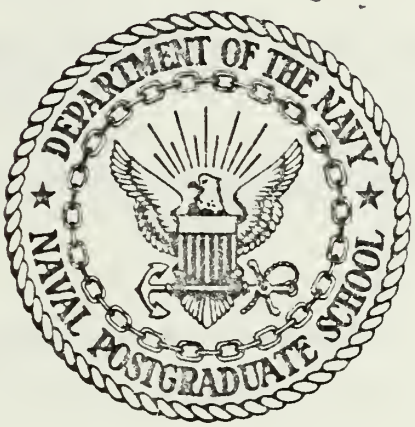
1984

Control system design language implementation of a gas turbine starting controller.

Riley, Richard Preston

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

CONTROL SYSTEM DESIGN LANGUAGE
IMPLEMENTATION OF A
GAS TURBINE STARTING CONTROLLER

by

Richard Preston Riley

June 1984

Thesis Advisor: A. A. Ross

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Control System Design Language Implementation of a Gas Turbine Starting Controller	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis; June 1984
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Richard Preston Riley	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943	12. REPORT DATE: June 1984
	13. NUMBER OF PAGES: 123
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

microcomputer; controller; CAD; CAD/CAM; CSDL; Computer System Design Language; automated design; controller design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis investigates the feasibility and utility of the Computer System Design Language (CSDL) and its design environment. The primary purpose of this design system is to automatically design microprocessor based controller prototypes given a description of the controller's behavior. CSDL is used to create a highly structured behavioral description which is used by the design environment to create a software and hardware listing.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

A "generic" gas turbine engine start malfunction controller is developed using CSDL and tested on a Prolog development system.

Approved for public release; distribution unlimited

Control System Design Language Implementation of a Gas Turbine Starting Controller

by

Richard Preston Riley
Lieutenant Commander, United States Navy
B.S., University of Oklahoma, 1972

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
June 1984

ABSTRACT

This thesis investigates the feasibility and utility of the Computer System Design Language (CSDL) and its design environment. The primary purpose of this design system is to automatically design microprocessor based controller prototypes given a description of the controller's behavior. CSDL is used to create a highly structured behavioral description which is used by the design environment to create a software and hardware listing. A "generic" gas turbine engine start malfunction controller is developed using CSDL and tested on a Prolog development system.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Lieutenant Colonel Alan Ross, and second reader, Professor Daniel Dolk, for their assistance and support in this thesis. I would also like to thank my wife, Donna for her patience, love and endurance (she watched the kids - while I played with the computer).

I. INTRODUCTION

The design of electronic devices for use in control applications has traditionally been a long, tedious process. In the past a designer would specify the desired behavior of the controller. From this behavioral description a definition of the inputs and outputs could be made. Next, functions would be derived mapping the input signals to the output signals using boolean algebra and truth tables. If the design were to be implemented using discrete logic gates, these functions would be used to define a "sum of products" equation describing the gates required to build the controller. Often the original equation could be minimized using Karnaugh Maps or Quine-McCluskey Minimization (reducing the number of gates required) and a savings in cost realised [Ref. 1: pp. 92-126]. Once the design was complete the project could be prototyped for further testing. Any follow-on designs would go through this same process until the desired configuration was obtained.

This traditional view of design grew out of an era when microprocessors and computer aided design principles were not yet accepted practice and when hardware was the primary cost in systems design. Today, the microelectornics industry has reduced the cost of a Zilog Z-80 microprocessor to less than $5.00 [Ref. 2: p. 536] and of a quad two-input

nand gate to less than $1.00 [Ref. 3: p. 124]. Clearly, the major costs of system development will be in the design and fabrication of printed circuit boards, data buses and interface hardware in addition to the generation of software (in the case of microprocessor based controllers), activities which are highly labor intensive. Additionally, Computer Aided Design (CAD) has been used successfully in a number of industries including aerospace, automotive and electronics. These industries have been most successful in implementing CAD in terms of "hardware" (airplane parts, auto parts and comuter circuits). By using computers to selectively access libraries of standard components, industry has been able to cut the costs and time required to develop complicated assemblies. [Ref. 4: pp. 63-66]

In most engineering projects a proof of concept is a required step in development before further expenses are incurred. Design errors must be eliminated early in the life of a project to avoid the excessively high repair costs when a device reaches the production stage. One of the most commonly accepted methods for proving a concept is to build a prototype and exhaustively test it. In designing a prototype the engineer usually reduces the scope of the problem to show only the essential or controversial aspects of the device. This reduces the manufacturing cost and complexity of the prototype. The problem with designing a prototype is that the level of effort required to design and

fabricate s single test device may be extremely high relative to the cost of a project as a whole, particularly if the device is software intensive. Additionally, it is desirable to have as many different implementations prototyped as possible, so one can chose the best design of a given project.

A designer, using traditional methods, would be required to write a behaviorial description, select hardware which best suits the task at hand and then write hardware-specific software to implement the design. This process, when used to produce several prototypes, can take as long as the time it takes to go through full systems development. Computer Aided Design (CAD) techniques can reduce time and cost required to produce workable prototypes by mechanizing the selection of hardware and software elements thus requiring the designer to produce only a behavioral description of the task. High level programming languages also follow this pattern of automated design in the sense that a library of machine or assembly language code is used to assemble a machine readable program from a "high level" problem description. The programmer is still responsible for defining the problem, but instead of having to do so at the machine level he can use a more "user friendly" medium-the high level language.

Automated design is being applied to the design of microprocessor based controllers through research currently being conducted at the Naval Postgraduate School under the

direction of LTC Alan Ross [Ref. 5]. This research project is an outgrowth of a doctoral thesis authored by Ross in which he implemented automated system design principles proposed by Matelan. [Ref. 6] This design environment requires a behavioral description in a high level language as the only input to generate both a hardware listing and a software listing which can then be compiled and linked for the controller under design. Ross's system, called the Computer System Design Language, is described in the next chapter.

The purpose of this research project is to validate the CSDL system by designing and building a prototype of a microprocessor based controller for a generic gas turbine engine using Ross's design environment. To date, two other projects have used Ross's system to design microprocessor based controllers but none have actually compiled source code or constructed the hardware necessary to support the design. This project will attempt to build a prototype of a microprocessor based controller solely on the hardware and software descriptions generated from CSDL using the Z-80 realization library.

II. BACKGROUND

A. CSDL

The CSDL concept is based on a design language proposed by M.N. Matelan [Ref. 7], Livermore Laboratory, in 1976. In his design, Matelan envisioned a design environment in which the user provides a description of a system using an input language. The design environment would then produce a microprocessor based prototype. In 1978, LTC Alan Ross [Ref. 8] further developed this concept and added the ability to build miltiprocessor based systems as well. This new language, designated Control System Design Language (CSDL), maps user defined contingency-task pairs to a "realization library" containing the software and hardware primitives (much like a compiler) to produce a program and hardware listing. To date, the primary use of CSDL has been in the development of prototypes of real-time controllers. This system can be extended to a much broader class of problems by incorporating the appropriate realization library.

CSDL is divided into five sections. The Identification Section contains a brief description of the control problem under consideration, the designer's name, the version or iteration number, revision information and other remarks. This section identifies and documents the design, but does not provide any information to the design system.

The Design Criteria Section provides the designer with a procedure to prioritize the choice of an implementation based on cost or power consumed or to select the first realization generated. This is the only section in which the designer has any input to CSDL concerning the choices of hardware or software primitives it will make in generating a system.

The Environment Section contains a declaration of all design variables. These design variables are defined in terms of the type of signal(s) generated or sensed (ie. TTL, RTL, etc.), type of arithmetic, precision and coding (ASCII, EBCDIC or BCD). This section is analagous to the declaration statements in PL/I or Pascal.

The Contingency Section contains declarations of those conditions that the device must respond to, the associated task that must be executed for each contingency, and the time constraints imposed upon each contingency-task pair. The timing constraints are determined by the maximum time allowed to recognize a contingency and the maximum time available to execute the corresponding response. Conditions include constructs such as if-then, do-while, and repeat-until.

The Procedures Section contains the routines which implement contingencies and tasks. By definition, contingencies are written as functions, while tasks are specified as procedures. Each routine contains the high level language descriptions necessary for the performance of its role in the system being produced.

B. DESIGN ENVIRONMENT

1. Overview

Once the designer completes the CSDL description of his problem, it can be processed through the design environment. Figure 1 shows a block diagram of the design environment. The user interface is through the CSDL description. Once entered into the system the CSDL description is decomposed into various symbol tables by the translator. The functional mapper maps each contingency-task pair to a set of primitives from the primitive library and analyzes the timing requirements. If the primitive chosen from the library meets the timing requirements, the process continues until the contingency-task list is exhausted. A monitor is then generated which will poll through each contingency within the time limits specified in the Contingency Section of the CSDL description. A final timing check is made to determined if the total system falls within the timing parameters set forth in the description. Software and hardware listings are generated as output, complete with an executive to drive the prototype. If the timing parameters cannot be met or a primitive cannot be matched in the library, the process aborts and attempts to use another realization library if one is available. There are realization libraries currently available for the Intel 8080 and the Zilog Z-80 processors and one under development for the Intel 8086.

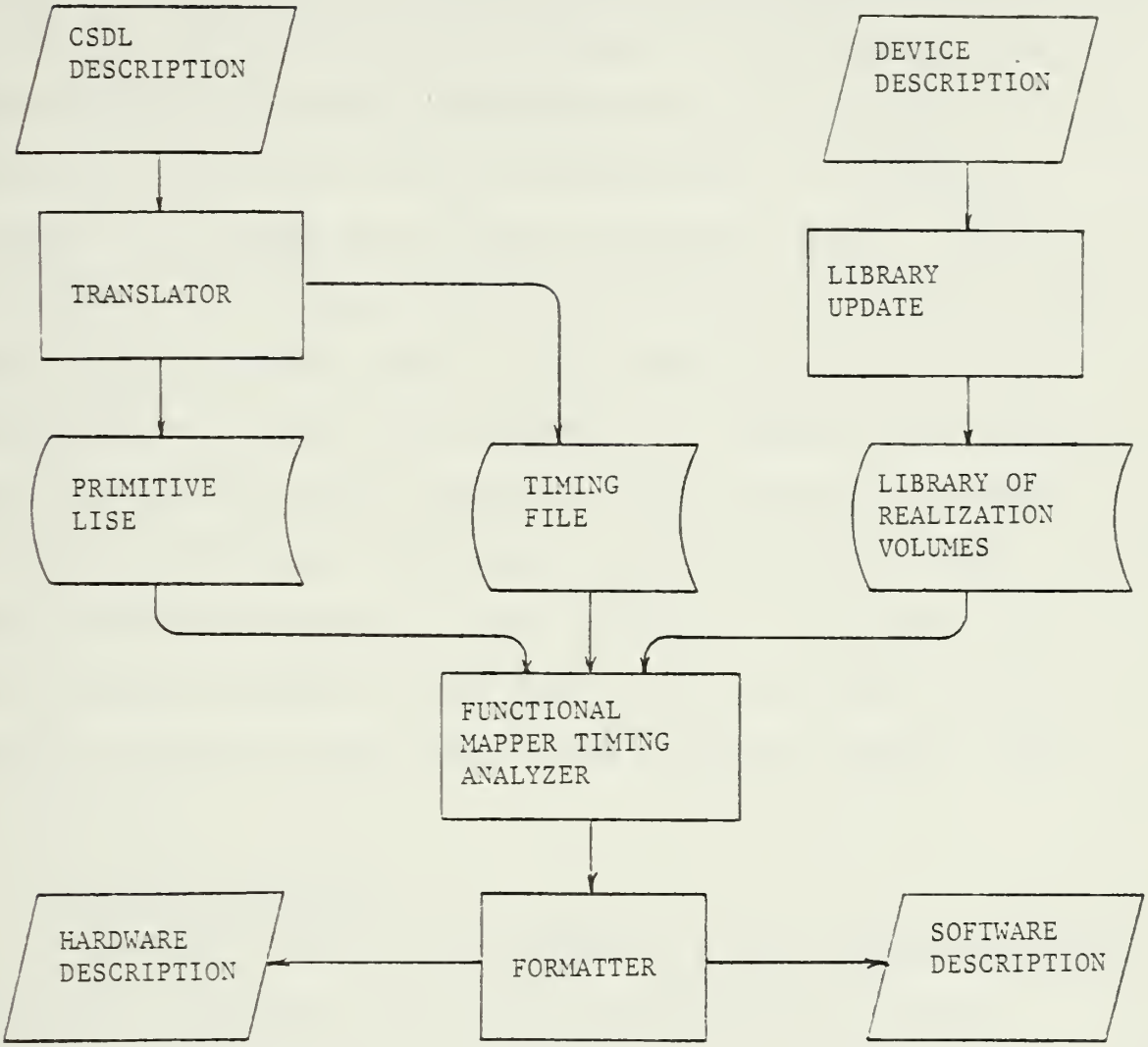


Figure 1 Current CSDL Design Environment

2. Functional Mapper

The Functional Mapper is the first module called by the design environment in its current configuration. Its main function is to map each primitive in the problem description (procedure section translated to a primitive listing) to a primitive in the realization library. The realization volume index is first searched for the primitive name from the current line in the primitive listing. The specifications within the primitive are compared with those found in the realization library. The criteria must match exactly or the design environment will abort the current design run and generate an error message. The output from the Functional Mapper is the FOR021.DAT file which is the Current Realization Table discussed in Ross's thesis. [Ref. 9: pp. 65-90]

3. Timing Analyser

The next module to be executed is the Timing Analyser. This module generates the monitor instructions to ensure that the implementation will meet the timing requirements set in the CSDL description. Since this monitor is a polled loop, this module assumed worst case conditions and assumed that all contingencies will be true and that their associated tasks are performed. The timing analyser calculates the length of time required to perform each contingency-task pair and compares this time with the time contained in the IADEFL file. This file is generated by the designer along

with the CSDL description in the current version of CSDL but will be developed from the CSDL description when the Translator module is complete. The IADEFL specifies the units of time (miliseconds, microseconds, etc.), the total period of time allowed for the contingency task pair, the time allowed for the contingency and task as separate structural units, the time for a timed block within the contingency, the time for a timed block within a task and a flag to indicate a background task with no specified time requirements. It should be stressed that the times listed in the IADEFL are maximum times and cannot be used to define events which require precise time intervals.

The output of this module is a set of monitor primitives which are appended to the primitive list provided by the designer. The current version of the design environment will support dual processor realizations if the timing requirements cannot be met with a single processor. The contingency-task pairs are divided among the two processors and separate monitors provided for each processor. This process can be forced by the designer if the time required to process each contingency-task pair is known. The timing parameters listed in the IADEFL can then be set artificially high, forcing the design environment to generate a dual processor implementation.

4. Formatter Module

The Formatter requires the primitive list, complete with the monitor, and the current application timing table from the Timing Analyzer. It then sequentially processes the "raw" primitive listing and uses the line numbers located in the application timing table as guides to enter the realization volume to extract the programming code from each primitive. The software and hardware listings are output to two separate files, FOR046.DAT for software and FOR047.DAT for hardware. The formatter is also responsible for splitting the "raw" primitive listing into two separate software listings if a dual processor implementation is generated. The design process is then terminated.

C. REALIZATION LIBRARY

Next to the actual design environment itself, the most important files in this system are the realization libraries. Each realization library contains the primitives necessary to implement all mathematical functions, all conditional testing, all logic functions and all hardware requirements for each software primitive for each microprocessor type (ie. Zilog Z80, Intel 8080, etc.). There are currently two volumes written, one for the Intel 8080 cpu and one for the Zilog Z-80 family. An additional library is under development for the Intel 8086 cpu.

The general format for each primitive is the same and must be stored as 80 column card images. The first 5

columns in each line serve as line numbers and are used for indexing by the design environment. The index to the realization volume resides in the first few lines of the realization volume and contains reproductions of the first line in each primitive. Their respective line numbers are retained and serve as pointers to each primitive in the file. Each realization library is limited to 9999 lines (the index is considered in the line count when the number of lines in a library is calculated).

There are ten specific formats which are recognized by the design environment: Primitive Title line, Comment line, Calc line, Attr line, Call line, Include line, If line, Begin Text line, End Text line, and Text line. The title line must contain an s or h (in lower case), to denote hardware or software, followed by the name of the primitive. The calling arguments, selection criteria, and attributes of the primitive are enclosed in parentheses following its name. The attributes are used to specify power consumption, latency and chips used. Attributes are unique for each primitive. Any or all of the attributes may be omitted but the commas that separate them must appear. The comment line is denoted by COM as the first characters of a line. The design environment ignores these lines and they produce no output. They are there to help document the code within the primitive. The Calc line allows the use of global variables within the system. An example of the use

of this variable is the ROM pointer which is used to keep track of the next address available in the controller's memory. The Calc line calculates the next position in memory based on the start location it receives from GLOBALS. The Attr line is used to calculate a value for an attribute which depends on the arguments passed to the primitive. Incl and Call lines are used to invoke other primitives from within a primitive. The difference between the two is that the output from a Call is inserted immediately following any previously generated output, the output from an Incl statement will be added to that of the primitive list after all other output from the including primitive has been produced. The If line provides a mechanism for the branching of instructions within the realization library. The Begin and End Text lines are used to mark the beginning and ending point of the code that is to be included as output from the design environment. The construction of primitives involves a detailed understanding of the assembly language for the particular microprocessor in use. Each primitive will vary in length depending on the complexity of the task it is required to perform. Figure 2 is an example of the s.add primitive from the current Z-80 library authored by Smith [Ref. 10: pp. 60-103]. The methodology involved in constructing a primitive listing is discussed by both Ross and Smith and will not be discussed in this thesis.

```
v0186s.add(rslt,arg1,arg2:0,16,0,16,0,16:
          13,71,21,9,0,186,195)
v0187com primitive to add arg1 and arg2 and store in rslt
v0188com list-rslt,arg1,arg2:precisions:s,t,e,c,i,addr
v0189 begin stext
v01901d hl, (<arg1>);6m  20t  4b  load arg1 in hl pair
v01911d bc, (<arg2>);6m  20t  4b  load arg2 in bc pair
v0192add hl, bc     ;3m  11t  1b  add
v01931d  (<rslt>),hl:6m  20t  4b  save result
v0194endtext
v0195calc romptr=romptr+13
```

Figure 2 S.ADD Primitive

The current version of the design environment has not implemented the translator or the library updater listed in Figure 1. There are two thesis projects under way which will provide a compiler and a user friendly front end to provide all the functions of the translator. The lack of a translator requires that any CSDL description processed by the design environment be hand compiled (create the primitive listing) before submission to the design environment.

D. PREVIOUS PROJECTS

In addition to the examples processed by Ross in his dissertation, two other thesis projects have used CSDL as a design tool. Pollock [Ref. 10] attempted to construct a microprocessor based fuel injection controller for a Datsun 280Z. He was not successful but did add numerous hardware and software primitives to the existing 8080 library to enable floating point operations and trancendental functions. He pointed out the need for a debugging tool to allow dynamic debugging of the system as well as the need to provide a structure to block code into groups. The present implementation of CSDL provides a polled monitor which will not break the monitor cycle into dependent modules. Dependent tasks must be nested within each contingency to ensure that they are completed. Pollock's primary contribution was the creation of a utility to format the realization library and make its construction easier. [Ref. 11]

Heilstedt [Ref. 12] expanded the horizons of possible applications by using CSDL to design digital filters based on the 8080 library. Heilstedt implemented the CSDL design environment on a Digital Equipment VAX 11/780 by converting a version of the design environment he received from Lawrence Livermore Laboratories, and to date, a complete validation of this program has not been done. Conversion problems were encountered which kept realizations from being generated when they were, in fact feasible. These

errors were caused by incompatibilities between the Livermore computer and the VAX used at the Naval Postgraduate School. It was impossible to fully exercise all branches of the design environment so some errors may still exist within the rarely used subroutines of the design environment. He did however, show that CSDL can be used to support devices other than microprocessor based controllers.

III. DESIGN

A microprocessor based digital controller may be regarded as a programmable electronic device which processes instructions in a sequential manner, senses signals from its environment and issues signals to affect (control) its environment. The environment consists of a set of electronic (analog and digital) devices or mechanical devices with appropriate interfaces. Microprocessor based controllers can be used to replace digital controllers based on discrete logic gates and offer the advantage of easily changing the logic driving the controller. Digital controllers using discrete components are "programmed" during the design phase when interconnections are designated between atomic parts. Reprogramming is difficult due to the many wiring changes that are required. In some instances, controllers can be designed with discrete components and a plug board arrangement included to provide some limited programming. The result of using discrete components is a very limited ability to reprogram. For prototypes, the ability to change the logic driving the controller is an extremely attractive concept as the behavior of the controller can be changed without having to rewire the prototype.

Projects using CSDL in the past have shown the ability of the language to define microprocessor based controllers. They have not shown that the design environment can produce a working prototype controller. All projects attempted thus far have relied on using separate, discrete components in the definition of hardware requirements within the realization library thus increasing the scope and difficulty of a given project. To reduce the complexity of this effort, the Prolog Standard Bus development system was chosen as the target machine [Ref. 13]. This development system consists of a card cage, power supply, Z-80 cpu card, keyboard card with a 20 key keypad, an alphanumeric readout with 8 light emitting diodes, and a dual UART using the standard RS-232 interface. The Z-80 realization library is constructed using the development system components (cards and cardcage) rather than discrete components. The Prolog system was chosen because of the local support available (Prolog systems are manufactured in Monterey) and the existence of several Prolog systems in the Electrical Engineering Department.

Pollock recognized that one might have problems in testing any design built from the CSDL design system and provided primitives to allow the setting of break points within software and a terminal connected to an RS-232 interface built into the controller. None of these primitives were tested in an actual implementation. LCDR

Stephen Hughes [Ref. 14], in a Naval Postgraduate School master's thesis, designed a software package which enables an Altos 8 bit microcomputer to communicate with the Prolog development system through a dual channel RS-232 port. One port connects to the Altos which acts as a host terminal and provides uploading and downloading of programs from the Altos memory or disk drives. A second port provides a connection for an ADM-3A terminal which provides a means to control the Prolog system when disconnected from the Altos. Hughes developed a PROM that enables the use of standard CP/M BIOS calls for input/output to the Prolog system. The utilization of a CP/M like operating system allows the use of several debugging aids like ZSID and DDT on software before it is downloaded to the Prolog. This feature proved to be extremely useful as this project developed.

Debugging in the CP/M environment is easily done using standard debugging software tools. A tool or set of tools is required to allow debugging on the target machine as well. The PROM developed by LCDR Hughes, installed on the Z-80 card, acts as a primitive monitor program for the Prolog system and provides functions for dumping memory, calls to the BIOS, changing values in memory and filling sections of memory with specific bytes. The example chosen to demonstrate the design environment's ability to design working controllers is based on the engine start sequence used in the Lockheed P-3 Antisubmarine Warfare aircraft.

The P-3 uses an Allison T-56 series gas turbine engine. For the purposes of demonstration the full T-56 implementation will not be discussed as it would add unnecessary complexity; a generic gas turbine engine will be described and used as the target. A complete discussion of the physics envolved in the jet cycle will be avoided and only the most salient points given.

The start controller is based on a "generic" axial flow gas turbine engine. The engine superficially resembles a T56-A-10 as used on Lockheed P-3A aircraft. During engine starts, the pilots and flight engineer are required to monitor numerous gauges widely spread apart in the cockpit. The chance of error in rapidly reading these gauges during the start sequence is high and increases with crew fatique.

The cockpit of the P-3 aircraft is configured such that the pilots sit on either side of a center pedestal which is approximately 30 inches wide. The flight engineer (the person who actually performs the engine start) sits immediately behind the pedestal. The primary engine instrumentation is located on a vertical dash panel in front of the pedestal. The view for all cockpit personnel is unobstructed to these instruments which include a Shaft Horsepower gauge, an RPM indicator, Fuel Flow indicator and Turbine Inlet Temperature indicator for each of four engines. The starter panel is located above the flight engineer on a overhead panel which is difficult for the

pilots to see. It contains the start selector switch and starter button. Additional instrumentation is located above the copilots seat on the right side of the aircraft and includes oil pressure and oil temeprature gauges. Other gauges on the overhead panel include bleed air pressure (air pressure is used to turn the starter on the P-3 engines), and miscellaneous indicator lights which will not be described as they are not within the scope of this project.

The basic start sequence is done entirely by the flight engineer but is monitored by both pilots. Due to the distance and viewing angles of the instruments, errors are likely to occur causing unnecessary shutdowns and delays in departure. These same instruments are used when restarting an engine in flight and the cost of errors in this environment pose a potential threat to flight safety.

Once the appropriate checklists and pre-start briefings are completed, the pilot in command directs the flight engineer to start the engines (or engine). The flight engineer first sets up the bleed air panel to allow high pressure air to reach the start selector valve. The fuel and ignition switch is placed in the "on" position to allow fuel to reach the combustion chambers and the ignitors to operate when sequenced by the speed sense control. An engine is selected with the start selector switch and the starter button pushed. This allows high pressure air to turn the starter. As the engine accelerates, the RPM

indicator is scanned by the cockpit crew. At 16 percent of normal RPM there must be indication of fuel flow and stabilized high pressure air to the starter. Oil pressure and TIT should be rising as ignition occurs in the burner cans causing the engine to rapidly accelerate toward a low RPM setting of a 72 percent of normal RPM. The start is considered complete when the engine reaches a stable low RPM condition with the start valve closed, air pressure returns to approximately 60 pounds and the ignitors are off.

The engine consists of three main sections: 1. compressor section, 2. combustion section and 3. turbine section. Air enters the compressor section and is compressed by a factor of 9.5:1. This increases the air density and raises its temperature 547 degrees above ambient. This highly compressed air enters the combustion chamber where fuel is injected and ignited by ignitors during the start sequence. The process of ignition is self-sustaining once the ignitors light the fuel/air mixture. The hot gases produced in the bombustion section pass to the turbine section where energy is extracted to turn the propeller, compressor and accessories (fuel control, pumps etc.).

During the start sequence readings are taken from the following gauges: fuel pressure, rpm, TIT (Turbine inlet temperature), oil pressure and clock. Figure 3 contains a summary of sensors for the start controller.

Signals	Signal Type	Allowable Limits
1. Fuel Flow	Analog	0-2000 lb/hr
2. TIT	Analog	0-1200 deb cent.
3. Ignitors	Digital	on-off
4. Oil Pressure	Analog	0-100 lbs/sq inch
5. RPM	Analog	0-120%
6. Start Switch	Digital	on-off
7. Fire Sensor	Digital	on-off

Figure 3 Start Controller Signal Summary

There are eight start malfunctions which can occur and require immediate shutdown to avoid engine damage. They are a turbine inlet temperature overtemp, stalled start, fire, low oil pressure, engine overspeed, no ignition, stagnated start and ignitors on after start. A fire condition requires the activation of the fire extinguishers in addition to a normal shutdown. Malfunctions used in this problem and the signals that will be used are summarized in Figure 4. These malfuncations are described in more detail in the following chapter.

Malfunction	Action (output)	Sensed From
1. TIT overtemp	Shutdown	TIT > 760deg
2. Stalled Start	Shutdown	RPM stabler at <60%
3. Fire	Shutdown/Activate fire extinguishers	Fire sensor
4. Low oil press	Shutdown	Oil pressure
5. Overspeed	Shutdown	RPM > 74%
6. No ignition	Shutdown	(Ignitors on and TIT>100) and RPM>16%
7. Stagnated Start	Shutdown	(PRM Stable < 60%) and TIT > 760 deg
8. Ignitors on after Start	Shutdown	(RPM > 65%) and ignitors on

Figure 4 Malfunction Summary

IV. DETAILED DESIGN

The actual design of a microprocessor-based controller using the CSDL design system starts with the conversion of the problem description into the Computer System Design Language representation. The syntax of the language is defined in Appendix A of Ross's doctoral dissertation [Ref. 15: p. A-1, A-7]. Chapter III of this paper describes the problem that this chapter will be used as an example design.

Figure 5 is a flow chart of the engine start controller. It consists of an initialization block to initialize all variables within the controller. At this point in the design process, the designer probably will not have an accurate picture of all variables required to express the problem. If the initialization block is not required it can be removed at the end of the design process. Initialization will generally consist of setting clocks or test parameters within the controller. After the initialization block a test to determine if the start switch has been pushed is performed. If the start switch has not been pushed the program loops back and continues the test until the start switch has been pushed. After the start button has been pushed, the monitor will move on to the next contingency test in the sequence. CSDL builds a polling loop as a monitor so the tests are

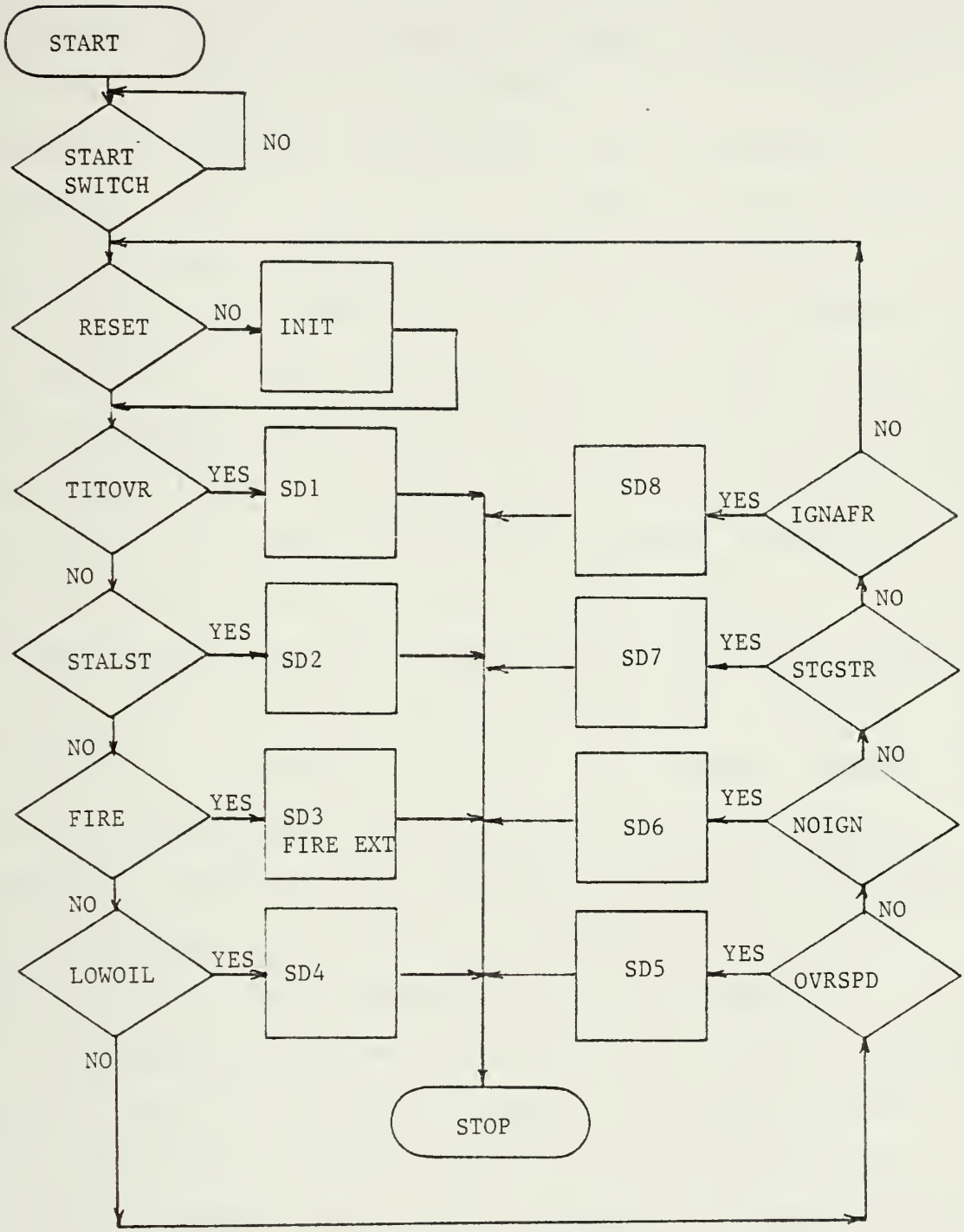


Figure 5 Start Controller Flowchart

arranged in a loop with a test (contingency) for each malfunction listed in a sequential manner. There are only two tasks to perform in this implementation, an engine shutdown or an engine shutdown with fire extinguisher activation. When a test fails, the flow of control passes to the next test in the sequence in an endless loop. Contingencies are tested until the engine start sequence is complete or until a malfunction occurs. When a malfunction occurs, a contingency test will be true and the associated task will be executed leading to an abnormal shutdown. At the end of the start sequence, mechanical devices trip an electrical switch which releases a solenoid within the start switch causing the start switch to "pop" and disconnect the start controller. With the completion of a systems flow chart, the designer should have a good understanding of what functions the controller is expected to perform and the types of signals are available for processing. The designer can then begin work on the generation of a CSDL representation of the problem.

The Identification section of the CSDL listing contains the designer's name, date of creation and project name. Since some form of version control is required for any project, a version number can be appended to the project name with no affect on the design environment. For this problem, the Identification section appears in Figure 6.

```
IDENTIFICATION
    Designer: Richard Riley
    Date: 1 Nov 83
    Project: Start Controller - Version 1.4
```

Figure 6 Identification Section

The Design Criteria section is the only section in which the designer can choose (to a limited degree at least) how the controller is to be built. In this design the metric parameter is listed as "first" to force the use of the first implementation generated. The design environment is capable of producing more than one implementation with the use of multiple realization libraries. We are using only one library, the Z-80, so this parameter is listed as "first". The volume parameter indicates the rankings (order) of the realization volumes for the design environment to use. In this case there are two volumes available, the 8080 and the Z-80, but only one will be used and the volume parameter is set to one. The monitor parameter is used to select the monitor strategy to use. So far, the design environment only supports a polled monitor and thus this parameter is also set to one. In future versions, the design environment will support an interrupt driven monitor and the design

criteria parameter can be used to choose the desired strategy. The Design Criteria section for this problem appears in Figure 7.

```
Design Criteria Section
    Metric = First ;
    Volumes = 1 ;
    Monitors = 1 ;
```

Figure 7 Design Criteria Section

The Environment section contains a description of the variables required by the controller to keep track of the signals it receives, test them and ultimately produce some sort of output. Since output signals must be generated from any controller, it makes sense to define them first, determine what input signals are required to make decisions and generate the output signals and finally to define the temporary variables (Arithmetic in CSDL terms) for use internally by the controller. These signals must be described by name, bit size, and signal type (TTL, RTL etc.).

There are basically two types of output signals used by this project, shutdown and fire extinguishing. For testing purposes, shutdown signals are designated for each type of

malfunction. In an actual design only one shutdown signal would be defined. From the flow chart in Figure 5 it can be seen that eight shutdown signals are defined. Since the object of this signal is to set a switch to deactivate the start sequence we need only define a single bit for testing purposes. For compatibility with the Prolog system input/ output channels, all signals types for this controller will be defined as TTL.

The input signals needed from the engine are RPM, oil pressure, ignitor switch on-or-off indication, start switch on-or-off indication, turbine inlet temperature (TIT) and fire sensor indication on-or-off. Additionally a reset signal has been added to aid in resetting the controller for testing purposes. In an actual implementation this signal would be excluded and the initialization sequence started when power is applied to the engine with a start selector switch.

The internal signals required for this controller are a clock signal for elapsed time and a flag for the stagnated start contingency. CSDL does not support a contingency which requires a precise elapsed time. Timing constraints given in the problem are used to build the monitor and to define the maximum time allowed between each contingency. The realization library contains a clock primitive which uses the counter-timer chip on the Z-80 card in the Prolog system. This signal is generated internal to the Prolog and

is defined in the arithmetic section. The stagnated start flag is a variable and is not sensed from the environment and thus must be defined within the arithmetic section as well.

The complete Environment section for this problem appears in Figure 8.

```
Environment
   Input:
        RPM,8,TTL ;Fire_Sense,1,TTL ;
        Oil Pres,8,TTL ; Ignitor,1,TTL ; TIT,16,TTL ;
        Start_Switch,1,TTL ; Reset_Switch,1,TTL ;
   Output:
        SD1,1,TTL ; SD2,1,TTL ; SD3,1,TTL ; SD4,1,TTL ;
        SD5,1,TTL ; SD6,1,TTL ; SD7,1,TTL ; SD8,1,TTL ;
        Fire_Ext,1,TTL ;
   Arithmetic:
        Clock,16,TTL ; STAGFLG,1,TTL ;
```

Figure 8 Environment Section

The Contingency section contains the timing requirements for each contingency/task pair. This section is built from the flow chart by using the diamonds as the contingency and the associated process box, in this case a shutdown or a shutdown and a fire extinguisher activation, as the task to be accomplished. The timing listed in this section determines how the design environment will construct the monitor and is used to determine if a feasible implementation can be constructed by adding up the times accumulated from the primitive list and comparing the sum to the times accumulated from the contingency list.

The first contingency/task pair encountered in this implementation is the test for a reset switch. Its associated task is to perform an initialization of the shutdown variables, clock, reset switch and to sense the start switch. An arbitrary time value of 100 milliseconds was chosen. The CSDL construct used for this example is the When-Do which will perform the task when the contingency is true (when the reset switch is pressed do the initialization task). In the CSDL syntax it appears as:

```
CONTINGENCY
   When Reset_Switch (100ms) do INIT;
```

The next contingency to be developed is for the clock. As mentioned above a primitive exits to perform the clock function but the timing is not supported by the design environment. The purpose of the clock is to keep track of time, and it must run for at least 1 minute - the maximum time a start will be allowed to continue if the RPM has not reached 72 percent of normal RPM. The clock must be initialized and the time accumulated in a variable to allow the stalled start and stagnated start contingencies to activate if necessary. The time is updated every second so the contingency must be timed at 1000 milliseconds. The task in this case is the clock function and there is really no contingency to test as we simply want to perform the clock function. The dummy contingency "every" is used in this case to keep the contingency/task pairing intact. The CSDL representation is:

every (1000ms) do clock;

The next contingency is the turbine inlet temperature (titovr) test. Once again this contingency task pair will use the "every" form to force the monitor to execute this task a minimum of every 100ms. In fact all the remaining tests will use the now familiar "every" form with a test period of no greater than 100ms. The stalled start task (stalstr) tests whether the engine has accelerated beyond 60 percent of normal rpm within 60 seconds. The fire task determines whether any of the fire sensors attached to the engine has shorted indicating a fire is present. The low oil pressure task (lowoil) determines if sufficient oil pressure is present in the engine to continue with the start sequence. The overspeed (ovrspd) task determines if the engine is accelerating beyond 74 percent of normal rpm indicating a failure of the engine speed governing system. The no ignition task (noign) determines whether the ignition has taken place by testing for rmp greater than 16 percent of normal and tit greater than ambient. The stagnated start (stagstr) task terminates the start sequence when rpm is less than 65 percent of normal and the tit is greater than 760 degrees. This condition will generally occur within the first forty seconds of the start and a clock value of forty seconds is included in the test. The ignition after start task determined whether the ingitors have shut off after the engine has started by sensing rpm greater than 65 percent of

normal and the ingitors switch in the on position. The complete contingency list appears in Figure 9.

```
Contingency:
   When Reset_Switch (100ms) do INIT ;
   Every (1000ms) do CLOCK ;
   Every (100ms) do TIT OVR ;
   Every (100ms) do STALSTR ;
   Every (100ms) do FIRE ;
   Every (100ms) do LOWOIL ;
   Every (100ms) do OVRSPD ;
   Every (100ms) do NOIGN ;
   Every (100ms) do STAGSTR ;
   Every (100ms) do IGNAFTR ;
```

Figure 9 Contingency Section

The Procedure section contains the contingencies and tasks to be performed when called upon by the monitor. The contingencies and tasks listed here form the logic by which the behavior of the controller is determined and is the portion where the most creativity on the designer's part is required.

The first functional element in the procedure section is the function Reset_Switch. Its only function is to sense the position of the reset switch and set a flag which will be tested by other functions and tasks in the procedure section. It is developed using the IF-THEN construct from Appendix A in Ross's thesis. [Ref. 14: p. A-3] In this construct a condition is tested by the IF and if true the

task after the THEN is executed. In each functional element a SENSE command is used to sense the variable in question. It may exist in a memroy location developed by the design environment or it may be taken from an input/output port known to the design environment. In either case the designer is only responsible for designating its existence, the design environment works out the details of where to find it. The function for Reset_Switch appears in Figure 10.

```
Function Reset_Switch;
   IF Reset = 1 then Reset_Switch = 1 ;
       ELSE Reset_Switch = 0 ;
   END IF;
End Reset_Switch ;
```

Figure 10 Reset Switch Function

The next functional element is a task to perform the initialization of variables. This task is fairly simple and consists mainly of assignment statements. At the end it senses the position of the start switch and sets the reset switch variable back to zero to avoid reinitializing the variable during subsequent processing. The structure of this task is:

```
Task INIT;
   CLOCK = 0 ;
   SD1 = 0 ;
   SD2 = 0 ;
   SD3 = 0 ;
   SD4 = 0 ;
   SD5 = 0 ;
   SD6 = 0 ;
   SD7 = 0 ;
   SD8 = 0 ;
   SD9 = 0 ;
   Reset_Switch = 0 ;
   Sense Start_Switch ;
```

Figure 11 Task INIT

The clock function is simulated by using a simple counter which increments the value of the clock at one second intervals. The IF-THEN construct is used to test the start switch position. If it is in the start position (true) the clock will start counting from zero. The construct for the clock function appears in Figure 12.

```
FUNCTION CLOCK:
   IF START_SWITCH THEN
      CLOCK=CLOCK + 1;
   END IF;
END CLOCK;
```

Figure 12 Function CLOCK

The turbine inlet temperature overtemp function uses the IF-THEN construct to test the value of the tit against a constant of 760. When the tit is greater than 760 a shutdown condition exists and the controller will initiate a shutdown. The task defined for titovr is given in Figure 13.

```
Task TITOVR:
    IF START_SWITCH then
        Sense (TIT) ;
        IF TIT > 760 THEN SD1 =1 ;
        ISSUE SD1;
        END IF;
    END IF;
END TITOVR;
```

Figure 13 Task TITOVR

The task for stalled start follows the same general format as the previously detailed structures in the Procedure section. The only difference is the use of a boolean operation (and) within the IF-THEN construct to test for two conditions to execute a shutdown sequence. The task for STALSTR appears in Figure 14.

```
TASK STALSTR :
    IF Start_Switch THEN
        Sense (RPM) ;
        IF CLOCK >= (60) and RPM <= (60)
           THEN SD2 = 1 ;
        END IF;
        ISSUE SD2 ;
    END IF;
END STALSTR;
```

Figure 14 Task STALSTR

By now a pattern should be apparent. The IF-THEN construct has been used heavily in this example to provide a means to test conditions before executing a shutdown sequence. This simple construct is used in all eight malfunctions that this controller will test for. A complete listing of the CSDL description appears in Appendix A.

The modularization of the monitor would allow the creation of an initialization module to initialize all variables, a start module, a run module to actually carry out the control desired and a reset module to restart the controller in the event of a failure (power fault, restart after some malfunction, etc.) This concept of modularization is not supported in the present implementation of CSDL. Although we thought this idea could be support through a series of primitives in the realization library, attempts to construct primitives to perform this function proved fruitless.

Once a CSDL representation of the design problem is generated the Environment, Contingency and Procedure sections must be converted into a primitive listing. The present design environment does not include a compiler, so for the time being, the CSDL description must be hand compiled. Most of the frustration with this project was a result of this process.

The primitive listing should be viewed as an assembly language program with the primitive names or titles being

the instruction set. The code that comprises each individual primitive is analagous to microcode. This view of the primitive list allows for an easier transition into the world of hand compiling. When code is compiled using an automated compiler certain rules for syntax must be followed so the compiler can properly choose the correct code to use. When hand compiling, no such rules exist and it becomes a more creative process as a result.

The syntax used within the CSDL description in Appendix A is the result of work being performed by LCDR Hill Carson in conjunction with the CSDL project. LCDR Carson has formalized the CSDL syntax and is in the process of writing a syntax directed compiler for the computer system design language. When completed it will take a CSDL description and compile a primitive listing which can be directly input to the design environment.

The primitive listing is one of two files the designer must input to the design environment. A few words of caution are in order at this point. The design environment resides in a file called NEWCSCL on the VAX under the VMS operating system. NEWCSDL is written in FORTRAN and is highly column dependent. The primitives in the primitive listing start in card column 6 with a lower case "s" or "t". NEWCSDL will not process uppercase letters so all references to primitives and variables must be in lower case. All names for varialbes must be less than 6 characters long;

names for procedures may be up to 10 characters long. There are essentially two primitive formats to be concerned with. The "t. generated for:<title>" is a title line and informs NEWCSDL that a new procedure is being generated. The "t" must appear in column 6 and the first letter of the title in column 23. The other format is for the "s.primname (operands: selection information)" primitive. The "s" must appear is column 6 and the "(" in column 18. An improper line error will result if these column dependencies are not recognized.

The CSDL description does not contain the data to make the calls to the primitives that generate a monitor for the controller. The first two primitives listed in every primitive list must be:

```
t.generated for  :system    **********************
s.main       (::)
```

The "t.generated for" is a title line which will name the contingency "system" and "s.main" is the task to build the monitor. Note the colons in the s.main primitive. These must appear in every primitive. The colons are used to separate variables, parameters and attributes of the primitive. If these values are null, the colons must still appear or an "improper line" error will result. The stars at the end of the "t.generated for" are used as markers to aid in reading the primitive listing and have no effect on NEWCSDL.

Each identifier defined in the environment section will need to be defined as a variable or one of two types of constants. Variables are assigned to high memory locations just below the stack area. Constants are assigned in low memory. The "s.initialcons" and "s.initialend" primitives were built to allow code which only needed to be executed once at the beginning of a program to be blocked together. In an initial run of a test program, attempts were made to use these primitives to "block" the initialization of variables and constants. The end result is that the variables appear in the assembled Z-80 code as variables but when the program is executed, the computer interprets these memory locations as instructions leading to disastrous results. This example is compiled using "s.initialcons" and "s.initialend" to show the results in the compile code. A separate test case is generated in the next chapter to show the proper method to declare variables.

The environment section generally will not include all the temporary variables required. As each structure within the procedure section is converted to a primitive list, a running listing of all temporary variables should be compiled. The syntax for the "s.var", "s.assigncons" and "s.cons" primitives are listed in Figure 15.

```
s.var          (varname:precision)
s.assigncons(varname,valueassigned:prec,prec)
s.cons(varname,valueassigned:prec,prec)
```

Figure 15 Syntax for Variables and Constants

The precision parameter is listed in bits and is generally given as 1, 8 or 16. The "s.var" primitive reserves a location but assigns no value to that location in memory. A variable value can be changed as a result of processing. An "s.assigncons" primitive has the same function as an "s.var" primitive except that an initial value can be assigned to the memory location chosen by NEWCSDL. The "s.cons" primitive assigns a constant to a memory location that cannot be changed during processing. A complete listing of the variables in this example can be seen in Appendix B.

The Contingency section is converted into a primitive listing by listing a title line, an "s.every" and a "s.var" for each entry. The use of the "s.every" primitive is peculiar to implementations in which the contingency/task is to be performed each time called. The a dummy variable of "each(1 through 8)" is used for all contingencies except the RESET contingency to help distinguish it from the "s.every" primitive. An example of a primitive list for a contingency appears in Figure 16. The Z-80 implementation of the

"s.every" construct does not require the use of the "s.proc" under the "t.generated for" primitive or the "s.exitproc" at the end. Other procedures may require these two "entry" and "exit" primitives. The "s.every" implementation in the 8080 library does require the use of the "s.proc" and "s.exitproc".

```
t. generated for:reset       ************************
s.every     (reset::)
s.var       (reset::)
t. generated for:each1        ***********************
s.every     (each1::)
s.var       (each1:8)
```

Figure 16 Example Listing for a Contingency

The functions and tasks listed in the Procedure section of the CSDL listing begin with a t.generated for title line, followed by a "s.proc" (procedure name). The body of the procedure is listed and terminated with an "s.exitproc" primitive. The creation of an IF-THEN statement using primitives follows the same pattern one would use in generating this structure from assembly language code. The first step is to sense the condition or variable to be tested. In the Z-80 library this is done with an "s.atod" primitive (analog to digital converter). This primitive will read a port from the analog to digital conversion card and record the value found in the variable listed in the parameters of the primitive. Once this value is stored it

is logically compared (a boolean and, greater than, etc.) with a constant (test value). This operation results in a flag being set in the Z-80. A conditional jump follows to a known location. In this example an "s.jmpf" or "jump on false" is used to jump to the end of the procedure if the test fails. This is the general structure used to implement the IF-THEN statement throughout the examples given in this thesis. An example, using the TITOVR procedure, is given in Figure 17.

```
t. generated for:titovr     **********************
s.proc        (titovr:)
s.atod        (stswt:8)
s.and         (temp2,temp1,stswt:8,8,8)
s.jmpf        (temp2,titl:8)
s.atod        (tit:8)
s.div         (titcon,tit,temp5:8,8,8)
s.ge          (trslt,titcon,temp4:8,8,8)
s.jmpf        (trslt,titl:8)
s.outled      (trslst:8)
s.loc         (titl:)
s.exitproc    (titovr:)
```

Figure 17 TITOVR Procedure Primitive Listing

Primitives vary in precision. All operations, whether numeric or logical must be between values of the same precision. There are conversion primitives which will convert between 8 and 16 bit values prior to any operation and their use is mandatory.

One other example procedure will be discussed since it relates to the use of the clock listing given in Figure 12. At the time the primitive list was generated, a new clock primitive was written which accessed the counter timer chip (ctc) on the Z-80 card within the Prolog system. The primitive listing reflects this new clock with a call to the new primitive. Since the initialization procedures for the clock primitive are internal to the primitive, the designer is relieved of the chore. Once set into motion the clock will generate a 16 bit number which can be read into a variable via the "s.rdtime" primitive. The variable is named as a parameter of the primitive. This value can then be manipulated and tested for use as a timer. The STALST procedure is used to illustrate this point. Figure 18 is a primitive list for STALST. "S.proc" builds a label in Z-80 format to start the procedure block. This label is called by the monitor when the contingency test is true. "S.atod" reads the start switch position, which will be 1 if on and 0 if off. "S.and" logically compares the values of the start switch with a variable called templ. Templ has a 1 loaded into its memory location. If the start switch value is 1 the true temp6 will be set to 1, if the value of 0 the temp6 will be set to 0. "S.jmpf" is a jump on false primitive. It will compare the value in temp6 to 1 and jump to the location defined by "stal" (to the end of the procedure). "S.atod" reads the rpm indication. "S.le" is a less than or

```
t.generated for:stalst     ************************
s. proc      (stalst:)
s.atod       (stswt:8)
s.and        (temp6,temp1,stswt:8,8,8)
s.jmpf       (temp6,stal:8)
s.atod       (rpm:8)
s.le         (rpmrlt,rpmcon,rpm:8,8,8)
s.rdtime     (clock:16)
s.le         (clkrlt,clock,clkcon:8,16,16)
s.and        (temp7,rpmrlt,clkrlt:8,8,8)
s.jmpf       (temp7,stal:8)
s.outled     (temp7:8)
s.loc        (stal:)
s.exitproc   (stalst:)
```

Figure 18 STALST Primitive Listing

equal to the constant, a 1 is stored in rpmrlt, if value of rpm is greater than the constant, a 0 is stored in rpmrlt. The clock (which has been accumulating time) is read by use of the s.rdtime primitive. This primitive reads the clock value and stores it in a variable named by the designer in the parameter section. The value of the clock is then compared to the constant clkcon. Clkrlt is set to 1 when the clock value is less than or equal to the constant and 0 if clock is greater than the constant. "S.and" logically and the values of clkrlt and rpmrlt and stores a 1 in temp7 if the result of the comparison is a true or a 0 if the result if false. "S.jmpf" jumps to the end of the primitive when the result of the "s.and" is false. When the "s.and" is true a malfunction exists and "s.outled" is used to light a led on the Prolog front panel as an indication of

the malfunction. The "s.loc" defines a label at the end of the procedure. "S.exitproc" causes the program to jump back to the monitor to execute the next procedure in the polling loop.

The complete primitive listing for the example problem can be seen in Appendix B. The format for entering values within the parameter section of each primitive is given in the Z-80 realization library.

The second file the designer must build is the Application Timing Table or IADEFL. This file is essentially an extract of the timing given in the contingency section of the CSDL description. The syntax for each line and the column locations for each entry of the IADEFL is:

```
1      7                  18          29      32
a <num>:  contingency name:  task name:  units:
      37     42      47      52   57       62    67
    rho,  beta1, beta2, order, pii, gamma1, gamma2,
    68
    bckgrd
```

Units is the time unit of all application table times (ie. "ms" for milliseconds). All values expressed beyond the units entry in the IADEFL are integer values representing time in the units specified by the units entry. Rho is the allowed period of the contingency/task pair, beta1 is the maximum allowed time duration of the contingency, and beta2 is the maximum time duration of the task. The backround entry is a flag to indicate that the task is to be treated as a backround task with no time period specified. All other entries can be left empty with space set aside to

maintain column dependencies. Order is the global order of the contingency/task pair, pii is the priority of the contingency task pair, gamma1 is the maximum allowed time duration of any timed block in the contingency and gamma2 is the maximum allowed duration of any timed block in the task. The first two lines of the IADEFL for this problem are given as an example in Figure 19. The small "a" tells

```
a001:system:    :ms      ,    ,    , , , , ,1
a002:reset :init:ms:100 ,100 ,100 , , , , ,0
```

Figure 19 Example IADEFL

the design environment that this is an application timing table entry. Line numbers are given immediately following the "a". The system line tells NEWCSDL that this is a background task with no associated time (the one in the last column sets this flag). Also note that no task is associated with the system line. The reset line includes a task (init). The units must be constant throughout this file, and in this case are listed as milliseconds. The timing for the contingency task pair is 100 milliseconds. The timing for the task alone is 100 milliseconds. This task is not a background task and must meet the timing parameters listed for the implementation to be feasible. All other timing parameters not listed are not required for

this example. Column dependencies are not preserved in this example to allow presentation within the format of this paper. A complete IADEFL is given in Appendix C.

With the completion of the Primitive list and IADEFL the designer can submit his work to NEWCSDL for processing. There are six files which must be present to enable NEWCSDL to function: NEWCSDL.COM, GLOBALS,DAT, PRIMITIVE.DAT, IADEFL.DAT, RELIZE.MAC, and MONTER.DAT. The file labled GLOBALS.DAT is a file that contains all of the global variables that are called by the RELIZE.MAC file. REALIZE.MAC is the realization library. MONTER.DAT contains primitives to build the software monitor.

When these files are present, NEWCSDL can be invoked by typing "NEWCSDL". The program will ask the user if an output to the terminal is desired. Answering this question with a yes will slow processing down to a snails pace. All error data can be output to a file to allow more rapid processing. The next question asked is to what detail to build the error file. All error data are output to a file call FOR099.DAT. There are three modes of operation which are selected from a menu. Mode 0 dumps all error messages to the FOR099 file. Mode 1 dumps all error meesages and adds the calculations for each line, ROM and RAM pointers. Mode 2 dumps a detailed trace including all calculations

which occurred within each primitive. With these questions answered the program will continue to process the file to completion or until a fatal error occurs.

When the program terminates, the FOR099 file should be checked first for any errors. A recommended procedure is to use mode 0 for initial processing through the design environment. This will reduce the number of error lines listed in the file. If errors are present and the reason is not immediately obvious, print the file and rerun the problem using mode 2. Mode 2 produces a very large file with all the calculations and linkages performed by NEWCSDL. With a printout of the much shorter mode 0 file, a search can be made through the mode 2 file with the editor on the VAX to find the point at which the error occurred.

A successful run through the design environment is signaled by the generation of a software listing and a hardware listing located in FOR046.DAT and FOR047.DAT respectively. Two additional files also occur; FOR021.DAT is an intermediate file and contains the primitive listing with the byte count, timing parameters, and beginning and ending line numbers for each primitive; FOR063.DAT reiterates the IADEFL.DAT file. The Z-80 listing for the start controller appears in Appendix D.

The software file can be downloaded to a Z-80 based machine via modem for further processing. This process will be discussed in the next chapter.

V. TESTING AND VALIDATION

A. OVERVIEW

The testing and validation process developed in this chapter follows the steps in the flowchart given in Figure 20. This process is iterative in nature and many loops through this system may be necessary before the prototype matches to the specifications. This is largely due to the experimental nature of the design process and immaturity of the realization libraries. The lack of a compiler makes the design process error prone because the designer is forced to hand compile the CSDL description. The realization libraries have not been validated beyond the benchchecking stage. This chapter will describe the method used for downloading programs from the VAX and the testing methodology used in verifying the Z-80 code generated by the design environment.

B. DOWNLOADING

The software listing in the FOR046.DAT file is of little use while it remains in the VAX system. Some means must be made available to transfer this listing to the target machine for testing, debugging and implementation. There are two methods available to the designer. One involves the use of a modem, the other a communications line connected directly to the VAX.

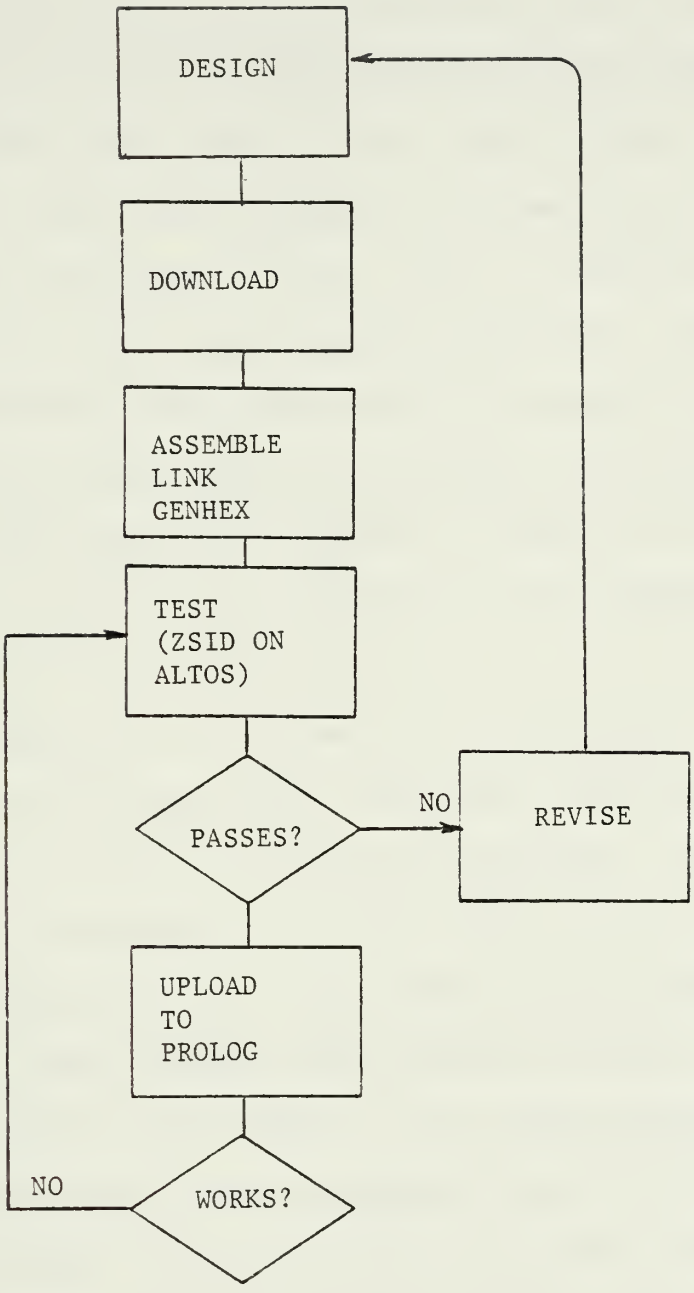


Figure 20 Testing and Validation Flow Chart

The modem, short for modulator-demodulator, is a device used to communicate digital data over telephone voice grade circuits. A modem must exist at both ends of the communications channel. Data from the VAX is transmitted through a modem, into the telephone circuit, then to the receiving modem, and finally into the receiving computer. Communications software must be used with both computers to set the parameters of the interfaces to the modems and within the modem itself. The speed of a modem is given as a baud rate and must be set the same at both ends of the communications channel.

The microcomputer lab is located on the third deck of Spanagel Hall and has one phone line with a 300/1200 baud modem attached to a single Altos microcomputer. The VAX has two phone lines connected to the VMS side, one used for 300 baud communications and the other used for 1200 baud communication. The designer must ensure that the modem baud rate select switch on the modem in the microcomputer lab is set to the same speed as that selected (via the phone number dialed) on the VAX). After booting up the CP/M operating system, the communications software can be loaded into either drive A or B of the Altos. Two public domain modem programs are available in the microlab, FCALL and MDM705. Either program works but MDM705 is capable of faster speeds. Documentation for both programs can be found on the utility disk located in the lab.

To start MDM705 simply type MDM705 and the "command" prompt will appear. Typing an "H" and a carriage return will allow the user to page through a four page help guide. To enter the terminal mode type a "T" at the command prompt and push the carriage return. The program is set to a default baud rate of 300 baud but can be changed to 1200 baud by following the procedures in the MDM705.DOC file on the utility desk. The user must ensure that the baud rate is set correctly for the telephone line dialed into the VAX. Once the line is dialed, the VAX will initiate communications with a "VMS or Unix?" prompt. Typing a V at this point connects the user with the VMS operating system and normal login procedures can be used to log on to the VAX. Once the login procedures have been completed the "$" prompt should appear. Type a "control Y" to open the text buffer in the modem program and then type "TY Filename.Filetype". The VAX responds by sending the file to the terminal. The user can see the file scroll up the screen. At the completion of the transfer type a "control R" to close the buffer. Type "control E" to get back to the command mode and type a "WRT" to write the file to disk. The text buffer is approximately 18 kilobytes long, so long files may have to be split and transmitted in separate pieces and then recombined after the transfer is complete.

File transfers using the communications line to the VAX are similar as far as the communications software is concerned except that speeds up to 4800 baud can be utilized. For long files, this extra speed can be very useful. This line is normally connected to the VSM terminal in room 527 of Spanagel Hall and has a default setting of 9600 baud. Unfortunately, the MDM705 software can't keep up with this speed. The speed can be changed at the terminal after logging on to the system. Once the speed has been changed the communications cable is disconnected from the terminal and reconnected to a cable attached to the Altos computer in the next room. With the cable set at 4800 baud, MDM705 software must be configured for 4800 baud as well. To reconfigure the baud rate to 4800 baud, use the procedures located in the MDM705 file. The instructions for file transfer from this point are the same as those for modem use. Be sure to reconnect the communications line to the terminal when finished. Logoff procedures can be performed at the Altos. The default baud rate of 9600 will be set at logoff.

C. PROGRAM ASSEMBLY

Once the file has been moved to the Altos, it can be assembled and linked like any other assembly language program, with one exception. In the Prolog memory map, using the PROM developed by LCDR Hughes [Ref. 16: p. 22], user memory starts at 400 hex. When linking a version of

the program targeted for the Prolog system the "L" switch must be set to 4000. This is done by putting a [L 4000] after the filename when invoking the L-80 linker (ie. L-80 filename [L 4000]). This will cause the linker to start the program loading at 4000 hex (the bottom of user memory in the Prolog).

Programs destined for the Prolog system must be converted to a hex format before transmission to the Prolog system. A program called GENHEX exists on the multiuser Altos in the microlab which can be used to convert linked files to hex format. This program will also run on a single user system is offloaded from the multiuser system on a standard IBM single density formatted eight inch floppy disk. The command for envoking this program is GENHEX filename 4000. Again the 4000 is there to force GENHEX to load the program starting at 4000 hex. This file contains a lot of storage overhead and will be approximately 46 kilobytes in length. When loaded into the Prolog system, this overhead is stripped off and the file will fit within the Prolog system's 16 kilobyte memory.

D. AMDS

Files are loaded into the Prolog system by using the development system created by LCDR Hughes. The Prolog system is connected to a single user Altos via the printer port. This connection is made through a flat ribbon cable attached to channel A of the dual UART located in the

Prolog's card cage. A second cable, comming from channel B of the dual UART, is connected to an Lier-Sigler ADM-3A terminal. The PROM will allow communication between the ADM-3A and the Prolog system for examining memory locations and changing values of memory locations. Uploading and downloading of programs to the Prolog system must take place through the Altos using a program called AMDS.COM. AMDS interfaces with the Altos and communicates with the Prolog system to creqte an effective work station. To transfer a file to the Prolog, select option "G" from the menu. AMDS will ask for the filename. After the filename is entered, AMDS will read the file into the Altos memory and download it to the Prolog. When the transfer is complete, AMDS will return to the main menu. Other options include dumping Prolog memory to the Altos screen, changing or revising values in memory, executing programs stored on the Prolog and locating specific byte sequences within memory. A complete users manual to the AMDS system can be found in Hughes thesis. [Ref. 17: pp. 39-60]

E. BENCH TEST

The first run of the sample problem proved to be a trying experience. All of the code in the Z-80 realization library was developed as separate primitives. When linked together to form a single unit, some primitives yielded unexpected results. A careful bench-check revealed some subtle problems in the monitor section. The "s.tabent"

primitive is responsible for building calls to the procedures listed in the primitive list. It added an "t" to the beginning of every contingency label and an "i" to the end of every task label. This would have caused fatal errors being generated within the assembler. Obviously changes had to be made to the s.tabent primitive.

F. CHANGING RELIZE.MAC

Changes to the primitive listing are not difficult but require the use of a format program. RELIZE.MAC must be stored as an 80 column card image file. This means that each line in the file must be padded to take the full 80 column card image when it is stored. Any changes made directly to RELIZE.MAC using the VAX editor will cause all lines to be stored as variable length lines. The procedure is to edit the realization library file in a version called INNAME.DAT, then invoke a program called FORMAT. The format program will build 80 column card images and an index from the primitive title lines and store theresults in a file designated by the user. The output file can be named REALIZE.MAC or stored under another file name for later use. The "s.tabent" primitive was changed using this procedure to remove the extra letter it inserted.

G. SIMPLE TEST CASE

Further checking revealed problems with the start addresses for the ROM and RAM pointers. These values are

entered in the GLOBALS file and are used as memory pointers for program instructions, variables and the stack. The Prolog memory map starts the user's memory at 4000 hex. The top of user memory is at 7fff hex or 32767 decimal. The pointers had been set to 4136 decimal and 36812 decimal. Changes were made to reflect the correct addressing scheme.

It became obvious that debugging 32 pages of Z-80 assembly language code was going to be a nontrival task. Time was slipping by and the major goals of this project had not been attained. A smaller program was needed which could be used to develop debugging methods for future work. The main elements in the test case were to test the primitives used in the IF-THEN construct and the output primitive, "s.outled". These primitives were the most used primitives in the sample problem.

A short program was developed using the Z-80 realization library which would flash an LED on the Prolog's front panel. The time for the LED to be turned on and off was determined by a counter which counts from 0 to 32,000. The counter value was tested using the IF-THEN construct and when the test passed would toggle the LED on and off. The primitive listing was processed using the procedures previously described. The program flow chart, primitive listing and timing application table appear in Appendix E. Assembly, linkage and conversion to the hex format proved to be a simple task and no problems were encountered.

Downloading to the Prolog system also went smoothly. It did not, however, execute properly and would not activate the led on the front panel.

H. TESTING USING ZSID

The testing of software produced by the design environment is simplified somewhat by the use of a CP/M type operating system. Digital Research's ZSID is an excellent tool for debugging and was used extensively in this project. After linking, ZSID was used to step through the program to check for proper operation. Logic errors and sequencing errors were easy to find using this technique.

ZSID can be used to trace programs loading at 4000 hex by using the following procedure. Invoke ZSID by typing "ZSID filename.filetype". ZSID will load the file starting at 100 hex. Attempts to execute a program with absolute jumps designed for a starting address of 4000 hex will cause the program to execute improperly. To move the program to a start address of 4000 hex type "M 100,5000,4000". This command will move a block of code starting at 100 hex and ending at 5000 hex to a start address of 4000 hex. The upper boundary of this block should be the same as the upper boundary for the program being tested. To set the program counter at 4000 hex type "G4000,4000". This command will set the program counter and set a break point at 4000 hex to halt execution. The trace mode of ZSID is used to actually step through the program and is invoked by "T,100 control p".

This command causes ZSID to trace through the first 100 steps of the program and put the results on a printer. The printout is optional but highly recommended as ZSID's trace function is very quick and data will scroll up the terminal too fast to be read.

I. PROBLEMS DISCOVERED

The use of this test method revealed several problems with the realization library. Two logic errors in the monitor section caused the program to halt before initializing variables and constants. The normal program flow is to build the stack in high memory, initialize all hardware, and then do the software initializations. Software initializations are defined by the "s.initalcons" primitive. The "s.initalcons" primitive heading in RELIZE.MAC indicates that it is to be used to "mark the beginning of things to be initialized". This statement is misleading in the sense that program code is expected between "s.initalcons" and "s.initalend", not variable and constant assignments. Variables and constants are assigned using the "DEFW" assembler command. This command is not a Z-80 instruction and is only used to tell the assembler to represent a particular value at a memory location. The result is that when the computer jumps to the start of this block it interprets the bytes in memory as instructions and not the variables these bytes actually represent. The "s.initalcons" and "s.iniltend" primitives must be present

as the monitor calls the block generated by these primitives as its first call. The fix is to include these two primitives in the primitive listing as "s.initalcons" followed immediately by "s.initalend" and place all variable and constant primitives at the end of the primitive listing.

A second problem occurred in the method used to test contingencies within the monitor. The "s.every" primitive sets a flag to "1" each time the contingency is called. The calling code within the monitor uses a "call z,<label>" instruction to call the associated task. The instruction used to generate the flag is a "cp 11111111b". This results in subtracting a -1 from 1, resulting in a positive two. This test will always fail and the monitor will never call the tasks associated with the contingencies and will continuously poll each contingency, fail, and accomplish absolutely nothing. A simple repair was made to the "s.every" primitive to load a "11111111b" into the contingency's flag to allevaite the problem.

The next test run on the Prolog system, with changes made to the primitive list and realization library, produced a program which activated the LED on the Prolog front panel but would not turn it off. The "s.outled" primitive was the culprit in this case. It was not designed to turn off a LED, only turn it on. A change to the logic structure within the primitive allows the designer to select the LED and turn it on or off as desired. This change produced a

working model, one which was completely generated by the design environment.

At this point the CSDL design environment had produced its first "controller". Although trivial in nature, it proves that a microprocessor-based controller can be specified using the design environment and usable hardware and software listings generated.

VI. CONCLUSIONS AND RECOMMENDATIONS

The completely successful completion of this project would have yielded a complete the gas turbine start controller, but time ran out. The major objectives of this thesis have been accomplished, however, by completing a much simpler test case. It was developed using the design techniques developed for the start controller and proves that the design environment is capable of producing Z-80 code which can be assembled, linked, loaded on the target machine and executed.

A major research effort remains in the construction and validation of the realization libraries. Most of the errors encountered were from subtle logic errors or errors of omission when the primitives were constructed. As has been shown, some primitives exhibit qualities which were not expected when they were originally designed. A series of small test cases are needed to reduce the scope of the task of validating primitives and to reduce the complexity of the programming required.

Some method should also be developed to freeze the hardware design and allow the designer to make changes in the behavioral description. If the design includes the fabrication of circuit boards from discrete components, a major investment in labor and materials will have been made.

To make small changes in the behavioral description which requires a major hardware change may not be cost effective. The designer should be allowed to freeze a hardware design and see if the new behavior can be supported with a new software package. The design environment can test the feasibility of this new design and report back if the design environment cannot support this change.

APPENDIX A
START CONTROLLER CSDL LISTING

```
"Generic Gas Turbine Start Malfunction Controller"
Identification
     Designer: "Richard Riley"
     Date:     "01 Nov 83"
     Project:  "Start Malfunction Controller (Ver 1.2)"
Design Criteria Section
     Metric = FIRST ;
     Volumes = 1 ;
     Monitors = 1 ;
Environment
     Input:
          RPM,8,TTL ;Fire Sense,1,TTL ;
          Oil Pres,8,TIL ; Ignitor,1,TIL ; TIT,16,TTL ;
          StartSwitch,1,TTL ; ResetSwitch,1,TTL ;
     Output:
          SD1,1,TTL ; SD2,1,TTL ; SD3,1,TTL ; SD4,1,TTL ;
          SD5,1,TTL ; SD6,1,TTL ; SD7,1,TTL ; SD8,1,TTL ;
          FireExt,1,TTL ;
     Arithmetic:
          Clock,16,TTL ; STAGFLG,1,TTL ;
Procedure
     Function ResetSwitch:
          Sense Reset ;
          IF Reset = 1 THEN ResetSwitch = true ;
               ELSE ResetSwitch = False ;
          End ResetSwitch ;
     Task INIT:
          CLOCK = 0 ;
          SD1 = 0 ;
          SD2 = 0 ;
          SD3 = 0 ;
          SD4 = 0 ;
          SD5 = 0 ;
          SD6 = 0 ;
          SD7 = 0 ;
          SD8 = 0 ;
          SD9 = 0 ;
          ResetSwitch = 0 ;
          Sense StartSwitch ;
     END INIT ;
     FUNCTION CLOCK:
          IF STARTSWITCH THEN
               CLOCK=CLICK + 1 ;
          END IF;
     END CLOCK ;
     Task TIT OVR:
          IF STARTSWITCH THEN ;
               Sense (TIT) ;
               IF TIT > 760 THEN SD1 = 1 FI ;
               ISSUE SD1 ;
          END IF;
     END OVR ;
     TASK STALSTR ;
```

```
    IF StartSwitch THEN
        SENSE (RPM) ;
        IF CLOCK >= (60) AND RPM <= (60)
            THEN SD2 = 1 FI ;
        ISSUE SD2 ;
    END IF ;
END STALSTR ;
TASK FIRE:
    IF StartSwitch THEN
        FIREEXT = 0 ;
        SENSE (FIRE SENSE) ;
        IF FIRE SENSE = 1 THEN SD3 = 1 ;
            FIREEXT = 1 FI ;
        ISSUE FIREEXT ;
        ISSUE SD3 ;
    END IF ;
END FIRE ;
TASK LOWOIL :
    IF StartSwitch THEN
        SENSE (OILPRES) ;
        IF CLOCK >= 10 AND OILPRES < 40
            THEN SD4 = 1 FI ;
        ISSUE SD4 ;
    END IF ;
END LOWOIL ;
TASK OVRSPD:
    IF StartSwitch THEN
        SENSE (RPM) ;
        IF RPM > 74 THEN SD5 = 1 FI ;
        ISSUE SD5 ;

    END IF ;
END OVRSPD ;
TASK NOIGN:
    IF StartSwitch THEN
        SENSE (IGNITOR) ;
        SENSE (RPM) ;
        IF RPM > 16 AND IGNITOR =1 THEN SD6 = 1 FI ;
        ISSUE SD6 ;
    END IF ;
END OVRSPD ;
TASK STAGSTR:
    IF StartSwitch THEN
        STAGFLG = 0 ;
        SENSE (RPM) ;
        SENSE (TIT) ;
        IF RPM < 60 AND TIT > 760 STAGFLAG = 1
        END IF ;
        IF STAGFLG = 1 AND CLOCK = 40 THEN SD7 = 1
        END IF ;
        ISSUE SD7 ;
    END IF ;
END STAGSTR ;
```

```
      TASK IGNAFTER :
          IF StartSwitch THEN
              SENSE (RPM) ;
              SENSE (IGN) ;
              IF RPM > 65 AND IGNITOR = 1
                  THEN SD8 = 1 END IF ;
              ISSUE SD8 ;
          END IF ;
      END IGNAFTER ;
Contingency:
      When ResetSwitch (100ms) do INIT ;
      Every (1000ms) do CLOCK ;
      Every (100ms) do TIT OVR ;
      Every (100ms) do STALSTR ;
      Every (100ms) do FIRE ;
      Every (100ms) do LOWOIL ;
      Every (100ms) do OVRSPD ;
      Every (100ms) do NOIGN ;
      Every (100ms) do STAGSTR ;
      Every (100ms) do IGNAFTR ;
Z
```

APPENDIX B
START CONTROLLER PRIMITIVE LISTING

```
p  t. generated for:system  ************************
p  s.main          (::)
p  s.clockcon25(::)
p  s.initialcons(::)
p  s.var           (stswt:8)
p  s.var           (rpm:8)
p  s.var           (firesn:8)
p  s.var           (oilprs:8)
p  s.var           (igntor:8)
p  s.var           (tit:8)
p  s.var           (clock:16)
p  s.var           (clkrlt:8)
p  s.var           (rpmrlt:8)
p  s.var           (restrt:8)
p  s.var           (temp2:8)
p  s.var           (temp3:8)
p  s.var           (temp6:8)
p  s.var           (temp7:8)
p  s.var           (temp8:8)
p  s.var           (temp9:8)
p  s.var           (temp10:8)
p  s.var           (temp11:8)
p  s.var           (temp12:8)
p  s.var           (temp14:8)
p  s.var           (temp15:8)
p  s.var           (temp17:8)
p  s.var           (temp18:8)
p  s.var           (temp19:8)
p  s.var           (temp21:8)
p  s.var           (temp22:8)
p  s.var           (temp23:8)
p  s.var           (temp25:8)
p  s.var           (titcon:16)
p  s.cons          (clkcon,-2401:16,16)
p  s.cons          (temp1,1:8,8)
p  s.cons          (temp4,76:8,8)
p  s.cons          (temp5,10:8,8)
p  s.cons          (rpmcon,60:8,8)
p  s.cons          (temp13,74:8,8)
p  s.cons          (temp16,16:8,8)
p  s.cons          (temp20,76:8,8)
p  s.cons          (temp24,65:8,8)
p  s.assigncons(sd1,0:8,8)
p  s.assigncons(sd2,0:8,8)
p  s.assigncons(sd3,0:8,8)
p  s.assigncons(sd4,0:8,8)
p  s.assigncons(sd5,0:8,8)
p  s.assigncons(sd6,0:8,8)
p  s.assigncons(sd7,0:8,8)
```

```
p    s.assigncons(sd8,0:8,8)
p    s.assigncons(sd9,0:8,8)
p    s.assigncons(firext,0:8,8)
p    s.initalend (::)
p    t. generated for:reset      **********************
p    s.every      (reset::)
p    s.var        (reset:1)
p    t. generated for:each1      **********************
p    s.every      (each1::)
p    s.var        (each1:8)
p    t. generated for:each2      **********************
p    s.every      (each2::)
p    s.var        (each2:8)
p    t. generated for:each3      **********************
p    s.every      (each3::)
p    s.var        (each3:8)
p    t. generated for:each4      **********************
p    s.every      (each4::)
p    s.var        (each4:8)
p    t. generated for:each5      **********************
p    s.every      (each5::)
p    s.var        (each5:8)
p    t. generated for:each6      **********************
p    s.every      (each6::)
p    s.var        (each6:8)
p    t. generated for:each7      **********************
p    s.every      (each7::)
p    s.var        (each7:8)
p    t. generated for:each8      **********************
p    s.every      (each8::)
p    s.var        (each8:8)
p    t. generated for:init       **********************
p    s.rocker     (restrt:8)
p    s.jmpf       (restrt,switch:8)
p    s.cold       (::)
p    s.proc       (init:)
p    s.loc        (switch:)
p    s.exitproc   (init,0:8)
p    t. generated for:titovr     **********************
p    s.proc       (titovr:)
p    s.atod       (stswt:8)
p    s.and        (temp2,temp1,stswt:8,8,8)
p    s.jmpf       (temp2,tit1:8)
p    s.atod       (tit:8)
p    s.div        (titcon,tit,temp5:8,8,8)
p    s.ge         (trslt,titcon,temp4:8,8,8)
p    s.jmpf       (trslt,tit1:8)
p    s.outled     (0,trslst:8,8)
p    s.loc        (tit1:)
```

```
p   s.exitproc  (titovr,0:8)
p   t. generated for:stalst     ************************
p   s.proc      (stalst:)
p   s.atod      (stswt:8)
p   s.and       (temp6,temp1,stswt:8,8,8)
p   s.jmpf      (temp6,stal:8)
p   s.atod      (rpm:8)
p   s.le        (rpmrlt,rpmcon,rpm:8,8,8)
p   s.rdtime    (clock:16)
p   s.le        (clkrlt,clock,clkcon:8,16,16)
p   s.and       (temp7,rpmrlt,clkrlt:8,8,8)
p   s.jmpf      (temp7,stal:8)
p   s.outled    (1,temp7:8,8)
p   s.loc       (stal:)
p   s.exitproc  (stalst,0:8)
p   t. generated for:fire       ************************
p   s.proc      (fire:)
p   s.atod      (stswt:8)
p   s.and       (temp8,temp1,stswt:8,8,8)
p   s.jmpf      (temp8,fir1:8)
p   s.atod      (firesn:8)
p   s.and       (temp9,temp1,firesn:8,8,8)
p   s.jmpf      (temp9,fir1:8)
p   s.outled    (2,temp9:8,8)
p   s.loc       (fir1:)
p   s.exitproc  (fire,0:8)
p   t. generated for:lowoil     ************************
p   s.proc      (lowoil:)
p   s.atod      (stswt:8)
p   s.and       (temp10,temp1,stswt:8,8,8)
p   s.jmpf      (temp10,low1:8)
p   s.atod      (oilprs:8)
p   s.and       (temp11,temp1,oilprs:8,8,8)
p   s.jmpf      (temp11,low1:8)
p   s.outled    (3,temp11:8,8)
p   s.loc       (low1:)
p   s.exitproc  (lowoil,0:8)
p   t. generated for:ovrspd     ************************
p   s.proc      (ovrspd:)
p   s.atod      (stswt:8)
p   s.and       (temp12,temp1,stswt:8,8,8)
p   s.jmpf      (temp12,ovr1:8)
p   s.atod      (rpm:8)
p   s.gt        (temp14,temp13,rpm:8,8,8)
p   s.jmpf      (temp14,ovr1:8)
p   s.outled    (4,temp14:8,8)
p   s.loc       (ovr1:)
p   s.exitproc  (ovrspd,0:8)
p   t. generated for:noign      ************************
```

```
p   s.proc    (noign:)
p   s.atod    (stswt:8)
p   s.and     (temp15,temp1,stswt:8,8,8)
p   s.jmpf    (temp15,ovrl:8)
p   s.atod    (rpm:8)
p   s.atod    (igntor:8)
p   s.gt      (temp17,temp16,rpm:8,8,8)
p   s.jmpf    (temp17,noign1:8)
p   s.and     (temp18,igntor,temp1:8,8,8)
p   s.jmpf    (temp18,noign1:8)
p   s.outled  (5,temp18:8,8)
p   s.loc     (noign1:)
p   s.exitproc (noign,0:8)          ************************
p   t. generated for:stgstr
p   s.proc    (stgstr:)
p   s.atod    (stswt:8)
p   s.and     (temp19,temp1,stswt:8,8,8)
p   s.jmpf    (temp19,stgl:8)
p   s.atod    (rpm:8)
p   s.atod    (tit:8)
p   s.div     (titcon,tit,temp5:16,16,16)
p   s.gt      (temp21,temp20,titcon:8,8,8)
p   s.jmpf    (temp21,stgl:8)
p   s.div     (rpm1,rpm,temp5:8,8,8)
p   s.le      (temp22,rpm1,temp20:8,8,8)
p   s.jmpf    (temp22,stgl:8)
p   s.outled  (6,temp22:8,8)
p   s.loc     (stgl:)
p   s.exitproc (stgstr,0:8)
p   t. generated for:ignafr         ************************
p   s.proc    (ignafr:)
p   s.atod    (stswt:8)
p   s.and     (temp23,temp1,stswt:8,8,8)
p   s.jmpf    (temp23,igrl:8)
p   s.atod    (rpm:8)
p   s.atod    (igntor:8)
p   s.gt      (temp24,rpm,temp24:8,8,8)
p   s.jmpf    (temp24,igrl:8)
p   s.and     (temp25,temp24,ign:8,8,8)
p   s.jmpf    (temp25,igrl:8)
p   s.outled  (7,temp25:8,8)
p   s.loc     (igrl:)
p   s.exitproc (ignafr,0:8)
```

APPENDIX C
START CONTROLLER IADEFL

```
a 001 :system  :                                                    .1
a 002 :reset   :init      :ms:100 ,100 ,100 ,100 .  .  .  .  .  .   .0
a 003 :each1   :titovr    :ms:100 ,100 ,100 ,100 .  .  .  .  .  .   .0
a 004 :each2   :stalst    :ms:100 ,100 ,100 ,100 .  .  .  .  .  .   .0
a 005 :each3   :fire      :ms:100 ,100 ,100 ,100 .  .  .  .  .  .   .0
a 006 :each4   :lowoil    :ms:100 ,100 ,100 ,100 .  .  .  .  .  .   .0
a 007 :each5   :ovrspd    :ms:100 ,100 ,100 ,100 .  .  .  .  .  .   .0
a 008 :each6   :noign     :ms:100 ,100 ,100 ,100 .  .  .  .  .  .   .0
a 009 :each7   :stystr    :ms:100 ,100 ,100 ,100 .  .  .  .  .  .   .0
a 010 :each8   :ignafr    :ms:100 ,100 ,100 ,100 .  .  .  .  .  .   .0
```

APPENDIX D
START CONTROLLER Z-80 LISTING

```
;
;       zilog z-80 based system
;
;
;
;
;
;
        .z80
        aseg
        org 32734       defb 0          ;ram pointer is pointing to top of memory - stack
@initvar:                               ;ram switch for software initializations
@stak:defs 32                           ;                       32b     define stack area
        org  16384      ;begin code after reserved interrupt area
@cold:ld  sp,@stak+32   ;3m     10t     3b      initialize stack pointer
        ld  a, 0                ;2m     7t      2b
        ld  (@initvar),a        ;3m     13t     3b      set initialize to false
        di                      ;1m     4t      1b      disable maskable interrupts
        jp @i0          ;3m     10t     3b      do hardware initializations
        jp @@0  ;3m 10t 3b
@i0:
        ld a,00110101b;2m 7t 1b counter 0 + load lsb then mb+ mode2+ hex
        out (0f3h),a    ;3m 11t 2b  set mode control
        ld a,050h       ;2m  7t 1b  lsb of 50000 decimal or c350 hex
        out (0f0h),a    ;3m 11t 2b  load counter time channel 0
        ld a,0c3h       ;2m  7t 1b  msb of 2000 bcd
        out (0f0h),a    ;3m 11t 2b  load counter time channel 0
        ld a,01110001b;2m 7t 1b counter 1 + load lsb then mb+ mode2+ hex
        out (0f3h),a    ;3m 11t 2b  set mode control
        ld a,00         ;2m  7t 1b  lsb of ffff+1 hex
        out (0f1h),a    ;3m 11t 2b  load counter time channel 1
        ld a,00h        ;2m  7t 1b  msb of ffff+1 hex
        out (0f1h),a    ;3m 11t 2b  load counter time channel 1
        jp @i1  ;3m 10t 3b
@@0:nop  ;1m  4t 3b  isolate the initialization
@inital: nop    ;1m     4t      1b mark top of initialization
        org  32732      ;16 bit variable stswt in ram
stswt:  defw  0         ;0m     0t      2b
        org  16424
        org  32730      ;16 bit variable rpm in ram
rpm:    defw  0         ;0m     0t      2b
        org  16424
        org  32728      ;16 bit variable firesn in ram
firesn: defw  0         ;0m     0t      2b
        org  16424
        org  32726      ;16 bit variable oilprs in ram
oilprs: defw  0         ;0m     0t      2b
        org  16424
        org  32724      ;16 bit variable igntor in ram
igntor: defw  0         ;0m     0t      2b
        org  16424
        org  32722      ;16 bit variable tit in ram
tit:    defw  0         ;0m     0t      2b
```

```
org  16424
org  32720        ;16 bit variable clock in ram
clock:  defw  0     ;0m    0t     2b
org  16424
org  32718        ;16 bit variable clkrlt in ram
clkrlt:  defw  0     ;0m    0t     2b
org  16424
org  32716        ;16 bit variable rpmrlt in ram
rpmrlt:  defw  0     ;0m    0t     2b
org  16424
org  32714        ;16 bit variable restrt in ram
restrt:  defw  0     ;0m    0t     2b
org  16424
org  32712        ;16 bit variable temp2 in ram
temp2:  defw  0     ;0m    0t     2b
org  16424
org  32710        ;16 bit variable temp3 in ram
temp3:  defw  0     ;0m    0t     2b
org  16424
org  32708        ;16 bit variable temp6 in ram
temp6:  defw  0     ;0m    0t     2b
org  16424
org  32706        ;16 bit variable temp7 in ram
temp7:  defw  0     ;0m    0t     2b
org  16424
org  32704        ;16 bit variable temp8 in ram
temp8:  defw  0     ;0m    0t     2b
org  16424
org  32702        ;16 bit variable temp9 in ram
temp9:  defw  0     ;0m    0t     2b
org  16424
org  32700        ;16 bit variable temp10 in ram
temp10:  defw  0     ;0m    0t     2b
org  16424
org  32698        ;16 bit variable temp11 in ram
temp11:  defw  0     ;0m    0t     2b
org  16424
org  32696        ;16 bit variable temp12 in ram
temp12:  defw  0     ;0m    0t     2b
org  16424
org  32694        ;16 bit variable temp14 in ram
temp14:  defw  0     ;0m    0t     2b
org  16424
org  32692        ;16 bit variable temp15 in ram
temp15:  defw  0     ;0m    0t     2b
org  16424
org  32690        ;16 bit variable temp17 in ram
temp17:  defw  0     ;0m    0t     2b
org  16424
org  32688        ;16 bit variable temp18 in ram
temp18:  defw  0     ;0m    0t     2b
org  16424
org  32686        ;16 bit variable temp19 in ram
```

```
temp19:  defw  0     ;0m    0t     2b
org  16424
org  32684         ;16 bit variable temp21 in ram
temp21:  defw  0     ;0m    0t     2b
org  16424
org  32682         ;16 bit variable temp22 in ram
temp22:  defw  0     ;0m    0t     2b
org  16424
org  32680         ;16 bit variable temp23 in ram
temp23:  defw  0     ;0m    0t     2b
org  16424
org  32678         ;16 bit variable temp25 in ram
temp25:  defw  0     ;0m    0t     2b
org  16424
org  32676         ;16 bit variable titcon in ram
titcon:  defw  0     ;0m    0t     2b
org  16424
clkcon:   defw  -2401  ;define a two byte integer
temp1:    defw  1   ;define a two byte integer
temp4:    defw  76  ;define a two byte integer
temp5:    defw  10  ;define a two byte integer
rpmcon:   defw  60  ;define a two byte integer
temp13:   defw  74  ;define a two byte integer
temp16:   defw  16  ;define a two byte integer
temp20:   defw  76  ;define a two byte integer
temp24:   defw  65  ;define a two byte integer
ld  a,0    ;2m    7t     2b     assign 0
ld (sd1),a    ;4m    13t    3b            to     sd1
ld  a,0    ;2m    7t     2b     assign 0
ld (sd2),a    ;4m    13t    3b            to     sd2
ld  a,0    ;2m    7t     2b     assign 0
ld (sd3),a    ;4m    13t    3b            to     sd3
ld  a,0    ;2m    7t     2b     assign 0
ld (sd4),a    ;4m    13t    3b            to     sd4
ld  a,0    ;2m    7t     2b     assign 0
ld (sd5),a    ;4m    13t    3b            to     sd5
ld  a,0    ;2m    7t     2b     assign 0
ld (sd6),a    ;4m    13t    3b            to     sd6
ld  a,0    ;2m    7t     2b     assign 0
ld (sd7),a    ;4m    13t    3b            to     sd7
ld  a,0    ;2m    7t     2b     assign 0
ld (sd8),a    ;4m    13t    3b            to     sd8
ld  a,0    ;2m    7t     2b     assign 0
ld (sd9),a    ;4m    13t    3b            to     sd9
ld  a,0    ;2m    7t     2b     assign 0
ld (firext),a    ;4m    13t    3b            to     firext
ld  a,  1        ;2m  7t  2b
ld  (@initvar),a;3m 13t 3b  set initvar to true
jp  @spvsr       ;3m 10t 3b  execute main moniter
;dummy procedure for every-period type contingency
@reset: nop      ;1m  4t 1b dummy function entry point
   ld a,11111111b;2m  7t 2b force function value
   ld (reset),a  ;3m 13t 3b to true value (1)
```

```
   ret             ;3m 10t 1b return to monitor
org  32674         ;16 bit variable reset in ram
reset:  defw  0     ;0m    0t     2b
org  16507
;dummy procedure for every-period type contingency
@each1: nop        ;1m  4t 1b dummy function entry point
   ld a,11111111b;2m  7t 2b force function value
   ld (each1),a  ;3m 13t 3b to true value (1)
   ret             ;3m 10t 1b return to monitor
org  32672         ;16 bit variable each1 in ram
each1:  defw  0     ;0m    0t     2b
org  16514
;dummy procedure for every-period type contingency
@each2: nop        ;1m  4t 1b dummy function entry point
   ld a,11111111b;2m  7t 2b force function value
   ld (each2),a  ;3m 13t 3b to true value (1)
   ret             ;3m 10t 1b return to monitor
org  32670         ;16 bit variable each2 in ram
each2:  defw  0     ;0m    0t     2b
org  16521
;dummy procedure for every-period type contingency
@each3: nop        ;1m  4t 1b dummy function entry point
   ld a,11111111b;2m  7t 2b force function value
   ld (each3),a  ;3m 13t 3b to true value (1)
   ret             ;3m 10t 1b return to monitor
org  32668         ;16 bit variable each3 in ram
each3:  defw  0     ;0m    0t     2b
org  16528
;dummy procedure for every-period type contingency
@each4: nop        ;1m  4t 1b dummy function entry point
   ld a,11111111b;2m  7t 2b force function value
   ld (each4),a  ;3m 13t 3b to true value (1)
   ret             ;3m 10t 1b return to monitor
org  32666         ;16 bit variable each4 in ram
each4:  defw  0     ;0m    0t     2b
org  16535
;dummy procedure for every-period type contingency
@each5: nop        ;1m  4t 1b dummy function entry point
   ld a,11111111b;2m  7t 2b force function value
   ld (each5),a  ;3m 13t 3b to true value (1)
   ret             ;3m 10t 1b return to monitor
org  32664         ;16 bit variable each5 in ram
each5:  defw  0     ;0m    0t     2b
org  16542
;dummy procedure for every-period type contingency
@each6: nop        ;1m  4t 1b dummy function entry point
   ld a,11111111b;2m  7t 2b force function value
   ld (each6),a  ;3m 13t 3b to true value (1)
   ret             ;3m 10t 1b return to monitor
org  32662         ;16 bit variable each6 in ram
each6:  defw  0     ;0m    0t     2b
org  16549
;dummy procedure for every-period type contingency
```

```
@each7: nop       ;1m  4t  1b dummy function entry point
   ld a,11111111b;2m  7t  2b force function value
   ld (each7),a  ;3m 13t  3b to true value (1)
   ret           ;3m 10t  1b return to monitor
org  32660       ;16 bit variable each7 in ram
each7:  defw  0    ;0m    0t     2b
org  16556
;dummy procedure for every-period type contingency
@each8: nop       ;1m  4t  1b dummy function entry point
   ld a,11111111b;2m  7t  2b force function value
   ld (each8),a  ;3m 13t  3b to true value (1)
   ret           ;3m 10t  1b return to monitor
org  32658       ;16 bit variable each8 in ram
each8:  defw  0    ;0m    0t     2b
org  16563
    in a,(0d0h)       ;3m 11t 2b  get status of switch 1
    bit 6,a    ;2m  8t 2b  set carry flag if true
    ld a, 0           ;2m  7t 2b  assume switch is off
    jp nc, $+4        ;3m 10t 3b  if 1 is off set restrt false
    cpl               ;1m  4t 1b  change result to true
    ld (restrt),a     ;3m 13t 3b  return the status of switch to restrt
ld a,(restrt) ;3m 13t 3b branch to switch if restrt is true
cp 0          ;2m  7t 2b
jp z, switch  ;3m 10t 3b
jp  @cold      ;3m    10t    3b
;procedure init
@init: nop    ;1m  4t     1b entry point for init
switch:  nop                  ; define location switch
ret            ;3m    10t    1b     return to monitor,exit init
;procedure titovr
@titovr: nop    ;1m  4t     1b entry point for titovr
jp $+01ch    ;3b start of initialization for first 8 bit a to d board
@i1:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh        ;3b
otir                ;2b
ld b,    2          ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 0  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp @i2  ;     3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch    ;3b  start of initialization for first 8 bit a to d board
@i2:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;2b  atodp = 4  port to be loaded
ld hl,$+00ch        ;3b
otir                ;2b
ld b,    2          ;2b  number of bytes for output
```

```
ld c, 0+3      ;2b  atodp = 4  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp   @i3      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6   ;3m 13t 3b  channel to be selected for input
out(0),a   ;3m 11t 2b  clear control
or  060h          ;1m  4t 1b  set start conv, addr latch
out(0),a   ;3m 11t 2b  issue a/d    control
in a,(0)   ;3m 11t 2b  read status
bit 7,  a         ;2m  8t 2b  check done bit
jr  z, $-4        ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (stswt),a ;3m 13t 3b  save results of input in stswt
ld  a,(temp1) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
ld  a,(stswt) ;4m    13t    3b
and b           ;1m    4t     1b
ld  (temp2),a ;4m    13t    3b
ld a,(temp2) ;3m 13t 3b branch to titl if temp2 is true
cp 0          ;2m  7t 2b
jp z, titl  ;3m 10t 3b
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i3:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;3b  atodp = 0  port to be loaded
ld hl,$+00eh         ;3b
otir                 ;2b
ld b,    2           ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 0  port to be loaded
otir                 ;2b
jp $+008h            ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp @i4  ;      3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i4:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;2b  atodp = 4  port to be loaded
ld hl,$+00ch         ;3b
otir                 ;2b
ld b,    2           ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 4  port to be loaded
otir                 ;2b
```

```
jp $+008h              ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp   @i5      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a    ;3m 11t 2b  clear control
or  060h           ;1m  4t 1b  set start conv, addr latch
out(0),a    ;3m 11t 2b  issue a/d    control
in a,(0)    ;3m 11t 2b  read status
bit 7,  a          ;2m  8t 2b  check done bit
jr  z, $-4         ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (tit),a ;3m 13t 3b  save results of input in tit
ld hl, (tit);5m    16t    3b     load arg1 in hl pair
bit 7, h         ;2m    8t     2b
ld b,0           ;2m    7t     2b
jp z, $+12       ;3m    10t    3b
ld a, h          ;1m    4t     1b
cpl              ;1m    4t     1b
ld h, a          ;1m    4t     1b
ld a, l          ;1m    4t     1b
cpl              ;1m    4t     1b
ld l, a          ;1m    4t     1b
inc hl           ;1m    6t     1b
ld b, 080h       ;1m    4t     1b
ld de, (temp5);6m    20t    4b     load arg2 in bc pair
bit 7, d         ;2m    8t     2b
ld a, 0          ;2m    7t     2b
jp z, $+12       ;3m    10t    3b
ld a, d          ;1m    4t     1b
cpl              ;1m    4t     1b
ld d, a          ;1m    4t     1b
ld a, e          ;1m    4t     1b
cpl              ;1m    4t     1b
ld e, a          ;1m    4t     1b
inc de           ;1m    6t     1b
ld a, 080h       ;2m    7t     2b
xor b            ;1m    4t     1b
ex af,af'        ;1m    4t     1b     save sign of rslt
ld c,l           ;1m    4t     1b
ld a,h           ;1m    4t     1b
ld b, 16d        ;2m    7t     2b
ld hl, 0         ;3m    10t    3b
rl c             ;2m    8t     2b     loop
rla              ;1m    4t     1b
adc hl, hl       ;3m    11t    1b
sbc hl,de        ;4m    15t    2b
```

```
jr  nc, $+3     ;3m    12t    2b     sub was ok
add hl, de      ;3m    11t    1b     restore accumulator
ccf             ;1m    4t     1b     calc result bit
djnz  $-11      ;3m    13t    2b     2m 8t on =0
rl  c           ;2m    8t     2b
rla             ;1m    4t     1b
ld  h,  a       ;1m    4t     1b
ld  l,  c       ;1m    4t     1b
ex af,af'       ;1m    4t     1b     restore sign of rslt
jp p, $+10      ;3m    10t    3b
ld a, h         ;1m    4t     1b
cpl             ;1m    4t     1b
ld h, a         ;1m    4t     1b
ld a, l         ;1m    4t     1b
cpl             ;1m    4t     1b
ld l, a         ;1m    4t     1b
inc hl          ;1m    6t     1b
ld  (titcon),hl;5m     16t    3b     save result
ld de,(temp4) ;6m 20t 4b if arg2 leargl then rslt=ffh  de=temp4
ld hl,(titcon) ;5m 16t 3b                                hl=titcon
ld  a,  h       ;1m  4t 1b
and a           ;1m    4t     1b     set sign flag of arg1
jp  p,$+13      ;3m    10t    3b     jump if arg1 is positive
ld  a,  d       ;1m    4t     1b     arg1 = -
and a           ;1m    4t     1b     set sign flag of arg2
jp  m,$+18      ;3m    10t    3b     arg2 = - arg1 = - comp backwards
ld  a,  0       ;2m    7t     2b     arg2 = + arg1 = - false
jp  $+24        ;3m 10t 3b
ld  a,  d       ;1m  4t 1b
and a           ;1m    4t     1b     set sign flag of arg2
jp  p,$+8       ;3m    10t    3b     arg2 = + arg1 = +
ld  a,11111111b;2m     7t     2b     arg2 = - arg1 = + true
jp  $+14        ;3m 10t 3b
sbc hl,de       ;4m 15t 2b
ld  a,00000000b;2m     7t     2b     result false arg2 >= arg1
jp  m,  $+7     ;3m    10t    3b
jp  m,  $+4     ;3m    10t    3b
cpl             ;1m    4t     1b     result true arg2 lt  arg1
 ld (trslt),a ;4m    13t    3b
ld a,(trslt) ;3m 13t 3b branch to titl if trslt is true
cp 0          ;2m  7t 2b
jp z, titl   ;3m 10t 3b
org 32657
@outled: defb 0 ;            set status of all lights off
org 16806
                ;trslst is displayed via lamp led number 0
ld a, 0         ;2m  7t 2b  write inhibit the alphanumeric display
out (0dlh),a    ;3m 11t 2b  send it to control port
ld a,(trslst)   ;3m 13t 3b  see if light should be on
and a           ;1m  4t 1b  set flags
ld a,(@outled) ;3m 13t 3b  recall what status of all lights are
res 0,a     ;2m  8t 2b
jp z,$+5        ;3m 10t 3b
```

```
set 0,a     ;2m  8t 2b
ld (@outled),a;3m 13t 3b save status of lamps
out (0d0h),a    ;3m 13t 3b light appropriate lamp
titl:  nop                  ; define location titl
ret             ;3m    10t     1b     return to monitor,exit titovr
;procedure stalst
@stalst: nop     ;1m  4t      1b entry point for stalst
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i5:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;3b  atodp = 0  port to be loaded
ld hl,$+00ch          ;3b
otir                  ;2b
ld b,     2           ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 0  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp @i6  ;       3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i6:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;2b  atodp = 4  port to be loaded
ld hl,$+00ch          ;3b
otir                  ;2b
ld b,     2           ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 4  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp    @i7      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a    ;3m 11t 2b  clear control
or  060h           ;1m  4t 1b  set start conv, addr latch
out(0),a    ;3m 11t 2b  issue a/d    control
in a,(0)    ;3m 11t 2b  read status
bit 7,  a          ;2m  8t 2b  check done bit
jr  z, $-4         ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (stswt),a ;3m 13t 3b  save results of input in stswt
ld  a,(templ) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
ld  a,(stswt) ;4m    13t    3b
```

```
and b           ;1m    4t     1b
ld  (temp6),a ;4m    13t    3b
ld a,(temp6) ;3m 13t 3b branch to stal if temp6 is true
cp 0          ;2m  7t 2b
jp z, stal   ;3m 10t 3b
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i7:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;3b  atodp = 0  port to be loaded
ld hl,$+00eh          ;3b
otir                  ;2b
ld b,    2            ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 0  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp @i8  ;      3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i8:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;2b  atodp = 4  port to be loaded
ld hl,$+00ch          ;3b
otir                  ;2b
ld b,    2            ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 4  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp   @i9      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a    ;3m 11t 2b  clear control
or  060h           ;1m  4t 1b  set start conv, addr latch
out(0),a    ;3m 11t 2b  issue a/d    control
in a,(0)    ;3m 11t 2b  read status
bit 7,  a          ;2m  8t 2b  check done bit
jr  z, $-4         ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (rpm),a ;3m 13t 3b  save results of input in rpm
ld de,(rpmcon) ;6m 20t 4b if arg1 learg2 then rslt=ffh  de=rpmcon
ld hl,(rpm) ;5m 16t 3b                                    hl=rpm
ld  a,  h       ;1m  4t 1b
and a           ;1m    4t     1b     set sign flag of arg2
jp  p,$+13      ;3m    10t    3b     jump if arg2 is positive
```

```
ld  a,  d       ;1m    4t     1b     arg2 = -
and a           ;1m    4t     1b     set sign flag of arg1
jp  m,$+18      ;3m    10t    3b     arg1 = - arg2 = - comp backwards
ld  a,  0       ;2m    7t     2b     arg1 = + arg2 = - false
jp  $+24        ;3m 10t 3b
ld  a,  d       ;1m  4t 1b
and a           ;1m    4t     1b     set sign flag of arg1
jp  p,$+8       ;3m    10t    3b     arg1 = + arg2 = +
ld  a,11111111b;2m    7t     2b     arg1 = - arg2 = + true
jp  $+14        ;3m 10t 3b
sbc hl,de       ;4m 15t 2b
ld  a,00000000b;2m    7t     2b     result false arg1 gt  arg2
jp  m,  $+7     ;3m    10t    3b
jp  m,  $+4     ;3m    10t    3b
cpl             ;1m    4t     1b     result true arg1 le arg2
 ld (rpmrlt),a ;4m    13t    3b
ld a,01000001b;2m 7t 1b counter 1 + latch + mode2+ hex
out (0f3h),a  ;3m 11t 2b  set mode control to latch channel 1
in l,(0f1h)   ;3m 12t 2b  get lsb
in h,(of1h)   ;3m 12t 2b  get msb
ld (clock),hl;5m 16t 3b  load time to clock
ld de,(clock) ;6m 20t 4b if arg1 learg2 then rslt=ffh  de=clock
ld hl,(clkcon) ;5m 16t 3b                                hl=clkcon
ld  a,  h       ;1m  4t 1b
and a           ;1m    4t     1b     set sign flag of arg2
jp  p,$+13      ;3m    10t    3b     jump if arg2 is positive
ld  a,  d       ;1m    4t     1b     arg2 = -
and a           ;1m    4t     1b     set sign flag of arg1
jp  m,$+18      ;3m    10t    3b     arg1 = - arg2 = - comp backwards
ld  a,  0       ;2m    7t     2b     arg1 = + arg2 = - false
jp  $+24        ;3m 10t 3b
ld  a,  d       ;1m  4t 1b
and a           ;1m    4t     1b     set sign flag of arg1
jp  p,$+8       ;3m    10t    3b     arg1 = + arg2 = +
ld  a,11111111b;2m    7t     2b     arg1 = - arg2 = + true
jp  $+14        ;3m 10t 3b
sbc hl,de       ;4m 15t 2b
ld  a,00000000b;2m    7t     2b     result false arg1 gt  arg2
jp  m,  $+7     ;3m    10t    3b
jp  m,  $+4     ;3m    10t    3b
cpl             ;1m    4t     1b     result true arg1 le arg2
 ld (clkrlt),a ;4m    13t    3b
ld  a,(rpmrlt) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
ld  a,(clkrlt) ;4m    13t    3b
and b           ;1m    4t     1b
ld  (temp7),a ;4m    13t    3b
ld a,(temp7) ;3m 13t 3b branch to stal if temp7 is true
cp 0          ;2m  7t 2b
jp z, stal  ;3m 10t 3b
                ;temp7 is displayed via lamp led number 1
ld a, 0         ;2m  7t 2b  write inhibit the alphanumeric display
out (0d1h),a   ;3m 11t 2b  send it to control port
```

```
ld a,(temp7)  ;3m 13t 3b  see if light should be on
and a          ;1m  4t 1b  set flags
ld a,(@outled) ;3m 13t 3b  recall what status of all lights are
res 1,a     ;2m  8t 2b
jp z,$+5       ;3m 10t 3b
set 1,a     ;2m  8t 2b
ld (@outled),a;3m 13t 3b save status of lamps
out (0d0h),a    ;3m 13t 3b light appropriate lamp
stal:  nop                  ; define location stal
ret            ;3m    10t    1b     return to monitor,exit stalst
;procedure fire
@fire: nop     ;1m  4t      1b entry point for fire
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i9:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;3b  atodp = 0  port to be loaded
ld hl,$+00eh         ;3b
otir                 ;2b
ld b,    2           ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 0  port to be loaded
otir                 ;2b
jp $+008h            ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp @i10  ;      3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i10:ld b, 3  ;2b  number of bytes to output
ld c, 0+1      ;2b  atodp = 4  port to be loaded
ld hl,$+00ch         ;3b
otir                 ;2b
ld b,    2           ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 4  port to be loaded
otir                 ;2b
jp $+008h            ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp   @i11      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6   ;3m 13t 3b  channel to be selected for input
out(0),a   ;3m 11t 2b  clear control
or  060h         ;1m  4t 1b  set start conv, addr latch
out(0),a   ;3m 11t 2b  issue a/d    control
in a,(0)   ;3m 11t 2b  read status
bit 7,  a        ;2m  8t 2b  check done bit
jr  z, $-4       ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
```

```
in  a,(0+2);3m 11t 2b  read a/d data
ld  (stswt),a ;3m 13t 3b  save results of input in stswt
ld  a,(temp1) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a      ;1m    4t     1b
ld  a,(stswt) ;4m    13t    3b
and b          ;1m    4t     1b
ld  (temp8),a ;4m    13t    3b
ld a,(temp8) ;3m 13t 3b branch to firl if temp8 is true
cp 0          ;2m  7t 2b
jp z, firl  ;3m 10t 3b
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@il1:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh        ;3b
otir                ;2b
ld b,    2          ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 0  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp @il2  ;     3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@il2:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;2b  atodp = 4  port to be loaded
ld hl,$+00ch        ;3b
otir                ;2b
ld b,    2          ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 4  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp   @il3     ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6   ;3m 13t 3b  channel to be selected for input
out(0),a   ;3m 11t 2b  clear control
or  060h         ;1m  4t 1b  set start conv, addr latch
out(0),a   ;3m 11t 2b  issue a/d    control
in a,(0)   ;3m 11t 2b  read status
bit 7,  a        ;2m  8t 2b  check done bit
jr  z, $-4       ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (firesn),a ;3m 13t 3b  save results of input in firesn
```

```
ld  a,(temp1) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a      ;1m    4t     1b
ld  a,(firesn) ;4m    13t    3b
and b          ;1m    4t     1b
ld  (temp9),a ;4m    13t    3b
ld a,(temp9) ;3m 13t 3b branch to fir1 if temp9 is true
cp 0          ;2m  7t 2b
jp z, fir1  ;3m 10t 3b
              ;temp9 is displayed via lamp led number 2
ld a, 0       ;2m  7t 2b  write inhibit the alphanumeric display
out (0d1h),a   ;3m 11t 2b  send it to control port
ld a,(temp9)  ;3m 13t 3b  see if light should be on
and a         ;1m  4t 1b  set flags
ld a,(@outled) ;3m 13t 3b  recall what status of all lights are
res 2,a    ;2m  8t 2b
jp z,$+5      ;3m 10t 3b
set 2,a    ;2m  8t 2b
ld (@outled),a;3m 13t 3b save status of lamps
out (0d0h),a   ;3m 13t 3b light appropriate lamp
fir1:  nop                 ; define location fir1
ret            ;3m    10t    1b     return to monitor,exit fire
;procedure lowoil
@lowoil: nop    ;1m  4t     1b entry point for lowoil
jp $+01ch    ;3b start of initialization for first 8 bit a to d board
@i13:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh       ;3b
otir               ;2b
ld b,    2         ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 0  port to be loaded
otir               ;2b
jp $+008h          ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp @i14  ;     3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch    ;3b  start of initialization for first 8 bit a to d board
@i14:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;2b  atodp = 4  port to be loaded
ld hl,$+00ch       ;3b
otir               ;2b
ld b,    2         ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 4  port to be loaded
otir               ;2b
jp $+008h          ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
```

```
jp   @il5      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a    ;3m 11t 2b  clear control
or  060h          ;1m  4t 1b  set start conv, addr latch
out(0),a    ;3m 11t 2b  issue a/d    control
in a,(0)    ;3m 11t 2b  read status
bit 7,  a         ;2m  8t 2b  check done bit
jr  z, $-4        ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (stswt),a ;3m 13t 3b  save results of input in stswt
ld  a,(temp1) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
ld  a,(stswt) ;4m    13t    3b
and b           ;1m    4t     1b
ld  (temp10),a ;4m    13t    3b
ld a,(temp10) ;3m 13t 3b branch to low1 if temp10 is true
cp 0          ;2m  7t 2b
jp z, low1  ;3m 10t 3b
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@il5:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh          ;3b
otir                  ;2b
ld b,    2            ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 0  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp @il6  ;      3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@il6:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;2b  atodp = 4  port to be loaded
ld hl,$+00ch          ;3b
otir                  ;2b
ld b,    2            ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 4  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp   @il7     ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
```

```
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a   ;3m 11t 2b  clear control
or  060h          ;1m  4t 1b  set start conv, addr latch
out(0),a   ;3m 11t 2b  issue a/d    control
in a,(0)   ;3m 11t 2b  read status
bit 7,  a          ;2m  8t 2b  check done bit
jr  z, $-4         ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (oilprs),a ;3m 13t 3b  save results of input in oilprs
ld  a,(templ) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
ld  a,(oilprs) ;4m    13t    3b
and b           ;1m    4t     1b
ld  (templl),a ;4m    13t    3b
ld a,(templl) ;3m 13t 3b branch to low1 if templl is true
cp 0          ;2m  7t 2b
jp z, low1  ;3m 10t 3b
                ;templl is displayed via lamp led number 3
ld a, 0         ;2m  7t 2b  write inhibit the alphanumeric display
out (0d1h),a    ;3m 11t 2b  send it to control port
ld a,(templl)  ;3m 13t 3b  see if light should be on
and a           ;1m  4t 1b  set flags
ld a,(@outled) ;3m 13t 3b  recall what status of all lights are
res 3,a    ;2m  8t 2b
jp z,$+5        ;3m 10t 3b
set 3,a    ;2m  8t 2b
ld (@outled),a;3m 13t 3b save status of lamps
out (0d0h),a    ;3m 13t 3b light appropriate lamp
low1:  nop                  ; define location low1
ret             ;3m    10t    1b     return to monitor,exit lowoil
;procedure ovrspd
@ovrspd: nop    ;1m  4t     1b entry point for ovrspd
jp $+01ch    ;3b start of initialization for first 8 bit a to d beard
@il7:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh        ;3b
otir                ;2b
ld b,    2          ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 0  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp @il8  ;     3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch    ;3b  start of initialization for first 8 bit a to d board
@il8:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;2b  atodp = 4  port to be loaded
```

```
ld hl,$+00ch          ;3b
otir                  ;2b
ld b,    2            ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 4  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh       ;1b
defb  080h       ;1b
defb  007h       ;1b
defb  04fh       ;1b
defb  007h       ;1b
jp   @i19      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a    ;3m 11t 2b  clear control
or  060h           ;1m  4t 1b  set start conv, addr latch
out(0),a    ;3m 11t 2b  issue a/d    control
in a,(0)    ;3m 11t 2b  read status
bit 7,  a          ;2m  8t 2b  check done bit
jr  z, $-4         ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (stswt),a ;3m 13t 3b  save results of input in stswt
ld  a,(temp1) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
ld  a,(stswt) ;4m    13t    3b
and b           ;1m    4t     1b
ld  (temp12),a ;4m    13t    3b
ld a,(temp12) ;3m 13t 3b branch to ovr1 if temp12 is true
cp 0          ;2m  7t 2b
jp z, ovr1  ;3m 10t 3b
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i19:ld b, 3  ;2b  number of bytes to output
ld c, 0+1      ;3b  atodp = 0  port to be loaded
ld hl,$+00eh          ;3b
otir                  ;2b
ld b,    2            ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 0  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh       ;1b
defb  080h       ;1b
defb  007h       ;1b
defb  04fh       ;1b
defb  007h       ;1b
jp @i20  ;      3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i20:ld b, 3  ;2b  number of bytes to output
ld c, 0+1      ;2b  atodp = 4  port to be loaded
ld hl,$+00ch          ;3b
otir                  ;2b
```

```
ld b,    2          ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 4  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp   @i21      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a    ;3m 11t 2b  clear control
or  060h          ;1m  4t 1b  set start conv, addr latch
out(0),a    ;3m 11t 2b  issue a/d    control
in a,(0)    ;3m 11t 2b  read status
bit 7,  a         ;2m  8t 2b  check done bit
jr  z, $-4        ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (rpm),a ;3m 13t 3b  save results of input in rpm
ld de,(rpm) ;6m 20t 4b if arg2 lt arg1 then rslt=ffh  de=rpm
ld hl,(temp13) ;5m 16t 3b                                  hl=temp13
ld  a,  h       ;1m  4t 1b
and a           ;1m    4t      1b      set sign flag of arg1
jp  p,$+13      ;3m    10t     3b      jump if arg1 is positive
ld  a,  d       ;1m    4t      1b      arg1 = -
and a           ;1m    4t      1b      set sign flag of arg2
jp  m,$+18      ;3m    10t     3b      arg2 = - arg1 = - comp backwards
ld  a,  0       ;2m    7t      2b      arg2 =/+ arg1 = - false
jp  $+24        ;3m 10t 3b
ld  a,  d       ;1m  4t 1b
and a           ;1m    4t      1b      set sign flag of arg2
jp  p,$+8       ;3m    10t     3b      arg2 = + arg1 = +
ld  a,11111111b;2m    7t      2b      arg2 = - arg1 = + true
jp  $+14        ;3m 10t 3b
sbc hl,de       ;4m 15t 2b
ld  a,00000000b;2m    7t      2b      result false arg2 gt  arg1
jp  z,  $+7     ;3m    10t     3b
jp  m,  $+4     ;3m    10t     3b
cpl             ;1m    4t      1b      result true arg2 le arg1
 ld (temp14),a ;4m    13t     3b
ld a,(temp14) ;3m 13t 3b branch to ovr1 if temp14 is true
cp 0          ;2m  7t 2b
jp z, ovr1  ;3m 10t 3b
                ;temp14 is displayed via lamp led number 4
ld a, 0         ;2m  7t 2b  write inhibit the alphanumeric display
out (0d1h),a    ;3m 11t 2b  send it to control port
ld a,(temp14)  ;3m 13t 3b  see if light should be on
and a           ;1m  4t 1b  set flags
ld a,(@outled) ;3m 13t 3b  recall what status of all lights are
res 4,a     ;2m  8t 2b
```

```
jp z,$+5        ;3m 10t 3b
set 4,a     ;2m  8t 2b
ld (@outled),a;3m 13t 3b save status of lamps
out (0d0h),a    ;3m 13t 3b light appropriate lamp
ovr1:  nop                  ; define location ovr1
ret             ;3m    10t    1b     return to monitor,exit ovrspd
;procedure noign
@noign: nop     ;1m  4t     1b entry point for noign
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i21:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;3b  atodp = 0  port to be loaded
ld hl,$+00eh          ;3b
otir                  ;2b
ld b,     2           ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 0  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp @i22  ;      3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i22:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;2b  atodp = 4  port to be loaded
ld hl,$+00ch          ;3b
otir                  ;2b
ld b,     2           ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 4  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp   @i23      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a    ;3m 11t 2b  clear control
or  060h          ;1m  4t 1b  set start conv, addr latch
out(0),a    ;3m 11t 2b  issue a/d    control
in a,(0)    ;3m 11t 2b  read status
bit 7,  a         ;2m  8t 2b  check done bit
jr  z, $-4        ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (stswt),a ;3m 13t 3b  save results of input in stswt
ld  a,(temp1) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
```

```
ld  a,(stswt) ;4m    13t    3b
and b          ;1m    4t     1b
ld  (temp15),a ;4m    13t    3b
ld a,(temp15) ;3m 13t 3b branch to ovr1 if temp15 is true
cp 0          ;2m  7t 2b
jp z, ovr1   ;3m 10t 3b
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i23:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh        ;3b
otir                ;2b
ld b,    2          ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 0  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp @i24  ;     3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i24:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;2b  atodp = 4  port to be loaded
ld hl,$+00ch        ;3b
otir                ;2b
ld b,    2          ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 4  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp   @i25     ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6   ;3m 13t 3b  channel to be selected for input
out(0),a   ;3m 11t 2b  clear control
or  060h         ;1m  4t 1b  set start conv, addr latch
out(0),a   ;3m 11t 2b  issue a/d    control
in a,(0)   ;3m 11t 2b  read status
bit 7,  a        ;2m  8t 2b  check done bit
jr  z, $-4       ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (rpm),a ;3m 13t 3b  save results of input in rpm
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i25:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh        ;3b
```

```
otir                    ;2b
ld b,    2              ;2b  number of bytes for output
ld c, 0+3       ;2b  atodp = 0  port to be loaded
otir                    ;2b
jp $+008h               ;3b
defb  0cfh       ;1b
defb  080h       ;1b
defb  007h       ;1b
defb  04fh       ;1b
defb  007h       ;1b
jp @i26  ;       3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch      ;3b  start of initialization for first 8 bit a to d board
@i26:ld b, 3   ;2b  number of bytes to output
ld c, 0+1       ;2b  atodp = 4  port to be loaded
ld hl,$+00ch            ;3b
otir                    ;2b
ld b,    2              ;2b  number of bytes for output
ld c, 0+3       ;2b  atodp = 4  port to be loaded
otir                    ;2b
jp $+008h               ;3b
defb  0cfh       ;1b
defb  080h       ;1b
defb  007h       ;1b
defb  04fh       ;1b
defb  007h       ;1b
jp   @i27       ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a    ;3m 11t 2b  clear control
or  060h            ;1m  4t 1b  set start conv, addr latch
out(0),a    ;3m 11t 2b  issue a/d    control
in a,(0)    ;3m 11t 2b  read status
bit 7,  a           ;2m  8t 2b  check done bit
jr  z, $-4          ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (igntor),a ;3m 13t 3b  save results of input in igntor
ld de,(rpm) ;6m 20t 4b if arg2 lt arg1 then rslt=ffh  de=rpm
ld hl,(temp16) ;5m 16t 3b                                   hl=temp16
ld  a,  h           ;1m  4t 1b
and a               ;1m    4t      1b      set sign flag of arg1
jp  p,$+13          ;3m    10t     3b      jump if arg1 is positive
ld  a,  d           ;1m    4t      1b      arg1 = -
and a               ;1m    4t      1b      set sign flag of arg2
jp  m,$+18          ;3m    10t     3b      arg2 = - arg1 = - comp backwards
ld  a,  0           ;2m    7t      2b      arg2 = + arg1 = - false
jp  $+24            ;3m 10t 3b
ld  a,  d           ;1m  4t 1b
and a               ;1m    4t      1b      set sign flag of arg2
jp  p,$+8           ;3m    10t     3b      arg2 = + arg1 = +
ld  a,11111111b;2m    7t      2b      arg2 = - arg1 = + true
```

```
jp  $+14        ;3m 10t 3b
sbc hl,de       ;4m 15t 2b
ld  a,00000000b;2m    7t     2b     result false arg2 gt  arg1
jp  z,  $+7     ;3m    10t    3b
jp  m,  $+4     ;3m    10t    3b
cpl             ;1m    4t     1b     result true arg2 le arg1
 ld (temp17),a ;4m    13t    3b
ld a,(temp17) ;3m 13t 3b branch to noign1 if temp17 is true
cp 0          ;2m  7t 2b
jp z, noign1  ;3m 10t 3b
ld  a,(igntor) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
ld  a,(temp1) ;4m    13t    3b
and b           ;1m    4t     1b
ld  (temp18),a ;4m    13t    3b
ld a,(temp18) ;3m 13t 3b branch to noign1 if temp18 is true
cp 0          ;2m  7t 2b
jp z, noign1  ;3m 10t 3b
                ;temp18 is displayed via lamp led number 5
ld a, 0         ;2m  7t 2b  write inhibit the alphanumeric display
out (0d1h),a    ;3m 11t 2b  send it to control port
ld a,(temp18)   ;3m 13t 3b  see if light should be on
and a           ;1m  4t 1b  set flags
ld a,(@outled) ;3m 13t 3b  recall what status of all lights are
res 5,a     ;2m  8t 2b
jp z,$+5        ;3m 10t 3b
set 5,a     ;2m  8t 2b
ld (@outled),a;3m 13t 3b save status of lamps
out (0d0h),a    ;3m 13t 3b light appropriate lamp
noign1:  nop                    ; define location noign1
ret             ;3m    10t    1b     return to monitor,exit noign
;procedure stgstr
@stgstr: nop    ;1m  4t     1b entry point for stgstr
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i27:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh        ;3b
otir                ;2b
ld b,    2          ;2b  number of bytes for output
ld c, 0+3     ;2b  atodp = 0  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp @i28  ;      3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i28:ld b, 3  ;2b  number of bytes to output
ld c, 0+1     ;2b  atodp = 4  port to be loaded
ld hl,$+00ch        ;3b
```

```
otir                    ;2b
ld b,    2              ;2b  number of bytes for output
ld c, 0+3       ;2b  atodp = 4  port to be loaded
otir                    ;2b
jp $+008h               ;3b
defb  0cfh       ;1b
defb  080h       ;1b
defb  007h       ;1b
defb  04fh       ;1b
defb  007h       ;1b
jp   @i29       ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a    ;3m 11t 2b  clear control
or  060h            ;1m  4t 1b  set start conv, addr latch
out(0),a    ;3m 11t 2b  issue a/d    control
in a,(0)    ;3m 11t 2b  read status
bit 7,  a           ;2m  8t 2b  check done bit
jr  z, $-4          ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (stswt),a ;3m 13t 3b  save results of input in stswt
ld  a,(temp1) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
ld  a,(stswt) ;4m    13t    3b
and b             ;1m    4t     1b
ld  (temp19),a ;4m    13t    3b
ld a,(temp19) ;3m 13t 3b branch to stg1 if temp19 is true
cp 0           ;2m  7t 2b
jp z, stg1   ;3m 10t 3b
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i29:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;3b  atodp = 0  port to be loaded
ld hl,$+00eh          ;3b
otir                    ;2b
ld b,    2              ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 0  port to be loaded
otir                    ;2b
jp $+008h               ;3b
defb  0cfh       ;1b
defb  080h       ;1b
defb  007h       ;1b
defb  04fh       ;1b
defb  007h       ;1b
jp @i30  ;      3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i30:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;2b  atodp = 4  port to be loaded
ld hl,$+00ch          ;3b
otir                    ;2b
ld b,    2              ;2b  number of bytes for output
```

```
ld c, 0+3      ;2b  atodp = 4  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp   @i31      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,6    ;3m 13t 3b  channel to be selected for input
out(0),a    ;3m 11t 2b  clear control
or  060h          ;1m  4t 1b  set start conv, addr latch
out(0),a    ;3m 11t 2b  issue a/d    control
in a,(0)    ;3m 11t 2b  read status
bit 7,  a         ;2m  8t 2b  check done bit
jr  z, $-4        ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(0+2);3m 11t 2b  read a/d data
ld  (rpm),a ;3m 13t 3b  save results of input in rpm
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i31:ld b, 3   ;2b  number of bytes to output
ld c, 0+1      ;3b  atodp = 0  port to be loaded
ld hl,$+00eh          ;3b
otir                  ;2b
ld b,    2            ;2b  number of bytes for output
ld c, 0+3      ;2b  atodp = 0  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp @i32  ;      3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i32:ld b, 3   ;2b  number of bytes to output
ld c, 4+1      ;2b  atodp = 4  port to be loaded
ld hl,$+00ch          ;3b
otir                  ;2b
ld b,    2            ;2b  number of bytes for output
ld c, 4+3      ;2b  atodp = 4  port to be loaded
otir                  ;2b
jp $+008h             ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp   @i33      ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
```

```
ld  a,0    ;3m 13t 3b  channel to be selected for input
out(4),a   ;3m 11t 2b  clear control
or  060h          ;1m  4t 1b  set start conv, addr latch
out(4),a   ;3m 11t 2b  issue a/d    control
in a,(4)   ;3m 11t 2b  read status
bit 7,  a         ;2m  8t 2b  check done bit
jr  z, $-4        ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(4+2);3m 11t 2b  read a/d data
ld  (tit),a ;3m 13t 3b  save results of input in tit
ld hl, (tit);5m    16t    3b     load arg1 in hl pair
bit 7, h        ;2m    8t     2b
ld b,0          ;2m    7t     2b
jp z, $+12      ;3m    10t    3b
ld a, h         ;1m    4t     1b
cpl             ;1m    4t     1b
ld h, a         ;1m    4t     1b
ld a, l         ;1m    4t     1b
cpl             ;1m    4t     1b
ld l, a         ;1m    4t     1b
inc hl          ;1m    6t     1b
ld b, 080h      ;1m    4t     1b
ld de, (temp5);6m    20t    4b     load arg2 in bc pair
bit 7, d        ;2m    8t     2b
ld a, 0         ;2m    7t     2b
jp z, $+12      ;3m    10t    3b
ld a, d         ;1m    4t     1b
cpl             ;1m    4t     1b
ld d, a         ;1m    4t     1b
ld a, e         ;1m    4t     1b
cpl             ;1m    4t     1b
ld e, a         ;1m    4t     1b
inc de          ;1m    6t     1b
ld a, 080h      ;2m    7t     2b
xor b           ;1m    4t     1b
ex af,af'       ;1m    4t     1b     save sign of rslt
ld c,l          ;1m    4t     1b
ld a,h          ;1m    4t     1b
ld b, 16d       ;2m    7t     2b
ld hl, 0        ;3m    10t    3b
rl c            ;2m    8t     2b     loop
rla             ;1m    4t     1b
adc hl, hl      ;3m    11t    1b
sbc hl,de       ;4m    15t    2b
jr  nc, $+3     ;3m    12t    2b     sub was ok
add hl, de      ;3m    11t    1b     restore accumulator
ccf             ;1m    4t     1b     calc result bit
djnz  $-11      ;3m    13t    2b     2m 8t on =0
rl  c           ;2m    8t     2b
rla             ;1m    4t     1b
ld  h,  a       ;1m    4t     1b
ld  l,  c       ;1m    4t     1b
```

```
ex af,af'       ;1m    4t     1b     restore sign of rslt
jp p, $+10      ;3m    10t    3b
ld a, h         ;1m    4t     1b
cpl             ;1m    4t     1b
ld h, a         ;1m    4t     1b
ld a, l         ;1m    4t     1b
cpl             ;1m    4t     1b
ld l, a         ;1m    4t     1b
inc hl          ;1m    6t     1b
ld  (titcon),hl;5m     16t    3b     save result
ld de,(titcon) ;6m 20t 4b if arg2 lt arg1 then rslt=ffh  de=titcon
ld hl,(temp20) ;5m 16t 3b                                hl=temp20
ld  a,  h       ;1m  4t 1b
and a           ;1m    4t     1b     set sign flag of arg1
jp  p,$+13      ;3m    10t    3b     jump if arg1 is positive
ld  a,  d       ;1m    4t     1b     arg1 = -
and a           ;1m    4t     1b     set sign flag of arg2
jp  m,$+18      ;3m    10t    3b     arg2 = - arg1 = - comp backwards
ld  a,  0       ;2m    7t     2b     arg2 = + arg1 = - false
jp  $+24        ;3m 10t 3b
ld  a,  d       ;1m  4t 1b
and a           ;1m    4t     1b     set sign flag of arg2
jp  p,$+8       ;3m    10t    3b     arg2 = + arg1 = +
ld  a,11111111b;2m     7t     2b     arg2 = - arg1 = + true
jp  $+14        ;3m 10t 3b
sbc hl,de       ;4m 15t 2b
ld  a,00000000b;2m     7t     2b     result false arg2 gt  arg1
jp  z,  $+7     ;3m    10t    3b
jp  m,  $+4     ;3m    10t    3b
cpl             ;1m    4t     1b     result true arg2 le arg1
 ld (temp21),a ;4m     13t    3b
ld a,(temp21) ;3m 13t 3b branch to stg1 if temp21 is true
cp 0          ;2m  7t 2b
jp z, stg1  ;3m 10t 3b
ld hl, (rpm);5m     16t     3b      load arg1 in hl pair
bit 7, h        ;2m    8t     2b
ld b,0          ;2m    7t     2b
jp z, $+12      ;3m    10t    3b
ld a, h         ;1m    4t     1b
cpl             ;1m    4t     1b
ld h, a         ;1m    4t     1b
ld a, l         ;1m    4t     1b
cpl             ;1m    4t     1b
ld l, a         ;1m    4t     1b
inc hl          ;1m    6t     1b
ld b, 080h      ;1m    4t     1b
ld de, (temp5);6m     20t     4b      load arg2 in bc pair
bit 7, d        ;2m    8t     2b
ld a, 0         ;2m    7t     2b
jp z, $+12      ;3m    10t    3b
ld a, d         ;1m    4t     1b
cpl             ;1m    4t     1b
ld d, a         ;1m    4t     1b
```

```
ld a, e          ;1m    4t     1b
cpl              ;1m    4t     1b
ld e, a          ;1m    4t     1b
inc de           ;1m    6t     1b
ld a, 080h       ;2m    7t     2b
xor b            ;1m    4t     1b
ex af,af'        ;1m    4t     1b     save sign of rslt
ld c,l           ;1m    4t     1b
ld a,h           ;1m    4t     1b
ld b, 16d        ;2m    7t     2b
ld hl, 0         ;3m    10t    3b
rl c             ;2m    8t     2b     loop
rla              ;1m    4t     1b
adc hl, hl       ;3m    11t    1b
sbc hl,de        ;4m    15t    2b
jr  nc, $+3      ;3m    12t    2b     sub was ok
add hl, de       ;3m    11t    1b     restore accumulator
ccf              ;1m    4t     1b     calc result bit
djnz  $-11       ;3m    13t    2b     2m 8t on =0
rl  c            ;2m    8t     2b
rla              ;1m    4t     1b
ld  h,  a        ;1m    4t     1b
ld  l,  c        ;1m    4t     1b
ex af,af'        ;1m    4t     1b     restore sign of rslt
jp p, $+10       ;3m    10t    3b
ld a, h          ;1m    4t     1b
cpl              ;1m    4t     1b
ld h, a          ;1m    4t     1b
ld a, l          ;1m    4t     1b
cpl              ;1m    4t     1b
ld l, a          ;1m    4t     1b
inc hl           ;1m    6t     1b
ld  (rpm1),hl;5m    16t    3b     save result
ld de,(rpm1) ;6m 20t 4b if arg1 learg2 then rslt=ffh  de=rpm1
ld hl,(temp20) ;5m 16t 3b                              hl=temp20
ld  a,  h        ;1m  4t 1b
and a            ;1m    4t     1b     set sign flag of arg2
jp  p,$+13       ;3m    10t    3b     jump if arg2 is positive
ld  a,  d        ;1m    4t     1b     arg2 = -
and a            ;1m    4t     1b     set sign flag of arg1
jp  m,$+18       ;3m    10t    3b     arg1 = - arg2 = - comp backwards
ld  a,  0        ;2m    7t     2b     arg1 = + arg2 = - false
jp  $+24         ;3m 10t 3b
ld  a,  d        ;1m  4t 1b
and a            ;1m    4t     1b     set sign flag of arg1
jp  p,$+8        ;3m    10t    3b     arg1 = + arg2 = +
ld  a,11111111b;2m    7t     2b     arg1 = - arg2 = + true
jp  $+14         ;3m 10t 3b
sbc hl,de        ;4m 15t 2b
ld  a,00000000b;2m    7t     2b     result false arg1 gt  arg2
jp  m,  $+7      ;3m    10t    3b
jp  m,  $+4      ;3m    10t    3b
cpl              ;1m    4t     1b     result true arg1 le arg2
```

```
 ld (temp22),a ;4m    13t    3b
ld a,(temp22) ;3m 13t 3b branch to stgl if temp22 is true
cp 0          ;2m  7t 2b
jp z, stgl  ;3m 10t 3b
                ;temp22 is displayed via lamp led number 6
ld a, 0         ;2m  7t 2b  write inhibit the alphanumeric display
out (0d1h),a    ;3m 11t 2b  send it to control port
ld a,(temp22)   ;3m 13t 3b  see if light should be on
and a           ;1m  4t 1b  set flags
ld a,(@outled) ;3m 13t 3b  recall what status of all lights are
res 6,a     ;2m  8t 2b
jp z,$+5        ;3m 10t 3b
set 6,a     ;2m  8t 2b
ld (@outled),a;3m 13t 3b save status of lamps
out (0d0h),a    ;3m 13t 3b light appropriate lamp
stgl:  nop                  ; define location stgl
ret             ;3m    10t    1b     return to monitor,exit stgstr
;procedure ignafr
@ignafr: nop     ;1m  4t     1b entry point for ignafr
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i33:ld b, 3  ;2b  number of bytes to output
ld c, 4+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh         ;3b
otir                 ;2b
ld b,    2           ;2b  number of bytes for output
ld c, 4+3     ;2b  atodp = 0  port to be loaded
otir                 ;2b
jp $+008h            ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp @i34  ;     3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i34:ld b, 3  ;2b  number of bytes to output
ld c, 4+1     ;2b  atodp = 4  port to be loaded
ld hl,$+00ch         ;3b
otir                 ;2b
ld b,    2           ;2b  number of bytes for output
ld c, 4+3     ;2b  atodp = 4  port to be loaded
otir                 ;2b
jp $+008h            ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp   @i35     ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,1   ;3m 13t 3b  channel to be selected for input
out(4),a   ;3m 11t 2b  clear control
```

```
or  060h          ;1m  4t 1b  set start conv, addr latch
out(4),a    ;3m 11t 2b  issue a/d    control
in a,(4)    ;3m 11t 2b  read status
bit 7,  a         ;2m  8t 2b  check done bit
jr  z, $-4        ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(4+2);3m 11t 2b  read a/d data
ld  (stswt),a ;3m 13t 3b  save results of input in stswt
ld  a,(temp1) ;4m    13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
ld  a,(stswt) ;4m    13t    3b
and b           ;1m    4t     1b
ld  (temp23),a ;4m    13t    3b
ld a,(temp23) ;3m 13t 3b branch to igr1 if temp23 is true
cp 0          ;2m  7t 2b
jp z, igr1  ;3m 10t 3b
jp $+01ch     ;3b start of initialization for first 8 bit a to d board
@i35:ld b, 3  ;2b  number of bytes to output
ld c, 4+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh         ;3b
otir                 ;2b
ld b,    2           ;2b  number of bytes for output
ld c, 4+3     ;2b  atodp = 0  port to be loaded
otir                 ;2b
jp $+008h            ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp @i36  ;      3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch     ;3b  start of initialization for first 8 bit a to d board
@i36:ld b, 3  ;2b  number of bytes to output
ld c, 4+1     ;2b  atodp = 4  port to be loaded
ld hl,$+00ch         ;3b
otir                 ;2b
ld b,    2           ;2b  number of bytes for output
ld c, 4+3     ;2b  atodp = 4  port to be loaded
otir                 ;2b
jp $+008h            ;3b
defb  0cfh      ;1b
defb  080h      ;1b
defb  007h      ;1b
defb  04fh      ;1b
defb  007h      ;1b
jp   @i37     ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,2   ;3m 13t 3b  channel to be selected for input
out(4),a    ;3m 11t 2b  clear control
or  060h          ;1m  4t 1b  set start conv, addr latch
out(4),a    ;3m 11t 2b  issue a/d    control
```

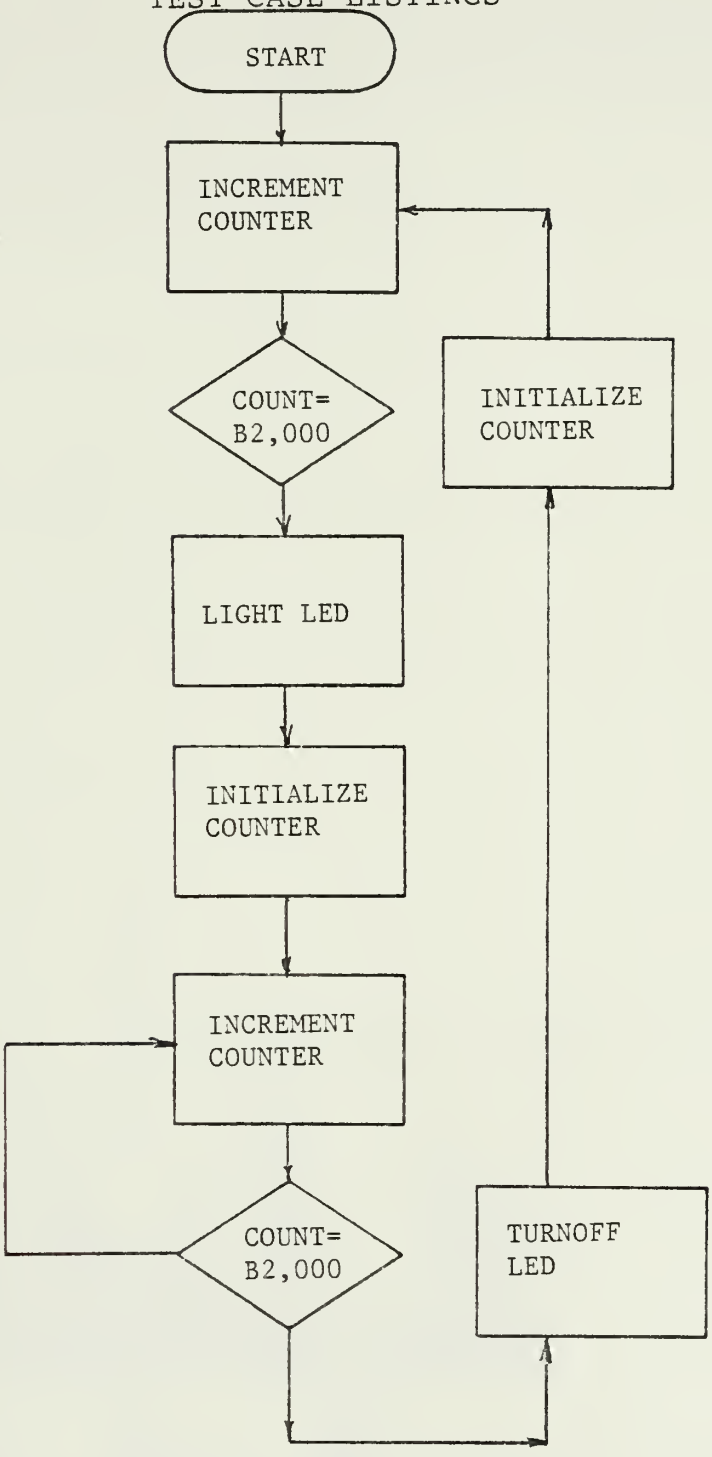
```
in a,(4)   ;3m 11t 2b  read status
bit 7,  a        ;2m  8t 2b  check done bit
jr  z, $-4       ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(4+2);3m 11t 2b  read a/d data
ld  (rpm),a ;3m 13t 3b  save results of input in rpm
jp $+01ch    ;3b start of initialization for first 8 bit a to d board
@i37:ld b, 3  ;2b  number of bytes to output
ld c, 4+1     ;3b  atodp = 0  port to be loaded
ld hl,$+00eh        ;3b
otir                ;2b
ld b,    2          ;2b  number of bytes for output
ld c, 4+3     ;2b  atodp = 0  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp @i38  ;     3b  jump to next hardware initalization
    nop ;1b  end of initialization for first 8 bit a to d board
jp $+01ch    ;3b  start of initialization for first 8 bit a to d board
@i38:ld b, 3  ;2b  number of bytes to output
ld c, 4+1     ;2b  atodp = 4  port to be loaded
ld hl,$+00ch        ;3b
otir                ;2b
ld b,    2          ;2b  number of bytes for output
ld c, 4+3     ;2b  atodp = 4  port to be loaded
otir                ;2b
jp $+008h           ;3b
defb  0cfh     ;1b
defb  080h     ;1b
defb  007h     ;1b
defb  04fh     ;1b
defb  007h     ;1b
jp   @i39     ;3b
    nop ;1b  end of initialization for second 8 bit a to d board
ld  a,3   ;3m 13t 3b  channel to be selected for input
out(4),a   ;3m 11t 2b  clear control
or  060h         ;1m  4t 1b  set start conv, addr latch
out(4),a   ;3m 11t 2b  issue a/d    control
in a,(4)   ;3m 11t 2b  read status
bit 7,  a        ;2m  8t 2b  check done bit
jr  z, $-4       ;2m  7t 2b  loop till done
com conversion time is 138 microseconds + one full execution of
com done polling loop
in  a,(4+2);3m 11t 2b  read a/d data
ld  (igntor),a ;3m 13t 3b  save results of input in igntor
ld de,(temp24) ;6m 20t 4b if arg2 lt arg1 then rslt=ffh  de=temp24
ld hl,(rpm) ;5m 16t 3b                                  hl=rpm
ld  a,  h      ;1m  4t 1b
```

```
and a           ;1m    4t     1b     set sign flag of arg1
jp  p,$+13      ;3m    10t    3b     jump if arg1 is positive
ld  a,  d       ;1m    4t     1b     arg1 = -
and a           ;1m    4t     1b     set sign flag of arg2
jp  m,$+18      ;3m    10t    3b     arg2 = - arg1 = - comp backwards
ld  a,  0       ;2m    7t     2b     arg2 = + arg1 = - false
jp  $+24        ;3m 10t 3b
ld  a,  d       ;1m  4t 1b
and a           ;1m    4t     1b     set sign flag of arg2
jp  p,$+8       ;3m    10t    3b     arg2 = + arg1 = +
ld  a,11111111b;2m     7t     2b     arg2 = - arg1 = + true
jp  $+14        ;3m 10t 3b
sbc hl,de       ;4m 15t 2b
ld  a,00000000b;2m     7t     2b     result false arg2 gt  arg1
jp  z,  $+7     ;3m    10t    3b
jp  m,  $+4     ;3m    10t    3b
cpl             ;1m    4t     1b     result true arg2 le arg1
 ld (temp24),a ;4m     13t    3b
ld a,(temp24) ;3m 13t 3b branch to igr1 if temp24 is true
cp 0          ;2m  7t 2b
jp z, igr1  ;3m 10t 3b
ld  a,(temp24) ;4m     13t    3b     rslt = arg1 .and. arg2
ld  b,  a       ;1m    4t     1b
ld  a,(ign) ;4m    13t    3b
and b           ;1m    4t     1b
ld  (temp25),a ;4m     13t    3b
ld a,(temp25) ;3m 13t 3b branch to igr1 if temp25 is true
cp 0          ;2m  7t 2b
jp z, igr1  ;3m 10t 3b
                ;temp25 is displayed via lamp led number 7
ld a, 0         ;2m  7t 2b  write inhibit the alphanumeric display
out (0d1h),a    ;3m 11t 2b  send it to control port
ld a,(temp25)  ;3m 13t 3b  see if light should be on
and a           ;1m  4t 1b  set flags
ld a,(@outled) ;3m 13t 3b  recall what status of all lights are
res 7,a     ;2m  8t 2b
jp z,$+5        ;3m 10t 3b
set 7,a     ;2m  8t 2b
ld (@outled),a;3m 13t 3b save status of lamps
out (0d0h),a    ;3m 13t 3b light appropriate lamp
igr1:  nop                  ; define location igr1
ret             ;3m    10t    1b     return to monitor,exit ignafr
;          =monitor section=
@spvsr:ld a,(@initvar);3m 13t 3b mark top of the polling loop and test
                ;             to see if the initializations have been
   and a        ;1m    4t 1b done.  if not do so
   jp z,@inital;3m   10t 3b
call  @reset  ;5m    17t    3b     test for contingency reset
ld  a,(reset)  ;4m    13t    3b     get contingency result
cp  11111111b  ;1m    4t     1b     check if result true
call z,@init ;5m    17t    3b     if true execute task
                ;if not true get next tabent or tabend to loop
call  @each1  ;5m    17t    3b     test for contingency each1
```

```
ld  a,(each1)  ;4m    13t    3b     get contingency result
cp  11111111b  ;1m    4t     1b     check if result true
call z,@titovr ;5m    17t    3b     if true execute task
                ;if not true get next tabent or tabend to loop
call  @each2   ;5m    17t    3b     test for contingency each2
ld  a,(each2)  ;4m    13t    3b     get contingency result
cp  11111111b  ;1m    4t     1b     check if result true
call z,@stalst ;5m    17t    3b     if true execute task
                ;if not true get next tabent or tabend to loop
call  @each3   ;5m    17t    3b     test for contingency each3
ld  a,(each3)  ;4m    13t    3b     get contingency result
cp  11111111b  ;1m    4t     1b     check if result true
call z,@fire ;5m    17t    3b     if true execute task
                ;if not true get next tabent or tabend to loop
call  @each4   ;5m    17t    3b     test for contingency each4
ld  a,(each4)  ;4m    13t    3b     get contingency result
cp  11111111b  ;1m    4t     1b     check if result true
call z,@lowoil ;5m    17t    3b     if true execute task
                ;if not true get next tabent or tabend to loop
call  @each5   ;5m    17t    3b     test for contingency each5
ld  a,(each5)  ;4m    13t    3b     get contingency result
cp  11111111b  ;1m    4t     1b     check if result true
call z,@ovrspd ;5m    17t    3b     if true execute task
                ;if not true get next tabent or tabend to loop
call  @each6   ;5m    17t    3b     test for contingency each6
ld  a,(each6)  ;4m    13t    3b     get contingency result
cp  11111111b  ;1m    4t     1b     check if result true
call z,@noign ;5m    17t    3b     if true execute task
                ;if not true get next tabent or tabend to loop
call  @each7   ;5m    17t    3b     test for contingency each7
ld  a,(each7)  ;4m    13t    3b     get contingency result
cp  11111111b  ;1m    4t     1b     check if result true
call z,@stgstr ;5m    17t    3b     if true execute task
                ;if not true get next tabent or tabend to loop
call  @each8   ;5m    17t    3b     test for contingency each8
ld  a,(each8)  ;4m    13t    3b     get contingency result
cp  11111111b  ;1m    4t     1b     check if result true
call z,@ignafr ;5m    17t    3b     if true execute task
                ;if not true get next tabent or tabend to loop
call  @each8   ;5m    17t    3b     test for contingency each8
ld  a,(each8)  ;4m    13t    3b     get contingency result
cp  11111111b  ;1m    4t     1b     check if result true
call z,@ignafr ;5m    17t    3b     if true execute task
                ;if not true get next tabent or tabend to loop
jp  @spvsr     ;go to the top of the polling loop of monitor table
    ; this space is deliberately void.  this is a dummy primitive.
    ; this space is deliberately void.  this is a dummy primitive.
    ; this space is deliberately void.  this is a dummy primitive.
    ; this space is deliberately void.  this is a dummy primitive.
    ; this space is deliberately void.  this is a dummy primitive.
    ; this space is deliberately void.  this is a dummy primitive.
    ; this space is deliberately void.  this is a dummy primitive.
    ; this space is deliberately void.  this is a dummy primitive.
```

```
        ; this space is deliberately void.  this is a dummy primitive.
     @i39:jp @spvsr ;3m 10t 3b  initialization of hardware is complete
                                ;           start top of main monitor loop
     end                                ;end of software listing ready for assembly
this realization consumes    0.060 watts of power
and contains    30. chips.
```

APPENDIX E
TEST CASE LISTINGS



TEST CASE FLOW CHART

```
p    t. generated for:system    **********************
p    s.main      (::)
p    t. generated for:each1     **********************
p    s.every     (each1::)
p    s.var       (each1:8)
p    t. generated for:each2     **********************
p    s.every     (each2::)
p    s.var       (each2:8)
p    t. generated for:light     **********************
p    s.proc      (light:)
p    s.outled    (0,temp5:8,8)
p    s.assign    (aa,temp4:16,16)
p    s.loc       (label1:)
p    s.add       (aa,aa,temp1:16,16,16)
p    s.eq        (bb,aa,temp2:8,16,16)
p    s.jmpf      (bb,label1:8)
p    s.exitproc  (light,0:8)
p    t. generated for:unlght    **********************
p    s.proc      (unlght::)
p    s.outled    (0,temp3:8,8)
p    s.assign    (aa,temp4:16,16)
p    s.loc       (label2:)
p    s.add       (aa,aa,temp1:16,16,16)
p    s.eq        (bb,aa,temp2:8,16,16)
p    s.jmpf      (bb,label2:8)
p    s.exitproc  (unlght,0:8)
p    s.initalcons(::)
p    s.initalend (::)
p    s.var       (aa:16)
p    s.var       (bb:8)
p    s.cons      (temp1,1:16,16)
p    s.cons      (temp2,32000:16,16)
p    s.cons      (temp3,0:8,8)
p    s.cons      (temp4,0:16,16)
p    s.cons      (temp5,1:8,8)
```

```
a 001 :system     :             :ms:     ,     ,     ,     ,     ,     ,     ,1
a 002 :each1      :light        :ms:500  ,500  ,500  ,     ,     ,     ,     ,0
a 003 :each2      :unlght       :ms:500  ,500  ,500  ,     ,     ,     ,     ,0
```

```
;
;               zilog z-80 based system
;
;
;
;
;
;
.z80
aseg
org 32734           ;ram pointer is pointing to top of memory - stack
@initvar:  defb 0           ;ram switch for software initializations
@stak:defs 32               ;              32b    define stack area
org   16384         ;begin code after reserved interrupt area
@cold:ld  sp,@stak+32       ;3m    10t    3b     initialize stack pointer
ld  a,  0               ;2m    7t     2b
ld  (@initvar),a        ;3m    13t    3b     set initialize to false
di                      ;1m    4t     1b     disable maskable interrupts
jp @i0          ;3m    10t    3b     do hardware initializations
;dummy procedure for every-period type contingency
@each1: nop         ;1m  4t 1b dummy function entry point
   ld a,11111111b;2m  7t 2b force function value
   ld (each1),a  ;3m 13t 3b to true value (1)
   ret           ;3m 10t 1b return to monitor
org  32732        ;16 bit variable each1 in ram
each1:  defw  0     ;0m    0t     2b
org  16403
;dummy procedure for every-period type contingency
@each2: nop         ;1m  4t 1b dummy function entry point
   ld a,11111111b;2m  7t 2b force function value
   ld (each2),a  ;3m 13t 3b to true value (1)
   ret           ;3m 10t 1b return to monitor
org  32730        ;16 bit variable each2 in ram
each2:  defw  0     ;0m    0t     2b
org  16410
;procedure light
@light: nop     ;1m  4t     1b entry point for light
org 32729
@outled: defb 0 ;            set status of all lights off
org 16411
                ;temp5 is displayed via lamp led number 0
ld a, 0         ;2m  7t 2b  write inhibit the alphanumeric display
out (0d1h),a    ;3m 11t 2b  send it to control port
ld a,(temp5)  ;3m 13t 3b  see if light should be on
and a           ;1m  4t 1b  set flags
ld a,(@outled) ;3m 13t 3b  recall what status of all lights are
res 0,a     ;2m  8t 2b
jp z,$+5        ;3m 10t 3b
set 0,a     ;2m  8t 2b
ld (@outled),a;3m 13t 3b save status of lamps
out (0d0h),a    ;3m 13t 3b light appropriate lamp
ld  hl,(temp4) ;6m    20t    4b     assign temp4
ld  (aa),hl  ;5m    16t    3b             to     aa
```

```
label1:  nop                    ; define location label1
ld hl, (aa);6m    20t    4b     load arg1 in hl pair
ld bc, (temp1);6m    20t    4b     load arg2 in bc pair
add hl, bc      ;3m    11t    1b     add
ld  (aa),hl;6m    20t    4b     save result
ld de,(aa) ;6m 20t 4b if arg1 = arg2 then rslt=ffh  de=aa
ld hl,(temp2) ;5m 16t 3b                                hl=temp2
sbc hl,de       ;4m 15t 2b
ld  a,11111111b;2m  8t 1b
jr  z, $+3      ;3m 13t 2b     result equal
cpl             ;1m  4t 1b     result not equal
 ld (bb),a ;4m 13t 3b
ld a,(bb) ;3m 13t 3b branch to label1 if bb is true
cp 0          ;2m  7t 2b
jp z, label1  ;3m 10t 3b
ret             ;3m    10t    1b     return to monitor,exit light
;procedure unlght
@unlght: nop     ;1m  4t     1b entry point for unlght
                ;temp3 is displayed via lamp led number 0
ld a, 0         ;2m  7t 2b  write inhibit the alphanumeric display
out (0d1h),a    ;3m 11t 2b  send it to control port
ld a,(temp3)  ;3m 13t 3b  see if light should be on
and a           ;1m  4t 1b  set flags
ld a,(@outled) ;3m 13t 3b  recall what status of all lights are
res 0,a     ;2m  8t 2b
jp z,$+5        ;3m 10t 3b
set 0,a     ;2m  8t 2b
ld (@outled),a;3m 13t 3b save status of lamps
out (0d0h),a    ;3m 13t 3b light appropriate lamp
ld  hl,(temp4) ;6m    20t    4b     assign temp4
ld  (aa),hl  ;5m    16t    3b          to     aa
label2:  nop                    ; define location label2
ld hl, (aa);6m    20t    4b     load arg1 in hl pair
ld bc, (temp1);6m    20t    4b     load arg2 in bc pair
add hl, bc      ;3m    11t    1b     add
ld  (aa),hl;6m    20t    4b     save result
ld de,(aa) ;6m 20t 4b if arg1 = arg2 then rslt=ffh  de=aa
ld hl,(temp2) ;5m 16t 3b                                hl=temp2
sbc hl,de       ;4m 15t 2b
ld  a,11111111b;2m  8t 1b
jr  z, $+3      ;3m 13t 2b     result equal
cpl             ;1m  4t 1b     result not equal
 ld (bb),a ;4m 13t 3b
ld a,(bb) ;3m 13t 3b branch to label2 if bb is true
cp 0          ;2m  7t 2b
jp z, label2  ;3m 10t 3b
ret             ;3m    10t    1b     return to monitor,exit unlght
@inital: nop  ;1m    4t     1b mark top of initialization
ld  a,  1       ;2m  7t  2b
ld  (@initvar),a;3m 13t 3b  set initvar to true
jp  @spvsr      ;3m 10t 3b  execute main moniter
org  32727       ;16 bit variable aa in ram
aa:  defw  0    ;0m    0t     2b
```

```
      org  16571
      org  32725        ;16 bit variable bb in ram
      bb:  defw  0     ;0m    0t      2b
      org  16571
      temp1:    defw  1  ;define a two byte integer
      temp2:    defw  32000  ;define a two byte integer
      temp3:    defw  0  ;define a two byte integer
      temp4:    defw  0  ;define a two byte integer
      temp5:    defw  1  ;define a two byte integer
      ;           =monitor section=
      @spvsr:ld a,(@initvar);3m 13t 3b mark top of the polling loop and test
                        ;             to see if the initializations have been
         and a          ;1m    4t 1b done.  if not do so
         jp z,@inital;3m   10t 3b
      call  @each1  ;5m    17t    3b     test for contingency each1
      ld  a,(each1)  ;4m    13t    3b     get contingency result
      cp  11111111b  ;1m    4t     1b     check if result true
      call z,@light ;5m    17t    3b     if true execute task
                        ;if not true get next tabent or tabend to loop
      call  @each2  ;5m    17t    3b     test for contingency each2
      ld  a,(each2)  ;4m    13t    3b     get contingency result
      cp  11111111b  ;1m    4t     1b     check if result true
      call z,@unlght ;5m    17t    3b     if true execute task
                        ;if not true get next tabent or tabend to loop
      call  @each2  ;5m    17t    3b     test for contingency each2
      ld  a,(each2)  ;4m    13t    3b     get contingency result
      cp  11111111b  ;1m    4t     1b     check if result true
      call z,@unlght ;5m    17t    3b     if true execute task
                        ;if not true get next tabent or tabend to loop
      jp  @spvsr     ;go to the top of the polling loop of monitor table
          ; this space is deliberately void.  this is a dummy primitive.
          ; this space is deliberately void.  this is a dummy primitive.
      @i0:jp @spvsr ;3m 10t 3b  initialization of hardware is complete
                           ;           start top of main monitor loop
      end                        ;end of software listing ready for assembly
this realization consumes   0.000 watts of power
and contains    0. chips.
```

LIST OF REFERENCES

1. Becher, W.D., Logical Design Using Integrated Circuits, Hayden Book Company, 1977.

2. "JDF Microdevices", Byte, V. 9, N. 2, February 1984.

3. Radio Shack, 1984 Catalog, 1984.

4. Allen, J.J., and others, CAD Systems, North-Holland Publishing Company, 1977.

5. Ross, A.A., Computer Aided Design of Microprocessor-Based Controllers, Ph.D. Thesis, University of California, Davis, 1978.

6. Matelan, M.N., The Automatic Design of Real Time Control Systems, Lawrence Livermore Laboratory, December 10, 1976.

7. Ibid.

8. Ross, A.A., Computer Aided Design of Microprocessor-Based Controllers, Ph.D. Thesis, University of California, Davis, 1978.

9. Ibid.

10. Smith, T.J., Implementation of a Zilog Z-80 Based Realization Library for the Computer Systems Design Environment, M.S. Thesis, Naval Postgraduate School, Monterey, California, March 1984.

11. Pollock, G.G., Further Development and Investigation of Computer-Aided Design of Microprocessor Systems, M.S. Thesis, University of California, Davis, December 1981.

12. Heilstedt, M.R., Automated Design of Microprocessor-Based Digital Filters, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1983.

13. Prolog Corporation, STD Bus Technical Manual and Product Catalog, August 1982.

14. Hughes, S.M., A Microprocessor Development System for the Altos Series Microcomputers, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1981.

15. Ross, A.A., Computer Aided Design of Microprocessor-Based Controllers, Ph.D. Thesis, University of California, Davis, 1978.

16. Hughes, S.M., A Microprocessor Development System for the Altos Series Microcomputers, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1981.

17. Ibid.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	LTC Alan Ross, Code 52Rs Naval Postgraduate School Monterey, California 93943	4
4.	Prof. Herschel H. Loomis, Code 62Lm Naval Postgraduate School Monterey, California 93943	1
5.	LCDR Richard P. Riley VP-17 FPO, San Francisco 96601	1
6.	Computer Technology Programs Office Code 37 Naval Postgraduate School Monterey, California 93943	1

208566

Thesis
R4745 Riley
c.1 Control system design language implementation of a gas turbine starting controller.