A REAL-TIME SIMULATOR OF A TURBOFAN ENGINE

Jonathan S. Litt
Propulsion Directorate
U.S. Army Aviation Research and Technology Activity - AVSCOM
Lewis Research Center
Cleveland, Ohio 44135

and

John C. DeLaat and Walter C. Merrill
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

E-4578

SUMMARY

A real-time digital simulator of a Pratt and Whitney F100 engine has been developed for real-time code verification and for actuator diagnosis during full-scale engine testing. This self-contained unit can operate in an open-loop stand-alone mode or as part of a closed-loop control system. It can also be used for control system design and development. Tests conducted in conjunction with the NASA Advanced Detection, Isolation, and Accommodation program show that the simulator is a valuable tool for real-time code verification and as a real-time actuator simulator for actuator fault diagnosis. Although currently a small perturbation model, advances in microprocessor hardware should allow the simulator to evolve into a real-time, full-envelope, full engine simulation.

INTRODUCTION

The F100 engine simulator was designed to support the Advanced Detection Isolation and Accommodation (ADIA) F100 engine test. The objective of the ADIA engine test was to demonstrate the application of analytical redundancy to the detection, isolation, and accommodation of engine sensor failures (ref. 1). That is, to show that the engine can continue to be controlled accurately - even during transients - with one or more of the engine sensors giving false readings. The objective of this engine test was also to demonstrate that the ADIA software works on a real engine and is, therefore, reliable and useful in a real environment. This software had already been successfully tested on a hybrid computer simulation of the engine (ref. 2). Due to the usual uncertainties associated with a full scale engine test, it was determined that should changes to the control computer's software be necessary, a simulation of the engine would be required for software verification. The simulator which has been developed is a portable box which could be taken into the Propulsion Systems Laboratory (PSL) to verify any changes in the control interface and monitoring (CIM) unit's (ref. 3) software before the CIM unit was used to control the engine. The simulator was installed in the PSL as shown in figure 1. Swapping a patch panel allows the CIM unit to control either the engine or the simulator. This change is completely transparent to the CIM unit. This technique minimizes risk to the engine which might otherwise occur if the controller's software contains a serious error.

The F100 engine is a high performance, twin-spool, low by-pass ratio, turbofan engine. Figure 2 shows the locations of the engine inputs which are defined in table I. Figure 3 shows the locations of the engine sensors defined, along with the other simulator outputs, in table II.

The simulator is based upon a HYTESS-like model (refs. 4 and 5) of the F100 engine without augmentation (afterburning). HYTESS is a simplified FORTRAN simulation of a generalized turbofan engine. To create the simulator, the original HYTESS code was revised to incorporate F100 specific parameters. Additionally, the executive was adapted from that of the ADIA code (ref. 6) which executes in the CIM unit.

This report describes the design and implementation of the F100 real-time portable engine simulator. The report discusses the simplified engine model and the actuator and sensor models used in the simulator. Next, the design of the microcomputer implementation, including the hardware and software design details, is described. A user's manual is included with step by step instructions of how to use the simulator. Performance comparisons with the real engine are presented. Finally, recommendations for future work are given.

MODEL

The original full nonlinear simulation of the F100 engine is a 13 000 line FORTRAN program. It incorporates detailed descriptions of both steady-state and dynamic engine operation throughout the entire flight envelope. This simulation very accurately reproduces the engine's performance but requires very large amounts of digital computer memory and processing time. The HYTESS turbofan engine simulation was developed to provide a structurally simpler alternative to engine simulation and thus reduce computer storage and processing requirements.

Since the main objective of the simulator is real-time execution, an F100 engine simulation with a HYTESS-like structure was used. The HYTESS-like model enables much more efficient calculation of the engine dynamics than the full nonlinear model. The penalty for this efficiency is (1) a small loss in accuracy and (2) the relationships between physical elements of the engine are lost.

The HYTESS-like model is set up in state space form using the vector differential equations

$$\begin{aligned} \dot{x} &= f(x,u,\Phi) \\ y &= g(x,u,\Phi) \end{aligned} \tag{1}$$

where x is the vector of intermediate engine variables or state variables, $\dot{x}$ is the derivative of x with respect to time, u is the vector of control inputs, Φ is the vector of environmental conditions, and y is the vector of engine outputs. Clearly, at steady-state points,

$$\begin{aligned} \dot{x} &= f(x_b,u_b,\Phi_b) = 0 \\ y_b &= g(x_b,u_b,\Phi_b) \end{aligned} \tag{2}$$

where the subscript b denotes a steady-state point on the operating line known as a base point. In other words, selecting y_b and Φ_b vectors determines steady-state x_b and u_b vectors such that the quadruple (x_b, y_b, u_b, Φ_b) satisfies equation (2). Typical base points representative of the entire flight envelope at a power lever angle of 83° are shown in figure 4.

Generally, state-space equations of a system linearized about the operating point (x_b, u_b, y_b) are of the form

$$\begin{aligned} \dot{x} &= F(x - x_b) + G(u - u_b) \\ y &= y_b + H(x - x_b) + D(u - u_b) \end{aligned} \tag{3}$$

where F, G, H, and D are system matrices of the appropriate dimensions. The full nonlinear F100 model was linearized at each base point using perturbation techniques. Thus, the state-space model is accurate in the neighborhood of a base point. The actual equations used in the model are of the form

$$\begin{aligned} \dot{x} &= F(y,\Phi)[x - x_{ss}] \\ y &= y_b(y,\Phi) + H(y,\Phi)[x - x_b(y,\Phi)] + D(y,\Phi)[u - u_b(y,\Phi)] \\ x_{ss} &= x_b(y,\Phi) - F^{-1}G(y,\Phi)[u - u_b(y,\Phi)] \end{aligned} \tag{4}$$

where the subscript ss denotes a steady-state point near a base point. This formulation was used to separate the dynamic and steady-state effects that the system matrix parameters have on the model outputs. It is clear that the equations for y in equations (3) and (4) are equivalent. To show that the equations for $\dot{x}$ are also the same, the equation for x_{ss} must be substituted into the equation for $\dot{x}$ in equation (4) as follows:

$$\begin{aligned} \dot{x} &= F(y,\Phi)[x - x_{ss}] \\ &= F(y,\Phi)[x - \{x_b(y,\Phi) - F^{-1}G(y,\Phi)[u - u_b(y,\Phi)]\}] \\ &= F(y,\Phi)x - F(y,\Phi)\{x_b(y,\Phi) - F^{-1}G(y,\Phi)[u - u_b(y,\phi)]\} \\ &= F(y,\Phi)x - F(y,\Phi)x_b(y,\Phi) + FF^{-1}G(y,\Phi)[u - u_b(y,\Phi)] \\ &= F(y,\Phi)[x - x_b(y,\Phi)] + G(y,\Phi)[u - u_b(y,\Phi)] \end{aligned}$$

Therefore the systems of equations in (3) and (4) are equivalent.

The linearized system is fourth order, in other words the state vector contains four elements whose derivatives are integrated to evolve the system in time. These elements represent actual engine variables: fan speed (N1), compressor speed (N2), burner exit slow response temperature, and fan turbine inlet slow response temperature. The first two elements are also the first two engine outputs. Using 109 base points, the original nonlinear simulation had been linearized to a set of 109 fourth order realizations. In the F100 model as in HYTESS, the elements of the matrices F, $F^{-1}G$, H, and D are nonlinear polynomials. These polynomials were determined by a curve-fitting algorithm used to regress each matrix element upon elements of y and Φ or upon elementary functions of y and Φ. Thus the polynomial matrices approximate

the data points, i.e., they approximate the system matrices determined by the use of perturbational techniques at each base point. Therefore, at each point in the envelope, the polynomials need only be evaluated to determine the system matrices. The definitions of these polynomials appear in reference 7.

The actuators and sensors are, for the most part, modeled as first-order lags with a small dead zone or other small nonlinearity included. In general, the nonlinearities are added after the lags are evaluated. This allows the sensor and actuator models to be evolved using closed-form equations. These equations are the standard zero-order hold z-transform solution of a linear first-order equation. Specifically,

$$y([k+1]T) = u(kT) - [u(kT) - y(kT)]e^{-T/\tau}$$

where T is the time step, $u(kT)$ is the input to the lag at time kT, $y(kT)$ is the output of the lag at time kT, and τ is the time constant of the lag. In some cases the output of this linear model is altered to incorporate a nonlinearity by, for instance, setting it equal to zero if its magnitude is less than some relatively small value. The time constants used are similar to those used on the hybrid simulation and are very close to those of the real instrumentation being modeled.

IMPLEMENTATION

The simulator consists of a rack-mountable microcomputer chassis, a dual floppy disk drive unit, and a CRT terminal. The microcomputer chassis has nine Multibus/IEEE 796 compatible expansion slots and a power supply. The chassis contains the five boards shown in figure 5. The simulator software executes on an INTEL 86/30 single board computer (ref. 8) with an 8086 microprocessor, an 8087 floating point coprocessor, and 256 Kb of random access memory. Table III lists the jumper connections used on the 86/30 board. A Zendex ZX-200A single board disk controller (ref. 9) is included to communicate with the disk drives. A data translation DT 1742-32 DI 32 channel, differential input A/D converter (ref. 10) accepts the analog control signals from the controller. Finally, there are two data translation DT 1842-8-V 8 channel D/A boards (ref. 10) which convert all of the simulated outputs to analog voltages. Table IV lists the pin connections from the D/A boards for each output variable.

The simulator software consists of 21 routines, 11 in FORTRAN and 10 in 8086 and 8087 assembly language (ref. 11). In addition, the simulation uses functions and utilities contained in four libraries. The routines share variables through common blocks of memory. These common blocks are listed in table V and their contents are described in table VI.

For proper stability and accuracy a good rule of thumb is that a numerical (Euler) integration time of not more than one quarter the control interval should be used in the simulation. Use of this rule will reduce the interaction between the simulation and the control by reducing any phase shift due to time delays in the simulation. The ADIA control interval was 40 msec. Thus a simulator integration time of 10 msec was the goal. As a full envelope simulation, the minimum achievable update time (integration time) for the simulation was approximately 40 msec or four times the desired interval. To overcome this problem, a drastic reduction in the cycle time of the algorithm was required.

It was possible to determine the execution time for each major subroutine. Most of the FORTRAN code had already been optimized (ref. 12) so the execution time for each routine was essentially the minimum possible. Several alternative solutions to the execution time problem were considered. These included using a faster processor, putting the simulation on multiple computers (parallel processing), and/or modifying the structure of the software. To avoid having to change the microcomputer hardware, the simulator software was modified. The simulator was changed from a full-envelope model to an operating point model. This was achieved by breaking the simulation into two loops: an initialization loop and a real-time run loop. Now the base points and the matrix elements are calculated in nonreal time (these are the longest routines) and then, in the real-time loop, the system equations are evolved as a set of linear equations to the new operating condition. This allowed the real-time loop cycle time to be reduced to 12 msec. The result is a linear model valid within a small region about a given operating point. This model gives excellent steady-state results and good transient results for small perturbations, such as small movements of the power lever angle (PLA). However, the model will not perform accurately for large perturbations such as large PLA movements.

Description of Modes

The simulation can operate in five different modes depending upon the application. The modes are: initialization/run, PSL/hybrid, calibration, open-loop/closed-loop, and actuators only. These modes are controlled by software switches described in table VII.

Initialization/run. - In the initialization mode, the simulator initializes variables to their desired steady-state operating point values. Initialized variables include the engine base points and the open-loop setpoints. In the run mode, the simulation enters the real-time loop. Here the state equations are evolved from the previous operating point to the desired operating point using Euler's method for numerical integration. These two modes are a consequence of the fact that the simulation is not fast enough to accurately model the whole flight envelope dynamically in real time.

Propulsion system lab (PSL)/hybrid. - The PSL mode scales the control signals and alters the simulator outputs to correspond to those of the engine in the PSL. Initially the inputs and outputs of the simulator were scaled identically to the inputs and outputs of the F100 Hybrid Simulation. These were all ±10 V, straight line representations of the engine inputs and outputs. However, the actual engine input and output devices consist of linear potentiometers, resolvers, thermocouples, flowmeters, and electro-hydraulic actuators. These devices typically do not accept or produce ±10 V, linear signals. Thus, while the system was in the PSL, the scaling for the control inputs from the CIM unit to the engine simulator had to be mapped to the equivalent scaling for the hybrid simulation. Likewise, the scaling for the outputs of the simulator's sensors had to be mapped to the equivalent scaling values for the actual engine sensors so the CIM unit received the same values the engine sensors would produce. For some variables the difference between the hybrid and the PSL mode was simply a different scale factor. In the nonlinear cases the PSL variable

had to be mapped though a look-up table or a polynomial curve in addition to being scaled. These curves and polynomials were determined experimentally during the engine test calibration procedure.

Calibration. - The calibration mode is used to test the input/output variable mappings. Once a map has been determined and implemented, it must be validated with the simulation and controller. The calibration mode allows the user to bypass the system evolution subroutines, independently set a variable to an intermediate value in engineering units, and examine the corresponding output value. In the same way, the simulator can receive analog inputs and the user can examine these values in engineering units once they have gone through conversion. Using this method, the user can determine if the values are being scaled correctly.

Closed loop/open loop. - In the closed-loop mode the simulator receives the control signals from an outside source such as the CIM unit. In the open-loop mode the simulator uses base point values stored in its memory for the control signals.

Actuator. - The actuator mode is used to simulate only the engine actuators. To ensure that the real actuators are all working correctly and since they are quite simple to model accurately, the simulator can be run in parallel with the engine and the simulated and actual actuator feedback values compared. The only difference between the actuator mode and the run mode of the simulator is that in the actuator mode TT2, the only independent variable which the actuators require beside the control signals, is obtained as input from the facility (Propulsion Systems Laboratory) rather than calculated by the simulator. Since no other information is required and the actuator calculations are fairly simple and accurate, the simulator is used as a full envelope real-time simulator for the actuators. The engine model outputs are not used in this mode since the base points are not recalculated at each operating point.

The modes are all defined by software switches which can be toggled using MINDS (ref. 13). MINDS is a program used to examine and to set values of memory locations. To the user, MINDS looks like an interpreter. The user types in commands and MINDS carries them out. MINDS executes in the background and is interrupted by the timer at the beginning of each initialization or run-time cycle (fig. 6). Even though it runs for only about 17 percent of the time in the run-time loop, to the user it appears to be running continuously. MINDS can be used to examine and/or set the software mode switches and also to examine and set any parameter within the simulation. In addition, MINDS can be used to collect transient data, that is, to examine memory locations periodically over time, and to display that information graphically. Due to memory constraints in the simulator, the transient data capability of MINDS was not included.

The simulator uses the CP/M-86 disk operating system for loading the simulator program from disk, for saving MINDS data to disk, and for communicating with the user terminal. This operating system has a limitation that the total space for code and data may not be more than 64 Kb. The total memory required for the simulator, including the reduced capability version of MINDS, is about 50 Kb, approximately two-thirds of which is code and one-third is data.

PROGRAM EXECUTION

After the system is booted, the program can be executed by typing the name of the disk drive where the program disk is located followed by a colon and the name of the program. When the RETURN key is pressed, the executable code is loaded from disk and executed.

The program starts execution in the executive (fig. 7). The executive initializes the update intervals, sets up the memory appropriately and takes care of the administrative details. Then it executes two routines, MSET (fig. 8) and MTRXST (fig. 9). They are routines for initialization of constants such as time constants, the exponents associated with each time constant, and the initial conditions. Once the program setup is complete it is not repeated since the setup information will never change. Following setup, the program enters the initialization loop by setting the interrupt timer (fig. 6). This loop does not have a real-time cycle constraint (it has no time dependency) but it repeats every 50 msec. Here it executes INLET (fig. 10) which calculates the ambient conditions based on the altitude and Mach number. Then it goes to EMODEL (fig. 11) which determines the base points and matrix elements by evaluating polynomials whose coefficients are functions of the ambient conditions. The scheduled values of engine variables are calculated in the subroutines RPFAND (fig. 12) and RPLIMD (fig. 13) which are called from EMODEL. The operating point is requested using MINDS. The operating point is automatically initialized to sea-level static, standard day conditions, 83° PLA. Base point values at this condition are also stored as the initial control values for that operating point in the open loop mode. Any extra time in this loop is used by MINDS to accept inputs from the user. He can change altitude and Mach number and the next time through the loop everything is recalculated for the new conditions. Since everything in the initialization loop is calculated directly, the loop need only be executed once after a change is made for the values to be correct. The user can also set the switch to go from the initialization loop to the run loop while in MINDS. Figure 14 shows the program flow as the initialization/run switch, RLOOP, is set and reset. The update interval is short enough to essentially guarantee that the loop will be executed at least once after the conditions are changed to obtain the correct values before the switch can be set. The program is ready to be used interactively once the MINDS prompt (>) appears.

Setting the appropriate software switch puts the program into the real-time mode. The run loop consists of the dynamic routines. This loop has an update interval of 12 msec and during that time the control input routine, actuator routine, the system evolution routine (numerical integration), and the output signal routine all execute. The first section receives the control signals from the CIM unit and converts the scaled integers to real numbers. ACTUAT (fig. 15) takes the real commanded values and evolves the actuator models to their value at the current time step. This output is used by EVOLVE (fig. 16) to integrate the differential equations describing the engine itself. Over time, the numerical integration will bring the simulation from its previous steady-state point up to the new steady-state point with a linear, non-realistic transient. The new steady-state point is, however, accurate and realistic. After EVOLVE executes, the engine outputs, actuator feedbacks, PLA, and the ambient conditions are converted to scaled integers and sent via D/A converters to the CIM unit. The I/O sections are part of the multiplexer interrupt service routine section of the executive (fig. 17). Any spare time

is used by the message generation routine or MINDS. The message generation routine takes priority over MINDS if it needs to execute but it is only used to print out error messages. A more in-depth description of the simulator's operation is given in appendix A.

Many of the routines listed above call their own subroutines which do table lookups or some other type of calculation. The relationships are shown in figure 18. A complete list of the routines with a description of each appears in appendix B.

Exception Handling

There are three noncatastrophic exceptions which, if they occur, will cause incorrect operation of the simulator. They are: (1) floating point, (2) divide by zero, and (3) update failure. The first two produce an interrupt and are handled by interrupt service routines (figs. 19 and 20). Update failures are detected by a flag check routine within the timer interrupt service routine (fig. 21). When the timer signaling the start of the real-time run loop interrupts the simulation, the service routine checks the update failure flag. Since the flag is reset near the end of the multiplexer interrupt service routine (fig. 17), a reset update failure flag indicates no problem. However, if the flag is still set at the start of the timer service routine, it means that the cycle was unable to finish the previous time through the loop and an update failure is declared.

One of the results of these interrupt service routines is to give the user an indication that the error took place by printing a message to the user terminal. This printing is done in the time remaining at the end of the real-time loop. Printing a message is a slow process and may take several cycles of the run loop to complete. Because more than one error might occur in a single cycle and each takes so long to print, a data structure is used to store the starting addresses of each error's corresponding message. Up to 15 addresses can be held in this circular queue. Figure 22 shows the way starting addresses of messages are saved. It is a more detailed version of the boxed area in figures 19 to 21.

After the digital-to-analog conversion of the simulator outputs in the run loop, the program checks the error message queue. If no printing is in progress and a message is waiting to be printed, the program will initiate printing the message at the head of the queue. Otherwise the program returns to the task which was interrupted by the current cycle of the run loop - either MINDS or printing a message. The message generation code, MESGEN (fig. 23), is actually a portion of the multiplexer interrupt routine and is shown in the boxed region of figure 17.

Figure 24 demonstrates the way the pointers move around the queue when two errors occur in rapid succession and are then printed out to the terminal device. The operation of the total noncatastrophic error handling system can be understood by tracing through the flowcharts in figures 17 and 19 to 23.

In general, the user would like to know the cause of any errors which occur. To help him determine what happened, both the divide interrupt service routine and the 8087 exception service routine save the instruction pointer and

the code segment of the instruction after which the error occurred. These values can be examined via MINDS to determine which line of code prompted the error. In addition, the 8087 exception handler stores the 8087 status word and the address of the 8087 environment. Since this saved information would be overwritten the next time a similar error occurs, these two service routines each set a latch to prevent new information from being stored. After the data have been examined, the user can use MINDS to reset the latches in preparation for the next error should one occur.

The only catastrophic error which is handled by an interrupt is a system bus timeout error (fig. 25). This error usually means that program control is lost and that execution has ceased. The timeout interrupt brings control to the service routine where it remains until the routine has executed and control is returned to the previously running instruction address. If this error occurs, a message is printed out immediately in the service routine. The message contains the location of the instruction pointer and code segment of the calling instruction which failed. This can be used to aid in reconstructing what caused the timeout.

A list of the messages which can be printed appears in table VIII.

SIMULATION RESULTS

The steady-state accuracy of the model is excellent. This is because the HYTESS-like model was based on the steady-state performance of a turbofan engine and the base point calculations which define steady-state performance in HYTESS were derived from steady-state data. Also, the steady-state and transient accuracies of the actuator simulation are excellent. The full engine transient performance for small perturbations about a given operating point is also quite good. The full engine large perturbation transient performance is quite limited since the engine is modeled as a linear system in the run loop.

CONCLUSIONS

Tests conducted in conjunction with the F100 Hybrid Simulation evaluation of the ADIA algorithm showed that the simulator works well as a real-time, steady-state and small perturbation substitute for the full hybrid, nonlinear simulation. The full-scale engine demonstration of the ADIA proved the capabilities of the simulator as a real-time code verifier and as a full envelope, real-time actuator simulator for actuator fault detection. This real-time, portable simulator capability will be valuable in future engine tests. With the rapid increases in microprocessor capabilities that have occurred since the F100 simulator was built, it is conceivable that full envelope, full engine simulation can now be achieved in real-time.

APPENDIX A

USER'S MANUAL FOR F100 ENGINE SIMULATOR

1. Turn on all of the equipment, i.e., the chassis, the disk drive, and the terminal.

2. Insert the system disk into drive a: and the program disk into drive b:.

3. Boot the simulator by pressing the RESET button on the chassis.

4. When the simulator has booted, load and start the program by typing b:⟨program-name⟩⟨RETURN⟩.

5. This causes the program to start executing. It goes through the one-time initialization routines, MSET and MTRXST, and enters the initialization loop containing INLET and EMODEL. In the spare time in this loop, MINDS runs, allowing the values of variables and flags to be changed. The MINDS variable definitions must either be entered by hand or loaded from disk. Choose the mode in which the program is to be run. This can be changed at any time very simply. The default mode is initialization/hybrid/open-loop. Each switch (flag) can be changed independently.

6. Altitude and Mach number, ALT and XM0 respectively, can be changed through MINDS. They are both initialized to 0.0, i.e., sea-level static conditions. Power lever angle is initialized to 83°. The ambient conditions, which are all calculated in INLET, depend on these values. The ambient conditions are initialized to standard day conditions, i.e., about 14.7 psi and about 59° F. For changes of these two variables to have any affect, the program must go through the initialization loop one time. The base points are calculated here and their values are stored for the additional purpose of being the set-points in the open-loop mode.

7. Setting the value of RLOOP to 1 puts the simulation into the real-time run loop. The routines take about 10 msec to run leaving approximately 2 msec for MINDS provided there are no error messages to be printed. In this mode, MINDS can be used to check the value of variables and to switch modes.

8. Setting RLOOP to 0 again returns the program to the initialization loop but leaves the value of every variable unchanged. Thus a transient can be stopped and restarted (if the program is in open-loop mode) or the operating conditions can be altered to move the system to another operating point.

9. To stop the simulation, reboot the system by pressing the RESET button on the chassis with the system disk in drive a:.

APPENDIX B

ROUTINE	DESCRIPTION
ACTUAT	This FORTRAN subroutine simulates the actuator dynamics and lags each one to approximate the sensor dynamics on each actuator's output. The scheduled value of nozzle area computed in RPLIMD is used in the nozzle area simulation except in the actuator mode where the scheduled value of nozzle area is calculated based on TT2 provided as an analog input to the simulation.
ALTABL	This FORTRAN subroutine takes the altitude and returns the δ and θ corresponding to the ambient conditions.
CIMDAC	This assembly language program takes a 16-bit scaled integer and converts the twelve most significant bits to analog for output over a specified digital-to-analog converter channel.
EMODEL	This FORTRAN subroutine calculates u_b, x_b, and y_b from the reference point schedules and computes the elements of the F, $F^{-1}G$, H, and D matrices by evaluating functions of the ambient conditions and reference point schedules.
EVOLVE	This FORTRAN subroutine uses Euler integration to compute the current values of the engine outputs. It also simulates the output sensor dynamics by lagging each output.
EXEC	This assembly language program is the main routine for the F100 simulation. It calls the other major subroutines and contains the interrupt service routines.
FUN1	This assembly language program does a table look-up and interpolation on a function of one variable.
FUN2	This assembly language program does a table look-up and interpolation on two functions simultaneously where the functions use the same independent variable and the values of the two functions are known for the same values of the independent variable.
FUN3	This assembly language program does a table look-up and interpolation on two functions simultaneously where the functions use the same independent variable and the values of the two functions are known for different values of the independent variable.
HFTA	This FORTRAN function returns enthalpy as a function of temperature.
INLET	This FORTRAN subroutine calculates the inlet conditions of the engine, TT2 and PT2, and the compressor inlet temperature, TT25. The dynamics of the temperature and pressure sensors are included for completeness but are multiplied by zero. The sensor dynamics are not used because the simulation is a steady-state model and is only accurate at the specified operating points, not between them.

MESSAGE This routine, part of the MINDS Library, prints an ASCII string on the console.

MINDS Microcomputer INteractive Data System is a program used to examine and change memory locations in 8086-based systems. Because of memory constraints, a reduced capability version of MINDS, SMINDS, was used. For more information, see reference 13.

MSET This FORTRAN subroutine initializes variables such as ambient conditions, engine states, engine outputs, actuator outputs, all both unlagged and lagged to simulate sensed values, time constants, associated exponentials, and integration step size.

MTRXST This FORTRAN subroutine initializes the elements of the H and D matrices which remain constant at all operating conditions. The flag DFLAG is initialized to 0. This flag is used in EMODEL to indicate whether or not to recalculate several of the matrix elements. The flag's value must be changed using MINDS.

NEFG This assembly language routine does table look-up and interpolation on a function of one variable using the slope/intercept method.

PRCMB This FORTRAN subroutine calculates properties of combustion.

PVAL This FORTRAN function evaluates a polynomial passed as an argument to it.

RFUN1S This assembly language program interpolates between points in a lookup table. It works with small-model programs, i.e., the code sections of all modules are combined and allocated within one segment. The program is described in more detail in reference 14.

RPFAND This assembly language program calculates reference point schedules for the engine variables. The first time through, the routine starts at the label RPFANINT which is slightly earlier in the code than RPFAND. This first part of the code initializes pointers which are used in successive calls of RPFAND.

RPLIMD This assembly language program is the continuation of RPFAND. It calculates more reference point schedules and limits for the scheduled values. The first time through, the routine starts at the label RPLIMINT which is slightly earlier in the code than RPLIMD. This first part of the code initializes pointers which are used in successive calls of RPLIMD.

TFHA This FORTRAN function returns temperature as a function of enthalpy.

REFERENCES

1. Merrill, W.C., et al.: Advanced Detection, Isolation, and Accommodation of Sensor Failures - Engine Demonstration. NASA TP-2836, 1988.

2. Merrill, W.C.; DeLaat, J.C.; and Bruton, W.M.: Advanced Detection, Isolation, and Accommodation of Sensor Failures - Real-Time Evaluation. NASA TP-2740, 1987.

3. DeLaat, J.C.; and Soeder, J.F.: Design of a Microprocessor-Based Control, Interface and Monitoring (CIM) Unit for Turbine Engine Controls Research. NASA TM-83433, 1983.

4. Merrill, W.C., et. al.: HYTESS - A Hypothetical Turbofan Engine Simplified Simulation. NASA TM-83561, 1984.

5. Merrill, W.C.: HYTESS II - A Hypothetical Turbofan Engine Simplified Simulation With Multivariable Control and Sensor Analytical Redundancy. NASA TM-87344, 1986.

6. DeLaat, J.C.; and Merrill, W.C.: A Real-Time Implementation of an Advanced Sensor Failure Detection, Isolation, and Accommodation Algorithm. AIAA Paper 84-0569, Jan. 1984 (NASA TM-83553).

7. Beattie, E.C., et al.: Sensor Failure Detection For Jet Engines. (PWA-5891-18, Pratt and Whitney Aircraft; NASA Contract NAS3-23282) NASA CR-168190, 1983.

8. iSBC 86/14 and iSBC 86/30 Single Board Computer Hardware Reference Manual. Order Number: 144044-001, Intel Corp., Santa Clara, CA, 1982.

9. ZX-200A Single Board Disk Controller. Publication Number-98-200A, Zendex Corp., Dublin, CA, 1983.

10. User Manual for DT1741 SERIES (DT1741, DT1742, DT1744, DT1751, DT1754, DT1841, DT1842, DT1843. Analog Input, Analog Output and Analog I/O Systems), Document UM-00048-3 (Ref. DTI-UM-1740-3), Data Translation, Inc., Marlboro, MA.

11. ASM86 Language Reference Manual. Order Number: 121703-002, Intel Corp., Santa Clara, CA, 1982.

12. DeLaat, J.C.: A Real-Time FORTRAN Implementation of a Sensor Failure Detection, Isolation and Accommodation Algorithm. Proceedings of the 1984 American Control Conference, Vol. 1, IEEE, 1984, pp. 572-573.

13. Soeder, J.F.: MINDS - A Microcomputer Interactive Data System for 8086-Based Controllers. NASA TP-2378, 1985.

14. Mackin, M.A.; and Soeder, J.F.: Floating-Point Function Generation Routines for 16-Bit Microcomputers. NASA TM-83783, 1984.

TABLE I. - SIMULATOR INPUTS

Input channel number	Variable	Description
8	WFCOM	Commanded main combustor fuel flow (WF)
9	AJCOM	Commanded exhaust nozzle area (AJ)
10	CIVVCM	Commanded fan inlet variable vane angle (CIVV)
11	RCVVCM	Commanded rear compressor variable vane angle (RCVV)
12	BLCCM	Commanded compressor bleed (BL) (bleed is used open-loop)
13	TT2ACT	Fan inlet temperature (used only in actuator mode)

TABLE II. - SIMULATOR OUTPUTS

Output channel number	Variable	Description
1	Timing DAC	Variable-height step output used to determine the running time of each subroutine
2	WFFBS	Sensed main combustor fuel flow (WF)
3	AJS	Sensed exhaust nozzle area (AJ)
4	CIVVS	Sensed fan inlet variable vane angle (CIVV)
5	RCVVS	Sensed rear compressor variable vane angle (RCVV)
6	BLFBS	Sensed compressor bleed (BL) (not used)
7	POS	Ambient (static) pressure (PO)
8	PT2	Fan inlet (total) pressure
9	TT2	Fan inlet temperature
10	TT25	Compressor inlet temperature
11	N1	Sensed fan speed
12	N2	Sensed compressor speed
13	PT4	Sensed combustor pressure
14	PT6	Sensed exhaust nozzle pressure
15	FTIT	Sensed fan turbine inlet temperature
16	PLA	Power lever angle

TABLE III. - INTEL 86/30 BOARD HARDWARE CONFIGURATION

86/30 Jumper connections	Description
7-11	2 wait states on EPROM access
13-14	2 wait states on I/O access
38-39	Timeout enabled
108-109	2716 select
111-112	2716 select
118-119	128K total ram on board
144-145	Ground NMI
151-152	Multibus interrupt 5/ to 8259 IR5
158-147	Timer 0 interrupt to 8259 IR2
175-176	1.23 MHz clock to CTR0 - out
178-179	1.23 MHz clock to CTR2
184-185	153.6 KHZ clock to CTR1
190-194	8753 out 2 to 8251 TXC
191-195	8753 out 2 to 8251 RXC
205-207	BCLK to Multibus
208-209	CCLK to Multibus
210-211	BPRO to Multibus - out
215-220	Out }
216-221	Out } Megabyte 0 (lowest) selected
217-222	Out }
219-224	Out }
225	n. c. }
226	n. c. } Off board address 0-1FFFFH
227	n. c. } (0 lowest address)
230-231	Out }
232-233	In } 128 K, 01FFFFH upper address
240-241	Out }
234-235	Out } All RAM available to bus, 128 K
236-237	Out }
33-34	Out Nonbus vectored interrupts
123-124-125	Out 2 K x 8 EPROM
189-193	DTR to DSR
202-203	ANYRQUEST line
213-212	CBEG line to ground
184-175	153.6 KHz clk to CTR0
133-165	Timeout interrupt to 8259 IR0
134-141	Timeout 1 int. to 8259 IR7 (for MINDS)
155-166	MINT to 8259 IR6

TABLE IV. - D/A BOARD PIN CONNECTIONS

Signal	DAC Board	Channel number	Pin number high	Pin number low	Comments
Timing DAC	1	1	17	18	
WFFBS		2	19	20	
AJS		3	21	22	
CIVVS		4	23	24	
RCVVS		5	25	26	
BLFBS		6	27	28	Not used
POS		7	29	30	
PT2	↓	8	31	32	
TT2	2	1	17	18	
TT25		2	19	20	
SNFSEN		3	21	22	N1 sensed
SNCSEN		4	23	24	N2 sensed
PT4		5	25	26	
PT6		6	27	28	
FTIT		7	29	30	
PLA	↓	8	31	32	

TABLE V. - ROUTINES AND ASSOCIATED COMMON BLOCKS

Common	Routine								
	ACTUAT	EVOLVE	EMODEL	INLET	MSET	MTRXST	RPFAND	RPLIMD	EXEC
XANDZ		X	X		X				X
FTICFC		X			X				
ACTOUT	X	X			X				X
MVCSHT	X	X		X	X				
AMBCND			X	X	X				X
FTSAV		X			X				
MATRIX		X	X			X			
BASEV		X	X	X					X
FILVAR		X							
EXPS	X	X		X	X				
JLCMN	X		X	X	X		X	X	X
RAMREC			X	X					
CONTROL	X								X

TABLE VI. - CONTENTS OF COMMON BLOCKS

Common	Contents
XANDZ	Engine states, engine outputs, sensed engine outputs
FTICFC	Fan turbine inlet temperature factors used to model FTIT sensor dynamics
ACTOUT	Actuator outputs, sensed actuator outputs
MVCSHT	Integration time step
AMBCND	Sensed ambient conditions
FTSAV	Slow and fast lag values used to model FTIT sensor dynamics
MATRIX	Matrices F, H, D, and $F^{-1}G$; and flag, DFLAG
BASEV	Base points for control inputs, engine states, and engine outputs
FILVAR	Intermediate variables used in system evolution routine
EXPS	Exponentials for all of the actuator and sensor dynamics
JLCMN	General variables that do not fit in another common
RAMREC	Variables used for ram recovery effect of the inlet
CONTROL	The control inputs to the actuators

TABLE VII. - SOFTWARE SWITCHES FOR MODE CHANGES

Software switch	Description
RLOOP	= 0, (default) program runs in initialization loop = 1, program runs in real-time run loop
PSL	= 0, (default) scaling of inputs and outputs corresponds to that of Hybrid simulation = 1, scaling of inputs and outputs corresponds to Propulsion Systems Laboratory hardware
CLLOOP	= 0, (default) program runs in open-loop mode, command signals are taken from memory (the values can be changed using MINDS) = 1, program runs in closed-loop mode, analog command signals are read in through A/D converters
CALIB	= 0, (default) each routine in run loop is executed fully = 1, only the A/D converter and D/A converter routines are executed in the run loop, ACTUAT and EVOLVE are not. Thus the effect of scale factors for both input and output can be checked directly using MINDS
ACTSIM	= 0, (default) scheduled AJ (nozzle area) is proportional to the steady-state scheduled value calculated in RPLIMD = 1, scheduled AJ is calculated as a function of TT2 read in by the simulation at each control interval. **This should only be used in the actuator simulation mode.**

TABLE VIII. - ERROR MESSAGES

Message	Description
8086 F-100 SIMULATION	Sign on message
UPDATE FAILURE OCCURRED	Update failure message
DIVIDE INTERRUPT OCCURRED	Divide interrupt message
FLOATING POINT EXCEPTION OCCURRED	Floating point exception message
SYSTEM BUS TIMEOUT!!!!! IP AT XXXX SEGMENT AT XXXX	Bus timeout message

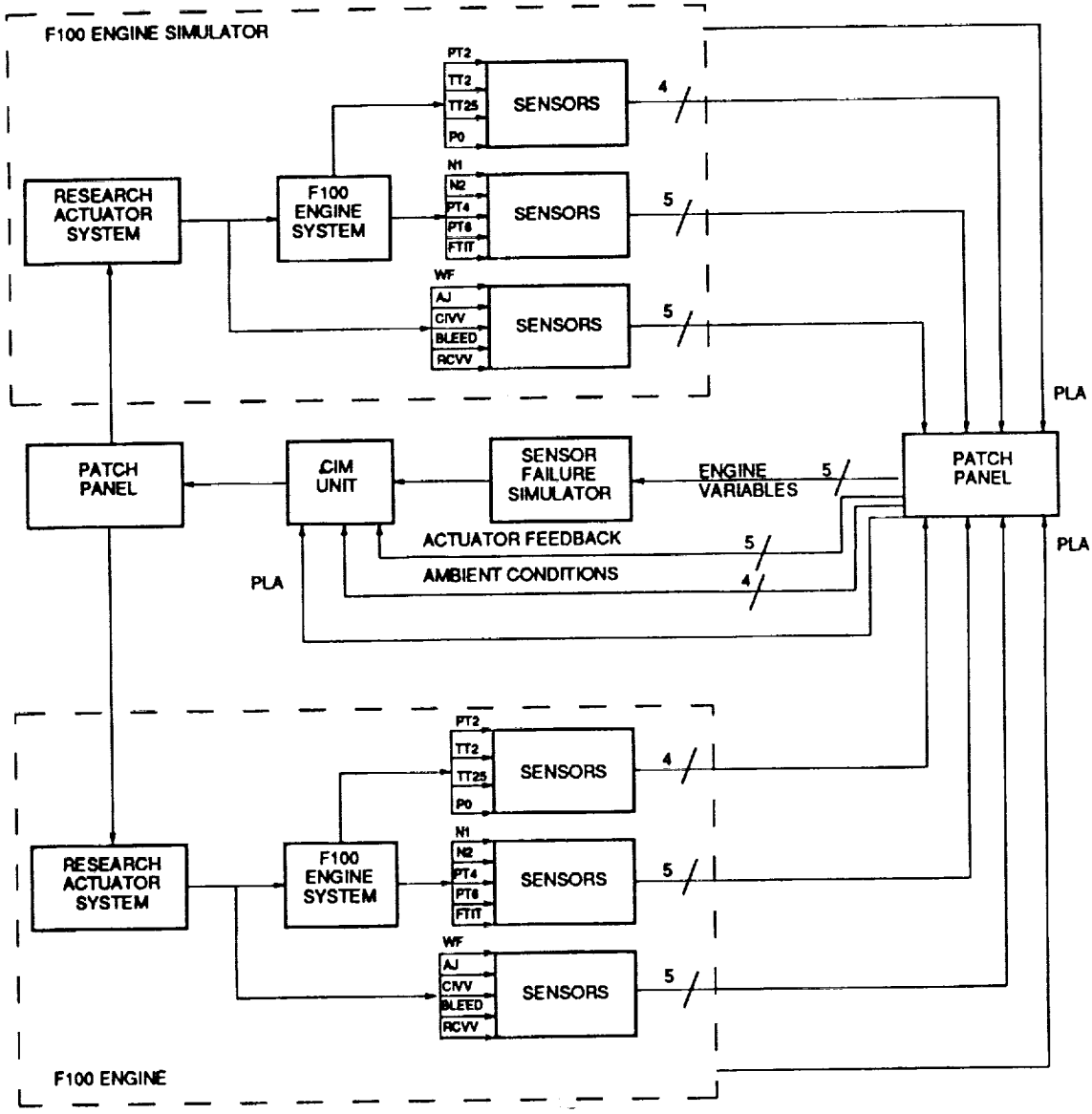

Figure 1. - Test setup in PSL.

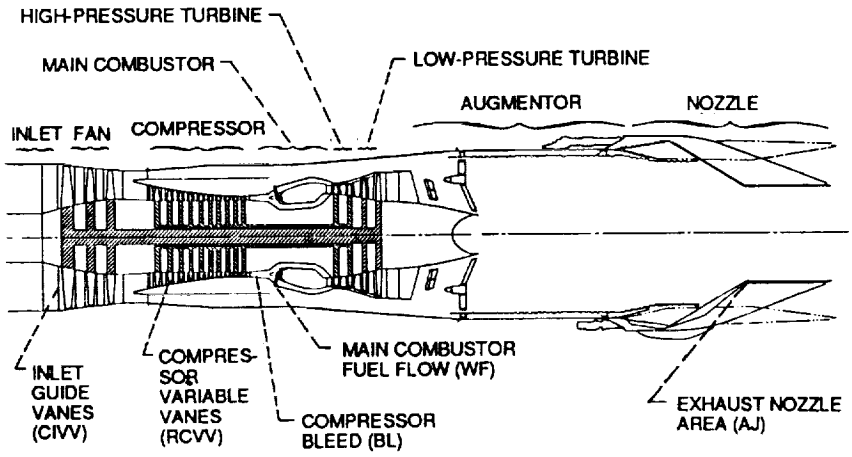

Figure 2. - F100 engine inputs.

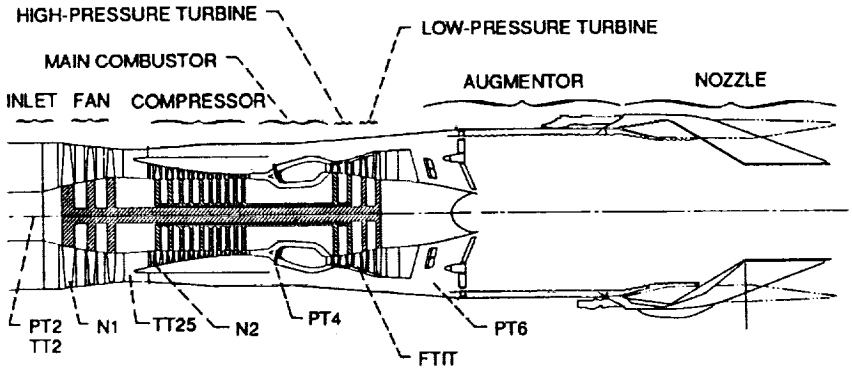

Figure 3. - F100 sense points.

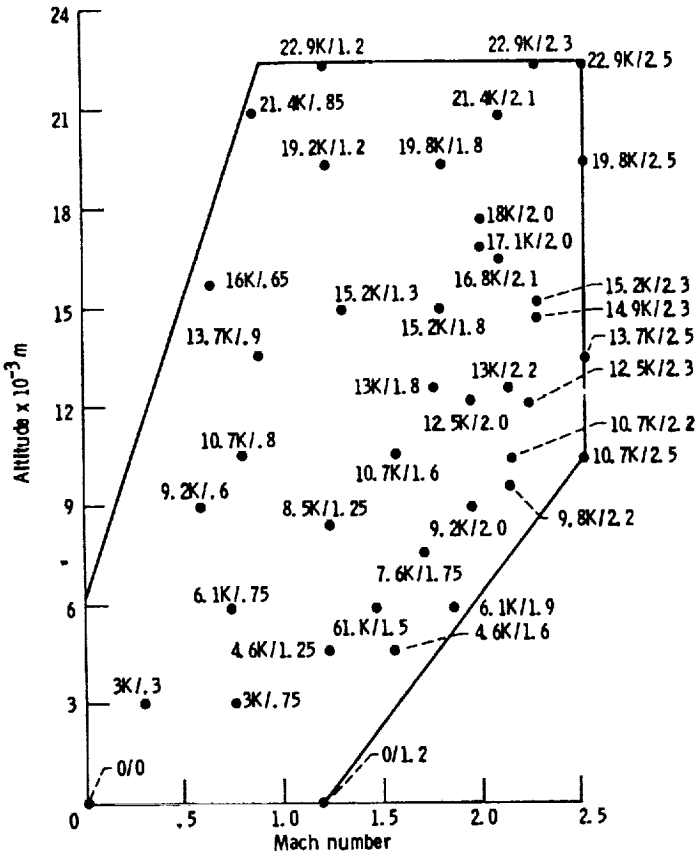

Figure 4. - Flight envelope for F100 model with power lever angle = 83 ^{o}.

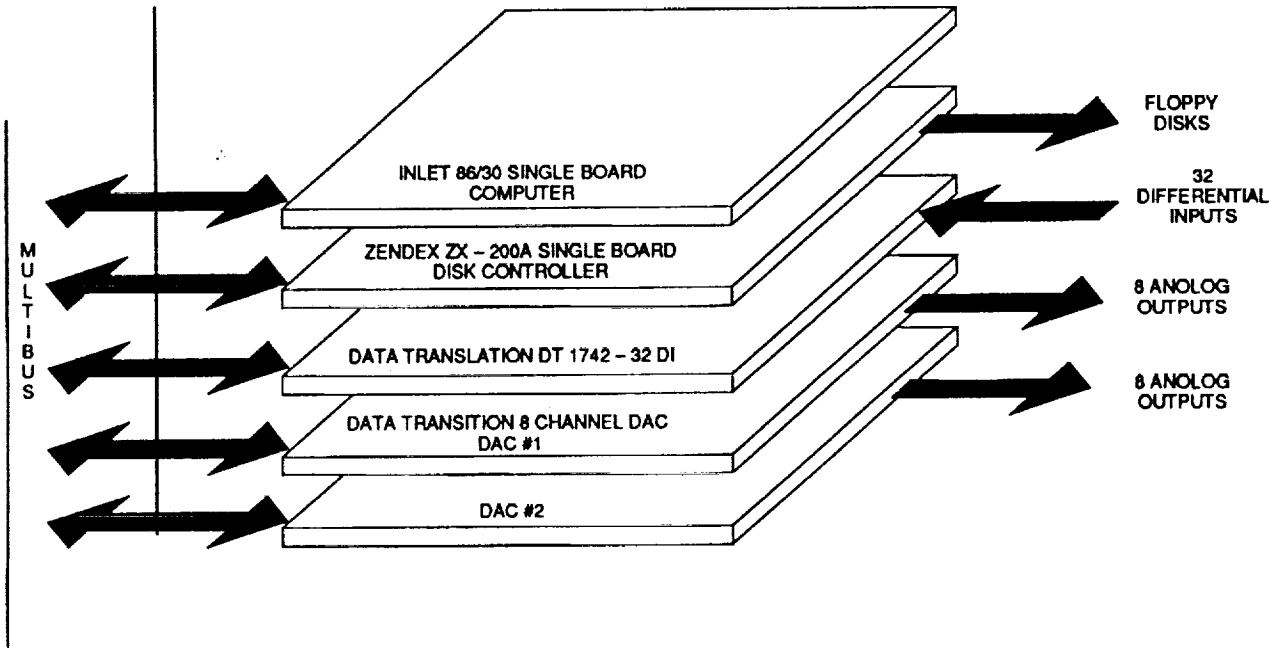

Figure 5. - Engine simulator hardware.

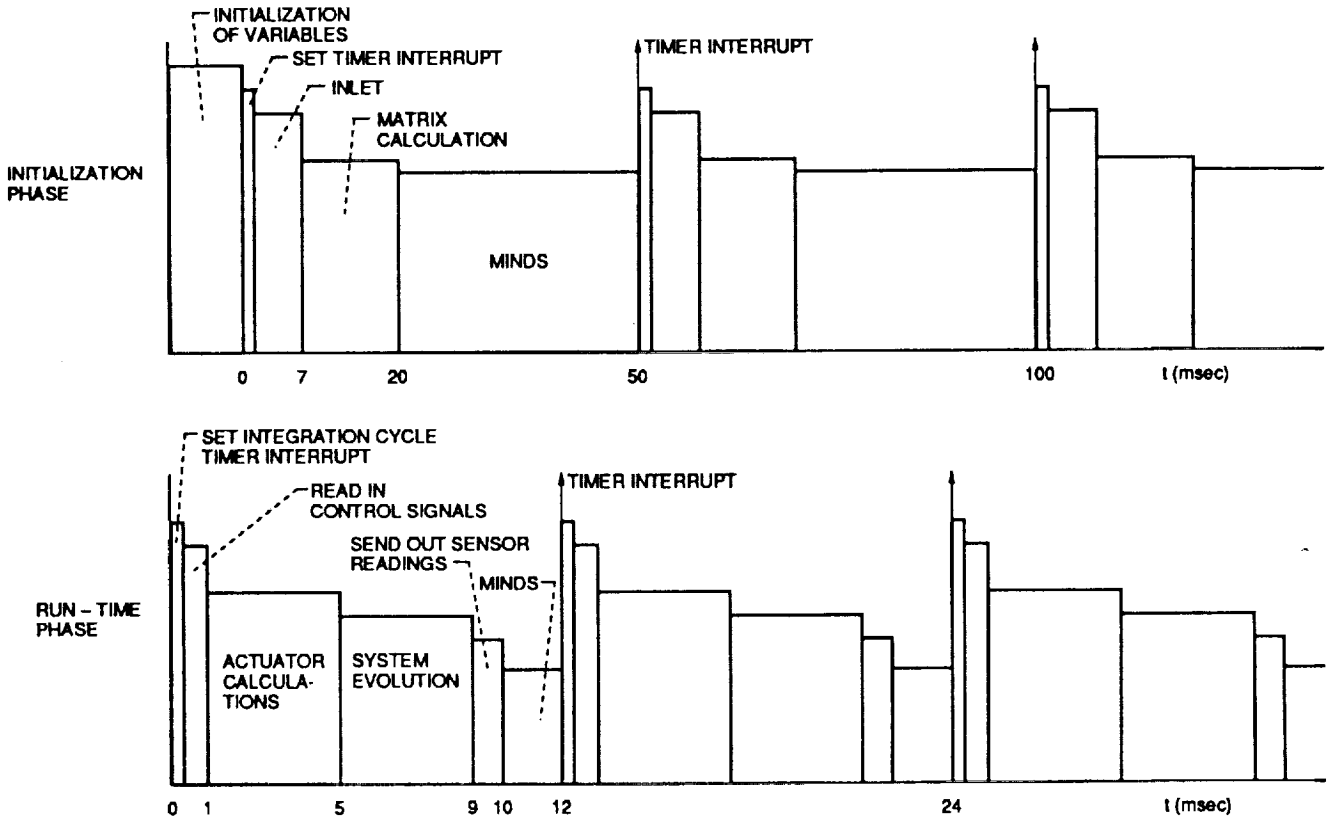

Figure 6. - Software timing diagrams.

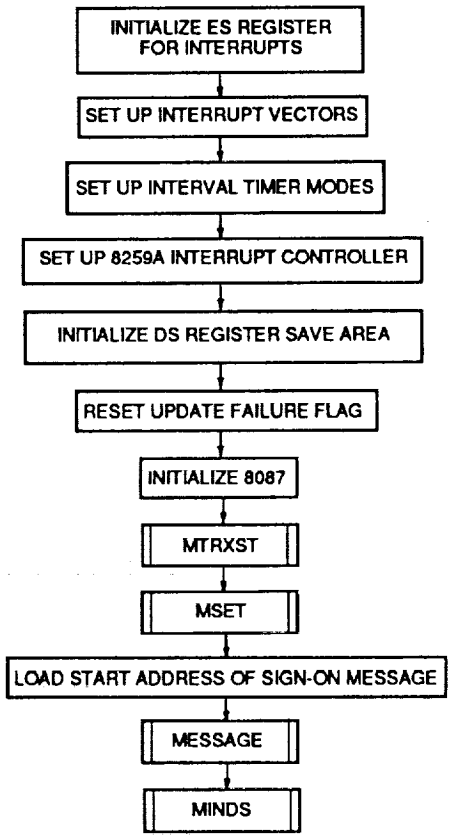

Figure 7. - Flow chart for executive routine, EXEC.

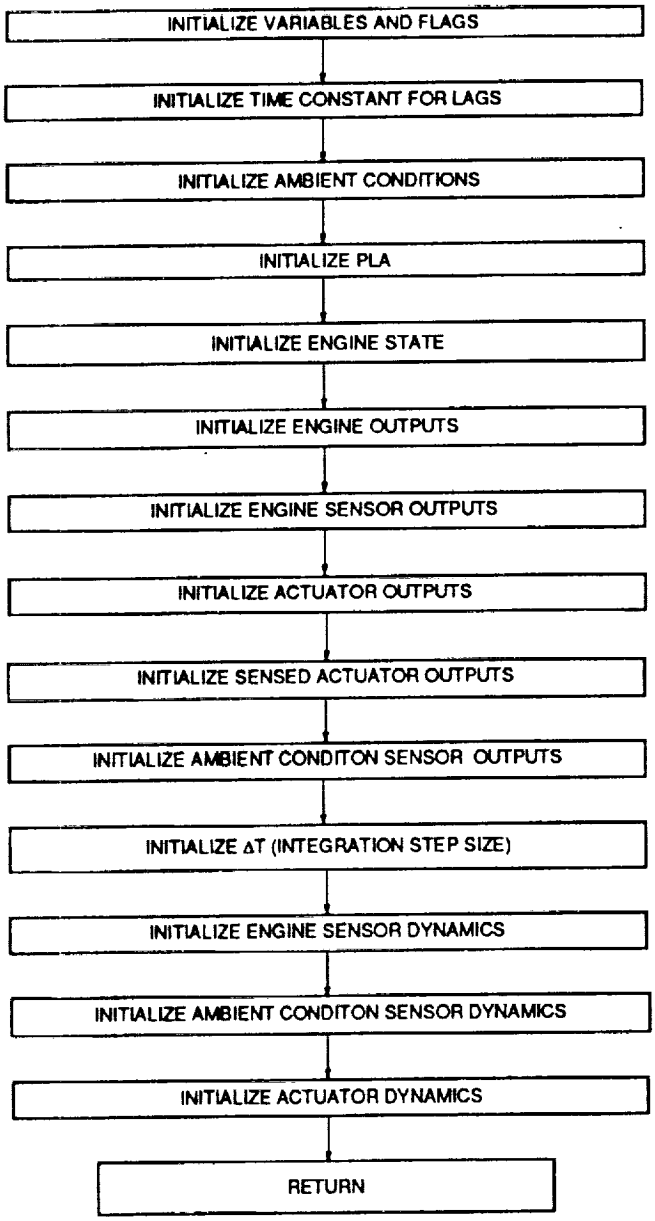

Figure 8. - Initilization routine, MSET.

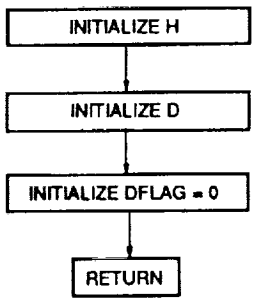

Figure 9. - Matrix initialization routine, MTRXST.

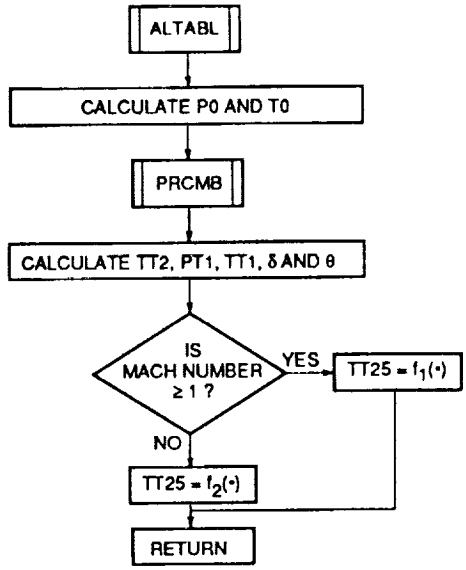

Figure 10. - Flow chart for inlet routine, INLET.

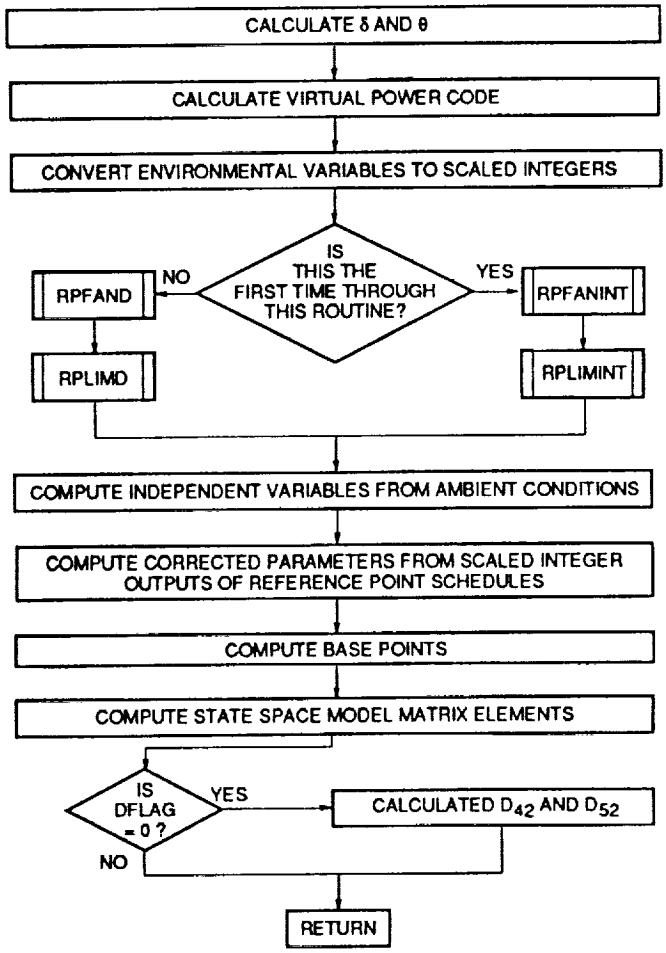

Figure 11. - Flow chart for routine to determine model at operating point, EMODEL.

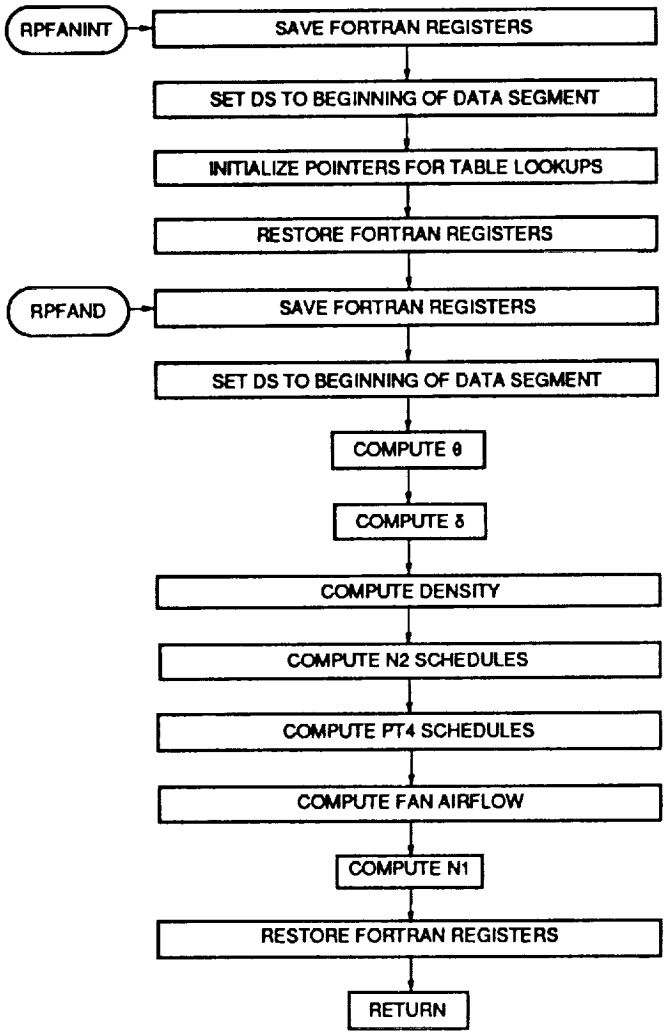

Figure 12. - Flow chart for setpoint calculation routine, RPFAND.

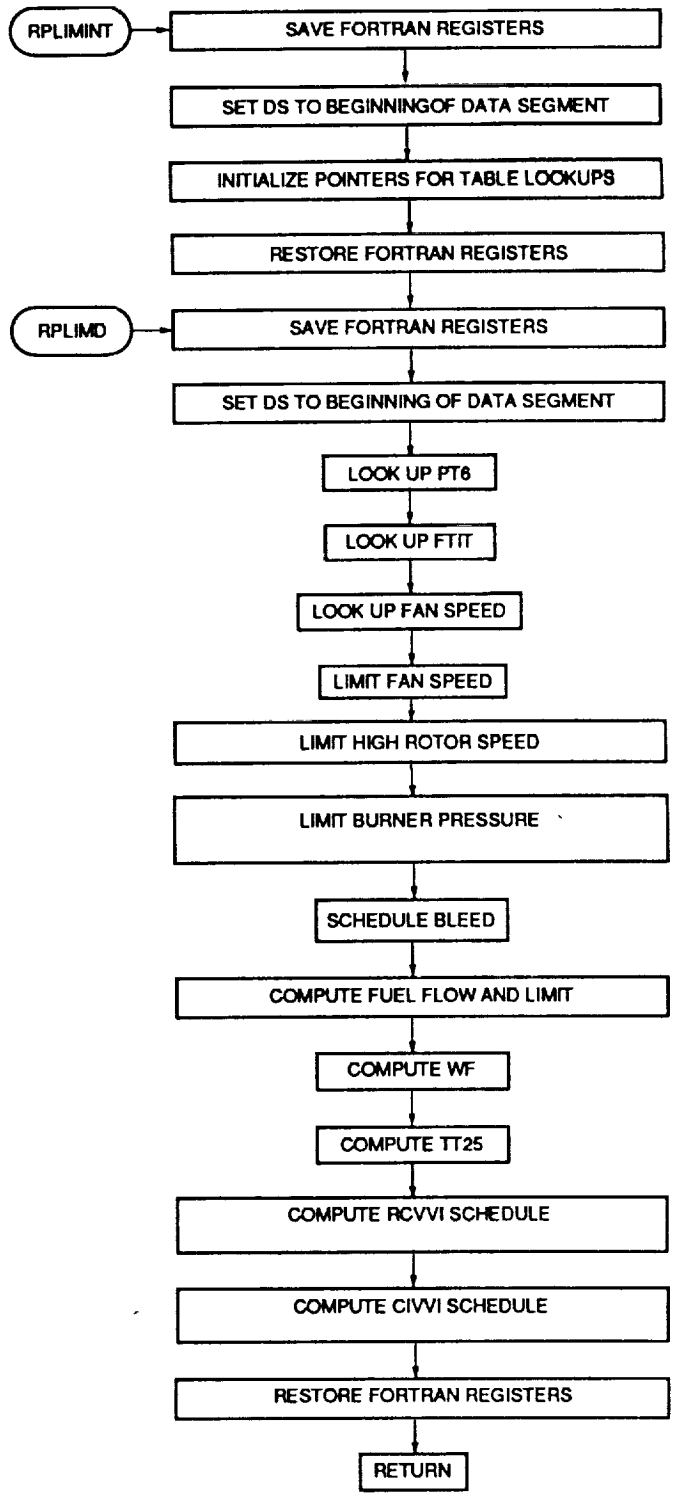

Figure 13. - Flow chart for setpoint calculation routine, RPLIMD.

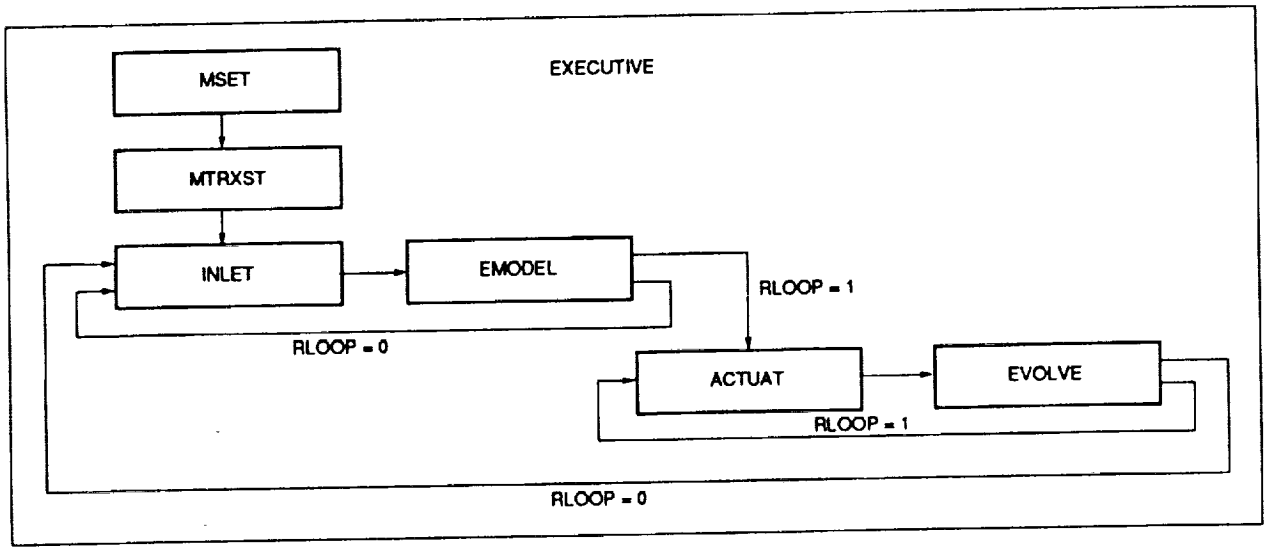

Figure 14. - Program flow.

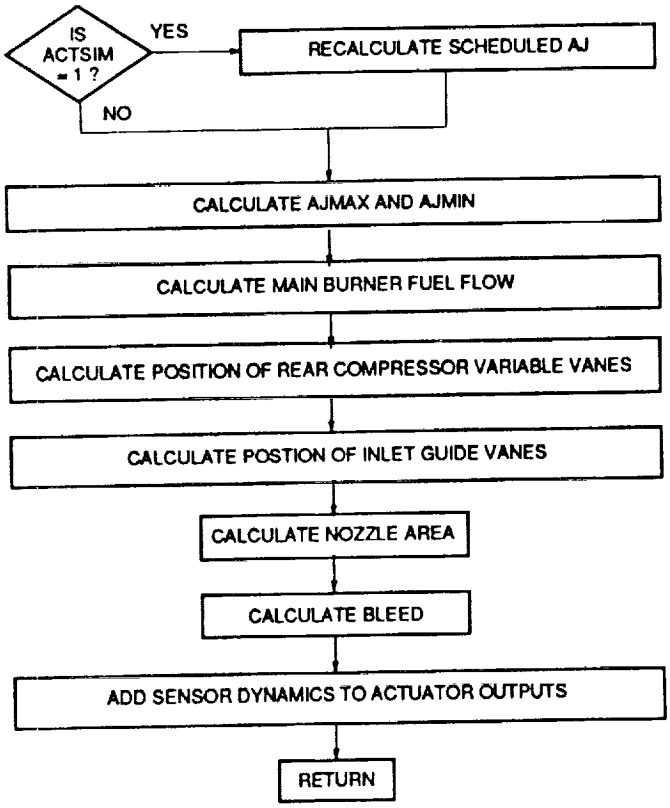

Figure 15. - Flow chart for actuator routine, ACTUAT.

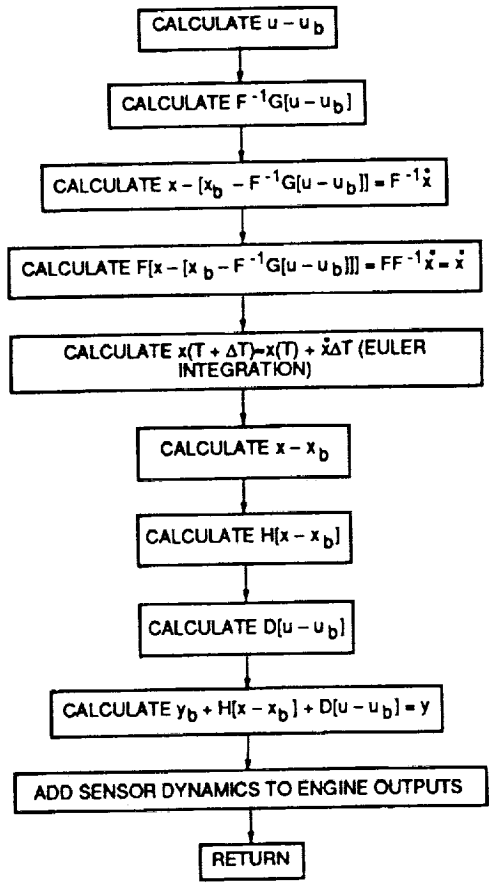

Figure 16. - Flow chart for system evolution routine, EVOLVE.

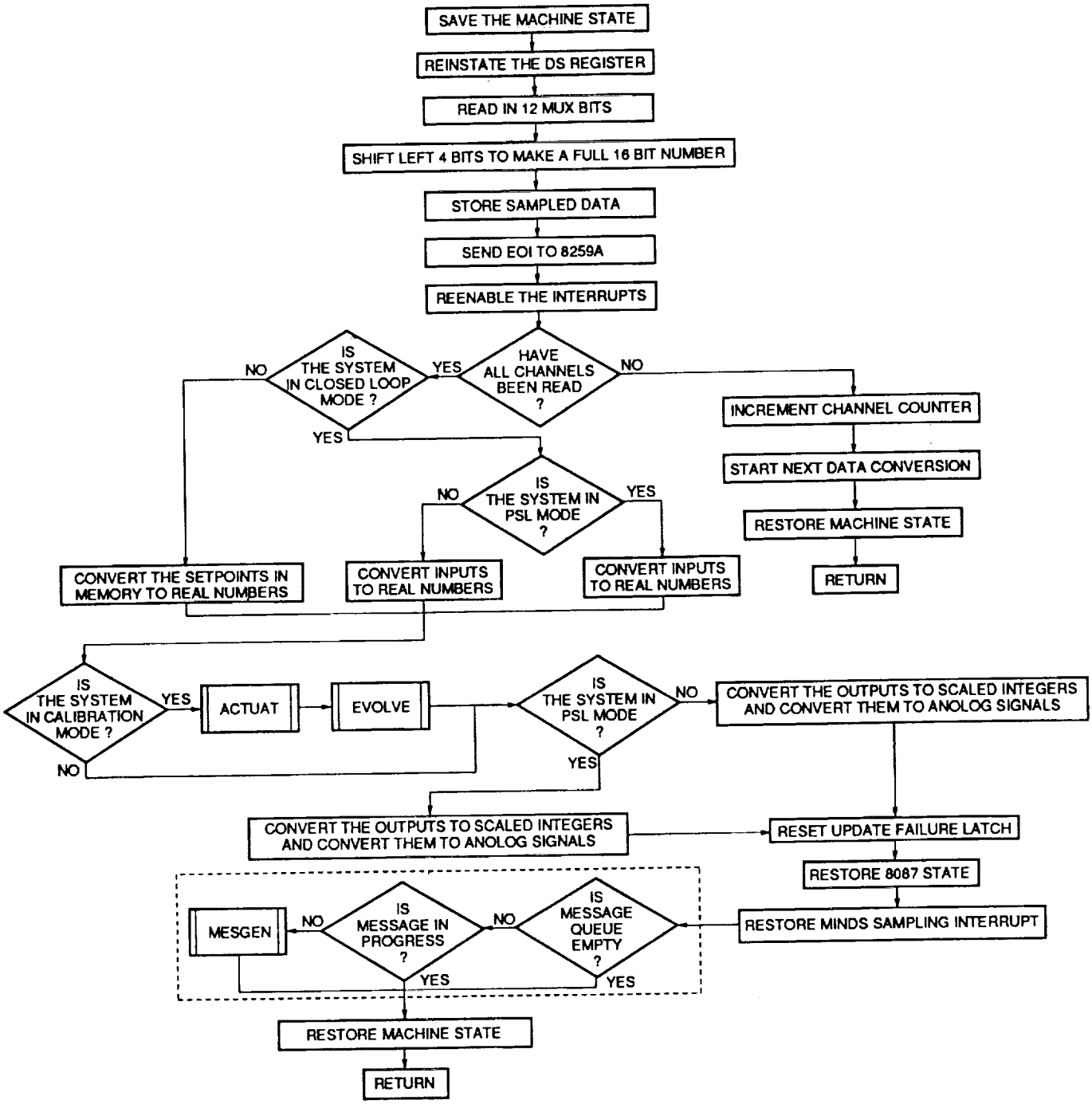

Figure 17. - Flow chart for multiplexer interrupt service routine.

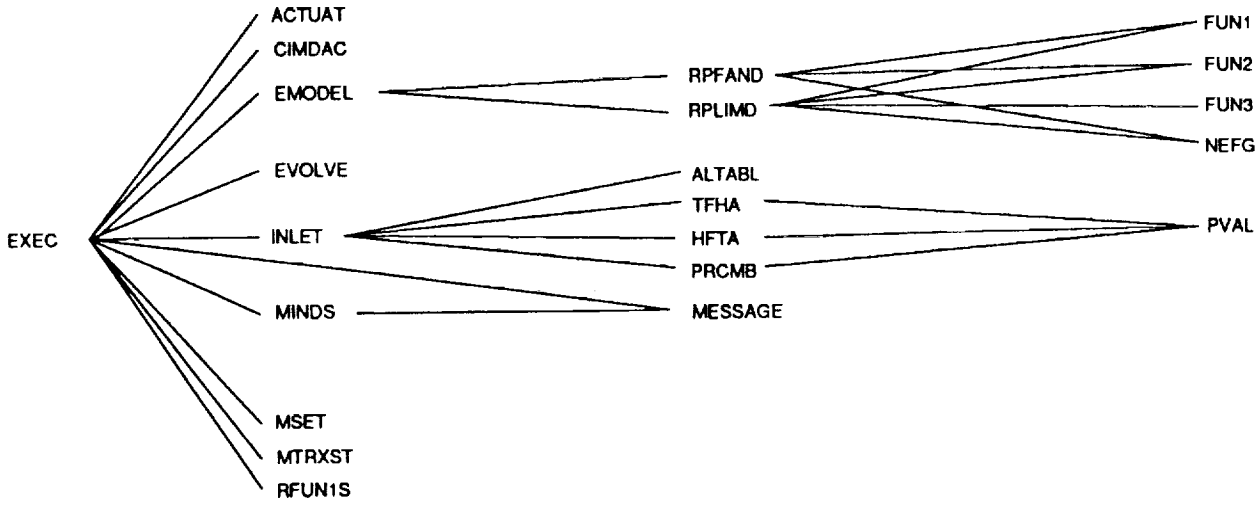

Figure 18. - Hierarchy of subroutine calls.

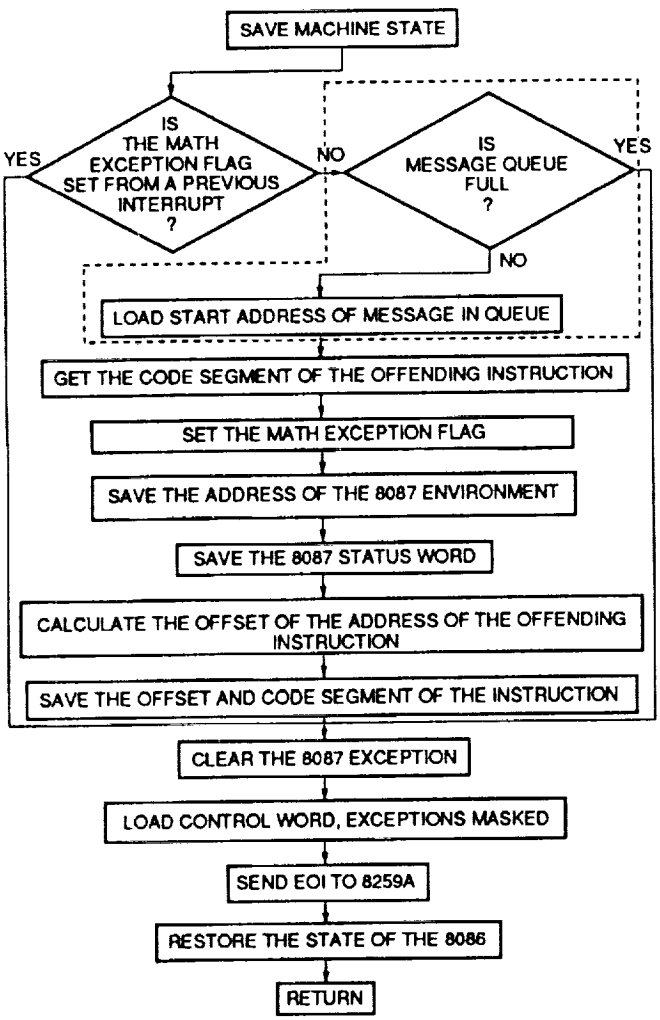

Figure 19. - Flow chart for 8087 exception interrupt service routine.

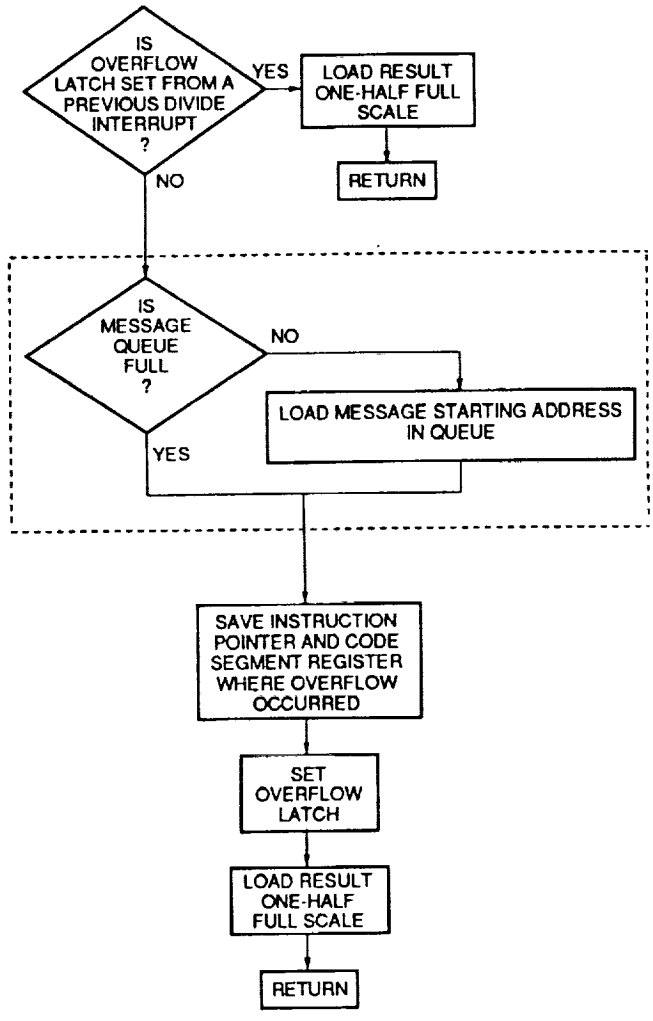

Figure 20. - Flow chart for divide interrupt service routine.

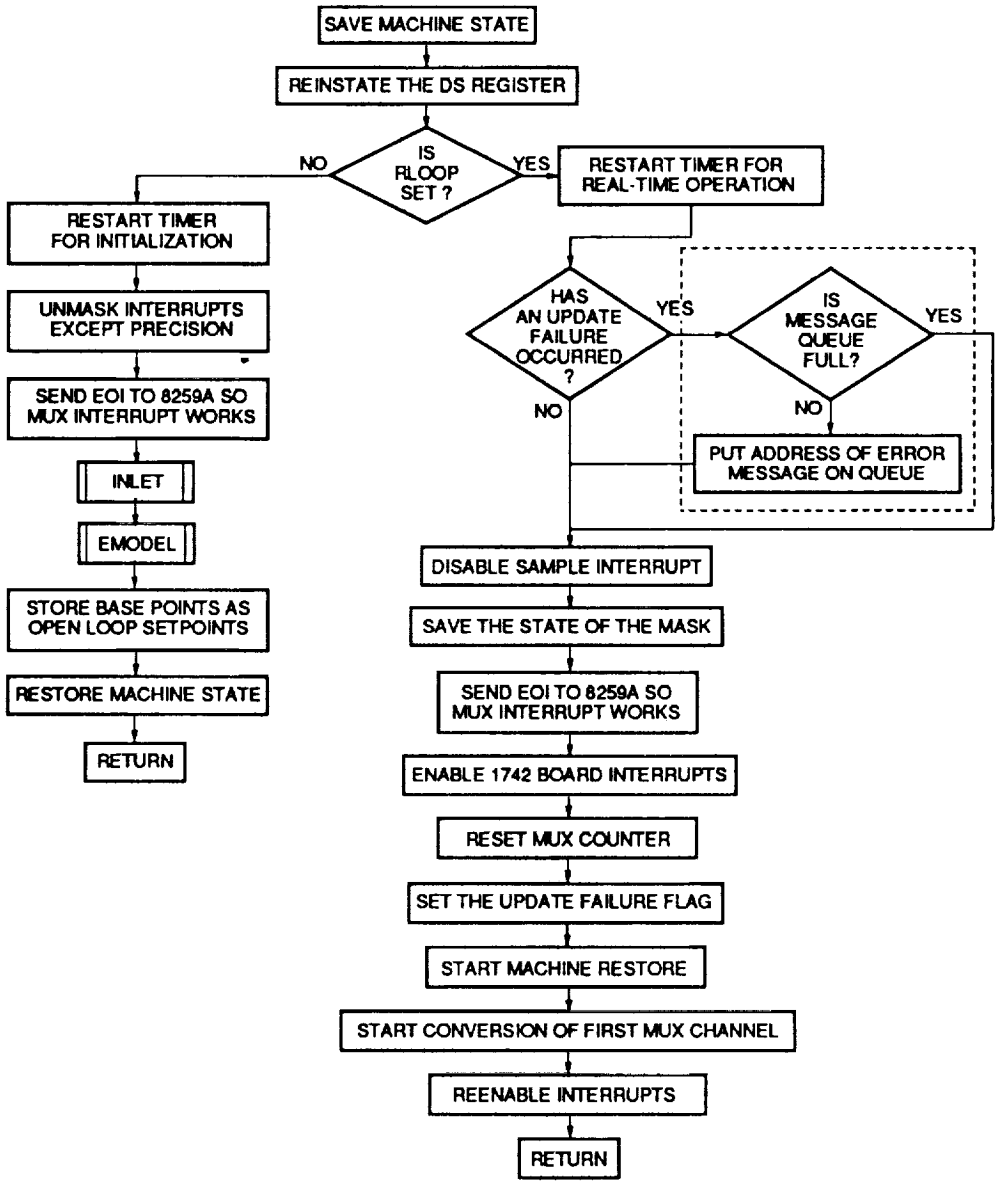

Figure 21. - Flow chart for timer interrupt service routine.

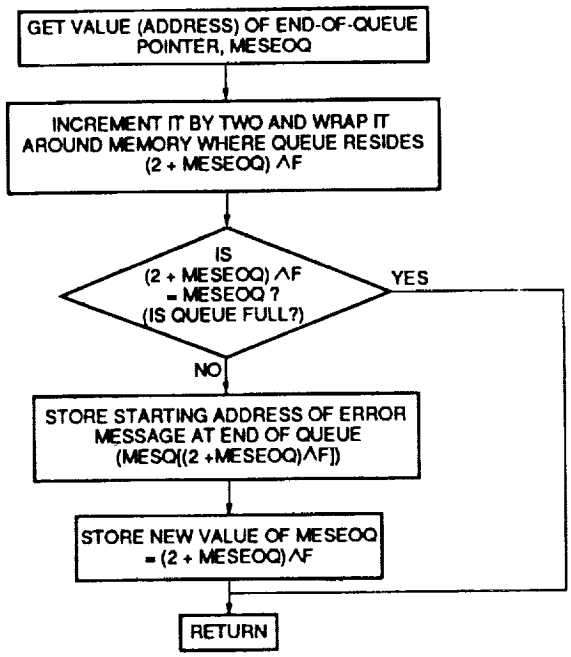

Figure 22. - Flow chart for routine for saving starting address of error messages.

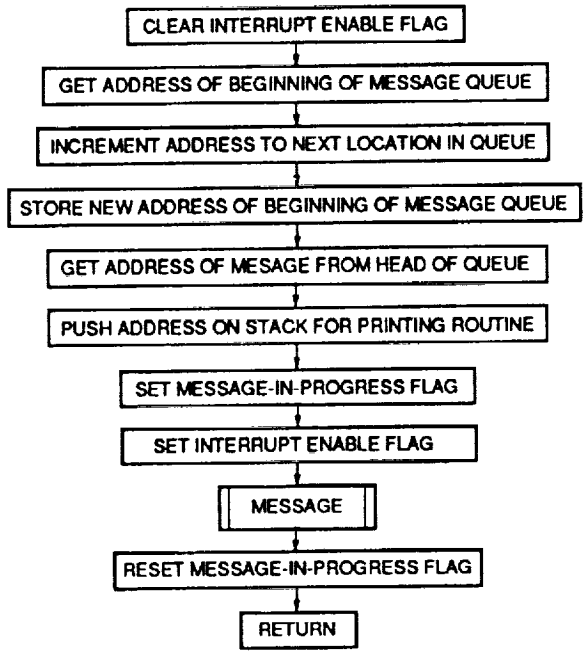

Figure 23. - Flow chart for message generation routine, MESGEN.

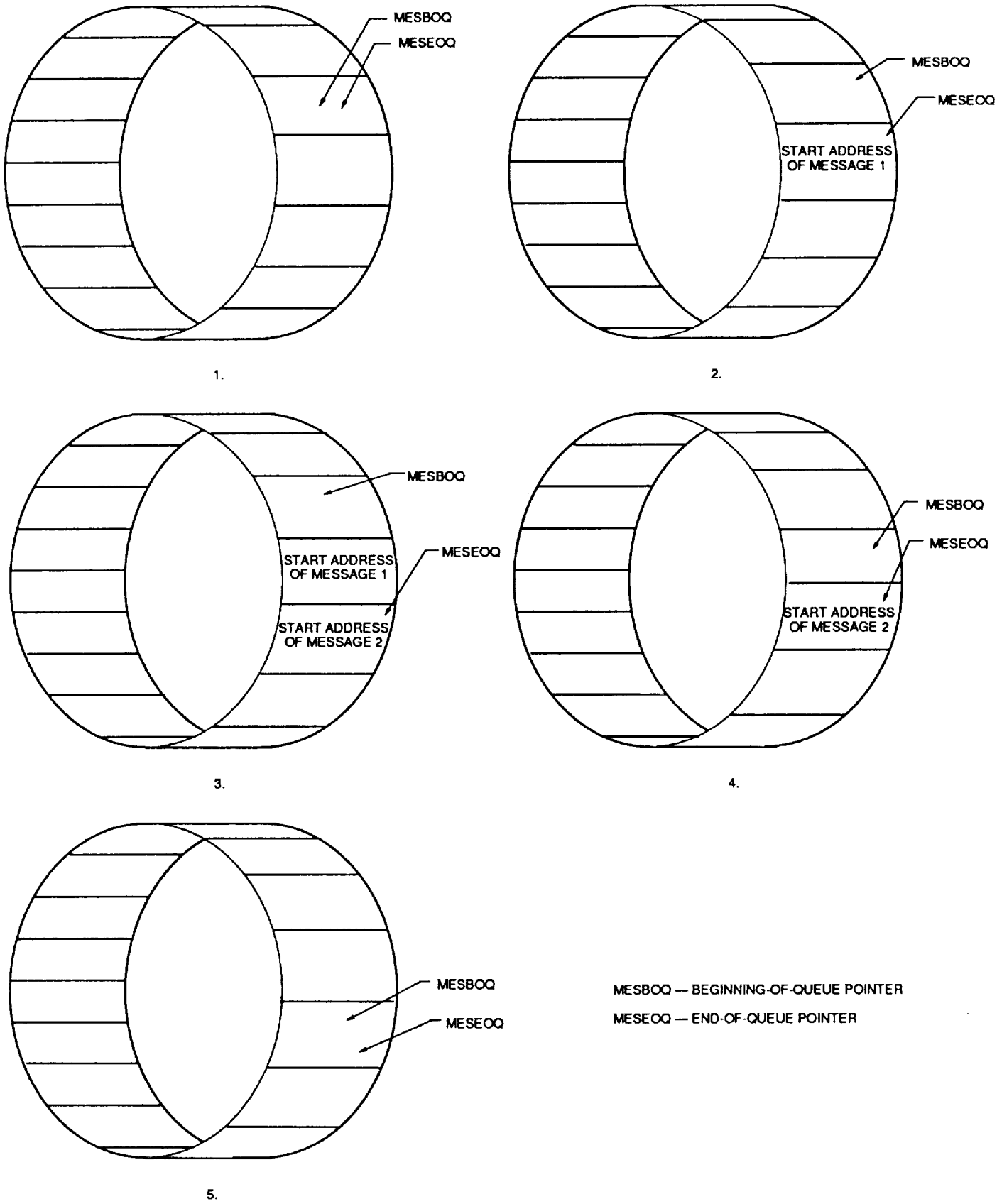

Figure 24. - Two messages entering queue and being printed.

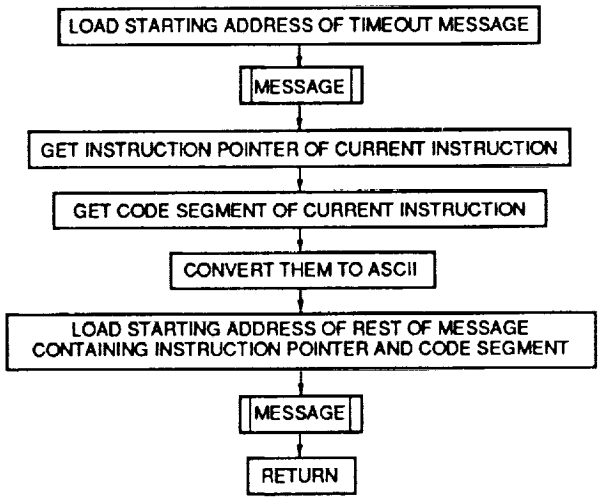

Figure 25. - Flow chart for system bus timeout interrupt service routine.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-100869 AVSCOM TR-89-C-001		

4. Title and Subtitle
A Real-time Simulator of a Turbofan Engine

5. Report Date
March 1989

6. Performing Organization Code

7. Author(s)
Jonathan S. Litt, John C. DeLaat, and Walter C. Merrill

8. Performing Organization Report No.
E-4578

10. Work Unit No.
505-62-01

9. Performing Organization Name and Address
NASA Lewis Research Center
Cleveland, Ohio 44135-3191
and
Propulsion Directorate
U.S. Army Aviation Research and Technology Activity—AVSCOM
Cleveland, Ohio 44135-3127

11. Contract or Grant No.

13. Type of Report and Period Covered
Technical Memorandum

12. Sponsoring Agency Name and Address
National Aeronautics and Space Administration
Washington, D.C. 20546-0001
and
U.S. Army Aviation Systems Command
St. Louis, Mo. 63120-1798

14. Sponsoring Agency Code

15. Supplementary Notes
Jonathan S. Litt, Propulsion Directorate; John C. DeLaat and Walter C. Merrill, NASA Lewis Research Center.

16. Abstract

A real-time digital simulator of a Pratt and Whitney F100 engine has been developed for real-time code verification and for actuator diagnosis during full-scale engine testing. This self-contained unit can operate in an open-loop stand-alone mode or as part of closed-loop control system. It can also be used for control system design and development. Tests conducted in conjunction with the NASA Advanced Detection, Isolation, and Accommodation program show that the simulator is a valuable tool for real-time code verification and as a real-time actuator simulator for actuator fault diagnosis. Although currently a small perturbation model, advances in microprocessor hardware should allow the simulator to evolve into a real-time, full-envelope, full engine simulation.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Microprocessor; Turbofan engine; Simulator; Real-time; Sensors; Actuators	Unclassified—Unlimited Subject Category 07

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No of pages	22. Price*
Unclassified	Unclassified	32	A03

*For sale by the National Technical Information Service, Springfield, Virginia 22161