ABSTRACT

DURING THE PAST ACADEMIC YEAR THE FOCAL POINT OF THIS PROJECT HAS BEEN TO ENHANCE THE ECONOMICAL FLIGHT SIMULATOR SYSTEM BY INCORPORATING IT INTO THE AERO ENGINEERING EDUCATIONAL ENVIRONMENT. TO ACCOMPLISH THIS GOAL IT HAS BEEN NECESSARY TO DEVELOP APPROPRIATE SOFTWARE MODULES THAT PROVIDE A FOUNDATION FOR STUDENT INTERACTION WITH THE SYSTEM. IN ADDITION EXPERIMENTS HAD TO BE DEVELOPED AND TESTED TO DETERMINE IF THEY WERE APPROPRIATE FOR INCORPORATION INTO THE BEGINNING FLIGHT SIMULATION COURSE, AERO-418. FOR THE MOST PART THESE GOALS HAVE BEEN ACCOMPLISHED. EXPERIMENTS HAVE BEEN DEVELOPED AND EVALUATED BY GRADUATE STUDENTS. MORE WORK NEEDS TO BE DONE IN THIS AREA. THE COMPLEXITY AND LENGTH OF THE EXPERIMENTS MUST BE REFINED TO MATCH THE PROGRAMMING EXPERIENCE OF THE TARGET STUDENTS. IT HAS BEEN DETERMINED THAT FEW UNDERGRADUATE STUDENTS ARE READY TO ABSORB THE FULL EXTENT AND COMPLEXITY OF A REAL-TIME FLIGHT SIMULATION. FOR THIS REASON THE EXPERIMENTS DEVELOPED ARE DESIGNED TO INTRODUCE BASIC COMPUTER ARCHITECTURES SUITABLE FOR SIMULATION, THE PROGRAMMING ENVIRONMENT AND LANGUAGES, THE CONCEPT OF MATH MODELS, EVALUATION OF ACQUIRED DATA, AND AN INTRODUCTION TO THE MEANING OF REAL-TIME.

THIS REPORT INCLUDES AN OVERVIEW OF THE SYSTEM ENVIRONMENT AS IT PERTAINS TO THE STUDENTS, AN EXAMPLE OF A FLIGHT SIMULATION EXPERIMENT PERFORMED BY THE STUDENTS, AND A SUMMARY OF THE EXECUTIVE PROGRAMMING MODULES CREATED BY THE STUDENTS TO ACHIEVE A USER-FRIENDLY MULTI-PROCESSOR SYSTEM SUITABLE TO AN AERO ENGINEERING EDUCATIONAL PROGRAM.

DUE TO THE RAPID CHANGING COMPUTER TECHNOLOGY RECOMMENDATIONS TO IMPROVE THE SYSTEM ARE INCLUDED. THESE RECOMMENDATIONS ARE DIRECTED TOWARD THE GRAPHICS PROBLEMS AND THE NEW COMPUTER ARCHITECTURES NOW AVAILABLE AT LOW COST.
The basic hardware architecture of the system has not been altered since the last proposal. The simulator is partitioned into three processors. The overall hardware is illustrated in Figure 1. The Intel 286/10 SBC acts as the command executive and graphics high-level controller. The Intel 386/22 SBC handles all real-time computations for control models and simulations. The Intel 186/78 SBC converts the high-level graphics commands into the appropriate graphics primitives and controls the Intel 82720 graphics processor.

Software development for any simulation is still conducted utilizing the operating system RMX86. No attempt has been made, as yet, to convert to RMX286. At present the Intel 310 development system has .5Mbytes of main memory. RMX286 requires at least .7Mbytes for configuration. Adding another .5Mbytes to the system would solve this problem but would introduce memory partitioning problems as it pertains to MULTIBUS 1. RMX286 operates in protected address mode. This forces a re-partitioning of MULTIBUS space and effects all the processors in the system. Memory strapping options for the 186/78 SBC and the 386/22 SBC are limited. This problem is still under investigation.

Input data to simulation models is currently created by software curve generation. Software modules have been developed this year to allow keyboard input to command the input function and control the type of input, i.e., a doublet, ramp, step, etc. The range of values and sample time are also controlled. Analog I/O is available but memory partitioning must be altered to accommodate the Robotrol RMB-731 analog I/O board.

The all "glass cockpit" concept of this project is still centered around the usage of an inexpensive graphics controller and an RGB color monitor. The 186/78 SBC controller board is the direct interface to the Prinston Graphics SRI2-P color monitor. The system still utilizes the Intel supplied VDI720 software package. This has proven to be inadequate for real-time displays because of software overhead. During the past academic year the graphics monitor has been utilized for the display of non real-time data. The students graph the results of a simulation run by displaying normalized curves of appropriate data in color. At present hardcopy of these displays can not be obtained.

The simulation system is centered around Intel's 310 development system. It will be proposed in the future to switch to an IBM PC/AT for all program development. This will make the 310 system the target system for simulation runs. In addition the students can develop software at a number of sites.
Figure 1.
The Intel Flight Simulator System is well suited for the educational environment because of the cost of the system and because it is a modular system. It can be viewed by the students as a complete flight simulator or as an educational tool designed to introduce senior aero students to the world of flight simulation. However the complexity of a multi-processing system without well established graphics has proven to be a disadvantage when the primary goal is modeling and the gathering of data.

For the past academic year the emphasis has been to develop the necessary software modules required to create a "student friendly" environment on the Intel 310 system. It was felt that student concentration should focus on aero modeling, input/output scaling, reduction of data for analysis, sample time, and frame time. Past experience has demonstrated that too much time can be spent introducing the system architecture, the programming environment, and the synchronization problems associated with a parallel processing system. The goal has been to minimize this overhead as it pertains to student involvement. Students concentrate on converting a control or simple simulation model into an equivalent set of equations. They create their own data bases and write their own integration algorithms. These program modules are linked and located for proper execution on the 386/22 processor. All software necessary to transfer their code to the 386 processor has already been developed. In addition the 386 processor supports a custom monitor designed to aid in the execution of the simulation.

To achieve the desired environment over 30 software procedures have been developed, linked and installed on the Intel 310 System during the past academic year. The students invoke the simulation software and follow the menu driven instructions. The menu instructions allow the student to perform the following operations:

1. Select and initialize input variables for a given run.

2. Select the input waveforms and limits. At present these include steps, ramps, and doublets. The inputs are software generated as the A/D convertor board cannot be installed due to memory constraints.

3. Download the simulation model to the 386/22 SBC for execution. After downloading the simulation model the initial data base is loaded by the 286/10 processor via shared memory. Startup, execution, and simulation run time are all controlled by the 286/10 processor via the command/executive menu.
4. The students can select the amount of data to be collected for display and can direct the data to the 310 system operator console, the printer, or the color graphics display. At present all data directed to the console or the printer is in "character" form only. Hardcopy graphics is not available at this time.

Figure 2 illustrates a simple block diagram of the system as viewed by the students. All input data, updated at a programmed frame rate, is loaded into a common buffer in shared memory. The 386/22 processor reads this data by sampling a "data$flag$in" flag in shared memory. If the flag is "true" the next computation cycle begins. The output results are stored in shared memory by the 386/22 processor and the "data$flag$in" flag is set "false". It is the responsibility of the Intel 286/10 processor to analyze this output data, format it for the proper display, and store the output data in a buffer located in the local memory of the 286/10 processor. The amount of data collected is controlled by the initialization menu and depends upon the selected frame time and the overall run time of the simulation. All code to control the color graphics display resides within the executive module on the 286/10 processor.

Figure 3 illustrates the basic flow control for the simulation model executing on the 386/22 SBC. While this flow control model is somewhat general purpose it is tailored to control elements of the experiment illustrated in Appendix I. If a different simulation experiment is to be run on the 386/22 processor it would be the responsibility of the students to alter the flow control of Figure 3 to meet the requirements of the simulation.

The experiment illustrated in Appendix I is a simple Pitch Attitude Hold System. The students are required to translate a block diagram of the system into a set of state variable equations. They then test the validity of the equations using MATLAB. After a correlation is obtained with the expected results the students program the equations using the high level language PLM86. They then prepare the equations, integrated with the necessary flow control illustrated in Figure 3, for downloading into the Intel 386/22 processor board. The downloading process is controlled by custom software residing on both the 286/10 processor and the 386/22 processor. The PLM86 program for the Pitch Attitude Hold System is also illustrated in Appendix I. Results obtained from this experiment were very encouraging, however the amount of effort put forth by the students exceeded that of a normal one or two week experiment.
Incorporation of the above experiment into a multi-processor system demands a user-friendly environment. For this reason the integration of the above experiment into the aero engineering curriculum required the use of a command/executive menu driven program consisting of over 30 software modules linked together and run as the primary task on the 286/10 processor. The main module program and an explanation of its primary functions is illustrated in Appendix 2. These modules controlled all data entry, console displays, printer output, and the formatting of results for output to the VDI720 graphics package. Results were displayed in color on the Princeton Graphics PGSR12-P color monitor. The displays could not be run in real-time because of the software overhead associated with the VDI720 graphics package.

To better understand the limitations associated the the VDI720 graphics package the following reviews the basic structure of VDI and summarizes its performance.

Intel provided the iVDI720 graphics package in ROM to handle graphics routines such as graphics initialization, line draw, text display, circles, etc. Unfortunately, the commands to the controller are difficult to understand and setup. This is mostly due to poor documentation on the part of Intel. Fortunately, Intel provided sample procedural binding to the iVDI720 and it is these procedural bindings that are used to access the iVDI commands.

The graphics controller is attached to the Multibus system as a logical device :VDI:. It is through this logical device name that ROM software can be accessed on the controller. The iVDI720 manual is vague on how to actually send the commands to the ROM. This is where the language binding procedures come in handy. The VDI language binding provides the procedures that send specific commands to the VDI device. The procedures send parameters in the format required by the VDI device.

To use the language binding, the graphics must be initialized by the procedure INITGRAPHICS(backgroundcolor). After initialization all other language binding procedures can be called into action. For the experiment in Appendix 1 lines were drawn using LINE(x1,y1,x2,y2). Text was displayed using TEXT(x,y,flag,count,pointer). Of course the appropriate setup needs to be done before calling these procedures. The above procedures are found in the file, VDLANG.EXT. It is well worth while to print out this file. While the file contains absolutely no comments, it does provide the user with a list of commands and required parameters.
SYNCHRONIZATION MODEL
FOR 386/22 <-> 286/10
COMMUNICATION

START

386/22 SYSTEM INITIALIZATION

WAIT FOR DATA#FLAG#IN = TRUE

CASE SELECTION 286/10 ON COMMAND

EXECUTE DYNAMIC TEST MODEL

EXECUTE FULL MATH MODEL

SET PAST DERIVATIVES = PRESENT DERIVATIVES

SET DATA#FLAG#IN = FALSE

FIGURE 3
Overall the performance of the VDI package is slow. Its performance is on par with graphics on an 8-bit IBM-PC class machine. There are no figures available which allow for a numerical value on the performance. But from the empirical results obtained over several experiments, both real-time and non-real-time, it is safe to say that the VDI package will not allow real-time output of a high speed process.

The only solution for this is to perform the graphics by direct access to the hardware. In general, this is not the purpose of the graphics module when utilized by aero engineering students. However, it is a good problem for a computer science major.
Over the past four years the Intel 286/10 based system has undergone considerable change. Most of these changes have involved adding additional hardware and software. In the beginning it was hoped that the system would provide an economical base for real-time flight simulation. Experience has demonstrated that the computing power of the 286/10 coupled to the 386/22 processor is sufficient to support a medium sized simulation that operates in real-time. However, the system will not support real-time instrument displays or any form of an out-the-window display. This is a disappointment considering that PC class machines support flight simulation models adequately as it pertains to the graphics. The models for these simulations may be weak but the displays do operate in pseudo real-time. It must be stated that the system is well suited to static displays like the one's generated for the experiment in Appendix I. It is unfortunate that hard copy of the displays is not available.

It is obvious that the major problem of the Intel system is the graphics coupled with system configuration limitations. The following suggested solutions would enhance the system a great deal.

1. Increase the 286/10 memory to 1 megabyte. The existing .5 megabytes is inadequate because of limited straping options. I/O can only be performed via the keyboard.

2. Change the operating system to RMX286 and run the 286/10 processor in protected mode. We have RMX286 but it can not be installed unless the memory is increased to at least .7 megabytes. Running in this mode will free up the strapping options and allow for real I/O. The disadvantage is the reconfiguring of all existing software to operate in protected mode.

3. Either rewrite all the graphics software or upgrade the graphics processor. Rewriting the graphics software is a labor intensive job best performed by computer science majors. Upgrading the graphics processor is a cost item coupled with the generation of new software. Either solution is not very attractive at this point as will be explained later.
4. Run the 386/22 processor board in protected mode. This will free up an additional 1 megabyte of memory. Running in protected mode the 386/22 processor can make use of its full 32-bit capability. This would increase its computing power by a factor of 3 or more when running math intensive programs that require a great deal of floating point arithmetic. To do this requires purchasing RMX386 and making a major configuration change to the entire system. This would be both costly and require a great deal of man-hours.

All in all the above solutions still do not create the type of system suitable to aero engineering majors who have limited computing experience at a system level. The multi-processing environment overshadows the main objective of introducing basic problems associated with flight simulation. This can be overcome if aero engineering majors were required to take a few more courses in computer science.

A better solution is to make the system so "canned" that the student need not know any aspect of the problems associated with a multi-processor system. To a large extent this has been the main objective of this project and has been successful for executing experiments such as the one outlined in Appendix I. However, it must noted that the students participating in these experiments were not aero engineering majors but electronic engineering majors. While the electronic majors did not fully understand the aero aspects of the experiments they were fully capable of generating the required support software to make the system appear user-friendly even to a novice computer user. For this reason the system is now fully capable of supporting static type experiments, minus the desired hard copy output.

While the above upgrades would provide for full up real-time flight simulation experiments this could not be achieved without considerable cost i.e., $5,000, and many man hours of software development. For this reason an alternate solution is proposed.
AN ALTERNATE SOLUTION AND CONCLUSIONS

The rapid changes in computer technology over the past four years have made systems like the Intel 310 obsolete. At best the 386/22 processor can be considered a 3 to 4 Mips machine when running in protected mode. Even the Intel 486 processor can only be considered an 8 Mips machine. With the advent of RISC technology coupled with new high speed graphics processors the modern "work station" is the way of the future. These work stations, varying in computer power from 10 to 30 Mips, provide a solution for introducing many aspects of flight simulation in the educational environment. Their cost continues to fall. It is now possible to purchase a 27 Mips machine for under $10,000. In addition there are several manufacturers, such as, Sun/Sparc, IBM/6000, DEC/3100, HP/Appollo,NeXT,DG/AViiON, etc. Not all of these work stations employ RISC technology but they all seem to be in the same class. Most come with 8 megabytes of main memory as standard and most support LAN technology to the fullest. Several of these workstations have DOS emulation as well as UNIX.

One of the major advantages of these machines, when viewed by an aero engineering major, is that the student does not have to have a complete understanding of the system-level hardware or software. They do require a working knowledge of UNIX but this is basic to most major curriculums. At the junior/senior level the students already have a working knowledge of UNIX.

With this idea in mind the system illustrated in Figure 4 is an example of what can be put together for under $20,000. This system would support any medium sized simulation, provide for all instrument display, give a good out-the-window display, and even support avionics displays. The system could operate in real-time, for both computations and graphics. The advantage to such a system is that the student can concentrate on the aero problem and put the system configuration problems in the background. In addition, the system is tailored to interface to a larger network providing a much bigger data base and the opportunity for many students to simultaneously work on one problem or one experiment.

Such a system is already being incorporated into the Electronic Engineering Department. It consists of 8 DEC/3100 work stations with DISC SERVERS. The Flight Simulation Laboratory at Cal Poly is now considering the purchase of two more work stations, either DEC/3100 or IBM/6000 class machines. These machines would be on their own network for high speed communications but would have
direct access to the department network for file transfer operations. Because the department network is connected campus wide students in aero engineering would have access to the two work stations reserved for flight simulation.

It is hoped that this system will create an environment where flight simulation experiments can become a permanent part of the aero engineering curriculum.

It should be noted that the Intel 310 system can still play an important part for senior project studies and master thesis work. This is particularly true for electronic engineering majors and computer science majors.
APPENDIX 1

PITCH ATTITUDE HOLD SYSTEM

EXPERIMENT
EL 520 PROJECT

PITCH ATTITUDE HOLD SYSTEM MODEL AND SIMULATION PROGRAM

PURPOSE

1. Gain experience in the formulation of continuous dynamic system models
defined in block diagram form.

2. Gain experience in the programming and checkout of a simulation model
running on two microprocessors that communicate with each other over
Multibus I. In addition, gain experience in the formulation of math-
intensive programs that utilize NDP coprocessors.

PROCEDURE

1. From the given block diagram formulate the equivalent system of equations
for the system simulation model.

2. Prepare a PLM-86 program that implements the simulation. The program
shall consist of 5 parts: 1. The main module; 2. A procedure to simulate
the input to the model; 3. The model; 4. A procedure to output the results
to a printer in numerical form and output the results to a CRT in
graphical form; 5. A procedure to handle communication between the 286/10
processor and the 386/22 processor.

3. NOTE: The simulation model should run on the 386 processor. All results are
to be passed to the 286 processor for scaling and output.

4. Refer to the block diagram. With the elevator servo locked at zero
position run the short period response of the airframe to the +/-10°
elevator doublet and verify your pitch rate response with the dynamic
check data.

5. Run the complete simulated pitch attitude hold system response to a +5° step
command $\Theta_c$ starting from zero initial conditions. Plot the following
variables versus time: $\Theta_c, \dot{\Theta}, \Theta$

6. Run the step response with $K_F \dot{\Theta} = 0$ simulating the loss of pitch rate
feedback.

NOTE: The following time constraints apply:
Step size (DT) = .001 sec
Sample time = .01 sec
Run time = 10 sec

7. Prepare one report that presents your methods, results and interpretation
of the system performance.
Pitch Attitude Hold System

Elevator position
Test input

Pitch attitude command

Elevator servos

\[
\frac{K_{es}}{\tau_{Es} S + 1} + SE
\]

Elevator input

Rate limit
\pm 90^\circ/\text{sec}

Short-period airframe

\[
\frac{K_\theta (S + \beta)}{S^2 + 2\xi \omega S + \omega^2}
\]

Pitch rate

\[
\frac{1}{S}
\]

Pitch attitude

Pitch gyro

\[
K_{FG} + K_{FG} S
\]

Position & rate feedback

K_{es} = 4.1

(\text{Kos is positive if trailing edge is down on elevator})

\[
\tau_{Es} = 0.03 \text{ sec}
\]

K_\theta = 0.0322

\beta = -0.53

\omega = 1.63 \text{ rad/s/sec}

\xi = 0.5 \text{ (damping coefficient)}

K_{FG} = 2.0 \text{ (position feedback)}

K_{FG} = 1.0 \text{ (rate feedback)}
T4B DYNAMIC CHECK

Elevator doublet (10 deg/2 sec)

Approach trim flight condition:

- \( H \) 500 ft
- \( KEAS \) 100 Knots
- Flaps 35°
- Gear down
PL/M-86, 86, V2.3 Compilation of Module Mathmodel
Object module placed in Math.obj
Compiler invoked by: :lang:plm86 Math.plm

#symbols
#define
#define large

1
MATHMODEL: DO;

2
/* COMMON BLOCK DECLARATION STATEMENT FOR VALUE PASSING */

3
DECLARE
STUFF STRUCTURE ((COMMAND, DATA$FLAG$IN, DUMMY, DUMMY2) BYTE,
(DATA$IN, DELTA$E, THETA$PRIME, THETA, TIME$STEP) REAL
AT (0E00000000));

4
DECLARE
(K$ES, Z$ES, K$THETA$PRIME, BETA, OMEGA, ZETA, K$=K$THETA$)

K$ = K$THETA$PRIME, F, DELTA$E$PRIME$PAST, Y$PRIME$PAST,
Y$PAST, THETA$PRIME$PAST, Y, DELTA$E$PRIME, Y$PRIME,
Z, Z$PRIME, Z$PAST, Z$PRIME$PAST, X) REAL
INITIAL (10.0, 0.05, 0.0522, 0.53, 1.63, 0.5, 1.0, 2.0);

/* PROCEDURE DEFINITIONS */

4
DYNAMIC$TEST: PROCEDURE;

/* TEST INPUT. ELEVATOR SERVO LOCKED AT ZERO DEGREES.
DATA$IN IS REALLY A SUPPLIED VALUE OF DELTA$E */

5
STUFF$DELTA$E = -1.0$STUFF$DATA$IN;

6
Z$PRIME = STUFF$DELTA$E$ - 2.0$ZETA$OMEGA$Z$ - OMEGA$OMEGA$X;

7
Z = Z + (STUFF$TIME$STEP/2.0)$*(3.0$Z$PRIME - Z$PRIME$PAST);

8
X = X + (STUFF$TIME$STEP/2.0)$*(3.0$X - Z$PAST);

9
STUFF$THETA$PRIME = K$THETA$PRIME$X (Z + BETA$);

10
END DYNAMIC$TEST;

11
FULL$MODEL: PROCEDURE;

/* DELTA$E IS IN DEGREES WHILE THETA$PRIME IS IN RADIANS/SECOND.
THETA IS TO BE IN DEGREES. THUS THE REQUIRED CONVERSION FACTOR
IN THE INTEGRATION AND FEEDBACK FORMULAE */

12
DELTA$E$PRIME = (K$ES$STUFF$DATA$IN$ - F) - STUFF$DELTA$E$)/TAU$ES;

13
IF (DELTA$E$PRIME > 90.0) THEN
ELSE IF (DELTA$E$PRIME < -90.0) THEN
DELTA$E$PRIME = -90.0;

STUFF.DELTA$E = STUFF.DELTA$E + (STUFF.TIME$STEP/2.0) * 
(3.0*DELTA$E$PRIME - DELTA$E$PRIME$PAST);

Y$PRIME = K$THETA$PRIME$PRIME + DELTA$E$PRIME + 
BETA$STUFF.DELTA$E - 2.0*BETA$OMEGA$Y - OMEGA$OMEGA$STUFF.THETA$PRIME;

Y = Y + (STUFF.TIME$STEP/2.0) * (3.0*Y$PRIME - Y$PRIME$PAST);

STUFF.THETA$PRIME = STUFF.THETA + (57.29577951)* (STUFF.TIME$STEP/2.0) * 
(3.0*STUFF.THETA$PRIME$PAST - THETA$PRIME$PAST);

F = K$THETA$PRIME$STUFF.THETA + 
(57.29577951)*K$THETA$PRIME$STUFF.THETA$PRIME;

END FULL$MODEL;

CALL INIT$REAL$MATH$UNIT; /* INITIALIZE MATH CHIP */

RESTART; /* INITIALIZE SYSTEM INITIAL CONDITIONS */

STUFF.DATA$FLAG$IN = 0;
STUFF.DELTA$E = 0.0;
STUFF.THETA$PRIME = 0.0;
STUFF.THETA = 0.0;
F = 0.0;
DELTA$E$PRIME$PAST = 0.0;
Y$PRIME$PAST = 0.0;
Y$PAST = 0.0;
THETA$PRIME$PAST = 0.0;
Y = 0.0;
DELTA$E$PRIME = 0.0;
Y$PRIME = 0.0;
Z = 0.0;
Z$PRIME = 0.0;
Z$PAST = 0.0;
X = 0.0;
K$THETA$PRIME = 2.0;

LOOP; /* MAIN LOOP */
DO WHILE (STUFF.DATA$FLAG$IN = 0);
END;

DO CASE STUFF.COMMAND;

CALL DYNAMIC$TEST;

CALL FULL$MODEL;

DO;
K$THETA$PRIME$ = 0.0;
CALL FULL$MODEL;
END;

GOTO RESTART;
END;

DELTA$PRIME$PAST$ = DELTA$PRIME$;
Y$PRIME$PAST$ = Y$PRIME$;
Y$PAST$ = Y;
THETA$PRIME$PAST$ = STUFF$THETA$PRIME$;
Z$PRIME$PAST$ = Z$PRIME$;
Z$PAST$ = Z;

STUFF$DATA$FLAG$IN$ = 0;  /* RESET DATA FLAG */
GOTO LOOP;  /* WAIT FOR NEXT DATA POINT */

END MATH$MODEL;
<table>
<thead>
<tr>
<th>ADDR</th>
<th>SIZE</th>
<th>NAME, ATTRIBUTES, AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000H</td>
<td>4</td>
<td>BETA ... REAL INITIAL</td>
</tr>
<tr>
<td>0002H</td>
<td>4</td>
<td>DELTAEPRIIME ... REAL INITIAL</td>
</tr>
<tr>
<td>0004H</td>
<td>4</td>
<td>DELTAEPRIIMEPAST ... REAL INITIAL</td>
</tr>
<tr>
<td>0020H</td>
<td>200</td>
<td>DYNAMICTEST ... PROCEDURE STACK=0002H</td>
</tr>
<tr>
<td>0158H</td>
<td>397</td>
<td>FULLMODEL ... PROCEDURE STACK=0004H</td>
</tr>
<tr>
<td>0600H</td>
<td>4</td>
<td>KEV ... REAL INITIAL</td>
</tr>
<tr>
<td>0618H</td>
<td>4</td>
<td>KITHETA ... REAL INITIAL</td>
</tr>
<tr>
<td>061CH</td>
<td>4</td>
<td>KITHEAPRIIME ... REAL INITIAL</td>
</tr>
<tr>
<td>0628H</td>
<td>4</td>
<td>KITHEAPRIIME ... REAL INITIAL</td>
</tr>
<tr>
<td>0800H</td>
<td></td>
<td>LOOP ... LABEL</td>
</tr>
<tr>
<td>0802H</td>
<td>289</td>
<td>MATHMODEL ... PROCEDURE STACK=0006H</td>
</tr>
<tr>
<td>0810H</td>
<td>4</td>
<td>CHEGA ... REAL INITIAL</td>
</tr>
<tr>
<td>0820H</td>
<td></td>
<td>RESTART ... LABEL</td>
</tr>
<tr>
<td>0820CH</td>
<td>24</td>
<td>STUFF ... STRUCTURE AT ABSOLUTE</td>
</tr>
<tr>
<td>0000H</td>
<td>1</td>
<td>COMMAND ... BYTE</td>
</tr>
<tr>
<td>0002H</td>
<td>1</td>
<td>DATAPLugin ... BYTE</td>
</tr>
<tr>
<td>0003H</td>
<td>1</td>
<td>DUMMY ... BYTE</td>
</tr>
<tr>
<td>0004H</td>
<td>1</td>
<td>DUMMY2 ... BYTE</td>
</tr>
<tr>
<td>0004H</td>
<td>4</td>
<td>DATAIN ... REAL</td>
</tr>
<tr>
<td>0004H</td>
<td>4</td>
<td>DELTAE ... REAL</td>
</tr>
<tr>
<td>0004H</td>
<td>4</td>
<td>THETAPRIIME ... REAL</td>
</tr>
<tr>
<td>0010H</td>
<td>4</td>
<td>THEA ... REAL</td>
</tr>
<tr>
<td>0014H</td>
<td>4</td>
<td>TIMESTEP ... REAL</td>
</tr>
<tr>
<td>0014H</td>
<td>4</td>
<td>TAUES ... REAL</td>
</tr>
<tr>
<td>0030H</td>
<td>4</td>
<td>THETAPRIIMEPAST ... REAL INITIAL</td>
</tr>
<tr>
<td>0050H</td>
<td>4</td>
<td>X ... REAL INITIAL</td>
</tr>
<tr>
<td>0054H</td>
<td>4</td>
<td>Y ... REAL INITIAL</td>
</tr>
<tr>
<td>005CH</td>
<td>4</td>
<td>YEAST ... REAL</td>
</tr>
<tr>
<td>0050H</td>
<td>4</td>
<td>YPRIME ... REAL</td>
</tr>
<tr>
<td>0050H</td>
<td>4</td>
<td>YPRIMEPAST ... REAL</td>
</tr>
<tr>
<td>0060H</td>
<td>4</td>
<td>Z ... REAL INITIAL</td>
</tr>
<tr>
<td>0060H</td>
<td>4</td>
<td>ZETA ... REAL</td>
</tr>
<tr>
<td>0060H</td>
<td>4</td>
<td>ZFAST ... REAL</td>
</tr>
<tr>
<td>0060H</td>
<td>4</td>
<td>ZPRIME ... REAL</td>
</tr>
<tr>
<td>0060H</td>
<td>4</td>
<td>ZPRIMEPAST ... REAL</td>
</tr>
</tbody>
</table>

**MODULE INFORMATION:**

- CODE AREA SIZE = 0378H  892D
- CONSTANT AREA SIZE = 001CH  2BD
- VARIABLE AREA SIZE = 0054H  642
- MAXIMUM STACK SIZE = 0066H  4D
- 127 LINES READ
- 0 PROGRAM WARNINGS
- 0 PROGRAM ERRORS

**DICTIONARY SUMMARY:**

- 64KB MEMORY AVAILABLE
SKB MEMORY USED (7%)
0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
LINK86  MATH.OBJ,
       /LIB/NDP87/DECON87.LIB,
       /LIB/NDP87/CER87.LIB,
       /LIB/NDP87/EH87.LIB,
       /LIB/NDP87/BOB87.LIB
TO MATH.LNK INITCODE

LINK86  MATH.LNK TO MATH.EXE INITCODE
       ADDRESSES: CLASSES (CODE(OA000H),
                   DATA(OA200H),
                   STACK(OA300H),
                   MEMORY(OA400H)),
                SEGMENTS (LIB_B7_INIT(OA5000H),
                           LIB_B7_PUB(OA6000H),
                           ??SEG(OA7000H))

COM86  MATH.EXE TO MATH.HEX
APPENDIX 2

286/10 COMMAND/EXECUTIVE MENU CONTROLLER
The pitch attitude simulator simulates the PD controller used to control the pitch attitude for an aircraft. This simulator consists of two parts, one running on the 80386 board and another on the 80286 board. The equations and calculations used to simulate the controller are performed on the 386 board, while the 236 program mainly collects signal simulation data that is output from the 386 and then displays this data. The communication between the two boards is through multibus and is flag driven.

The user first loads the simulation module onto the 386 board. He then executes the main program called main_const on the 386 system. The input signal to the controller is a simulated version of the command signal that would be obtained if it had been input into the computer by an analog to digital converter. Thus, the user must set a sample time which is used to construct this simulated command signal and is also used for simulating the controller. The user may also set the total simulation time, and the time between display updates. Once the signal and times have been selected the simulation begins.

The 386 uses timer 2 of the 8254 timer to control when the next sample of the input signal is to be sent to the 386. For example, if a 1ms sample time has been selected, then the 8254 is set so that it reaches terminal count every 1ms. When we detect that it has reached terminal count, timer_interrupt routine is called which sets the continue_simulation flag and sets theta in of the controller to the next value of the input signal. (This makes it look like we are actually sampling the input signal every 1ms and then passing it to the 386 simulator). The timer is then reloaded and will count off another 1ms period. The continue_simulation_flag tells the 386 that a new value for the input signal is ready to be processed.

Also, at this time we check to see if we should be recording the data that has been output by the 386 board. This data consists of various signals from the controller such as theta out, derivative of theta out, and delta error. This determination is made by looking at the value for time step size. This entire process is repeated until we reach the end of the simulation.

The 386 program will wait until it sees continue_simulation flag set to true. It then will process the new data that the 386 has given it and store the new values of the output signals that we are measuring. It is assumed that the 386 will reset continue_simulation flag after it has processed the data and will then remain idle until continue_simulation flag becomes high again.

The 386 program must be passed the sample time and the simulation type from the 386. The simulation type alters the controller model that will be used in the simulation. A 1 indicates a elevator dailbit will be used and that the elevator servo should be locked a 0 degrees. A 2 indicates that the full model is to be used, and a 3 indicates that the full model should be used but the velocity feedback coefficient should be set to 0. Finally, a 4 indicates that a new simulation run is about to be started and that the 386 program should reset itself.

All communication between the two programs is done through multibus starting at address F000000. The names of the variables passed between the two programs are simulation_type, sample_time, new_theta_in.
Once the simulation has been completed, the user can display the results in either a tabular format (to the screen or printer) or in a graphical format on the color monitor. After viewing the variables he desires, he may quit the simulator or run a new simulation

main_module: do;

#include(bsd:/user/controls/paul/basetype.p1m)
#include(bsd:/user/controls/paul/386io86.h)
#include(/usr/lib/exit.ex8)
#include(bsd:/user/controls/paul/add.io.h)
#include(bsd:rm:/lib/vdi/vdlaqi88.ex8)
#include(bsd:/user/controls/paul/colors)

/global variables used in this program

all variables that have been absoluted at 260000 or above
are used to pass information back and forth between this
program and the simulator running on the 386 board

variables

end_simulation

variable_name

simulation

input_signal

simulation_type

continue_simulation

simulation_time

sample_time

time_step_size

1st,con

literally '1001',

literally '10001',

literally '4',

literally '0',

literally '1',

literally '2',

literally '3',

literally '4',

string,

structure (dat (data_array_size: real),

input_signal: signal_array_size: real),

byte at (0EO000H),

byte at (0EO001H) initial(0),
new_variable
new_derivative

simulation_time
sample_time
absolute_sample_time
time_step_size

/* the following are used to program the counter 2 of the 8254 timer to count off the sample time */

8254_counter_2_addr
8254_counter_2_byte_high
8254_counter_2_byte_low
timer_rate
8254_control_word_addr

/* some general input/output procedures */

answer_to_question - this procedure returns either Y or N in response to a question that is contained in output_string

parameters -
output_string_ptr - pointer to the string containing the question to be asked

answer_to_question:

procedure(output_string_ptr) byte;

declare
output_string_ptr pointer,
output_string based output_string_ptr string,
response byte,
done byte,
input_char byte;

call clear_screen;

call write_in(con,output_string);

done = false;

do while (not done);
    do while (not keypressed);
    end;

    input_char = character$in;

    if ((input_char = 'n') or (input_char = 'N')) then
        done = true;
        response = 'N';
    end;
    else if ((input_char = 'y') or (input_char = 'Y')) then

get_real_parameter: procedure(output_string_ptr, real_value_ptr, default);
declare
  output_string_ptr pointer,
  real_value_ptr pointer,
  real_value based real_value_ptr real,
  default byte,
  realstring string;
call writeLn(con, output_string_ptr);
if (default = true) then
do;
call real_string(real_value, @realstring);
call writeLn(con, @(0, ['default = '?''])
call writeLn(con, @realstring);
call writeLn(con, @(0, [' ?'])
end;
call readln(true, @realstring);
if (realstring.len <> 0) then
call string_len(realstring, real_value_ptr)
end get_real_parameter;

get_integer_parameter: procedure(output_string_ptr, integer_value_ptr, default);
declare
  output_string_ptr pointer,
  integer_value_ptr pointer,
call write_in(con,output_string_ptr);

if (default = true) then
  do;
    call integer_string(integer_value,@integerstring);
    call write_in(con,@(0,'I default = ""'));
    call write_in(con,@(0,'1 : ""'));
  end;

  call readln(true,@integerstring);

  if (integerstring.len <> 0) then
    do;
      call string_to_real(@integerstring,@temp_real_value);
      integer_value = fix(temp_real_value);
    end;
  end %get_integer_parameter;

/***************************************************************************/
** routines to setup timer 2 of the 8254 to count off the sample time **
** before we send the next value of the input signal to the 386 */
/***************************************************************************/
load_8254_timer_2 - this procedure reloads the timer 2 of the 8254 with its count
parameters - none
***************************************************************************/
load_8254_timer_2:
  procedure;

  output(8254_counter_2_addr) = 8254_counter_2_byte_high;
  output(8254_counter_2_addr) = 8254_counter_2_byte_low;

end load_8254_timer_2;

/***************************************************************************/
** setup_sample_time - this procedure calculates the values that need to **
** be loaded into the 8254 in order for it to count off the sample time. **
** parameters - **
**  sample_time - the amount of time that the 8254 is supposed to count **
** off before we send the next value of the input to the 386 **
***************************************************************************/
set_up_sample_timer:
  procedure(sample_time);

  declare
    i8254_mode_control_word literally '0001',
    timer_count: integer,
    sample_time: real;

  timer_count = fix(sample_time / timer_rate);

  i8254_counter_2_byte_high = high(unsigned(timer_count));
  i8254_counter_2_byte_low = low(unsigned(timer_count));
end set_up_sample_timer;

/*****************************************************************************/
check_for_interrupt - this procedure checks the status bit of the
8254 to see if it has reached terminal count yet
It returns true if it has
/*****************************************************************************/

check_for_interrupt:
procedure byte;

declare
i8254_read_back_control_word literally '000H',
mask_byte literally '00H',
status_counter_2 byte;

output(i8254_control_word_addr) = i8254_read_back_control_word;
status_counter_2 = input(i8254_counter_2_addr) and mask_byte;

if (status_counter_2 = mask_byte) then return(true);
else return(false);
end check_for_interrupt;

/*****************************************************************************/
routine which handles communication between 386 and 286
*****************************************************************************/

/*****************************************************************************/
timer_interrupt_routine - this procedure takes the next value of the
input signal and gives it to the 386 simulator
If we are supposed to store the output signals for this particular sample then it will place
the values for delta_e, deriv_theta_out, and theta_out in the simulation data array.
It also sets continue simulation flag for the 386
and reloads the timer.
*****************************************************************************/

timer_interrupt_routine:
procedure(time_index, data_storage_index, store_variables);

declare
time_index integer,
data_storage_index integer,
store_variables byte,
result_string string;

call load_8254_timer_2;

if (store_variables = true) then
do:
simulation(delta_e).dat(data_storage_index) = new_delta_e;
simulation(theta_out).dat(data_storage_index) = new_theta_out;
simulation(deriv_theta_out).dat(data_storage_index) = new_deriv_theta_out;
simulation(theta_ref).dat(data_storage_index) = input_signal(time_index, data_storage_index) = new_theta_ref;
end do;
new_theta_ref = input_signal(time_index);

continue_simulation_flag = true;
end timer_interruptRoutine;

/*
 * routines which build the input signal for the controller
 *
 * simulate_doublet_input_to_model - builds a +/- 10 degree 2 second elevator
doublet input to the airframe
 */

simulate_doublet_input_to_model: procedure (simulation_time, sample_time);
begin
  declare
    index: integer,
    simulation_time: real,
    sample_time: real,
    (time_1, time_2, time_3): real;

  time_1 = (0.5/sample_time);
  time_2 = (1.5/sample_time);
  time_3 = (2.0/sample_time);

  do index = 0 to fix(simulation_time/sample_time);
    input_signal(index) = 0.0;
  end;

  do index = 1 to fix(time_1);
    input_signal(index) = (-10.001 * float(index)/time_1);
  end;

  do index = fix(time_1) to fix(time_2);
    input_signal(index) = (20.000 * float(index) - time_1)/
                         (time_2 - time_1) + -10.001;
  end;

  do index = fix(time_2) to fix(time_3);
    input_signal(index) = (-10.000 * float(index) - time_2)/
                         (time_3 - time_2) + 10.001;
  end;

end simulate_doublet_input_to_model;

/*
 * simulate_step_input_to_model - builds a 5 degree step input to the controller
 */

simulate_step_input_to_model: procedure (simulation_time, sample_time, time_step_size);
begin
  declare
    index: integer,
    simulation_time: real,
    sample_time: real,
    time_step_size: real;

  time_1 = (0.5/sample_time);
  time_2 = (1.5/sample_time);
  time_3 = (2.0/sample_time);

  do index = 0 to fix(simulation_time/sample_time);
    input_signal(index) = 0.0;
  end;

  do index = 1 to fix(time_1);
    input_signal(index) = (-10.001 * float(index)/time_1);
  end;

  do index = fix(time_1) to fix(time_2);
    input_signal(index) = (20.000 * float(index) - time_1)/
                         (time_2 - time_1) + -10.001;
  end;

  do index = fix(time_2) to fix(time_3);
    input_signal(index) = (-10.000 * float(index) - time_2)/
                         (time_3 - time_2) + 10.001;
  end;

end simulate_step_input_to_model;
declare
  index: integer,
  simulation_time: real,
  sample_time: real,
  time_step_size: real;

do index = 0 to (2 * fix(time_step_size/sample_time));
  input_signal(index) = 0.0;
end;

do index = (2 * fix(time_step_size/sample_time)) to fix(simulation_time/sample_time);
  input_signal(index) = 5.00;
end;
e nd simulate_step_input_to_model;

*******************************************************************************
get_time_constraints -- this procedure asks the user for the simulation
time, sample time, and time step size for the
current simulation run.

parameters-
  ptr_sample_time - pointer to the value for the sample time
  ptr_time_step_size - pointer to the value for the time step size
  ptr_simulation_time - pointer to the value for the simulation time

*******************************************************************************/
get_time_constraints:
  procedure(ptr_sample_time, ptr_time_step_size, ptr_simulation_time);

  declare
    ptr_sample_time: pointer,
    ptr_time_step_size: pointer,
    ptr_simulation_time: pointer;

call clear$screen;

simulation_time = 10.00;
sample_time = 0.001;
time_step_size = 0.010;

call get_real_parameter(@0,'input sample time ~'), ptr_sample_time, true);

call get_real_parameter(@0,'input time step size ~'), ptr_time_step_size, true);

call get_real_parameter(@0,'input simulation time ~'), ptr_simulation_time, true);

e nd get_time_constraints;

*******************************************************************************
get_simulation_type -- this procedure asks the user for the type of simulation
he wishes to run. It then sets the correct simulation
type and builds the appropriate input signal to
be used in the simulation.
sample_time - time between successive samples of the input signal

'time_step_size' - time between display updates of the simulation data

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

get_simulation_type:

procedure(simulation_time, sample_time,

time_step_size);

declare

simulation_time
sample_time
'time_step_size'
temp_simulation_type

real;
real;
real;
integer;

call clear$screen;
call write_in(cono, @O, 'Simulations that you may run:

1 - doublet dynamic check
2 - 5 degree step
3 - 5 degree step [no feedback];
call get_integer_parameter(@O, 'simulation type desired : ')
@temp_simulation_type, false);
simulation_type = low(unsig(temp_simulation_type));

do case (simulation_type - 1);
call simulate_doublet_input_to_model(simulation_time, sample_time);
call simulate_step_input_to_model(simulation_time, sample_time, time_step_size);
call simulate_step_input_to_model(simulation_time, sample_time, time_step_size);
end;

end get_simulation_type;

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

setup_simulation - this procedure gets the simulation time, sample time
and time_step_size. It then programs the timer for
the correct interrupt time, and determines the type
of simulation the user wishes to run

parameters-

ptr_simulation_time - pointer to the value for the simulation time
ptr_time_step_size - pointer to the value for the time step size
ptr_sample_time - pointer to the value for the sample time

setup_simulation:

procedure(ptr_sample_time, ptr_time_step_size,

ptr_simulation_time);

declare

ptr_sample_time
ptr_time_step_size
ptr_simulation_time
output_string
sample_time
time_step_size
simulation_time

pointer,
pointer,
pointer,
string,
based ptr_sample_time real,
based ptr_time_step_size real,
based ptr_simulation_time real;

call get_time_constraints(@sample_time, @time_step_size, @simulation_time);
call get_simulation_type(simulation_time, sample_time, time_step_size);
call set_up_sample_timer(sample_time);
end setup_simulation;

/*
 run_simulation - this procedure runs the entire simulation. It first
 sets up the simulation and initializes the appropriate
 variables. It then checks the timer to see if a next
 value of the input signal should be sent to the 386.
 It also determines whether the current data from the 386
 should be stored in the simulation data. It repeats
 this process until the simulation has been completed
 */

run_simulation:

parameters:
 ptr_sample_time - pointer to the value for the sample time
 ptr_time_step_size - pointer to the value for the time step size
 ptr_simulation_time - pointer to the value for the simulation time

procedure(ptr_sample_time, ptr_time_step_size, ptr_simulation_time);

declare
 ptr_sample_time, integer,
 data_storage_index, integer,
 ptr_time_step_size, pointer,
 ptr_simulation_time, pointer,
 sample_time, based ptr_sample_time real,
 time_step_size, based ptr_time_step_size real,
 simulation_time, based ptr_simulation_time real,

time_index, integer,
delta_time, real,
simulation_done, byte,
store_variables, byte;

/* initialize the timer, input signal, and simulation variables */
call setup_simulation(>@sample_time, @time_step_size, @simulation_time);

continue_simulation_flag = false;
simulation_done = false;

/* data storage index is a pointer into the simulation data array where
 we will be storing the next set of simulation data */
data_storage_index = 0;

/* time index is a pointer into the input signal array that tells us what
 the next value of the input signal given to the 386 will be */
time_index = 0;
delta_time = time_step_size;

do while (not simulation_done);
    /* if we have gone over the total simulation time then we are done */
    if (time_index > fix(simulation_time/sample_time)) then
        simulation_done = true;

    /* see if the timer has reached terminal count yet */
    if (check_for_interrupt = true) then
        do;
/* if so next check to see if we need to store the current simulation data */
if (delta_time >= (time_step_size - sample_time)) then
    do:
        delta_time = 0.0;
        store_variables = true;
        data_storage_index = data_storage_index + 1;
    end;
else delta_time = delta_time + sample_time;
/* send out the next data point in our input signal to the */
/* 386 simulation program */
call timer_interruptroutine(time_index, data_storage_index,
    store_variables);
/* increment to the next point in our input signal that will */
/* be given to the 386 when terminal count is reached again */
time_index = time_index + 1;
end;
end run_simulation;

get_variables_to_be_printed - this procedure asks the user which variables
from the simulation he wishes to display in tabular format.

parameters:
variable_info_ptr - pointer to a structure. One of the fields of
this structure (graph_variable) is set to true
if the variable should be displayed

get_variables_to_be_printed: procedure(variable_info_ptr);
    declare:
        variable_info_ptr     pointer,
        variable_info         based variable_info_ptr(1) graph,
        output_string         string,
        selection             integer,
        done_choosing         byte,
        i                     integer;
    do i = 0 to (variables - 1);
        variable_info(i).graph_variable = false;
    end;

call clear$screen;

call write$ln(con,@(0,'Choose the variables that you wish to be printed:

^' ));
    do i = 0 to (variables - 1);
        call integer_string(i+1,output_string);
        call append_string(output_string,@(0,' ',' '));
        call append_string(output_string,@(0,' ',' '));
        call append_string(output_string,@(0,' ',' ')[i]);
        call append_string(output_string,@(0,' ',' '));
        call append_string(output_string,@(2,'n'));
    end;
call write$ln(con,@output_string);
end;
call integer_string((variables+1),@output_string);
call append_string(@output_string,@(0, 'print the variables you have chosen\n'));
call writeLn(con,@output_string);
call writeLn(con,@(2,'\n'));
done_choosing = false;
do while (done_choosing = false);
call get_integer_parameter(@(0,'number of the variable to be printed : 
'),@selection,false);
if (selection = (variables + 1)) then done_choosing = true;
else variable_info(selection - 1).graph_variable = true;
end;
end get_variables_to_be_printed;

*******************************************************************************
print_simulation_data - this procedure prints the simulation data in a table format. This data may be directed to either the printer or the screen

parameters-
simulation_time - the total time for the simulation
sample_time - time between successive samples of the input signal
time_step_size - time between display updates of the simulation data
*******************************************************************************
print_simulation_data: procedure(sample_time,time_step_size, simulation_time);

declare
sample_time real,
time_step_size real,
simulation_time real,
out word,
time real,
result_string string,
data_string string,
(i,j,k) integer,
number_of_data_points integer,
variable_info(variables) graph;

if (answer_to_question(@(0,'would you like the output to go to the printer ? 
')) = 'Y') then out = 1st;
else out = con;

/* ask the user which variables he wants to print out */
call get_variables_to_be_printed(@variable_info);
call clearscreen;

number_of_data_points = fix(simulation_time/time_step_size);
do i = 0 to (variables - 1);
  if (variable_info(i).graph_variable = true) then do;
    do case i;
call write_in(out,@(0, 'simulation data for delta erro: \n'));
call write_in(out,@(0, 'simulation data for theta out: \n'));
call write_in(out,@(0, 'simulation data for derivative of the

ta out: \n'));
end;
call write_in(out,@(0, 'time value: \n'));
call write_in(out,@(0, 'time = 0.0000;'))
do j = 1 to number_of_data_points;
call real_string(time, &result_string);
call append_string(&result_string, @(0, '\n'));
call real_string(simulation(i), &data_string);
call append_string(&result_string, @(data_string));
call append_string(&result_string, @(0, '\n'));
call write_in(out, &result_string);
time = time + time_step_size;
end;
end;
end print_simulation_data;

/************************** setup_graphics_display - this procedure sets up the graphics display by clearing the screen, drawing the graph axes, and labeling the time axes. It also returns the size of the display screen and sets up the text sizes
parameters-
maximum_x_ptr - pointer to the maximum x value of the display
maximum_y_ptr - pointer to the maximum y value of the display
start_time - the starting time of the data to be displayed
end_time - the ending time of the data to be displayed
***************************/
setup_graphics_display: procedure(maximum_x_ptr, maximum_y_ptr, start_time, end_time);
declare
maximum_x_ptr
maximum_y_ptr
maximum_x
maximum_y
start_time
end_time
text_string

maximum_x = 640;
maximum_y = 476;

/* draw the x and y axes */
call line(75,0,75,480);
call line(0,28,640,28);

/* set the type of text we will be displaying on our graph */
call set$text$font$index(1);
call set$character$height(6);
call set$character$path(0);
call set$character$orientation(0,0,2,2);
call set$character$spacing(1.0);
call real_string(start_time,@text_string);
call text(80,20,0,text_string.len,@text_string.text(0));
call real_string(end_time,@text_string);
call text(550,20,0,text_string.len,@text_string.text(0));
end setup_graphics_display;

get_variables_to_be_graphed - this procedure asks the user which variables
from the simulation he wishes to display as
graphs on the color monitor

parameters-
variable_info_ptr - pointer to a structure. One of the fields of
this structure (graph_variable) is set to true
if the variable should be displayed

 purposely comments out code

get_variables_to_be_graphed: procedure(variable_info_ptr);
declare
    variable_info_ptr pointer,
    variable_info based variable_info_ptr(1) graph,
    output_string string,
    selection integer,
    done_choosing byte,
    i integer;

/* initialize variable_info so that no variables will be graphed
unless the user asks for them to be graphed right now */
do i = 0 to (variables - 1);
    variable_info(i).graph_variable = false;
end;
call clear$screen;

/* display the variables that the user may graph */
call write$ln(con,@0,'Choose the variables that you wish to be graphed:


');
do i = 0 to (variables - 1);
    call integer_string(i+1,output_string);
call append_string(output_string,@0,' ','
');
call append_string(output_string,variable_name(i));
call append_string(output_string,@2,'
');
call write$ln(con,output_string);
end;
call write$ln(con,@2,'
');
/* ask him which ones he wants to graph */
call integer_string((variables+1),output_string);
call append_string(output_string,@0,'
call writeLn(con,@(2,1n'));

done_choosing = false;
do while (done_choosing = false);
    call get_integer_parameter(0,number_of_the_variable_to_be_displayed);
    if (selection = (variables + 1)) then done_choosing = true;
    else variable_info(selection - 1).graph_variable = true;
end;
end get_variables_to_be_graphed;

get_time_duration_to_graph — this procedure asks which portion of the
simulation data the user wishes to display.
It then calculates the starting and ending
data points to be displayed based on this info

parameters -

start_data_pt_ptr — pointer to the value of the starting data pt to be
displayed

end_data_pt_ptr — pointer to the value of the ending data pt to be
displayed

start_time_ptr — pointer to the starting time of the data to be displayed

end_time_ptr — pointer to the ending time of the data to be displayed

get_time_duration_to_graph: procedure(start_data_pt_ptr,end_data_pt_ptr,
                              start_time_ptr,end_time_ptr,
                              simulation_time,number_of_data_pts);

declare
    start_data_pt_ptr pointer,
    end_data_pt_ptr pointer,
    start_time_ptr pointer,
    end_time_ptr pointer,
    simulation_time real,
    number_of_data_pts integer,
    start_dataPt based start_data_pt_ptr integer,
    end_dataPt based end_data_pt_ptr integer,
    start_time based start_time_ptr real,
    end_time based end_time_ptr real,
    realstring string;

call clear$screen;
call writeLn(con,@(2,'the total simulation time was : '''));
call real_string(simulation_time,#realstring);
call writeLn(con,#realstring);
call writeLn(con,@(2,' seconds\n'));
call writeLn(con,@(2,1n'));
call writeLn(con,@(2,'time period to be displayed :\n'));
call writeLn(con,@(2,1n'));
call get_real_parameter(0,'starting time ~'),@start_time,true);
call get_real_parameter(0,'ending time ~'),@end_time,true);

/* given the starting and ending times, determine the first and last
data point in our data that we will be displaying */
end_data_pt = fix(float(number_of_data_pts - 1) * (end_time/simulation_time)) + 1;
end get_time_duration_to_graph;

 масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло масло мас
if (answer_to_question('@(0,'do you wish to resize your graphs ?~')') = 'Y') then
  do i = 0 to (variables - 1);
    call clear$screen;
    textstring.len = 0;
    call append_string(@textstring,@(0,'do you wish to resize ~'));
    call append_string(@textstring,@variable_name(i));
    call append_string(@textstring,@(0,' ?~'));
    if (variable_info(i).graph_variable = true) then
      if (answer_to_question(@textstring) = 'Y') then
        do;
          variable_info(i).max_value = simulation(i).dat(start_data_pt);
          variable_info(i).min_value = simulation(i).dat(start_data_pt);
          do j = (start_data_pt + 1) to end_data_pt;
            if (simulation(i).dat(j) > variable_info(i).max_value) then
              variable_info(i).max_value = simulation(i).dat(j);
            else if (simulation(i).dat(j) < variable_info(i).min_value) then
              variable_info(i).min_value = simulation(i).dat(j);
          end;
        end;
      end:
    if (answer_to_question('@(0,'do you wish to resize the graph by hand ?~')') = 'Y') then
      do;
        call get_real_parameter('@(0,' maximum graph value');
        call get_real_parameter('@(0,' minimum graph value);
        if (variable_info(i).min_value = variable_info(i).max_value) then
          do;
            if (variable_info(i).min_value = 0.0) then offset = 1.0;
            else offset = abs(variable_info(i).min_value * 0.5);
            variable_info(i).min_value = variable_info(i).min_value - offset;
            variable_info(i).max_value = variable_info(i).max_value + offset;
          end;
        end;
      end:
  end get_graph_sizes;

فذ Residents of this procedure labels the axes of the graph with the
parameters -
  variable_info_ptr - pointer to a structure that contains info about the
      variable such as its max and min values
  values_x - the x coordinate where the max and min values are
      to be written on the screen
  max_val_y_ptr - pointer to the y coordinate on the screen where the
      maximum value for the variable should be written
  minval_y_ptr - pointer to the y coordinate on the screen where the
      minimum value for the variable should be written
  name_x_ptr - pointer to the x coordinate where the name of the
      variable should be written
  name_y_ptr - pointer to the y coordinate where the name of the
      variable should be written
  color_ptr - pointer to the color that all this stuff should be
      written in
  var_num - the number of the variable to be displayed

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

label_graph:
  procedure(variable_info_ptr,values_x,
      maxval_y_ptr,minval_y_ptr,
      name_x_ptr, name_y,color_ptr, var_num);
  declare
    variable_info_ptr   pointer,
    maxval_y_ptr       pointer,
    minval_y_ptr       pointer,
    name_x_ptr         pointer,
    color_ptr          pointer,
    variable_info      based variable_info_ptr(1) graph,
    maxval_y           based maxval_y_ptr word,
    minval_y           based minval_y_ptr word,
    name_x             based name_x_ptr word,
    name_y             word,
    values_x           word,
    color              based color_ptr word,
    var_num            integer,
    text_string        string;
  call set$line$color(color);
  call set$text$color(color);
  color = color + 1;
  if (color > 15) then color = 1;
  call real_string(variable_info(var_num).max_value, text_string);
  call text(values_x,maxval_y,0,text_string.len,text_string текст(0));
  call real_string(variable_info(var_num).min_value, text_string);
  call text(values_x,minval_y,0,text_string.len,text_string текст(0));
  / * change the y positions for the max value and min value so when we
      put on the max and min values for the next graph they wont be written
      over the max and min values for this graph */
  maxval_y = maxval_y - 12;
  minval_y = minval_y + 12;
  call text(name_x,name_y,0,variable_name(var_num).len,
      @variable_name(var_num).текст(0));
  name_x = name_x + (8 * variable_name(var_num).len);
end label_graph;
parameters:

simulation_time - the total time for the simulation
sample_time - time between successive samples of the input signal
time_step_size - time between display updates of the simulation data

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

graph_simulation_data: procedure(sample_time, time_step_size, simulation_time);

declare
sample_time
real,
time_step_size
real,
simulation_time
real,
start_time
real,
end_time
real,
start_data_pt
integer,
end_data_pt
integer,
number_of_data_pts
integer,
i
integer,
j
integer,
done_graphing
byte,
first_point
byte,
maxval_y
word,
minval_y
word,
name_x
word,
color
word,
maximum_x
integer,
maximum_y
integer,
x_position
integer,
y_position
integer,
old_x_position
integer,
old_y_position
integer,
graph_step_size
integer,
offset_x
integer,
offset_y
integer,
space_between_data_pts
integer,
max_value
real,
min_value
real,
graph_size
real,

variable_info(variables) graph;

/* start time and end time define to portion of the simulation data that we will be displaying in our graph */
start_time = 0.0;
end_time = simulation_time;

/* set boundaries on the portion of the screen in which we will be graphing and set the spacing between consecutive data points */
offset_x = 80;
offset_y = 30;
done_graphing = false;
space_between_data_pts = 1;

number_of_data_pts = fix(simulation_time/time_step_size);
maxval_y = 470;
name_x = 80;
call get_variables_to_be_graphed(@variable_info);
call get_time_duration_to_graph(@start_data_pt, @end_data_pt,
   @start_time, @end_time, @simulation_time, @number_of_data_pts);
call get_graph_sizes(@variable_info, @start_data_pt, @end_data_pt);
call setup_graphics_display(@maximum_x, @maximum_y, @start_time, @end_time);
/* determine the spacing on the screen between consecutive data points.
   if it is less than 2 pixels then condense the number of data points
   we will display */
graph_step_size = (@maximum_x - @offset_x)/(@end_data_pt - @start_data_pt);
if (graph_step_size <= 1) then
   do;
      graph_step_size = 2;
      space_between_data_pts = ((@end_data_pt - @start_data_pt)/
         (@maximum_x - @offset_x)/2)) + 1;
   end;
else space_between_data_pts = 1;
color = 2;
/* graph the variables */
do @i = 0 to (@variables - 1);
   if (@variable_info(i). graph_variable = true) then
      do;
         call label_graph(@variable_info(i).i, @maxval_y
            , @minval_y, @name_x, @unsigned(@offset_y - 20), @color);
         first_point = true;
         x_position = @offset_x;
         j = @start_data_pt + 1;
         do while (j <= @end_data_pt);
            old_y_position = @y_position;
            y_position = @fix(@variable_info(i).max_value)/
               @variable_info(i).graph_size * @float(@maximum_y - @offset_y));
            if (first_point = true) then
               do;
                  first_point = false;
                  call graph_line(x_position, @y_position,
                     x_position, @y_position);
               end;
            else call graph_line(x_position, @y_position,
               old_x_position, old_y_position);
            old_x_position = x_position;
            x_position = x_position + graph_step_size;
            j = j + space_between_data_pts;
         end;
      end;
   end;
if (answer_to_question(@i(0, 'do you wish to make some more graphs? '))
   = 'N') then done_graphing = true;
end;
end graph_simulation_data;
display_simulation_results - this procedure will display the simulation
data in both tabular and graphical format
if the user desires either one

parameters:
simulation_time - the total time for the simulation
sample_time - time between successive samples of the input signal
time_step_size - time between display updates of the simulation data

display_simulation_results:  procedure(sample_time, time_step_size, simulation_time);

declare
  sample_time  real,
  time_step_size  real,
  simulation_time  real;

if (answer_to_question(0,'do you wish to display the simulation data in ta-
bular format ?') = 'Y') then
call print_simulation_data(sample_time, time_step_size, simulation_time);
if (answer_to_question(0,'do you wish to display the simulation data in gr-
aphical format ?') = 'Y') then
call graph_simulation_data(sample_time, time_step_size, simulation_time);
end display_simulation_results;

/* initialization of program routines */

setup_output_devices - this procedure establishes con as a pointer to output
info to the screen, and lst as a pointer to output
info to the printer

parameters - none

setup_output_devices:  procedure;

declare
  device_open  byte;

device_open = open#connection;
  lst = lst#out;
  con = con#out;
end setup_output_devices;

init_variables - initialization of the signal names that we will be
recording simulation data for

parameters - none

init_variables:  procedure;

declare
  1 integer;

d0 i = 0 to (variables - 1);
  variables = i + 1;
call append_string(@variable_name(0),@0,'theta in~');
call append_string(@variable_name(1),@0,'delta error~'));
call append_string(@variable_name(2),@0,'theta out~'));
call append_string(@variable_name(3),@0,'deriv of theta out~'));
end init_variables;

/**
 * initialize the 80287 math chip */
call init$real$math$unit;

/* open a connection to the screen and to the printer */
call setup_output_devices;

/* initialize the names of the signals that we will be tracking during
the simulation */
call init_variables;

/* tell the program running on the 386 to reset itself */
simulation_type = end_simulation;
continue_simulation_flag = true;

/* run multiple simulations until the user is done */
done = false;
do while (not done);
call run_simulation(@sample_time,@time_step_size,@simulation_time);
call display_simulation_results(sample_time,time_step_size,simulation_time);
if (answer_to_question(@0,'would you like to run another simulation ?~')
)= 'N') then done = true;

/* tell the simulator on the 386 to reset itself */
simulation_type = end_simulation;
continue_simulation_flag = true;
end;

/* exit the program */
call dq$exit(0);

end main_module;