 bit variable AC8 in ram"
org 1023

ram
AC8: db 0
"from address pointer"
org 983046
"8 bit variable AC9 in ram"
org 1023

ram
AC9: db 0
"from address pointer"
org 983046
"8 bit variable INTPER in ram"
org 1023

INTPER: db 0
"from address pointer"
org 983046


```
#define 8-bit storage
org 1023

in ram
MSG0:  do 0
org 983046

#define 8-bit storage
org 1023

in ram
MSG1:  do 0
org 983046

#define 8-bit storage
org 1023

in ram
MSG2:  do 0
org 983046

#define 8-bit storage
org 1023

in ram
MSG3:  do 0
org 983046

#define 8-bit storage
org 1023

in ram
MSG4:  do 0
org 983046

#define 8-bit storage
org 1023

in ram
MSG5:  do 0
org 983046

#define 8-bit storage
org 1023

in ram
MSG6:  do 0
org 983046

#define 8-bit storage
org 1023

in ram
MSG7:  do 0
org 983046

#define 8-bit storage
org 1023

in ram
MSG8:  do 0
org 983046

#define 8-bit storage
org 1023

in ram
MSG9:  do 0
org 983046

#define 8-bit storage

```
in ram
TMLOCA: db 0
org 983046
;define 8-bit storage
org 1023
;8 bit variable TMLUCA

in ram
TLOGIN: db 0
org 983046
;define 8-bit storage
org 1023
;8 bit variable TLOGIN

in ram
TLOGOU: db 0
org 983046
;define 8-bit storage
org 1023
;8 bit variable TLOGOU

in ram
NEXTMS: db 0
org 983046
;define 8-bit storage
org 1023
;8 bit variable NEXTMS

in ram
NEXTAC: db 0
org 983046
;define 8-bit storage
org 1023
;8 bit variable NEXTAC

in ram
TPOLL: db 0
org 983046
;define 8-bit storage
org 1023
;8 bit variable TPOLL

in ram
COUNT: db 0
org 983046
;define 8-bit storage
org 1023
;8 bit variable COUNT

in ram
CLOCK: db 0
org 983046
;define 8-bit storage
org 1023
;8 bit variable CLOCK

;PROCEDURE KEYINM
@KEYINM: nop
;entry point for

KEYINM
;detect condition-type input (16-bit)
in AX,0 ;sense environmental data
mov KEYFLG,AX
;test for equality between KEYFLG and AC01 (16-bit)
mov @T01,1 ;presuppose equality
mov AX,KEYFLG ;fetch KEYFLG
cmp AX,AC01 ;compare arguments
jz $+4 ;send routine if
true

branch on false

not equal, AT01 = 0

load value into accumulator

AL, AT01

false(=0)

assign value of one variable to another variable (16-bit)

AX, AT01

assign AT01

KEYINM, AX

ito KEYINM

@01: define location @01

; procedure KBINPM

@KBINPM: entry point for KBINPM

assign value of one variable to another variable (16-bit)

AX, AT01

assign AT01

MENU, AX

ito MENU

send condition-type output (16-bit)

AX, MENU

issue control output

0, AX

detect condition-type input (16-bit)

AX, 1

sense environmental data

KEYCHA, AX

test for equality between KEYCHA and AT01 (16-bit)

AT01, 1

presuppose equality

AX, KEYCHA

fetch KEYCHA

AX, AT01

compare arguments

$+4

end routine if true

branch on false

load value into accumulator

AL, AT01

false(=0)

assign value of one variable to another variable (16-bit)

AX, AT01

assign AT01

MINTAC, AX

ito MINTAC

@02: define location @02

test for equality between KEYCHA and AT03 (16-bit)

AT01, 1

presuppose equality

AX, KEYCHA

fetch KEYCHA

AX, AT03

compare arguments

$+4

end routine if true
mov AT01,0 ;not equal, AT01 ≠ 0
branch on false
mov AL,AT01 ;load value into accumulator
cmp AL,0 ;compare to zero
jz @03 ;jump to @03 if false(≠0)

;assign value of one variable to another variable
(16-bit)

mov AX,AC01 ;assign AC01
mov MMMGSDS,AX ;to MMMGSDS
@03:
nop ;define location @03
;test for equality between KEYCHA and @C04 (16-bit)
mov AT01,1 ;presuppose equality
mov AX,KEYCHA ;fetch KEYCHA
cmp AX,AC04 ;compare arguments
jz $+4 ;send routine if true

true

mov AT01,0 ;not equal, AT01 ≠ 0
branch on false
mov AL,AT01 ;load value into accumulator
cmp AL,0 ;compare to zero
jz @04 ;jump to @04 if false(≠0)

false(≠0)

;assign value of one variable to another variable
(16-bit)

mov AX,AC01 ;assign AC01
mov MLOCAT,AX ;to MLOCAT
@04:
nop ;define location @04
;test for equality between KEYCHA and @C05 (16-bit)
mov AT01,1 ;presuppose equality
mov AX,KEYCHA ;fetch KEYCHA
cmp AX,AC05 ;compare arguments
jz $+4 ;send routine if true

true

mov AT01,0 ;not equal, AT01 ≠ 0
branch on false
mov AL,AT01 ;load value into accumulator
cmp AL,0 ;compare to zero
jz @05 ;jump to @05 if false(≠0)

false(≠0)

;assign value of one variable to another variable
(16-bit)

mov AX,AC01 ;assign AC01
mov MCLOCK,AX ;to MCLOCK
@05:
nop ;define location @05
;test for equality between KEYCHA and @C06 (16-bit)
mov AT01,1 ;presuppose equality
mov AX,KEYCHA ;fetch KEYCHA
cmo AX,3C06 ;compare arguments
jz $+4 ;send routine if true
mov @T01,0 ;not equal, @T01 = 0
branch on false
mov AL,@T01 ;load value into accumulator
cmo AL,0 ;compare to zero
jz @06 ;jump to @06 if false(≠0)

;assign value of one variable to another variable
(16-bit)
mov AX,3C01 ;assign @C01
mov MLOGIN,AX ;to MLOGIN
@06: nop ;define location @06

;assign value of one variable to another variable
(16-bit)
mov AX,3C02 ;assign @C02
mov KEYINM,AX ;to KEYINM

;procedure MINTAC
@MINTAC: nop ;entry point for MINTAC

;detect condition-type input (16-bit)
in AX,2 ;sense environmental data
mov KEYCHA,AX

;procedure INTAC
@INTAC: nop ;entry point for INTAC

;assign value of one variable to another variable
(16-bit)
mov AX,3C01 ;assign @C01
mov MENU,AX ;to MENU

;send condition-type output (16-bit)
mov AX,MENU ;issue control
out 2,AX

;detect condition-type input (16-bit)
in AX,3 ;sense environmental data
mov KEYCHA,AX

;test for equality between KEYCHA and @C02 (16-bit)
mov @T01,1 ;presuppose equality
mov AX,KEYCHA ;fetch KEYCHA
cmo AX,3C02 ;compare arguments
jz $+4 ;send routine if true

branch on false
mov AL,@T01 ;load value into accumulator
cmo AL,0 ;compare to zero
false(=0)
jz @07
;jump to @07 if false(=0)
;assign value of one variable to another variable
(16-bit)
    mov AX, @C01 ;assign @C01
    mov KEYINM, AX ;sto KEYINM

@07:
;define location @07
;jtest for equality between KEYCHA and @C01 (16-bit)
    mov @T01, 1 ;presuppose equality
    mov AX, KEYCHA ;fetch KEYCHA
    cmp AX, @C01 ;compare arguments
    jz $+4 ;end routine if
true
;jbranch on false
mov AL, @T01 ;load value into accumulator

;assign value of one variable to another variable
(16-bit)
    mov AX, @C01 ;assign @C01
    mov SYMANU, AX ;sto SYMANU

@08:
;define location @08
;jtest for equality between KEYCHA and @C03 (16-bit)
    mov @T01, 1 ;presuppose equality
    mov AX, KEYCHA ;fetch KEYCHA
    cmp AX, @C03 ;compare arguments
    jz $+4 ;end routine if
true
;jbranch on false
mov AL, @T01 ;load value into accumulator

;assign value of one variable to another variable
(16-bit)
    mov AX, @C01 ;assign @C01
    mov SMAUTO, AX ;sto SMAUTO

@09:
;define location @09
;jtest for equality between KEYCHA and @C02 (16-bit)
    mov SMAUTO, AX ;sto SMAUTO
assign value of one variable to another variable

;procedure SYMANU
@SMANU:
;jentry point for SYMANU
; detect condition-type input (16-bit)
in AX,4 ; sense environmental data
mov KEYCHA,AX
;
; procedure *MANUAL
@MANUAL: nop ; entry point for
MANUAL

; test if $AC0$ not equal $@AC02$ then $@T01 = 1$ (16-bit)
mov $@T01,1 ; presuppose
inequality
mov AX,$AC0 ; ifetch $AC0
cmp AX,$AC02 ;icompare arguments
jne $S+4 ; send routine if
true
mov $@T01,0 ; equal, $@T01 = 0

; branch on false
mov AL,$@T01 ; load value into accumulator
cmp AL,$@T01 ;icompare to zero
jz @11 ; jump to @11 if false(=0)

; assign value of one variable to another variable (16-bit)
mov AX,$AC02 ;assign $AC02
mov POLL,AX ; to POLL
; send condition-type output (16-bit)
mov AX,POLL ;issue control
out 4,AX

@10: nop ;define location @10
; test if $AC1$ not equal $@AC02$ then $@T01 = 1$ (16-bit)
mov $@T01,1 ; presuppose
inequality
mov AX,$AC1 ; ifetch $AC1
cmp AX,$AC02 ;icompare arguments
jne $S+4 ; send routine if
true
mov $@T01,0 ; equal, $@T01 = 0

; branch on false
mov AL,$@T01 ; load value into accumulator
cmp AL,$@T01 ;icompare to zero
jz @11 ; jump to @11 if false(=0)

; assign value of one variable to another variable (16-bit)
mov AX,$AC01 ;assign $AC01
mov POLL,AX ; to POLL
; send condition-type output (16-bit)
mov AX,POLL ;issue control
out 6,AX

@11: nop ; define location @11
;test if AC2 not equal @C02 then @T01 = 1 (16-bit)
comed @T01,1 ;paresuppose
inequality
mov AX,AC2 ;fetch AC2
com AX,@C02 ;compare arguments
jne $+4 ;send routine if true
true
mov @T01,0 ;equal, @T01 = 0
branch on false
mov AL, @T01 ;load value into accumulator
com AL,0 ;compare to zero
jz @12 ;jump to @12 if false(=0)
;assign value of one variable to another variable (16-bit)
mov AX, @C03 ;assign @C03
mov POLL, AX ;to POLL
;send condition-type output (16-bit)
mov AX, POLL ;issue control
out 8, AX
@12: nop ;define location @12
;test if AC3 not equal @C02 then @T01 = 1 (16-bit)
mov @T01,1 ;paresuppose
inequality
mov AX,AC3 ;fetch AC3
com AX,@C02 ;compare arguments
jne $+4 ;send routine if true
true
mov @T01,0 ;equal, @T01 = 0
branch on false
mov AL, @T01 ;load value into accumulator
com AL,0 ;compare to zero
jz @13 ;jump to @13 if false(=0)
;assign value of one variable to another variable (16-bit)
mov AX, @C04 ;assign @C04
mov POLL, AX ;to POLL
;send condition-type output (16-bit)
mov AX, POLL ;issue control
out 10, AX
@13: nop ;define location @13
;test if AC4 not equal @C02 then @T01 = 1 (16-bit)
mov @T01,1 ;paresuppose
inequality
mov AX,AC4 ;fetch AC4
com AX,@C02 ;compare arguments
jne $+4 ;send routine if true

92
;branch on false
    mov @T01, 0  ;equal, @T01 = 0
    mov AL, @T01  ;load value into accumulator
    cmp AL, 0  ;compare to zero
    jz @14  ;jump to @14 if false(=0)

;assign value of one variable to another variable
    (16-bit)

    mov AX, @C05  ;assign @C05
    mov POLL, AX  ;to POLL
    ;send condition-type output (16-bit)
    mov AX, POLL  ;issue control
    out 12, AX

@14:
    mov  ;define location @14
    test if AC5 not equal @C02 then @T01 = 1 (16-bit)
    mov @T01, 1  ;presuppose
    inequality
    mov AX, AC5  ;fetch AC5
    cmp AX, @C02  ;compare arguments
    jne $+4  ;send routine if true
    true
    mov @T01, 0  ;equal, @T01 = 0
    jz @15  ;jump to @15 if false(=0)

;branch on false
    mov AL, @T01  ;load value into accumulator
    cmp AL, 0  ;compare to zero
    jz @15  ;jump to @15 if false(=0)

;assign value of one variable to another variable
    (16-bit)

    mov AX, @C06  ;assign @C06
    mov POLL, AX  ;to POLL
    ;send condition-type output (16-bit)
    mov AX, POLL  ;issue control
    out 14, AX

@15:
    mov  ;define location @15
    test if AC6 not equal @C02 then @T01 = 1 (16-bit)
    mov @T01, 1  ;presuppose
    inequality
    mov AX, AC6  ;fetch AC6
    cmp AX, @C02  ;compare arguments
    jne $+4  ;send routine if true
    true
    mov @T01, 0  ;equal, @T01 = 0
    jz @16  ;jump to @16 if false(=0)

;branch on false
assign value of one variable to another variable (16-bit)

mov AX,AC07 ;assign AC07
mov POLL,AX ;to POLL

;send condition-type output (16-bit)
mov AX,POLL ;issue control
out 16,AX

@16:

nop ;define location @16

;test if AC7 not equal AC02 then @T01 = 1 (16-bit)
mov @T01,1 ;presuppose

inequality

mov AX,AC7 ;fetch AC7
cmc AX,AC02 ;compare arguments
jne $+4 ;send routine if true

false(=0)

;assign value of one variable to another variable (16-bit)

mov AX,AC08 ;assign AC08
mov POLL,AX ;to POLL

;send condition-type output (16-bit)
mov AX,POLL ;issue control
out 18,AX

@17:

nop ;define location @17

;test if AC8 not equal AC02 then @T01 = 1 (16-bit)
mov @T01,1 ;presuppose

inequality

mov AX,AC8 ;fetch AC8
cmc AX,AC02 ;compare arguments
jne $+4 ;send routine if true

false(=0)

;assign value of one variable to another variable (16-bit)

mov AX,AC09 ;assign AC09
mov POLL,AX ;to POLL

;send condition-type output (16-bit)
mov AX,POLL ;issue control
out 20,AX
@18:  nop                      :define location @18
itest if AC9 not equal @C02 then @T01 = 1 (16-bit)
    mov @T01, 1              :presuppose
inequality
    mov AX, AC9               :fetch AC9
    cmp AX, @C02              :compare arguments
    jne $+4                   :send routine if
true
    mov @T01, 0               :equal, @T01 = 0
:branch on false
    mov AL, @T01              :load value into
accumulator
    cmp AL, 0                 :compare to zero
    jz @19                    :jump to @19 if
false(=0)
    ;assign value of one variable to another variable
    (16-bit)
    mov AX, @C10              :assign @C10
    mov POLL, AX              :to POLL
send condition-type output (16-bit)
    mov AX, POLL              :issue control
    out 22, AX                :send

@19:  nop                      :define location @19
    ;assign value of one variable to another variable
    (16-bit)
    mov AX, @C02              :assign @C02
    mov SMMANU, AX            :to SMMANU
    ; procedure SMAUTO
@SMAUTO:  nop                   :entry point for
SMAUTO
    ;detect condition-type input (16-bit)
    in AX, 5                  :sense environmental data
    mov KEYCHA, AX
    ; procedure AUTO
@AUTO:  nop                   :entry point for
AUTO
    ;assign value of one variable to another variable
    (16-bit)
    mov AX, @C03              :assign @C03
    mov MENU, AX              :to MENU
send condition-type output (16-bit)
    mov AX, MENU              :issue control
    out 24, AX                :send
detect condition-type input (16-bit)
    in AX, 6                  :sense environmental data
    mov KEYCHA, AX
; test for equality between KEYCHA and @C02 (16-bit)
    mov @T01, 1               :presuppose equality
    mov AX, KEYCHA            :fetch KEYCHA

95
true
mov @T01,0 ;not equal, @T01 = 0
branch on false
mov AL, @T01 ;load value into accumulator

cmp AL, 0 ;compare to zero
jz @20 ;jump to @20 if

false(=0)
;assign value of one variable to another variable
(16-bit)

mov AX, @C01 ;assign @C01
mov KEYINM, AX ;to KEYINM

@20: mov @O :define location @20

;test for equality between KEYCHA and @C01 (16-bit)
mov @T01, @1 ;presuppose equality
mov AX, KEYCHA ;fetch KEYCHA
cmp AX, @C01 ;compare arguments
jz @20 ;jump to @20 if
true
mov @T01,0 ;not equal, @T01 = 0
branch on false
mov AL, @T01 ;load value into accumulator

cmp AL, 0 ;compare to zero
jz @21 ;jump to @21 if

false(=0)
;assign value of one variable to another variable
(16-bit)

mov AX, @C11 ;assign @C11
mov INTER, AX ;to INTER

@21: mov @O :define location @21

;test for equality between KEYCHA and @C03 (16-bit)
mov @T01, @1 ;presuppose equality
mov AX, KEYCHA ;fetch KEYCHA
cmp AX, @C03 ;compare arguments
jz @21 ;jump to @21 if
true
mov @T01,0 ;not equal, @T01 = 0
branch on false
mov AL, @T01 ;load value into accumulator

cmp AL, 0 ;compare to zero
jz @22 ;jump to @22 if

false(=0)
;assign value of one variable to another variable
(16-bit)

mov AX, @C12 ;assign @C12
mov INTER, AX ;to INTER

@22: mov @O :define location @22
;test for equality between KEYCHA and @C04 (16-bit)

mov @T01,1 ;premise equality
mov AX,KEYCHA ;fetch KEYCHA
cmp AX,@C04 ;compare arguments
jz $+4 ;end routine if

true

mov @T01,0 ;not equal, @T01 = 0

for branch on false

mov AL,@T01 ;load value into accumulator

false(=0)

;assign value of one variable to another variable
(16-bit)

mov AX,@C13 ;assign @C13
mov INTPER,AX ;to INTPER

@23:

nop ;define location @23

;test for equality between KEYCHA and @C05 (16-bit)

mov @T01,1 ;premise equality
mov AX,KEYCHA ;fetch KEYCHA
cmp AX,@C05 ;compare arguments
jz $+4 ;end routine if

true

mov @T01,0 ;not equal, @T01 = 0

for branch on false

mov AL,@T01 ;load value into accumulator

false(=0)

;assign value of one variable to another variable
(16-bit)

mov AX,@C14 ;assign @C14
mov INTPER,AX ;to INTPER

@24:

nop ;define location @24

;test for equality between KEYCHA and @C06 (16-bit)

mov @T01,1 ;premise equality
mov AX,KEYCHA ;fetch KEYCHA
cmp AX,@C06 ;compare arguments
jz $+4 ;end routine if

true

mov @T01,0 ;not equal, @T01 = 0

for branch on false

mov AL,@T01 ;load value into accumulator

false(=0)

;assign value of one variable to another variable
(16-bit)

97
nov AX, @C06 ;assign @C06
nov INTPER, AX ;to INTPER
a25: "nop ;define location a25
;test for equality between KEYCHA and @C07 (16-bit)
nov @T01, 0 ;test for equality
nov AX, KEYCHA ;if fetch KEYCHA
cmo AX, @C07 ;compare arguments
jz @+4 ;send routine if true
;branch on false
nov AL, @T01 ;load value into accumulator
cmo AL, 0 ;compare to zero
jz @26 ;jump to @26 if false(=0)
;assign value of one variable to another variable (16-bit)
nov AX, @C01 ;assign @C01
nov INTPER, AX ;to INTPER
a26: "nop ;define location a26
;assign value of one variable to another variable (16-bit)
nov AX, CLOCK ;assign CLOCK
nov INTIM, AX ;to INTIM
;assign value of one variable to another variable (16-bit)
nov AX, @C02 ;assign @C02
nov SMAUTO, AX ;to SMAUTO
;assign value of one variable to another variable (16-bit)
nov AX, @C01 ;assign @C01
nov TPOLL, AX ;to TPOLL
; ;procedure TPOLL
@TPOLL: ;entry point for TPOLL
TPOLL ;subtract 16-bit CLOCK - INTIM = @T01
nov AX, CLOCK ;fetch subtrahend
sub AX, INTIM ;fetch and subtract
minuend
nov @T01, AX ;store answer in @T01
;test if @T01 > INTPER then @T01 = 1
(16-bit)
nov @T01, 0 ;test if @T01 > INTPER then @T01 = 1
arq2
nov AX, @T01 ;if fetch @T01
cmo AX, INTPER ;compare arguments
jz @+4 ;send routine if true
;branch on false
nov @T01, 0 ;not > , @T01 = 0
; branch on false
mov AL, @T01 ; load value into accumulator

; assign value of one variable to another variable
false(=0)

mov AL, O ; compare to zero
jz @27 ; jump to @27 if

false(=0)

; assign value of one variable to another variable
(16-bit)
mov AX, @C01 ; assign @C01
mov TPOLL, AX ; to TPOLL

@27: mov ; define location @27

; procedure POLLAU
; entry point for POLLAU:

POLLAU:
test if AC0 not equal @C02 then @T01 = 1 (16-bit)
mov @T01, 1 ; presuppose

inequality
mov AX, AC0 ; fetch AC0
cmo AX, @C02 ; compare arguments
jne $+4 ; send routine if

true
mov @T01, 0 ; equal, @T01 = 0

; branch on false
mov AL, @T01 ; load value into accumulator

false(=0)

; assign value of one variable to another variable
(16-bit)
mov AX, @C02 ; assign @C02
mov POLL, AX ; to POLL

; send condition-type output (16-bit)
mov AX, POLL ; issue control
out 26, AX

@28: mov ; define location @28

; test if AC1 not equal @C02 then @T01 = 1 (16-bit)
mov @T01, 1 ; presuppose

inequality
mov AX, AC1 ; fetch AC1
cmo AX, @C02 ; compare arguments
jne $+4 ; send routine if

true
mov @T01, 0 ; equal, @T01 = 0

; branch on false
mov AL, @T01 ; load value into accumulator

false(=0)

jz @29 ; jump to @29 if
assign value of one variable to another variable

(16-bit)

mov AX, @C01 ; assign @C01
mov POLL, AX ; to POLL

; send condition-type output (16-bit)

mov AX, POLL ; issue control
out 28, AX

@29:  ; define location @29

; test if AC2 not equal @C02 then @T01 = 1 (16-bit)

mov @T01, 1 ; presuppose

inequality

mov AX, AC2 ; fetch AC2
cmp AX, @C02 ; compare arguments
jne $+4 ; send routine if true

true

mov @T01, 0 ; equal, @T01 = 0

branch on false

mov AL, @T01 ; load value into accumulator

false(=0)

; assign value of one variable to another variable

(16-bit)

mov AX, @C03 ; assign @C03
mov POLL, AX ; to POLL

; send condition-type output (16-bit)

mov AX, POLL ; issue control
out 30, AX

@30:  ; define location @30

; test if AC3 not equal @C02 then @T01 = 1 (16-bit)

mov @T01, 1 ; presuppose

inequality

mov AX, AC3 ; fetch AC3
cmp AX, @C02 ; compare arguments
jne $+4 ; send routine if true

true

mov @T01, 0 ; equal, @T01 = 0

branch on false

mov AL, @T01 ; load value into accumulator

false(=0)

; assign value of one variable to another variable

(16-bit)

mov AX, @C04 ; assign @C04
mov POLL, AX ; to POLL

; send condition-type output (16-bit)

mov AX, POLL ; issue control
out 32, AX
&31: mov @T01,1 ;define location &31

; test if AC4 not equal AC02 then &T01 = 1 (16-bit)

inequality

mov AX,AC4

; fetched AC4

cmp AX,AC02

; compare arguments

jne &+4 ; send routine if true

true

mov @T01,0

; equal, &T01 = 0

branch on false

mov AL,&T01

; load value into accumulator

false(=0)

; assign value of one variable to another variable

(jz &32 ; compare to zero  

sub routine if false)

&32:

mov AX,AC05

; assign AC05

mov POLL,AX

; to POLL

; send condition-type output (16-bit)

mov AX,POLL

; issue control

out &+4,AX

&33:

mov @T01,1

; define location &33

; test if AC5 not equal AC02 then &T01 = 1 (16-bit)

inequality

mov AX,AC5

; fetched AC5

cmp AX,AC02

; compare arguments

jne &+4 ; send routine if true

true

mov @T01,0

; equal, &T01 = 0

branch on false

mov AL,&T01

; load value into accumulator

false(=0)

; assign value of one variable to another variable

(jz &33 ; compare to zero  

sub routine if false)
true

;branch on false
mov AL, @T01, 0
mov AL, @T01
;load value into accumulator
cmo AL, 0
;jz @34
;jump to @34 if false(=0)

false(=0)
;assign value of one variable to another variable (16-bit)
mov AX, @C07
mov POLL, AX
;send condition-type output (16-bit)
mov AX, POLL
;issue control
out 3A, AX

@34:
 nop
 ;define location @34

;test if AC7 not equal @C02 then @T01 = 1 (16-bit)
mov @T01, 1
;presuppose inequality

mov AX, AC7
cmo AX, 3C02
jne $+4
;send routine if true

true

;branch on false
mov @T01, 0
mov AL, @T01
;load value into accumulator
cmo AL, 0
;jz @35
;jump to @35 if false(=0)

false(=0)
;assign value of one variable to another variable (16-bit)
mov AX, @C08
mov POLL, AX
;send condition-type output (16-bit)
mov AX, POLL
;issue control
out 40, AX

@35:
 nop
 ;define location @35

;test if AC8 not equal @C02 then @T01 = 1 (16-bit)
mov @T01, 1
;presuppose inequality

mov AX, AC8
cmo AX, 3C02
jne $+4
;send routine if true

true

;branch on false
mov @T01, 0
mov AL, @T01
;load value into accumulator
cmo AL, 0
;jz @36
;jump to @36 if
false(=0)
  ;assign value of one variable to another variable
  (16-bit)
  mov AX,AC09 ;assign AC09
  mov POLL,AX ;to POLL
  ;send condition-type output (16-bit)
  mov AX,POLL ;issue control
  out 42,AX
  @36:  nop ;define location @36
  ;test if AC9 not equal AC02 then AT01 = 1 (16-bit)
  mov AT01,1 ;presuppose
  inequality
  mov AX,AC9 ;fetch AC9
  cmp AX,AC02 ;compare arguments
  jne $+4 ;send routine if
  true
  mov AT01,0 ;equal, AT01 = 0
  ;branch on false
  mov AL,AT01 ;load value into accumulator
  cmp AL,0 ;compare to zero
  jz @37 ;jump to @37 if
false(=0)
  ;assign value of one variable to another variable
  (16-bit)
  mov AX,AC10 ;assign AC10
  mov POLL,AX ;to POLL
  ;send condition-type output (16-bit)
  mov AX,POLL ;issue control
  out 44,AX
  @37:  nop ;define location @37
  ;assign value of one variable to another variable
  (16-bit)
  mov AX,AC02 ;assign AC02
  mov TPOll,AX ;to TPOll
  ;procedure MSGDS
  @MSGDS:  nop ;entry point for
MSGDS
  ;detect condition-type inout (16-bit)
  in AX,7 ;sense environmental data
  mov KEYCHA,AX
  ;procedure MSGDSP
  @MSGDSP:  nop ;entry point for
MSGDSP
  ;assign value of one variable to another variable
  (16-bit)
  mov AX,MSG0 ;assign MSG0
  mov MSGVDT,AX ;to MSGVDT
  ;send condition-type output (16-bit)
mov AX, MSGVDT ; issue control
out 4b, AX

; assign value of one variable to another variable (16-bit)
mov AX, MSG1 ; assign MSG1
mov MSGVDT, AX ; to MSGVDT

; send condition-type output (16-bit)
mov AX, MSGVDT ; issue control
out 4b, AX

mov AX, MSG2 ; assign MSG2
mov MSGVDT, AX ; to MSGVDT

; send condition-type output (16-bit)
mov AX, MSGVDT ; issue control
out 50, AX

mov AX, MSG3 ; assign MSG3
mov MSGVDT, AX ; to MSGVDT

; send condition-type output (16-bit)
mov AX, MSGVDT ; issue control
out 52, AX

mov AX, MSG4 ; assign MSG4
mov MSGVDT, AX ; to MSGVDT

; send condition-type output (16-bit)
mov AX, MSGVDT ; issue control
out 54, AX

mov AX, MSG5 ; assign MSG5
mov MSGVDT, AX ; to MSGVDT

; send condition-type output (16-bit)
mov AX, MSGVDT ; issue control
out 56, AX

mov AX, MSG6 ; assign MSG6
mov MSGVDT, AX ; to MSGVDT

; send condition-type output (16-bit)
mov AX, MSGVDT ; issue control
out 58, AX

mov AX, MSG7 ; assign MSG7
mov MSGVDT, AX ; to MSGVDT

; send condition-type output (16-bit)
mov AX, MSGVDT ; issue control
out 60, AX

104
; assign value of one variable to another variable
(16-bit)
omv AX, MSG8 ; assign MSG8
omv MSGVDT, AX ; to MSGVDT
; send condition-type output (16-bit)
omv AX, MSGVDT ; issue control
out 62, AX
; assign value of one variable to another variable
(16-bit)
omv AX, MSG9 ; assign MSG9
omv MSGVDT, AX ; to MSGVDT
; send condition-type output (16-bit)
omv AX, MSGVDT ; issue control
out 64, AX
; assign value of one variable to another variable
(16-bit)
omv AX, @C02 ; assign @C02
omv MSGGDS, AX ; to MSGGDS
:
; procedure MLOCAT
@MLOCAT: nno ; entry point for
MLOCAT
; detect condition-type input (16-bit)
in AX, 8 ; sense environmental data
omv KEYCHA, AX
;
; procedure LOCATI
@LOCATI: nno ; entry point for
LOCATI
; assign value of one variable to another variable
(16-bit)
omv AX, @C04 ; assign @C04
omv MENU, AX ; to MENU
; send condition-type output (16-bit)
omv AX, MENU ; issue control
out 66, AX
; detect condition-type input (16-bit)
in AX, 9 ; sense environmental data
omv KEYCHA, AX
; test for equality between KEYCHA and @C02 (16-bit)
omv @T01, 1 ; presuppose equality
omv AX, KEYCHA ; fetch KEYCHA
cmp AX, @C02 ; compare arguments
jz 8+4 ; send routine if true
;
; branch on false
omv AL, @T01 ; load value into accumulator
cmp AL, 0 ; compare to zero
jz @38 ; jump to @38 if
false(=0)  
;assign value of one variable to another variable
(16-bit)
mov AX, @C01  ;assign @C01
mov KEYINH, AX  ;to KEYINH
@38:  noo  ;define location @38
;test for equality between KEYCHA and @C01 (16-bit)
mov @T01, 1  ;oresuppose equality
mov AX, KEYCHA  ;fetch KEYCHA
cmo AX, @C01  ;compare arguments
jz $+4  ;end routine if
true
;branch on false
mov AL, @T01  ;load value into accumulator
cmo AL, 0  ;compare to zero
jz @39  ;jum to @39 if false(=0)
false(=0)  
;assign value of one variable to another variable
(16-bit)
mov AX, @C02  ;assign @C02
mov NEWPOS, AX  ;to NEWPOS
@39:  noo  ;define location @39
;test for equality between KEYCHA and @C03 (16-bit)
mov @T01, 1  ;oresuppose equality
mov AX, KEYCHA  ;fetch KEYCHA
cmo AX, @C03  ;compare arguments
jz $+4  ;end routine if
true
;branch on false
mov AL, @T01  ;load value into accumulator
cmo AL, 0  ;compare to zero
jz @40  ;jum to @40 if false(=0)
false(=0)  
;assign value of one variable to another variable
(16-bit)
mov AX, @C02  ;assign @C02
mov NEWPOS, AX  ;to NEWPOS
;assign value of one variable to another variable
(16-bit)
mov AX, @C01  ;assign @C01
mov TMLOCA, AX  ;to TMLOCA
@40:  noo  ;define location @40
;assign value of one variable to another variable
(16-bit)
mov AX, @C02  ;assign @C02
mov MLOCAT, AX  ;to MLOCAT
; procedure TMLOCA
@TMLOCA:  mov KEYCHA,AX
            ; entry point for TMLOCA
            ; detect condition-type input (16-bit)
            ; in AX,10  ; sense environmental data
            mov MANPOS,AX
            mov POSITI,AX  ; to POSITI
            ; assign value of one variable to another variable (16-bit)
            mov AX,MANPOS  ; assign MANPOS
            mov AX,MANPOS
            ; assign value of one variable to another variable (16-bit)
            mov AX,AC02   ; assign AC02
            mov TMLOCA,AX  ; to TMLOCA
            ; procedure MANLOC
@MANLOC:   ; entry point for MANLOC
            ; detect condition-type input (16-bit)
            ; in AX,11  ; sense environmental data
            mov MANPOS,AX
            mov POSITI,AX  ; to POSITI
            ; assign value of one variable to another variable (16-bit)
            mov AX,MANPOS  ; assign MANPOS
            mov AX,MANPOS
            ; assign value of one variable to another variable (16-bit)
            mov AX,AC02   ; assign AC02
            mov TMLOCA,AX  ; to TMLOCA
            ; procedure MCLOCK
@MCLOCK:   ; entry point for MCLOCK
            ; detect condition-type input (16-bit)
            ; in AX,12  ; sense environmental data
            mov KEYCHA,AX
            ; procedure CLOCKS
@CLOCKS:   ; entry point for CLOCKS
            ; assign value of one variable to another variable (16-bit)
            mov AX,AC05   ; assign AC05
            mov MENU,AX   ; to MENU
            ; send condition-type output (16-bit)
            mov AX,MENU   ; issue control
            out 68,AX
            ; detect condition-type input (16-bit)
            ; in AX,13  ; sense environmental data
            mov KEYCHA,AX
            ; assign value of one variable to another variable (16-bit)
            mov AX,AC02   ; assign AC02
            mov MCLOCK,AX  ; to MCLOCK
            ; procedure MLOGIN
@MLOGIN:   ; entry point for MLOGIN
            ; detect condition-type input (16-bit)
; procedure LOGINO
@LOGINO: mov ; entry point for

LOGINO

; assign value of one variable to another variable
(16-bit)

mov AX,AC06 ; assign AC06
mov MENU,AX ; to MENU

; send condition-type output (16-bit)
mov AX,MENU ; issue control
out 70,AX

; detect condition-type input (16-bit)
in AX,15 ; sense environmental data
mov KEYCHA,AX

; test for equality between KEYCHA and AC02 (16-bit)
mov @T01,1 ; presuppose equality
mov AX,KEYCHA ; fetch KEYCHA
cmp AX,AC02 ; compare arguments
jz $+4 ; send routine if true

mov @T01,0 ; not equal, @T01 = 0

; branch on false
mov AL,@T01 ; load value into accumulator
cmp AL,0 ; compare to zero
jz @41 ; jump to @41 if false(=0)

; assign value of one variable to another variable
(16-bit)

mov AX,AC01 ; assign AC01
mov KEYINM,AX ; to KEYINM

@41: nop ; define location @41

; test for equality between KEYCHA and AC01 (16-bit)
mov @T01,1 ; presuppose equality
mov AX,KEYCHA ; fetch KEYCHA
cmp AX,AC01 ; compare arguments
jz $+4 ; send routine if true

mov @T01,0 ; not equal, @T01 = 0

; branch on false
mov AL,@T01 ; load value into accumulator
cmp AL,0 ; compare to zero
jz @42 ; jump to @42 if false(=0)

; assign value of one variable to another variable
(16-bit)

mov AX,AC01 ; assign AC01
mov TLOGIN,AX ; to TLOGIN

108
@42:   nop  ;define location @42
       ;test for equality between KEYCHA and @C03 (16-bit)
       mov   @T01,1  ;presuppose equality
       mov   AX,KEYCHA  ;fetch KEYCHA
       cmp   AX,@C03  ;compare arguments
       jz    $+4  ;end routine if true

true
       mov   @T01,0  ;not equal, @T01 = 0
       branch on false
       mov   AL,@T01  ;load value into accumulator
       cmp   AL,0  ;compare to zero
       jz    @43  ;jump to @43 if false(=0)

false(=0)
       ;assign value of one variable to another variable
       (16-bit)
       mov   AX,@C01  ;assign @C01
       mov   TLQGIN,AX  ;to TLQGIN

@43:   nop  ;define location @43
       ;assign value of one variable to another variable
       (16-bit)
       mov   AX,@C02  ;assign @C02
       mov   MLOGIN,AX  ;to MLOGIN

;procedure TLQGIN
@TLQGIN:   nop  ;entry point for TLQGIN

;detect condition-type input (16-bit)
   in   AX,16  ;sense environmental data
       mov   KEYCHA,AX

;procedure LOGIN
@LOGIN:   nop  ;entry point for LOGIN

;assign value of one variable to another variable
       (16-bit)
       mov   AX,@C02  ;assign @C02
       mov   ACNUM,AX  ;to ACNUM

;detect condition-type input (16-bit)
   in   AX,17  ;sense environmental data
       mov   ACNUM,AX

;test for equality between NEXTAC and @C02 (16-bit)
       mov   @T01,1  ;presuppose equality
       mov   AX,NEXTAC  ;fetch NEXTAC
       cmp   AX,@C02  ;compare arguments
       jz    $+4  ;end routine if true

true
       mov   @T01,0  ;not equal, @T01 = 0

;test for equality between AC0 and @C02 (16-bit)
       mov   @T02,1  ;presuppose equality
       mov   AX,AC0  ;fetch AC0

109
; compare arguments
jz $+4 ; end routine if

true

mov @T02,0 ; not equal, @T02 = 0

; logical and. (16-bit) @T01 .and. @T02 = @T01
mov AX, @T01
and AX, @T02
mov @T01, AX

; branch on false
mov AL, @T01 ; load value into accumulator

false(=0)

; assign value of one variable to another variable
(16-bit)

mov AX, ACNU4 ; assign ACNU4
mov AC0, AX ; to AC0

@44:

mov : define location @44

; test for equality between NEXTAC and AC01 (16-bit)
mov @T01,1 ; store true equality
mov AX, NEXTAC ; ifetch NEXTAC
mov AX, @C01 ; compare arguments
jz $+4 ; end routine if

true

mov @T01,0 ; not equal, @T01 = 0

; test for equality between AC1 and @C02 (16-bit)
mov @T02,1 ; store true equality
mov AX, AC1 ; ifetch AC1
mov AX, @C02 ; compare arguments
jz $+4 ; end routine if

true

mov @T02,0 ; not equal, @T02 = 0

; logical and. (16-bit) @T01 .and. @T02 = @T01
mov AX, @T01
and AX, @T02
mov @T01, AX

; branch on false
mov AL, @T01 ; load value into accumulator

false(=0)

; assign value of one variable to another variable
(16-bit)

mov AX, ACNU4 ; assign ACNU4
mov AC1, AX ; to AC1

@45:

mov : define location @45

; test for equality between NEXTAC and AC03 (16-bit)
mov @T01,1 ; store true equality
mov AX, NEXTAC ; ifetch NEXTAC
cma AX, @C03 ; compare arguments
jz $+4 ; end routine if

true
mov @T01, 0 ; not equal, @T01 = 0

; test for equality between AC2 and @C02 (16-bit)
mov @T02, 1 ; presuppose equality
mov AX, AC2 ; fetch AC2
cma AX, @C02 ; compare arguments
jz $+4 ; end routine if

true
mov @T02, 0 ; not equal, @T02 = 0

; logical and, (16-bit) @T01, and @T02 = @T01
mov AX, @T01
and AX, @T02
mov @T01, AX

; branch on false
mov AL, @T01 ; load value into accumulator

false(=0)

; assign value of one variable to another variable
(16-bit)
mov AX, ACNUM ; assign ACNUM
mov AC2, AX ; to AC2

$46:
mov ; define location $46

; test for equality between NEXTAC and @C04 (16-bit)
mov @T01, 1 ; presuppose equality
mov AX, NEXTAC ; fetch NEXTAC
cma AX, @C04 ; compare arguments
jz $+4 ; end routine if

true
mov @T01, 0 ; not equal, @T01 = 0

; test for equality between AC3 and @C02 (16-bit)
mov @T02, 1 ; presuppose equality
mov AX, AC3 ; fetch AC3
cma AX, @C02 ; compare arguments
jz $+4 ; end routine if

true
mov @T02, 0 ; not equal, @T02 = 0

; logical and, (16-bit) @T01, and @T02 = @T01
mov AX, @T01
and AX, @T02
mov @T01, AX

; branch on false
mov AL, @T01 ; load value into accumulator

false(=0)

; assign value of one variable to another variable

111
(16-bit)

mov AX,ACNUM ;assign ACNUM
mov AC3,AX ;to AC3

@47: nnp ;define location @47

jtest for equality between NEXTAC and AC05 (16-bit)

mov ATO1,1 ;presuppose equality
mov AX,NEXTAC ;ifetch NEXTAC
cmp AX,AC05 ;icmpare arguments
jz $+4 ;send routine if

true

mov ATO1,0 ;not equal, ATO1 = 0

jtest for equality between AC4 and AC02 (16-bit)

mov ATO2,1 ;presuppose equality
mov AX,AC4 ;ifetch AC4
cmp AX,AC02 ;icmpare arguments
jz $+4 ;send routine if

true

mov ATO2,0 ;not equal, ATO2 = 0

jlogical and. (16-bit) ATO1 .and. ATO2 = ATO1

mov AX,ATO1
and AX,ATO2
mov ATO1,AX

jbranch on false

mov AL,ATO1 ;load value into accumulator

cmp AL,0 ;icmpare to zero
jz $+8 ;jump to @48 if

false (=0)

;assign value of one variable to another variable
(16-bit)

mov AX,ACNUM ;assign ACNUM
mov AC4,AX ;to AC4

@48: nnp ;define location @48

jtest for equality between NEXTAC and AC06 (16-bit)

mov ATO1,1 ;presuppose equality
mov AX,NEXTAC ;ifetch NEXTAC
cmp AX,AC06 ;icmpare arguments
jz $+4 ;send routine if

true

mov ATO1,0 ;not equal, ATO1 = 0

jtest for equality between AC5 and AC02 (16-bit)

mov ATO2,1 ;presuppose equality
mov AX,AC5 ;ifetch AC5
cmp AX,AC02 ;icmpare arguments
jz $+4 ;send routine if

true

mov ATO2,0 ;not equal, ATO2 = 0

jlogical and. (16-bit) ATO1 .and. ATO2 = ATO1

mov AX,ATO1
and AX,ATO2
mov ATO1,AX

112
; branch on false
mov AL, @T01

; load value into accumulator
mov AL, 0
jz @49

; false(=0)
; assign value of one variable to another variable
(16-bit)

mov AX, ACNUM
mov AC5, AX
@49:

; define location @49

; test for equality between NEXTAC and @C07 (16-bit)
mov @T01, 1
mov AX, NEXTAC
mov AX, @C07
jz $+4

; end routine if true

true
mov @T01, 0

; not equal, @T01 = 0

; test for equality between AC6 and @C02 (16-bit)
mov @T02, 1
mov AX, AC6
mov AX, @C02
jz $+4

; end routine if true

true
mov @T02, 0

; not equal, @T02 = 0

; logical and, (16-bit) @T01 and @T02 = @T01
mov AX, @T01
and AX, @T02
mov @T01, AX

; branch on false
mov AL, @T01

; load value into accumulator
mov AL, 0
jz @50

; false(=0)
; assign value of one variable to another variable
(16-bit)

mov AX, ACNUM
mov AC6, AX
@50:

; define location @50

; test for equality between NEXTAC and @C08 (16-bit)
mov @T01, 1
mov AX, NEXTAC
mov AX, @C08
jz $+4

; end routine if true

true
mov @T01, 0

; not equal, @T01 = 0

; test for equality between AC7 and @C02 (16-bit)
mov @T02, 1
mov AX, AC7
mov AX, @C02
mov AX, @C07
mov AX, @C02
; compare arguments
jz $+4

; end routine if true

true
true
mov @T02, 0
; not equal, @T02 = 0
; logical and (16-bit) @T01 . and @T02 = @T01
mov AX, @T01
and AX, @T02
mov @T01, AX
branch on false
mov AL, @T01
; load value into accumulator

false(=0)
; assign value of one variable to another variable
(16-bit)
mov AX, ACNUM
; assign ACNUM
mov AC7, AX
; to AC7
@51:
mov
; define location @51
jz @52
; test for equality between NEXTAC and @C09 (16-bit)
mov @T01, 1
; presuppose equality
mov AX, NEXTAC
; fetch NEXTAC
cmp AX, @C09
; compare arguments
jz @54
; send routine if
true
mov @T01, 0
; not equal, @T01 = 0
; test for equality between ACR and @C02 (16-bit)
mov @T02, 1
; presuppose equality
mov AX, ACR
; fetch ACR
cmp AX, @C02
; compare arguments
jz @54
; send routine if
true
mov @T02, 0
; not equal, @T02 = 0
; logical and (16-bit) @T01 . and @T02 = @T01
mov AX, @T01
and AX, @T02
mov @T01, AX
branch on false
mov AL, @T01
; load value into accumulator

false(=0)
; assign value of one variable to another variable
(16-bit)
mov AX, ACNUM
; assign ACNUM
mov AC8, AX
; to AC8
@52:
mov
; define location @52
jz @52
; test for equality between NEXTAC and @C10 (16-bit)
mov @T01, 1
; presuppose equality
mov AX, NEXTAC
; fetch NEXTAC
cmp AX, @C10
; compare arguments
true

mov @T01,0  ;not equal, @T01 = 0

;test for equality between AC9 and @C02 (16-bit)
mov @T02,1  ;suppose equality
mov AX,AC9   ;fetch AC9
cmp AX,@C02  ;compare arguments
jz $+4       ;send routine if

true

mov @T02,0  ;not equal, @T02 = 0

;logical and, (16-bit) @T01 .and. @T02 = @T01
mov AX,@T01
and AX,@T02
mov @T01,AX

;branch on false
mov AL,@T01  ;load value into accumulator
cmp AL,0     ;compare to zero
jz @53       ;jump to @53 if

false(=0)

;assign value of one variable to another variable
(16-bit)

mov AX,ACNUM  ;assign ACNUM
mov AC9,AX    ;to AC9

@53:

;add 16-bit NEXTAC + @C01 = @T01
mov AX,NEXTAC ;fetch first argument
add AX,@C01   ;add second argument
to first
mov @T01,AX   ;store answer in @T01

false(=0)

;assign value of one variable to another variable
(16-bit)

mov AX,@T01   ;assign @T01
mov NEXTAC,AX ;to NEXTAC

;test for equality between NEXTAC and @C14 (16-bit)
mov @T01,1    ;suppose equality
mov AX,NEXTAC ;fetch NEXTAC
cmp AX,@C14   ;compare arguments
jz $+4        ;send routine if

true

mov @T01,0   ;not equal, @T01 = 0

;branch on false
mov AL,@T01   ;load value into accumulator
cmp AL,0     ;compare to zero
jz @54       ;jump to @54 if

false(=0)

;assign value of one variable to another variable
(16-bit)

mov AX,@C02  ;assign @C02

115
;assign value of one variable to another variable (16-bit)

; procedure TL3GOU
TL3GOU: mov AX,4C02 ; assign AC02
mov TLOGIN,AX ; to TLOGIN

;procedure LOGOUT
LOGOUT: mov AX,ACNUM,AX ; to ACNUM
mov ACNUM,AX ; detect condition-type input (16-bit)
in AX,19 ; sense environmental data
mov KEYCHA,AX

; assign value of one variable to another variable (16-bit)

mov AX,AC02 ; assign AC02
mov AC0,AX ; to AC0

; define location @55

; test for equality between AC0 and ACNUM (16-bit)

mov @T01,1 ; presuppose equality
mov AX,AC0 ; fetch AC0
cmp AX,ACNUM ; compare arguments
jz $+4 ; end routine if true

mov @T01,0 ; not equal, @T01 = 0
jz @55 ; jump to @55 if false(=0)

; assign value of one variable to another variable (16-bit)

mov AX,AC02 ; assign AC02
mov AC0,AX ; to AC0

@55: mov AX,AC1 ; to AC1
mov AC1,AX ; fetch AC1
cmp AX,ACNUM ; compare arguments
jz $+4 ; end routine if true

mov @T01,0 ; not equal, @T01 = 0
jz @55 ; jump to @55 if false(=0)
accumulator

cmp AL, 0; compare to zero

jz @56; jump to @56 if false(=0)

; assign value of one variable to another variable

(16-bit)

mov AX, @C01; assign @C01
mov @C01, AX; to AC0

@56: nob; define location @56

; test for equality between AC2 and ACNUM (16-bit)

mov @T01, 1; foresuppose equality
mov AX, AC2; fetch AC2
cmp AX, ACNUM; compare arguments
jz $+4; send routine if true

mov @T01, 0; not equal, @T01 = 0

branch on false

mov AL, @T01; load value into accumulator

false(=0)

; assign value of one variable to another variable

(16-bit)

mov AX, @C03; assign @C03
mov @C03, AX; to AC0

@57: nob; define location @57

; test for equality between AC3 and ACNUM (16-bit)

mov @T01, 1; foresuppose equality
mov AX, AC3; fetch AC3
cmp AX, ACNUM; compare arguments
jz $+4; send routine if true

mov @T01, 0; not equal, @T01 = 0

branch on false

mov AL, @T01; load value into accumulator

false(=0)

; assign value of one variable to another variable

(16-bit)

mov AX, @C04; assign @C04
mov @C04, AX; to AC0

@58: nob; define location @58

; test for equality between AC4 and ACNUM (16-bit)

mov @T01, 1; foresuppose equality
mov AX, AC4; fetch AC4
cmp AX, ACNUM; compare arguments
jz $+4; send routine if true

117
branch on false

mov AL, @T01

; load value into accumulator

cmp AL, 0

; compare to zero

jz @59

; jump to @59 if false(=0)

; assign value of one variable to another variable

(16-bit)

mov AX, @C05

; assign @C05

mov AC0, AX

; to AC0

@59:

nop

; define location @59

; test for equality between AC5 and ACNUM (16-bit)

mov @T01, 1

; presuppose equality

mov AX, AC5

; fetch AC5

cmp AX, ACNUM

; compare arguments

jz $+4

; end routine if true

branch on false

mov AL, @T01

; load value into accumulator

cmp AL, 0

; compare to zero

jz @60

; jump to @60 if false(=0)

; assign value of one variable to another variable

(16-bit)

mov AX, @C06

; assign @C06

mov AC0, AX

; to AC0

@60:

nop

; define location @60

; test for equality between AC6 and ACNUM (16-bit)

mov @T01, 1

; presuppose equality

mov AX, AC6

; fetch AC6

cmp AX, ACNUM

; compare arguments

jz $+4

; end routine if true

branch on false

mov AL, @T01

; load value into accumulator

cmp AL, 0

; compare to zero

jz @61

; jump to @61 if false(=0)

; assign value of one variable to another variable

(16-bit)

mov AX, @C07

; assign @C07

mov AC0, AX

; to AC0

@61:

nop

; define location @61

; test for equality between AC7 and ACNUM (16-bit)

mov @T01, 1

; presuppose equality

mov AX, AC7

; fetch AC7
true
mov @T01,0
:branch on false
mov AL,@T01
accumulator
cmp AL,0
jz @62
:jump to @62 if
false(=0)
:assign value of one variable to another variable
(16-bit)
mov AX,@C08
mov AC0,AX
@62:
:define location @62
:compare equality between AC9 and ACNUM (16-bit)
mov @T01,1
:foresuppose equality
mov AX,AC9
:fetch AC9
cmp AX,ACNUM
jz $+4
:send routine if
true
mov @T01,0
:branch on false
mov AL,@T01
accumulator
cmp AL,0
jz @63
:jump to @63 if
false(=0)
:assign value of one variable to another variable
(16-bit)
mov AX,@C09
mov AC0,AX
@63:
:define location @63
:compare equality between AC9 and ACNUM (16-bit)
mov @T01,1
:foresuppose equality
mov AX,AC9
:fetch AC9
cmp AX,ACNUM
jz $+4
:send routine if
true
mov @T01,0
:branch on false
mov AL,@T01
accumulator
cmp AL,0
jz @64
:jump to @64 if
false(=0)
:assign value of one variable to another variable
(16-bit)
mov AX,@C10
mov AC0,AX
@64:
:define location @64
mov AX,AC10
mov AC10,AC0
; assign value of one variable to another variable
(16-bit)

mov AX, @CO2 ; assign value of one variable to another variable
mov TLOGOU, AX ; to TLOGOU

; procedure POSCH
@POSCH: ; entry point for
POSCH

; detect condition-type input (16-bit)
in AX, 20 ; sense environmental data
mov NEPOS, AX

; test for equality between NEPOS and @CO1 (16-bit)
mov @T01, 1 ; presuppose equality
mov AX, NEPOS ; fetch NEPOS
cmp AX, @CO1 ; compare arguments
jz $+4 ; end routine if true

true
mov @T01, 0 ; not equal, @T01 = 0

; branch on false
mov AL, @T01 ; load value into accumulator
cmp AL, @T01 ; compare to zero
jz @65 ; jump to @65 if false(=0)

false(=0)

; assign value of one variable to another variable
(16-bit)
mov AX, @CO1 ; assign @CO1
mov POSCH, AX ; to POSCH

@65: ; define location @65

; procedure POSUPD
@POSUPD: ; entry point for
POSUPD

; detect condition-type input (16-bit)
in AX, 21 ; sense environmental data
mov POSITI, AX

; assign value of one variable to another variable
(16-bit)
mov AX, @CO2 ; assign value of one variable to another variable
mov POSCH, AX ; to POSCH

; procedure MSGIN
@MSGIN: ; entry point for
MSGIN

; detect condition-type input (16-bit)
in AX, 22 ; sense environmental data
mov MSGREA, AX

; test for equality between MSGREA and @CO1 (16-bit)
mov @T01, 1 ; presuppose equality
mov AX, MSGREA ; fetch MSGREA
cmp AX, @CO1 ; compare arguments
true

jz $+4 ; end routine if

for branch on false

mov @T01, 0 ; not equal, @T01 = 0

accumulator

cmp AL, @T01 ; load value into accumulator

false(=0)

; assign value of one variable to another variable

(16-bit)

mov AX, @AC01 ; assign @AC01
mov MSGIN, AX ; to MSGIN

@66: ; define location @66

procedure MSGSTO

@MSGSTO: ; entry point for MSGSTO

; detect condition-type input (16-bit)

in AX, 23 ; sense environmental data

mov MESSAG, AX

; send condition-type output (16-bit)

mov AX, MSGRCV ; issue control
out 72, AX

; test for equality between NEXTMS and @AC02 (16-bit)

mov @T01, 1 ; recompute equality
mov AX, NEXTMS ; fetch NEXTMS
cmp AX, @AC02 ; compare arguments
jz $+4 ; end routine if

true

for branch on false

mov @T01, 0 ; not equal, @T01 = 0

accumulator

cmp AL, 0 ; compare to zero

false(=0)

; assign value of one variable to another variable

(16-bit)

mov AX, MESSAG ; assign MESSAG
mov @AC0, AX ; to @AC0

@67: ; define location @67

; test for equality between NEXTMS and @AC01 (16-bit)

mov @T01, 1 ; recompute equality
mov AX, NEXTMS ; fetch NEXTMS
cmp AX, @AC01 ; compare arguments
jz $+4 ; end routine if

true

for branch on false

mov AL, @T01 ; load value into
accumulator

false(=0)

; assign value of one variable to another variable
(16-bit)

false(=0)

; assign value of one variable to another variable
(16-bit)

false(=0)

; assign value of one variable to another variable
(16-bit)
false(=0)
  ;assign value of one variable to another variable
  (16-bit)
   mov  AX, MESSAG  ;assign MESSAG
   mov  AC0, AX  ;to AC0
   @71:  mov  AL,0  ;define location @71
   @71:  test for equality between NEXTMS and AC06 (16-bit)
   mov  @T01,1  ;presuppose equality
   mov  AX, NEXTMS  ;fetch NEXTMS
   cmc  AX, AC06  ;compare arguments
   jz  $+4  ;end routine if
true
  ;branch on false
  mov  @T01,0  ;not equal, @T01 = 0
accumulator
   mov  AL,0  ;load value into
   jz  @71  ;jump to @71 if
false(=0)
  ;assign value of one variable to another variable
  (16-bit)
   mov  AX, MESSAG  ;assign MESSAG
   mov  AC5, AX  ;to AC5
   @72:  mov  AL,0  ;define location @72
   @72:  test for equality between NEXTMS and AC07 (16-bit)
   mov  @T01,1  ;presuppose equality
   mov  AX, NEXTMS  ;fetch NEXTMS
   cmc  AX, AC07  ;compare arguments
   jz  $+4  ;end routine if
true
  ;branch on false
  mov  @T01,0  ;not equal, @T01 = 0
accumulator
   mov  AL,0  ;load value into
   jz  @72  ;jump to @72 if
false(=0)
  ;assign value of one variable to another variable
  (16-bit)
   mov  AX, MESSAG  ;assign MESSAG
   mov  AC6, AX  ;to AC6
   @73:  mov  AL,0  ;define location @73
   @73:  test for equality between NEXTMS and AC08 (16-bit)
   mov  @T01,1  ;presuppose equality
   mov  AX, NEXTMS  ;fetch NEXTMS
CMO AX, ·9C08  ; compare arguments  
JZ $+4  ; send routine if

true

MOV @T01, 0  ; not equal, @T01 ≠ 0

fork branch on false

MOV AL, @T01  ; load value into accumulator

false(=0)

assign value of one variable to another variable (16-bit)

MOV AX, ·MESSAG  ; assign ·MESSAG
MOV AC7, AX  ; to AC7

@74:  NOO  ; define location @74

Jtest for equality between NEXTMS and @C09 (16-bit)

MOV @T01, 1  ; presuppose equality
MOV AX, NEXTMS  ; ifetch NEXTMS
CMO AX, @C09  ; compare arguments
JZ $+4  ; send routine if

true

fork branch on false

MOV AL, @T01  ; load value into accumulator

false(=0)

assign value of one variable to another variable (16-bit)

MOV AX, ·MESSAG  ; assign ·MESSAG
MOV AC9, AX  ; to AC9

@75:  NOO  ; define location @75

Jtest for equality between NEXTMS and @C10 (16-bit)

MOV @T01, 1  ; presuppose equality
MOV AX, NEXTMS  ; ifetch NEXTMS
CMO AX, @C10  ; compare arguments
JZ $+4  ; send routine if

true

fork branch on false

MOV AL, @T01  ; load value into accumulator

false(=0)

assign value of one variable to another variable (16-bit)

MOV AX, ·MESSAG  ; assign ·MESSAG
MOV AC9, AX  ; to AC9

@76:  NOO  ; define location @76
; Add 16-bit NEXTMS + @C01 = @T01
mov AX, NEXTMS        ; fetch first argument
add AX, @C01          ; add second argument
test for equality between NEXTMS and @C14 (16-bit)
mov @T01, 1            ; presuppose equality
mov AX, NEXTMS         ; fetch NEXTMS
cmp AX, @C14           ; compare arguments
jz $+4                 ; end routine if true
jz 77                   ; jump to @77 if false(=0)

mov AL, @T01           ; load value into accumulator
cmp AL, 0              ; compare to zero
jz 77                   ; jump to @77 if false(=0)

mov AX, @C02           ; assign @C02
mov NEXTMS, AX         ; to NEXTMS

; Define 8-bit data constants
@C01 equ 1
@C02 equ 0
@C03 equ 2
@C04 equ 3
@C05 equ 4
@C06 equ 5
@C07 equ 6
@C08 equ 7
@C09 equ 8
@C10 equ 9
@C11 equ 30

; Define location @77
@77:          

125
`C12  equ  20
; define 8-bit data constant
`C13  equ  15
; define 8-bit data constant
`C14  equ  10
; define 8-bit storage
`org  1023 ; 8 bit variable  `AT01

in ram
`AT01:    do  0
`org  986521 ; from address pointer
; define 8-bit storage
`org  1023 ; 8 bit variable  `AT02

in ram
`AT02:    do  0
`org  986521 ; from address pointer
; - monitor section -
; `@spsr:  mov  AX, `@table ; initialize table pointer
`@sloop:  mov  `@ontr, AX ; to beginning
`@mloop:  mov  BX, `@ontr
  inc  BX
  inc  BX
  inc  BX
  mov  `@ontr, BX
  jno  BX

; - data section -
; `@ontr:  `org  1023 ; address pointer
`@table:  dw  0 ; table entry
`@tentry:  `org  986521 ; from address pointer
; (define too)
  jno  @KEYINM ; test for contingency KEYINM
  jno  @MINTAC ; test for contingency MINTAC
  jno  @MANU ; test for contingency MANU
  jno  @MAUTO ; test for contingency MAUTO
  jno  @POLL ; test for contingency POLL
  jno  @LOCAT ; test for contingency LOCAT
  jno  @POSCH ; test for contingency POSCH
contingency POSCH  jno  @MEMGSOS  ;test for
contingency MSGOS  jno  @MCLOCK  ;test for
contingency MCLOCK  jno  @MLOGIN  ;test for
contingency MLOGIN  jno  @TLOGIN  ;test for
contingency TLOGIN  jno  @TLOGOU  ;test for
contingency TLOGOJ  jno  @MSGIN  ;test for
contingency MSGIN  jno  @MSGIN  ;test for
contingency MSGIN  jno  @SOSVR  ;igo to start of

; KEYINM:  call @KEYINM
contingency code KEYINM  cmo  KEYINM, 1  ;execute
contingency result to

execute KBINPM  jnz  $ + 5
KBINPM  if true  call @KBINPM  ;execute task

; INTAC:  call @INTAC
contingency code INTAC  cmo  INTAC, 1  ;execute
contingency result to

execute INTAC  jnz  $ + 5
INTAC  if true  call @INTAC  ;execute task

; SMANU:  call @SMANU
contingency code SMANU  cmo  SMANU, 1  ;execute
contingency result to

execute MANUAL  jnz  $ + 5
MANUAL  if true  call @MANUAL  ;execute task


; $SMAUTO: call $SMAUTO
contingency code $SMAUTO
contingency result to
jnz $ + 5
execute AUTO
call $AUTO
if true
jno $m100
; $TPOLL: call $TPOLL
contingency code $TPOLL
contingency result to
jnz $ + 5
execute POLLAU
POLLAU if true
jno $m100
; $MLOCAT: call $MLOCAT
contingency code $MLOCAT
contingency result to
jnz $ + 5
execute LOCATI
LOCATI if true
jno $m100
; $TMLOCA: call $TMLOCA
contingency code $TMLOCA
contingency result to
jnz $ + 5
execute MANLOC
MANLOC if true
jno $m100
; $POSCH: call $POSCH
contingency code $POSCH
contingency result to
jnz $ + 5
execute POSUPD
PJSUPD if true
  call $POSUPD
  jno @m100
 ;
@MMGDS:  call $MMGDS
contingency code $MMGDS
cmp $MMGDS, 1
contingency result to
execute MSGDSP
  jnz $ + 5
MSGDSP if true
  jno @m100
 ;
@MCLOCK:  call $MCLOCK
contingency code $MCLOCK
cmp $MCLOCK, 1
contingency result to
execute CLOCKS
  jnz $ + 5
CLOCKS if true
  jno @m100
 ;
@MLOGIN:  call $MLOGIN
contingency code $MLOGIN
cmp $MLOGIN, 1
contingency result to
execute LOGINO
  jnz $ + 5
LOGINO if true
  jno @m100
 ;
@TLOGIN:  call $TLOGIN
contingency code $TLOGIN
cmp $TLOGIN, 1
contingency result to
execute LOGIN
  jnz $ + 5
if true
  jno @m100
 ;
@TLOGOU:  call $TLOGOU
contingency code $TLOGOU
cmp $TLOGOU, 1
execute task $POSUPD
return to monitor
execute $MMGDS
compare true flag (1) if false do not execute task
return to monitor execute
execute
execute contingency code $MMGDS
compare
execute task
return to monitor execute
execute
execute contingency code $MCLOCK
compare true flag (1) if false do not execute task
return to monitor execute
execute
execute
execute contingency code $MLOGIN
compare true flag (1) if false do not execute task
return to monitor execute
execute
execute
execute contingency code $TLOGIN
compare true flag (1) if false do not execute task
return to monitor execute
execute
execute
execute contingency code $TLOGOU
compare
execute task LOGINO
return to monitor execute
execute
execute
contingency result to 

execute LOGOUT call @LOGOUT if true jnz $ + 5 if false do not execute task

LOGOUT if true jno @mloo return to monitor

; @MSGIN: call @MSGIN execute

contingency code MSGIN cmd MSGIN,1 compare

contingency result to 

execute MSGSTO call @MSGSTO if true jnz $ + 5 if false do not execute MSGSTO if true

MSGSTO if true jno @mloo return to monitor end software listing

complete

dis realisatit consumes 71.180 watts of power and contains 162. chips.
APPENDIX E

NEWMCSL.FOR OUTPUT
LISTING (HARDWARE)

central processing unit
device: intel 8086 microprocessor (max-mode, no
ndo), ic1
connections:
pins
16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,39,38,37,36,35 = a(0:19)
pins 16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,39 =
d(0:15)
pin 17 (nmi) = gnd
pin 18 (intr) = gnd
pin 19 = clk
pin 34 = che-bar
pin 33 (mv/max-bar) = gnd
pin 32 (rd-bar) = n.c.
pin 31 (rq-bar/qti0-bar) = n.c.
pin 30 (rq-bar/qti0-bar) = n.c.
pin 29 (lock-bar) = n.c.
pin 28 = s2-bar
pin 27 = s1-bar
pin 26 = s0-bar
pin 25 (qs0) = n.c.
pin 24 (qs1) = n.c.
pin 23 (test-bar) = gnd
pin 22 = ready
pin 21 = reset
pins 1,20 = gnd
pin 40 = +5v
clock generator (0.125 us
device: intel 8284 clock gen and driver for 8086

cou, ic2
connections:
pin 1 (csync) = gnd
pin 3 (aen1-bar) = gnd
pin 4 (rdv1) = +5v
pin 5 (ready) = ready
pin 6 (rdv2) = gnd
pin 7 (aen2-bar) = gnd
pin 8 = clk
pin 10 = reset
pin 11 = res-bar
pin 13 (f/c-bar) = gnd
pin 14 (efi) = gnd
pin 15 (async-bar) = +5v
pins 10,17 (xtal(1:2)) = device: 24 mhz crystal
octal bus transceiver/data bits 0:7
device: intel 8286 octal bus transceiver, ic3
connections:
  pins 19,18,17,16,15,14,13,12 (db(0:7)) = do(0:7)
  pin 1 (a0) = d(0)
  pin 2 (a1) = d(1)
  pin 3 (a2) = d(2)
  pin 4 (a3) = d(3)
  pin 5 (a4) = d(4)
  pin 6 (a5) = d(5)
  pin 7 (a6) = d(6)
  pin 8 (a7) = d(7)
  pin 9 (oe-bar) = not, den
  pin 11 (t) = dt/r-bar
  pin 10 = gnd
  pin 20 = +5v

octal bus transceiver/data bits 8:15
device: intel 8286 octal bus transceiver, ic4
connections:
  pins 19,18,17,16,15,14,13,12 (db(0:7)) = do(8:15)
  pin 1 (a0) = d(0)
  pin 2 (a1) = d(1)
  pin 3 (a2) = d(2)
  pin 4 (a3) = d(3)
  pin 5 (a4) = d(4)
  pin 6 (a5) = d(5)
  pin 7 (a6) = d(6)
  pin 8 (a7) = d(7)
  pin 9 (oe-bar) = not, den
  pin 11 (t) = dt/r-bar
  pin 10 = gnd
  pin 20 = +5v

bus controller
device: intel 8288 bus controller for 8086 cou, ic5
connections:
  pin 19 = s0-bar
  pin 3 = s1-bar
  pin 18 = s2-bar
  pin 2 = clk
  pin 5 = ale
  pin 16 = den
  pin 4 = dt/r-bar
  pin 6 (ae-bar) = qnd
  pin 1 (ioe) = +5v
  pin 7 = mrjc-bar
  pin 9 = mrwc-bar
  pin 11 = iowc-bar
  pin 13 = iarc-bar
  pin 14 = inter-bar
octal latch/address bits 0:7
device: intel 8282 octal latch for 8086 CPU, ic6
connections:
o pins 1, 2, 3, 4, 5, 6, 7, 8 (di(0:7)) = a(0:7)
o pins 19, 18, 17, 16, 15, 14, 13, 12 (do(0:7)) = a(0:7)
o pin 9 (oe-bar) = qnd
o pin 11 (sto) = ale
o pin 10 = qnd
o pin 20 = +5v

octal latch/address bits 8:15
device: intel 8282 octal latch for 8086 CPU, ic7
connections:
o pins 1, 2, 3, 4, 5, 6, 7, 8 (di(0:7)) = a(8:15)
o pins 19, 18, 17, 16, 15, 14, 13, 12 (do(0:7)) = a(8:15)
o pin 9 (oe-bar) = qnd
o pin 11 (sto) = ale
o pin 10 = qnd
o pin 20 = +5v

octal latch/address bits 16:19
device: intel 8282 octal latch for 8086 CPU, ic8
connections:
o pins 1, 2, 3, 4 (di(0:3)) = a(16:19)
o pins 19, 18, 17, 16 (do(0:3)) = a(16:19)
o pins 5, 6, 7, 8 (di(3:7)) = qnd
o pin 9 (oe-bar) = qnd
o pin 11 (sto) = ale
o pin 10 = qnd
o pin 20 = +5v

address decoder/address for memory select
device: intel 8205 1-of-8 binary decoder, ic9
connections:
o pin 15 (o(0)) = csu-bar(1)
o pin 14 (o(1)) = csu-bar(2)
o pin 13 (o(2)) = csu-bar(3)
o pin 12 (o(3)) = csu-bar(4)
o pin 11 (o(4)) = csu-bar(5)
o pin 10 (o(5)) = csu-bar(6)
o pin 9 (o(6)) = csu-bar(7)
o pin 7 (o(7)) = csu-bar(8)
o pin 1 = a(15)
o pin 2 = a(16)
o pin 3 = a(17)
o pin 4 (e1-bar) = a(0)
o pin 5 (e2-bar) = a(18)
o pin 6 (e3) = not, a(19)
o pin 8 = qnd
o pin 16 = +5v

address decoder/address for memory select
device: intel 8205 1-of-8 binary decoder, ic10

connections:
- oin 15 (o(0)) = csu-bar(9)
- oin 14 (o(1)) = csu-bar(10)
- oin 13 (o(2)) = csu-bar(11)
- oin 12 (o(3)) = csu-bar(12)
- oin 11 (o(4)) = csu-bar(13)
- oin 10 (o(5)) = csu-bar(14)
- oin 9 (o(6)) = csu-bar(15)
- oin 7 (o(7)) = csu-bar(16)
- oin 1 = a(15)
- oin 2 = a(16)
- oin 3 = a(17)
- oin 4 = e1-bar = a(0)
- oin 5 = e2-bar = a(19)
- oin 6 = e3 = a(18)
- oin 8 = gnd
- oin 16 = +5v

address decoder/address for memory select
device: intel 8205 1-of-8 binary decoder, ic11

connections:
- oin 15 (o(0)) = csu-bar(17)
- oin 14 (o(1)) = csu-bar(18)
- oin 13 (o(2)) = csu-bar(19)
- oin 12 (o(3)) = csu-bar(20)
- oin 11 (o(4)) = csu-bar(21)
- oin 10 (o(5)) = csu-bar(22)
- oin 9 (o(6)) = csu-bar(23)
- oin 7 (o(7)) = csu-bar(24)
- oin 1 = a(15)
- oin 2 = a(16)
- oin 3 = a(17)
- oin 4 = e1-bar = a(0)
- oin 5 = e2-bar = a(19)
- oin 6 = e3 = a(18)
- oin 8 = gnd
- oin 16 = +5v

address decoder/address for memory select
device: intel 8205 1-of-8 binary decoder, ic12

connections:
- oin 15 (o(0)) = csu-bar(25)
- oin 14 (o(1)) = csu-bar(26)
- oin 13 (o(2)) = csu-bar(27)
- oin 12 (o(3)) = csu-bar(28)
- oin 11 (o(4)) = csu-bar(29)
- oin 10 (o(5)) = csu-bar(30)
- oin 9 (o(6)) = csu-bar(31)
- oin 7 (o(7)) = csu-bar(32)
- oin 1 = a(15)
- oin 2 = a(16)
- oin 3 = a(17)
pin 4 (el-bar) = a(0)
pin 5 (e2-bar) = a(18)
pin 6 (e3) = .not. a(19)
pin 8 = gnd
pin 16 = +5v

address decoder/address for memory select
device: intel 8205 1-of-9 binary decoder, ic13
connections:
pin 15 (o(0)) = csl-bar(1)
pin 14 (o(1)) = csl-bar(2)
pin 13 (o(2)) = csl-bar(3)
pin 12 (o(3)) = csl-bar(4)
pin 11 (o(4)) = csl-bar(5)
pin 10 (o(5)) = csl-bar(6)
pin 9 (o(6)) = csl-bar(7)
pin 7 (o(7)) = csl-bar(8)
pin 1 = a(15)
pin 2 = a(16)
pin 3 = a(17)
pin 4 (el-bar) = bhe-bar
pin 5 (e2-bar) = a(18)
pin 6 (e3) = .not. a(19)
pin 8 = gnd
pin 16 = +5v

address decoder/address for memory select
device: intel 8205 1-of-8 binary decoder, ic14
connections:
pin 15 (o(0)) = csl-bar(9)
pin 14 (o(1)) = csl-bar(10)
pin 13 (o(2)) = csl-bar(11)
pin 12 (o(3)) = csl-bar(12)
pin 11 (o(4)) = csl-bar(13)
pin 10 (o(5)) = csl-bar(14)
pin 9 (o(6)) = csl-bar(15)
pin 7 (o(7)) = csl-bar(16)
pin 1 = a(15)
pin 2 = a(16)
pin 3 = a(17)
pin 4 (el-bar) = bhe-bar
pin 5 (e2-bar) = a(19)
pin 6 (e3) = a(18)
pin 8 = gnd
pin 16 = +5v

address decoder/address for memory select
device: intel 8205 1-of-8 binary decoder, ic15
connections:
pin 15 (o(0)) = csl-bar(17)
pin 14 (o(1)) = csl-oar(18)
pin 13 (o(2)) = csl-bar(19)
pin 12 (o(3)) = csl-bar(20)
pin 11 (o(4)) = csl-bar(21)
pin 10 (o(5)) = csl-bar(22)
pin 9 (o(6)) = csl-bar(23)
pin 7 (o(7)) = csl-bar(24)
pin 1 = a(15)
pin 2 = a(16)
pin 3 = a(17)
pin 4 (e1-oar) = bhe-bar
pin 5 (e2-oar) = a(18)
pin 6 (e3) = a(19)
pin 8 = gnd
pin 16 = +5v

address decoder/address for memory select
device: intel 8205 1-of-8 binary decoder, ic16
connections:
  pin 15 (o(0)) = csl-bar(25)
  pin 14 (o(1)) = csl-bar(26)
  pin 13 (o(2)) = csl-bar(27)
  pin 12 (o(3)) = csl-bar(28)
  pin 11 (o(4)) = csl-bar(29)
  pin 10 (o(5)) = csl-bar(30)
  pin 9 (o(6)) = csl-bar(31)
  pin 7 (o(7)) = csl-bar(32)
  pin 1 = a(15)
  pin 2 = a(16)
  pin 3 = a(17)
  pin 4 (e1-oar) = bhe-bar
  pin 5 (e2-oar) = a(18)
  pin 6 (e3) = not. a(19)
  pin 8 = gnd
  pin 16 = +5v

condition mode input interface hardware to sense
signal KEYFLG
device: intel 8212 8 bit i/o port, ic 17
connections:
  oins 3,5,7,9,16,18,20,22(di(1:8)) = KEYFLG(1:8)
remainder to
around
  oins 4,6,8,10,15,17,19,21(do(1:8)) = db(1:8)
  pin 2 (md) = gnd
  pin 11 (sta) = gnd
  pin 1 (dsl-bar) = not. (decode a(0:7) value 0)
  pin 13 (ds2) = ino. and. doin
  pin 24 = +5v
  pin 12 = gnd

condition mode input interface hardware to sense
signal KEYFLG
device: intel 8212 8 bit i/o port, ic 18
connections:
  oins 3,5,7,9,16,18,20,22(di(9:16)) = KEYFLG(9:16)
remainder to
ground
d in 4, 6, 8, 10, 15, 17, 19, 21 (do(9:16)) = db(9:16)
d in 2 (md) = gnd

d in 11 (sto) = gnd

d in 1 (dsl-bar) = .not. (decode a(8:15) value 0)
d in 13 (ds2) = inp .and. doin
d in 24 = +5v

d in 12 = gnd
16 bit output port composed of two 8 bit ports
x1 is for low order byte
x9 is for high order byte
condition-mode output interface hardware to issue
signal: x1

device: intel 8212 8-bit i/o port, ic 19
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = x1(1:8) ; if

8 are req

d in 2 (md) = +5v

d in 11 (stb) = gnd

d in 1 (dsl-bar) = wr-bar

d in 13 (ds2) = out .and. (decode a(0:7) value 0)
d in 24 (vcc) = +5v

d in 12 (gnd) = gnd

condition-mode output interface hardware to issue
signal: x9

device: intel 8212 8-bit i/o port, ic 20
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = x9(1:8) ; if

8 are req

d in 2 (md) = +5v

d in 11 (stb) = gnd

d in 1 (dsl-bar) = wr-bar

d in 13 (ds2) = out .and. (decode a(0:7) value 1)
d in 24 (vcc) = +5v

d in 12 (gnd) = gnd

condition mode input interface hardware to sense
signal KEYCH

device: intel 8212 8 bit i/o port, ic 21
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = KEYCH(1:8)
remainder to

ground

pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = db(1:8)
d in 2 (md) = gnd

d in 11 (stc) = gnd

d in 1 (dsl-bar) = .not. (decode a(0:7) value 1)
d in 13 (ds2) = inp .and. doin

137
pin 24 = +5v
pin 12 = gnd

condition mode input interface hardware to sense

signal KEYCHA
device:intel 8212 8 bit i/o port,ic 22
connections:
oins 3,5,7,9,16,18,20,22(di(9:16)) = KEYCHA(9:16)
remainder to
ground
oins 4,6,8,10,15,17,19,21(do(9:16)) = db(9:16)
pin 2 (md) = gnd
pin 11 (sto) = gnd
pin 1 (d1- bar) = .not. (decode a(8:15) value 1)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd

condition mode input interface hardware to sense

signal KEYCHA
device:intel 8212 8 bit i/o port,ic 23
connections:
oins 3,5,7,9,16,18,20,22(di(1:8)) = KEYCHA(1:8)
remainder to
ground
pins 4,6,8,10,15,17,19,21(do(1:8)) = db(1:8)
pin 2 (md) = gnd
pin 11 (sto) = gnd
pin 1 (d1- bar) = .not. (decode a(0:7) value 2)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd

condition mode input interface hardware to sense

signal KEYCHA
device:intel 8212 8 bit i/o port,ic 24
connections:
oins 3,5,7,9,16,18,20,22(di(9:16)) = KEYCHA(9:16)
remainder to
ground
pins 4,6,8,10,15,17,19,21(do(9:16)) = db(9:16)
pin 2 (md) = gnd
pin 11 (sto) = gnd
pin 1 (d1- bar) = .not. (decode a(8:15) value 2)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd

16 bit output port composed of two 8 bit ports
  x17 is for low order byte
  x25 is for high order byte

condition-mode output interface hardware to issue
signal: x17
device: intel 8212 8-bit i/o port, ic 25
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,10,15,17,19,21 (do(1:8)) = x17(1:8) ; if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = qnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out  .and. (decode a(0:7) value 2)
pin 24 (vcc) = +5v
pin 12 (gnd) = qnd
condition-mode output interface hardware to issue

signal: x25
device: intel 8212 8-bit i/o port, ic 26
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x25(1:8) ; if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = qnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out  .and. (decode a(0:7) value 3)
pin 24 (vcc) = +5v
pin 12 (gnd) = qnd
condition mode input interface hardware to sense

signal KEYCHA
device: intel 8212 8 bit i/o port, ic 27
connections:
pins 3,5,7,9,16,18,20,22(di(1:8)) = KEYCHA(1:8)
remainder to
ground
pins 4,6,8,10,15,17,19,21(do(1:8)) = db(1:8)
pin 2 (md) = qnd
pin 11 (stb) = qnd
pin 1 (ds1-bar) = not  .and. (decode a(0:7) value 3)
pin 13 (ds2) = inp  .and. doin
pin 24 = +5v
pin 12 = qnd
condition mode inout interface hardware to sense

signal KEYCHA
device: intel 8212 8 bit i/o port, ic 28
connections:
pins 3,5,7,9,16,18,20,22(di(9:16)) = KEYCHA(9:16)
remainder to
ground
pins 4,6,8,10,15,17,19,21(do(9:16)) = db(9:16)
pin 2 (md) = qnd
pin 11 (stb) = qnd
oin 1 (dls-bar) = not. (decode a(8:15) value 3)
oin 13 (ds2) = inp .and. doin
oin 24 = +5v
oin 12 = gnd

condition mode input interface hardware to sense
signal KEYCHA
device:intel 8212 8 bit i/o oort,ic 29
connections:
oin 3,5,7,9,16,18,20,22(d1(1:8)) = KEYCHA(1:8)
remainder to
ground
oin 4,6,8,10,15,17,19,21(d0(1:8)) = do(1:8)
oin 2 (md) = gnd
oin 11 (stb) = gnd
oin 1 (dls-bar) = not. (decode a(0:7) value 4)
oin 13 (ds2) = inp .and. doin
oin 24 = +5v
oin 12 = gnd

condition mode input interface hardware to sense
signal KEYCHA
device: intel 8212 8 bit i/o oort, ic 30
connections:
oin 3,5,7,9,16,18,20,22(di(9:16)) = KEYCHA(9:16)
remainder to
ground
oin 4,6,8,10,15,17,19,21(d9(9:16)) = db(9:16)
oin 2 (md) = gnd
oin 11 (stb) = gnd
oin 1 (dls-bar) = not. (decode a(8:15) value 4)
oin 13 (ds2) = inp .and. doin
oin 24 = +5v
oin 12 = gnd

16 bit output port composed of two 8 bit ports
x33 is for low order byte
x41 is for high order byte
condition-mode output interface hardware to issue
signal: x33
device: intel 8212 8-bit i/o oort, ic 31
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x33(1:8) ; if
8 are req
oin 2 (md) = +5v
oin 11 (stb) = gnd
oin 1 (dls-bar) = wr-bar
oin 13 (ds2) = out .and. (decode a(0:7) value 4)
oin 24 (vcc) = +5v
oin 12 (qnd) = gnd

condition-mode output interface hardware to issue
device: intel 8212 8-bit i/o port, ic 32
connections:
pins 3,5,7,9,16,19,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x41(1:8) ; if
8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out , and. (decode a(0:7) value 5)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd
16 bit output port composed of two 8 bit ports
x49 is for low order byte
x57 is for high order byte
condition-mode output interface hardware to issue

signal: x49
device: intel 8212 8-bit i/o port, ic 33
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x49(1:8) ; if
8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out , and. (decode a(0:7) value 6)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd
condition-mode output interface hardware to issue

signal: x57
device: intel 8212 8-bit i/o port, ic 34
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x57(1:8) ; if
8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out , and. (decode a(0:7) value 7)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd
16 bit output port composed of two 8 bit ports
x65 is for low order byte
x73 is for high order byte
condition-mode output interface hardware to issue

signal: x65
device: intel 8212 8-bit i/o port, ic 35
connections:
pins 3,5,7,9,16,19,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x65(1:8) ; if
8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (dsl-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a[0:7] value 9)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue signal: \text{xx7}3
device: intel 8212 8-bit i/o port, ic 36
connections:
pins 3,5,7,9,16,19,20,22 (di[1:8]) = db[1:8]
pins 4,6,8,10,15,17,19,21 (do[1:8]) = \text{xx73}[1:8]; if

8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (dsl-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a[0:7] value 9)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16-bit output port composed of two 8-bit ports
xx81 is for low order byte
xx89 is for high order byte

condition-mode output interface hardware to issue signal: \text{xx8}1
device: intel 8212 8-bit i/o port, ic 37
connections:
pins 3,5,7,9,16,19,20,22 (di[1:8]) = db[1:8]
pins 4,6,8,10,15,17,19,21 (do[1:8]) = \text{xx81}[1:8]; if

8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (dsl-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a[0:7] value 10)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue signal: \text{xx8}9
device: intel 8212 8-bit i/o port, ic 38
connections:
pins 3,5,7,9,16,18,20,22 (di[1:8]) = db[1:8]
pins 4,6,8,10,15,17,19,21 (do[1:8]) = \text{xx89}[1:8]; if

8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (dsl-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a[0:7] value 11)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16-bit output port composed of two 8-bit ports
xx97 is for low order byte
xx105 is for high order byte

142
condition-mode output interface hardware to issue signal: x97
device: intel 8212 8-bit i/o port, ic 39
connections:
<table>
<thead>
<tr>
<th>pins</th>
<th>(di(1:8)) = dh(1:8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, 5, 7, 9, 16, 19, 20, 22</td>
<td></td>
</tr>
<tr>
<td>4, 6, 9, 10, 15, 17, 19, 21</td>
<td></td>
</tr>
</tbody>
</table>

if 8 are req
- pin 2 (md) = +5v
- pin 11 (stb) = gnd
- pin 1 (dsl-bar) = wr-bar
- pin 13 (ds2) = out .and. (decode a(0:7) value 12)
- pin 24 (vcc) = +5v
- pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
- x113 is for low order byte
- x121 is for high order byte

condition-mode output interface hardware to issue signal: x113
device: intel 8212 8-bit i/o port, ic 41
connections:
<table>
<thead>
<tr>
<th>pins</th>
<th>(di(1:8)) = dh(1:8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, 5, 7, 9, 16, 18, 20, 22</td>
<td></td>
</tr>
<tr>
<td>4, 6, 8, 10, 15, 17, 19, 21</td>
<td></td>
</tr>
</tbody>
</table>

if 8 are req
- pin 2 (md) = +5v
- pin 11 (stb) = gnd
- pin 1 (dsl-bar) = wr-bar
- pin 13 (ds2) = out .and. (decode a(0:7) value 13)
- pin 24 (vcc) = +5v
- pin 12 (gnd) = gnd

condition-mode output interface hardware to issue signal: x121
device: intel 8212 8-bit i/o port, ic 42
connections:
<table>
<thead>
<tr>
<th>pins</th>
<th>(di(1:8)) = dh(1:8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, 5, 7, 9, 16, 19, 20, 22</td>
<td></td>
</tr>
<tr>
<td>4, 6, 8, 10, 15, 17, 19, 21</td>
<td></td>
</tr>
</tbody>
</table>

if 8 are req
- pin 2 (md) = +5v
- pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 15)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
x129 is for low order byte
x137 is for high order byte
condition-mode output interface hardware to issue
signal: x129
device: intel 8212 8-bit i/o port, ic 43
connections:
    pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x129(1:8)

if 8 are req
    pin 2 (md) = +5v
    pin 11 (stb) = gnd
    pin 1 (ds1-bar) = wr-bar
    pin 13 (ds2) = out .and. (decode a(0:7) value 16)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

signal: x137
device: intel A212 8-bit i/o port, ic 44
connections:
    pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x137(1:8)

if 8 are req
    pin 2 (md) = +5v
    pin 11 (stb) = gnd
    pin 1 (ds1-bar) = wr-bar
    pin 13 (ds2) = out .and. (decode a(0:7) value 17)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
x145 is for low order byte
x153 is for high order byte
condition-mode output interface hardware to issue
signal: x145
device: intel 8212 8-bit i/o port, ic 45
connections:
    pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x145(1:8)

if 8 are req
    pin 2 (md) = +5v
    pin 11 (stb) = gnd
    pin 1 (ds1-bar) = wr-bar
    pin 13 (ds2) = out .and. (decode a(0:7) value 18)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

signal: x153
device: intel 8212 8-bit i/o port, ic 46
connections:
  pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
  pins 4,6,8,10,15,17,19,21 (do(1:8)) = x153(1:8)

; if 8 are req
  pin 2 (md) = +5v
  pin 11 (stb) = gnd
  pin 1 (d$1-bar) = wr-bar
  pin 13 (d$s2) = out .and. (decode a(0:7) value 19)
  pin 24 (vcc) = +5v
  pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
  x161 is for low order byte
  x169 is for high order byte

condition-mode output interface hardware to issue signal: x161
device: intel 8212 8-bit i/o port, ic 47
connections:
  pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
  pins 4,6,8,10,15,17,19,21 (do(1:8)) = x161(1:8)

; if 8 are req
  pin 2 (md) = +5v
  pin 11 (stb) = gnd
  pin 1 (d$1-bar) = wr-bar
  pin 13 (d$s2) = out .and. (decode a(0:7) value 20)
  pin 24 (vcc) = +5v
  pin 12 (gnd) = gnd

condition-mode output interface hardware to issue signal: x169
device: intel 8212 8-bit i/o port, ic 48
connections:
  pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
  pins 4,6,8,10,15,17,19,21 (do(1:8)) = x169(1:8)

; if 8 are req
  pin 2 (md) = +5v
  pin 11 (stb) = gnd
  pin 1 (d$1-bar) = wr-bar
  pin 13 (d$s2) = out .and. (decode a(0:7) value 21)
  pin 24 (vcc) = +5v
  pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
  x177 is for low order byte
  x185 is for high order byte

condition-mode output interface hardware to issue signal: x177
device: intel 8212 8-bit i/o port, ic 49
connections:
  pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
  pins 4,6,8,10,15,17,19,21 (do(1:8)) = x177(1:8)

; if 8 are req
  pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 22)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

Condition-mode output interface hardware to issue signal: x185
device: intel 8212 8-bit i/o port, ic 50
connections:
pins 5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,5,8,10,15,17,19,21 (do(1:8)) = x185(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 23)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

Condition mode input interface hardware to sense signal KEYCHA
device: intel 8212 8 bit i/o port, ic 51
connections:
pins 3,5,7,9,16,18,20,22(di(1:8)) = KEYCHA(1:8)
remainder to ground
pines 4,6,8,10,15,17,19,21(do(1:8)) = db(1:8)
pin 2 (md) = gnd
pin 11 (stb) = gnd
pin 1 (ds1-bar) = .not. (decode a(0:7) value 5)
pin 13 (ds2) = ino .and. doin
pin 24 = +5v
pin 12 = gnd

Condition mode input interface hardware to sense signal KEYCHA
device: intel 8212 8 bit i/o port, ic 52
connections:
pins 3,5,7,9,10,18,20,22(di(9:16)) = KEYCHA(9:16)
remainder to ground
pines 4,6,8,10,15,17,19,21(do(9:16)) = db(9:16)
pin 2 (md) = gnd
pin 11 (stb) = gnd
pin 1 (ds1-bar) = .not. (decode a(8:15) value 5)
pin 13 (ds2) = ino .and. doin
pin 24 = +5v
pin 12 = gnd

16 bit output port composed of two 8 bit ports
x193 is for low order byte
x201 is for high order byte
condition-mode output interface hardware to issue signal: \( x_{193} \)
- device: Intel 8212 8-bit i/o port, ic 53
- connections:
  - pins 3, 5, 7, 9, 16, 19, 20, 22: \( di(1:8) = db(1:8) \)
  - pins 4, 6, 8, 10, 15, 17, 19, 21: \( do(1:8) = x_{193}(1:8) \)

\( \text{if 8 are req} \)
- pin 2 (\( md \)) = +5v
- pin 11 (\( stb \)) = gnd
- pin 1 (\( ds1-bar \)) = \( wr-bar \)
- pin 13 (\( ds2 \)) = out
- pin 24 (\( vcc \)) = +5v
- pin 12 (\( gnd \)) = gnd

condition-mode output interface hardware to issue signal: \( x_{201} \)
- device: Intel 8212 8-bit i/o port, ic 54
- connections:
  - pins 3, 5, 7, 9, 16, 18, 20, 22: \( di(1:8) = db(1:8) \)
  - pins 4, 6, 9, 10, 15, 17, 19, 21: \( do(1:8) = x_{201}(1:8) \)

\( \text{if 8 are req} \)
- pin 2 (\( md \)) = +5v
- pin 11 (\( stb \)) = gnd
- pin 1 (\( ds1-bar \)) = \( wr-bar \)
- pin 13 (\( ds2 \)) = out
- pin 24 (\( vcc \)) = +5v
- pin 12 (\( gnd \)) = gnd

condition mode inout interface hardware to sense signal KEYCHA
- device: Intel 8212 8-bit i/o port, ic 55
- connections:
  - pins 3, 5, 7, 9, 16, 18, 20, 22: \( di(1:8) = \text{KEYCHA}(1:8) \)
- remainder to

around
- pins 4, 6, 8, 10, 15, 17, 19, 21: \( do(1:8) = db(1:8) \)
- pin 2 (\( md \)) = gnd
- pin 11 (\( stb \)) = gnd
- pin 1 (\( ds1-bar \)) = \( not \cdot (\text{decode a}(0:7) \text{ value 6}) \)
- pin 13 (\( ds2 \)) = \( \text{and} \cdot \text{doin} \)
- pin 24 = +5v
- pin 12 = gnd

condition mode inout interface hardware to sense signal KEYCHA
- device: Intel 8212 8-bit i/o port, ic 56
- connections:
  - pins 3, 5, 7, 9, 16, 18, 20, 22: \( di(9:16) = \text{KEYCHA}(9:16) \)
- remainder to

around
- pins 4, 6, 8, 10, 15, 17, 19, 21: \( do(9:16) = db(9:16) \)
- pin 2 (\( md \)) = gnd
Pin 11 (stb) = gnd
Pin 1 (ds1-bar) = .not. (decode a(8:15) value 6)
Pin 13 (ds2) = inb .and. doin
Pin 24 = +5v
Pin 12 = gnd

16 bit output port composed of two 8 bit ports
x209 is for low order byte
x217 is for high order byte

condition-mode output interface hardware to issue

signal: x209
device: intel 8212 8-bit i/o port, ic 57
connections:
opins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
opins 4,6,8,10,15,17,19,21 (do(1:8)) = x209(1:8)

if 8 are req
Pin 2 (md) = +5v
Pin 11 (stb) = gnd
Pin 1 (ds1-bar) = wr-par
Pin 13 (ds2) = out .and. (decode a(0:7) value 26)
Pin 24 (vcc) = +5v
Pin 12 (gnd) = gnd

condition-mode output interface hardware to issue

signal: x217
device: intel 8212 8-bit i/o port, ic 58
connections:
opins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
opins 4,6,8,10,15,17,19,21 (do(1:8)) = x217(1:8)

if 8 are req
Pin 2 (md) = +5v
Pin 11 (stb) = gnd
Pin 1 (ds1-bar) = wr-par
Pin 13 (ds2) = out .and. (decode a(0:7) value 27)
Pin 24 (vcc) = +5v
Pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
x225 is for low order byte
x233 is for high order byte

condition-mode output interface hardware to issue

signal: x225
device: intel 8212 8-bit i/o port, ic 59
connections:
opins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
opins 4,6,8,10,15,17,19,21 (do(1:8)) = x225(1:8)

if 8 are req
Pin 2 (md) = +5v
Pin 11 (stb) = gnd
Pin 1 (ds1-bar) = wr-par
Pin 13 (ds2) = out .and. (decode a(0:7) value 28)
Pin 24 (vcc) = +5v
Pin 12 (gnd) = gnd

condition-mode output interface hardware to issue
signal: x233

device: intel 8212 8-bit i/o port, ic 60

connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = x233(1:8)

if 8 are req
pin 2 (nd) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out and. (decode a(0:7) value 29)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
x241 is for low order byte
x249 is for high order byte

condition-mode output interface hardware to issue

signal: x241

device: intel 8212 8-bit i/o port, ic 61

connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = x241(1:8)

if 8 are req
pin 2 (nd) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out and. (decode a(0:7) value 30)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue

signal: x249

device: intel 8212 8-bit i/o port, ic 62

connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = x249(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out and. (decode a(0:7) value 31)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
x257 is for low order byte
x265 is for high order byte

condition-mode output interface hardware to issue

signal: x257

device: intel 8212 8-bit i/o port, ic 63

connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = x257(1:8)

if 8 are req
condition-mode output interface hardware to issue signal: x265
device: intel 8212 8-bit i/o port, ic 64
connections:
  pins 3,5,7,9,16,19,20,22 (di(1:8)) = db(1:8)
  pins 4,6,8,10,15,17,19,21 (do(1:8)) = x265(1:8)

; if 8 are req
  pin 2 (md) = +5v
  pin 11 (stb) = gnd
  pin 1 (ds1-bar) = wr-bar
  pin 13 (ds2) = out .and. (decode a(0:7) value 32)
  pin 24 (vcc) = +5v
  pin 12 (gnd) = gnd
16 bit output port composed of two 8 bit ports
  x273 is for low order byte
  x281 is for high order byte

condition-mode output interface hardware to issue signal: x273
device: intel 8212 8-bit i/o port, ic 65
connections:
  pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
  pins 4,6,8,10,15,17,19,21 (do(1:8)) = x273(1:8)

; if 8 are req
  pin 2 (md) = +5v
  pin 11 (stb) = gnd
  pin 1 (ds1-bar) = wr-bar
  pin 13 (ds2) = out .and. (decode a(0:7) value 33)
  pin 24 (vcc) = +5v
  pin 12 (gnd) = gnd
16 bit output port composed of two 8 bit ports
  x273 is for low order byte
  x281 is for high order byte

condition-mode output interface hardware to issue signal: x281
device: intel 8212 8-bit i/o port, ic 66
connections:
  pins 3,5,7,9,16,19,20,22 (di(1:8)) = db(1:8)
  pins 4,6,8,10,15,17,19,21 (do(1:8)) = x281(1:8)

; if 8 are req
  pin 2 (md) = +5v
  pin 11 (stb) = gnd
  pin 1 (ds1-bar) = wr-bar
  pin 13 (ds2) = out .and. (decode a(0:7) value 34)
  pin 24 (vcc) = +5v
  pin 12 (gnd) = gnd
16 bit output port composed of two 8 bit ports
  x289 is for low order byte
  x297 is for high order byte
condition-mode output interface hardware to issue signal: x289

device: intel 8212 8-bit i/o port, ic 67
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x289(1:8)

; if A are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 3b)
pin 24 (vcc) = +5v
pin 12 (qnd) = gnd

condition-mode output interface hardware to issue
signal: x297

device: intel 8212 8-bit i/o port, ic 68
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x297(1:8)

; if B are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 37)
pin 24 (vcc) = +5v
pin 12 (qnd) = gnd

16-bit output port composed of two 8-bit ports
x305 is for low order byte
x313 is for high order byte

condition-mode output interface hardware to issue
signal: x305

device: intel 8212 8-bit i/o port, ic 69
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x305(1:8)

; if B are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 38)
pin 24 (vcc) = +5v
pin 12 (qnd) = gnd

condition-mode output interface hardware to issue
signal: x313

device: intel 8212 8-bit i/o port, ic 70
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x313(1:8)

; if B are req
pin 2 (md) = +5v
pin 11 (stb) = gnd

151
oin 1 (ds1-bar) = wr-bar
oin 13 (ds2) = out .and. (decode a(0:7) value 39)
pin 24 (VCC) = +5V
pin 12 (GND) = GND
16 bit output port composed of two 8 bit ports
x321 is for low order byte
x329 is for high order byte
condition-mode output interface hardware to issue signal: x321
device: intel 8212 8-bit i/o port, ic 71
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = x321(1:8)

if 8 are req
pin 2 (md) = +5V
pin 11 (stb) = GND
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 40)
pin 24 (VCC) = +5V
pin 12 (GND) = GND
condition-mode output interface hardware to issue signal: x329
device: intel 8212 8-bit i/o port, ic 72
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = x329(1:8)

if 8 are req
pin 2 (md) = +5V
pin 11 (stb) = GND
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 41)
pin 24 (VCC) = +5V
pin 12 (GND) = GND
16 bit output port composed of two 8 bit ports
x337 is for low order byte
x345 is for high order byte
condition-mode output interface hardware to issue signal: x337
device: intel 8212 8-bit i/o port, ic 73
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = x337(1:8)

if 8 are req
pin 2 (md) = +5V
pin 11 (stb) = GND
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 42)
pin 24 (VCC) = +5V
pin 12 (GND) = GND
condition-mode output interface hardware to issue signal: x345
device: intel 8212 8-bit i/o port, ic 74
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = do(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x345(1:8)

; if A are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 45)
pin 24 (vcc) = +5v
pin 12 (qnd) = gnd

10 bit output port composed of two 8 bit ports
x353 is for low order byte
x361 is for high order byte

condition mode output interface hardware to issue
signal: x353
device: intel 8212 8-bit i/o port, ic 75
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = do(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x353(1:8)

; if B are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 45)
pin 24 (vcc) = +5v
pin 12 (qnd) = gnd

condition mode output interface hardware to issue
signal: x361
device: intel 8212 8-bit i/o port, ic 76
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = do(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x361(1:8)

; if B are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 45)
pin 24 (vcc) = +5v
pin 12 (qnd) = gnd

condition mode input interface hardware to sense
signal KEYCHA
device: intel 8212 8-bit i/o port, ic 77
connections:
pins 3,5,7,9,16,18,20,22(di(1:8)) = KEYCHA(1:8)
remainder to

ground
pins 4,6,8,10,15,17,19,21(do(1:8)) = do(1:8)
pin 2 (md) = gnd
pin 11 (stb) = gnd

153
pin 1 (ds1-bar) = .not. (decode a(0:7) value 7)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd

condition mode input interface hardware to sense signal KEYCHA
device: intel 8212 8 bit i/o port, ic 78
connections:
pins 3,5,7,9,16,18,20,22 (di(9:16)) = KEYCHA(9:16)
remainder to ground
pines 4,6,8,10,15,17,19,21 (do(9:16)) = db(9:16)
pin 2 (md) = and
pin 11 (stb) = gnd
pin 1 (ds1-bar) = not (decode a(8:15) value 7)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd
16 bit output port composed of two 8 bit ports
x369 is for low order byte
x377 is for high order byte
condition-mode output interface hardware to issue signal: x369
device: intel 8212 8-bit i/o port, ic 79
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,9,10,15,17,19,21 (do(1:8)) = x369(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 4b)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd
condition-mode output interface hardware to issue signal: x377
device: intel 8212 8-bit i/o port, ic 80
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,9,10,15,17,19,21 (do(1:8)) = x377(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 47)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd
16 bit output port composed of two 8 bit ports
x385 is for low order byte
x393 is for high order byte
condition-mode output interface hardware to issue
signal: x385
device: Intel 8212 8-bit I/O port, IC 41
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (DI:1:8) = DB(1:8)
pins 4, 6, 9, 10, 15, 17, 19, 21 (DO:1:8) = x385(1:8)

if 8 are req
pin 2 (md) = +5V
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 4A)
pin 24 (vcc) = +5V
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue
signal: x393
device: Intel 8212 8-bit I/O port, IC 82
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (DI:1:8) = DB(1:8)
pins 4, 6, 9, 10, 15, 17, 19, 21 (DO:1:8) = x393(1:8)

if 8 are req
pin 2 (md) = +5V
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 49)
pin 24 (vcc) = +5V
pin 12 (gnd) = gnd

16-bit output port composed of two 8-bit ports
x401 is for low order byte
x409 is for high order byte

condition-mode output interface hardware to issue
signal: x401
device: Intel 8212 8-bit I/O port, IC 83
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (DI:1:8) = DB(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (DO:1:8) = x401(1:8)

if 8 are req
pin 2 (md) = +5V
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 50)
pin 24 (vcc) = +5V
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue
signal: x409
device: Intel 8212 8-bit I/O port, IC 84
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (DI:1:8) = DB(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (DO:1:8) = x409(1:8)

if 8 are req
pin 2 (md) = +5V
pin 11 (stb) = gnd

155
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 51)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
x417 is for low order byte
x425 is for high order byte
condition-mode output interface hardware to issue

signal: x417
device: intel 8212 8-bit i/o port, ic 85
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x417(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 52)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue

signal: x425
device: intel 8212 8-bit i/o port, ic 86
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x425(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 53)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
x433 is for low order byte
x441 is for high order byte
condition-mode output interface hardware to issue

signal: x433
device: intel 8212 8-bit i/o port, ic 87
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x433(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 54)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue

signal: x441

156
device: intel 8212 8-bit i/o port, ic 88
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = do(1:9)
pins 4,6,9,10,15,17,19,21 (do(1:8)) = x441(1:9)

; if 8 are req
pin 2 (md) = 5v
pin 1 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 55)
pin 24 (vcc) = 5v
pin 12 (gnd) = gnd
16 bit output port composed of two 8 bit ports
x449 is for low order byte
x457 is for high order byte
condition-mode output interface hardware to issue
signal: x449

device: intel 8212 8-bit i/o port, ic 89
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = do(1:9)
pins 4,6,9,10,15,17,19,21 (do(1:8)) = x449(1:9)

; if 8 are req
pin 2 (md) = 5v
pin 1 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 56)
pin 24 (vcc) = 5v
pin 12 (gnd) = gnd
condition-mode output interface hardware to issue
signal: x457

device: intel 8212 8-bit i/o port, ic 90
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = do(1:8)
pins 4,6,9,10,15,17,19,21 (do(1:8)) = x457(1:8)

; if 8 are req
pin 2 (md) = 5v
pin 1 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 57)
pin 24 (vcc) = 5v
pin 12 (gnd) = gnd
16 bit output port composed of two 8 bit ports
x465 is for low order byte
x473 is for high order byte
condition-mode output interface hardware to issue
signal: x465

device: intel 8212 8-bit i/o port, ic 91
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = do(1:8)
pins 4,6,9,10,15,17,19,21 (do(1:8)) = x465(1:8)

; if 8 are req
pin 2 (md) = 5v
pin 11 (stb) = gnd
pin 1 (dsl-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 58)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue

signal: x473
device: intel 8212 8-bit i/o port, ic 92
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,9,10,15,17,19,21 (do(1:8)) = x473(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (dsl-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 59)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
x481 is for low order byte
x489 is for high order byte

condition-mode output interface hardware to issue

signal: x481
device: intel 8212 8-bit i/o port, ic 93
connections:
pins 3,5,7,9,16,19,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x481(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (dsl-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 60)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue

signal: x489
device: intel 8212 8-bit i/o port, ic 94
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x489(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (dsl-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 61)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
x497 is for low order byte
x505 is for high order byte

condition-mode output interface hardware to issue

158
signal: x497
device: intel 8212 8-bit i/o port, ic 95
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x497(1:8)

; if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 62)

pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue

signal: x505
device: intel 8212 8-bit i/o port, ic 96
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x505(1:8)

; if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 63)

pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

16 bit output port composed of two 8 bit ports
x513 is for low order byte
x521 is for high order byte

condition-mode output interface hardware to issue

signal: x513
device: intel 8212 8-bit i/o port, ic 97
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x513(1:4)

; if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 64)

pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue

signal: x521
device: intel 8212 8-bit i/o port, ic 98
connections:
pins 3,5,7,9,16,18,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x521(1:4)

; if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 65)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition mode input interface hardware to sense

signal KEYCHA
device: intel 8212 8 bit i/o port, ic 99
connections:
pins 3,5,7,9,16,18,20,22(di(1:8)) = KEYCHA(1:8)
remainder to

ground
pins 4,6,8,10,15,17,19,21(do(1:8)) = db(1:8)
pin 2 (md) = gnd
pin 11 (stop) = gnd
pin 1 (ds1-bar) = .not. (decode a(0:7) value 8)
pin 13 (ds2) = ino .and. doin
pin 24 = +5v
pin 12 = gnd

condition mode input interface hardware to sense

signal KEYCHA
device: intel 8212 8 bit i/o port, ic 100
connections:
pins 3,5,7,9,16,18,20,22(di(9:16)) = KEYCHA(9:16)
remainder to

ground
pins 4,6,8,10,15,17,19,21(do(9:16)) = db(9:16)
pin 2 (md) = gnd
pin 11 (stop) = gnd
pin 1 (ds1-bar) = .not. (decode a(8:15) value 8)
pin 13 (ds2) = ino .and. doin
pin 24 = +5v
pin 12 = gnd

16 bit output port composed of two 8 bit ports
x529 is for low order byte
x537 is for high order byte

condition-mode output interface hardware to issue

signal: x529
device: intel 8212 8-bit i/o port, ic 101
connections:
pins 3,5,7,9,16,18,20,22(di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21(do(1:8)) = x529(1:8)

; if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 66)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition-mode output interface hardware to issue

signal: x537

160
device: intel 8212 8-bit i/o port, ic 102
connections:
pins 3, 5, 7, 9, 16, 19, 20, 22 (di(1:8)) = db(1:8)
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = x537(1:8)

if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 67)
pin 24 (vcc) = +5v
pin 12 (gnd) = gnd

condition mode input interface hardware to sense

signal KEYCHA
device: intel 8212 8 bit i/o port, ic 103
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(1:8)) = KEYCHA(1:8)
remainder to
ground
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = db(1:8)
pin 2 (md) = gnd
pin 11 (stb) = gnd
pin 1 (ds1-bar) = .not. (decode a(0:7) value 9)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd

condition mode input interface hardware to sense

signal KEYCHA
device: intel 8212 8 bit i/o port, ic 104
connections:
pins 3, 5, 7, 9, 16, 18, 20, 22 (di(9:16)) = KEYCHA(9:16)
remainder to
ground
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(9:16)) = db(9:16)
pin 2 (md) = gnd
pin 11 (stb) = gnd
pin 1 (ds1-bar) = .not. (decode a(9:15) value 9)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd

condition mode input interface hardware to sense

signal KEYCHA
device: intel 8212 8 bit i/o port, ic 105
connections:
pins 3, 5, 7, 9, 16, 14, 20, 22 (di(1:8)) = KEYCHA(1:8)
remainder to
ground
pins 4, 6, 8, 10, 15, 17, 19, 21 (do(1:8)) = do(1:8)
pin 2 (md) = gnd

161
pin 11 (sto) = gnd
pin 1 (dsl-bar) = .not. (decode a(0:7) value 10)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd

condition mode inout interface hardware to sense
signal KEYCHA
device:intel 8212 8 bit i/o port,ic 106
connections:
pins 3,5,7,9,16,18,20,22(di(9:16)) = KEYCHA(9:16)
remainder to
ground
pins 4,6,8,10,15,17,19,21(do(9:16)) = db(9:16)
pin 2 (md) = gnd
pin 11 (sto) = gnd
pin 1 (dsl-bar) = .not. (decode a(8:15) value 10)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd

condition mode inout interface hardware to sense
signal MANPOS
device:intel 8212 8 bit i/o port,ic 107
connections:
pins 3,5,7,9,16,18,20,22(di(1:8)) = MANPOS(1:8)
remainder to
ground
pins 4,6,8,10,15,17,19,21(do(1:8)) = do(1:8)
pin 2 (md) = gnd
pin 11 (sto) = gnd
pin 1 (dsl-bar) = .not. (decode a(0:7) value 11)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd

condition mode inout interface hardware to sense
signal MANPOS
device:intel 8212 8 bit i/o port,ic 108
connections:
pins 3,5,7,9,16,18,20,22(di(9:16)) = MANPOS(9:16)
remainder to
ground
pins 4,6,8,10,15,17,19,21(do(9:16)) = db(9:16)
pin 2 (md) = gnd
pin 11 (sto) = gnd
pin 1 (dsl-bar) = .not. (decode a(0:7) value 11)
pin 13 (ds2) = inp .and. doin
pin 24 = +5v
pin 12 = gnd

condition mode inout interface hardware to sense

162
signal KEYCHA
  device: intel 8212 8 bit i/o port, ic 109
connections:
pins 3,5,7,9,16,18,20,22(di(1:8)) = KEYCHA(1:8)
remainder to ground
  pins 4,6,8,10,15,17,19,21(do(1:8)) = do(1:8)
pin 2 (md) = gnd
pin 11 (stb) = gnd
pin 1 (ds1-bar) = not. (decode a(0:7) value 12)
pin 13 (ds2) = inp *and* doin
pin 24 = +5v
pin 12 = gnd

condition mode output interface hardware to sense
signal KEYCHA
  device: intel 8212 8 bit i/o port, ic 110
connections:
pins 3,5,7,9,16,18,20,22(di(9:16)) = KEYCHA(9:16)
remainder to ground
  pins 4,6,8,10,15,17,19,21(do(9:16)) = do(9:16)
pin 2 (md) = gnd
pin 11 (stb) = gnd
pin 1 (ds1-bar) = not. (decode a(8:15) value 12)
pin 13 (ds2) = inp *and* doin
pin 24 = +5v
pin 12 = gnd

16 bit output port composed of two 8 bit ports
  x545 is for low order byte
  x553 is for high order byte

condition-mode output interface hardware to issue signal: x545
  device: intel 8212 8-bit i/o port, ic 111
connections:
pins 3,5,7,9,16,19,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x545(1:8)

; if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (ds1-bar) = wr-bar
pin 13 (ds2) = out *and* (decode a(0:7) value 68)
pin 24 (vcc) = +5v
pin 12 (and) = gnd

condition-mode output interface hardware to issue signal: x553
  device: intel 8212 8-bit i/o port, ic 112
connections:
pins 3,5,7,9,16,19,20,22 (di(1:8)) = db(1:8)
pins 4,6,8,10,15,17,19,21 (do(1:8)) = x553(1:8)
; if 8 are req
pin 2 (md) = +5v
pin 11 (stb) = gnd
pin 1 (dsl-bar) = wr-bar
pin 13 (ds2) = out .and. (decode a(0:7) value 69)
pin 24 (vcc) = +5v
pin 12 (qnd) = gnd

condition node input interface hardware to sense

signal KEYCHA
device:intel 8212 8 bit i/o oort,ic 113
connections:
pins 3,5,7,9,16,18,20,22(di(1:8)) = KEYCHA(1:8)

remainder to

ground

pins 4,6,8,10,15,17,19,21(do(1:8)) = db(1:8)
pin 2 (md) = qnd
pin 11 (stb) = qnd
pin 1 (dsl-bar) = .not. (decode a(0:7) value 13)
pin 13 (ds2) = inp .and. dbin
pin 24 = +5v
pin 12 = qnd
INITIAL DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Address</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, VA 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, CA 93943</td>
</tr>
<tr>
<td>3.</td>
<td>2</td>
<td>Commandant (G-PTE-1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U.S. Coast Guard</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2100 Second Street SW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Washington, D.C. 20593</td>
</tr>
<tr>
<td>4.</td>
<td>4</td>
<td>LTC Alan A. Ross, USAF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Code 52RS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, CA 93943</td>
</tr>
<tr>
<td>5.</td>
<td>1</td>
<td>Capt. Bradford D. Mercer, USAF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Code 522I</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Monterey, CA 93943</td>
</tr>
<tr>
<td>6.</td>
<td>2</td>
<td>LCDR F. Sutter Fox, USCG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c/o Commandant (G-TPP-2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U.S. Coast Guard</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2100 Second Street SW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Washington, D.C. 20593</td>
</tr>
</tbody>
</table>
END

FILMED

1-85

DTIC