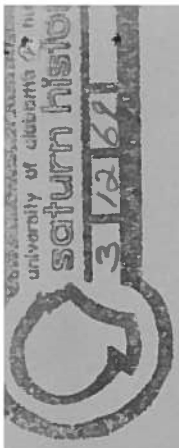
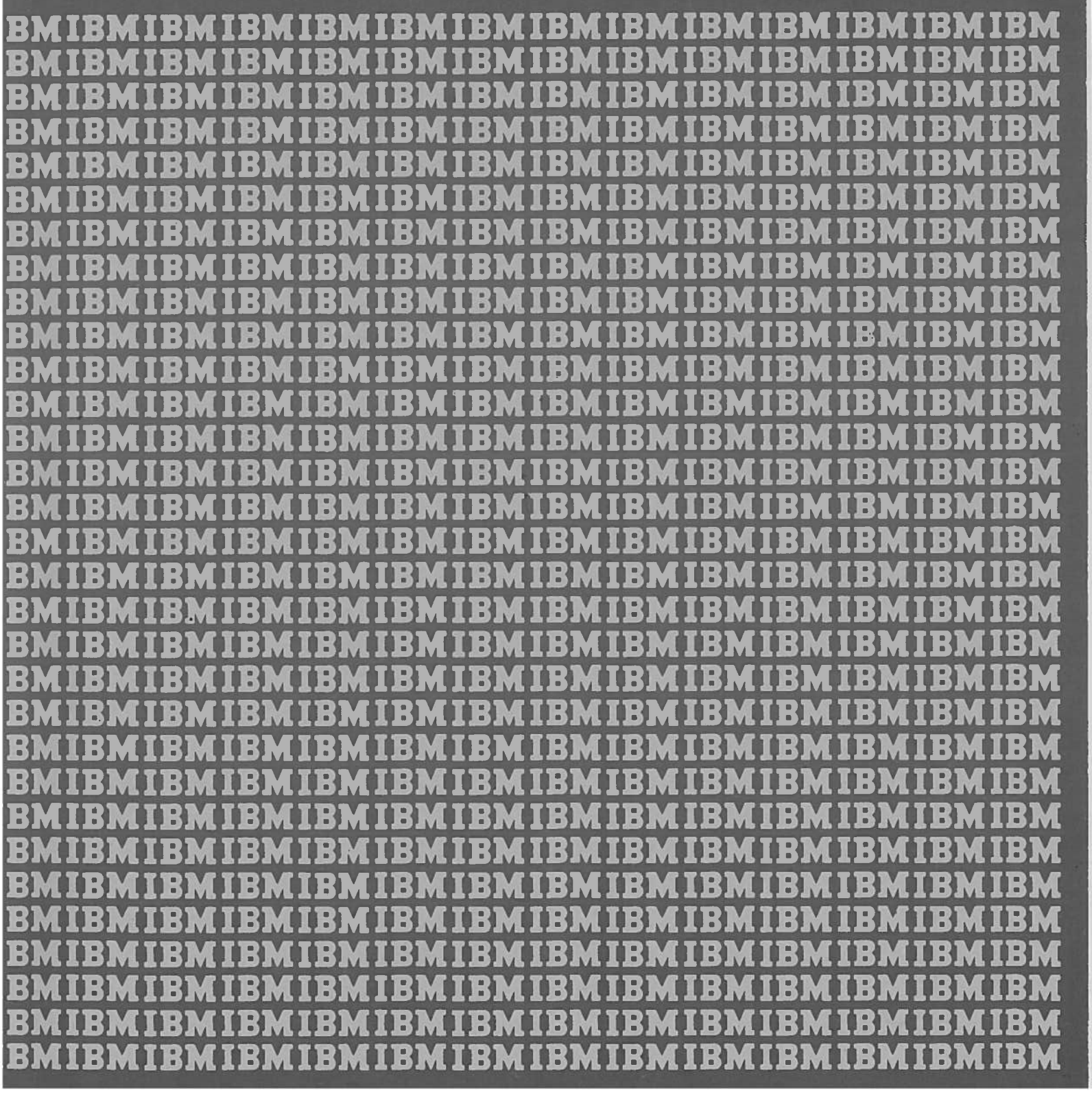


Flight Software Development Laboratory

X1.16

IBM No. 68-U60-0022



IBM No. 68-U60-0022

FLIGHT SOFTWARE DEVELOPMENT LABORATORY

by

T. H. Witzel
and
J. S. Hughes

Presented at the 3rd Conference of
the American Institute of Aeronautics
and Astronautics on Flight, Test, Simulation,
and Support, March 10-12, 1969

International Business Machines Corporation
Federal Systems Division
Space Systems Center
Huntsville, Alabama

FLIGHT SOFTWARE DEVELOPMENT LABORATORY

T. H. Witzel and J. S. Hughes
Space Systems Center
International Business Machines Corporation
Huntsville, Alabama

Abstract

A man-in-the-loop computer facility has been created using a digital computer, display terminal, and space vehicle flight computer to enable programmers to check out flight programs in a simulated space flight environment. The simulation requires a real time multiprogrammed environment, which is supplied by a control system capable of scheduling programs on 32 levels of priority interrupt as well as answering demands for service at the display terminal. A special interface device permits visibility and control of the flight program as it executes in the flight computer. On-line inputs from the programmer at the display terminal and outputs from data collection and reduction routines to the display screen are executed in real time. The Flight Software Development Laboratory* has proved to be very useful in reducing program preparation time and increasing flight program confidence.

Introduction

The Flight Software Development Laboratory (Figure 1) has been created to aid programmer/engineers in the development of programs that will operate in a spaceborne computer aboard the Apollo/Saturn IB and V Launch Vehicles. The Flight Computer operates as an integral part of various vehicle subsystems in the Instrument Unit. The subsystems provide onboard navigation, guidance, control, sequencing, data compression, and ground communications. These functions are illustrated in Figure 2. Continued emphasis is placed on flight software reliability because it is an essential element in overall vehicle performance. No opportunity exists to test or exercise the flight program in its actual flight environment prior to a mission. Therefore, to ensure the integrity of the flight program, simulators are used to accomplish flight testing. The purpose of this paper is to present the organization of one such simulator that has been created with the sole purpose of the development and checkout of Saturn flight software. The emphasis throughout the design and implementation of the Laboratory has been that it must be user-oriented for program checkout. Before the existence of the Laboratory, available facilities for checking out flight programs were oriented to hardware checkout. Although such facilities can be, and have been, rigged for program checkout, they have not provided the type of assistance required to produce the quality of software demanded by spaceborne computers. The Laboratory is believed to be unique in the capabilities it provides to the programmer/engineer in controlling and affecting the operation of the flight computer in a real time environment.

Flight software development begins with a set of explicit engineering requirements: equation and logic definition, range of variables, and expected performance data. After an intensive analysis of the requirements, the flight software is designed and organized to meet these engineering requirements with minimal flight computer memory and reasonable flexibility. After the flight program has been flowed, scaled (fixed point computer), coded,

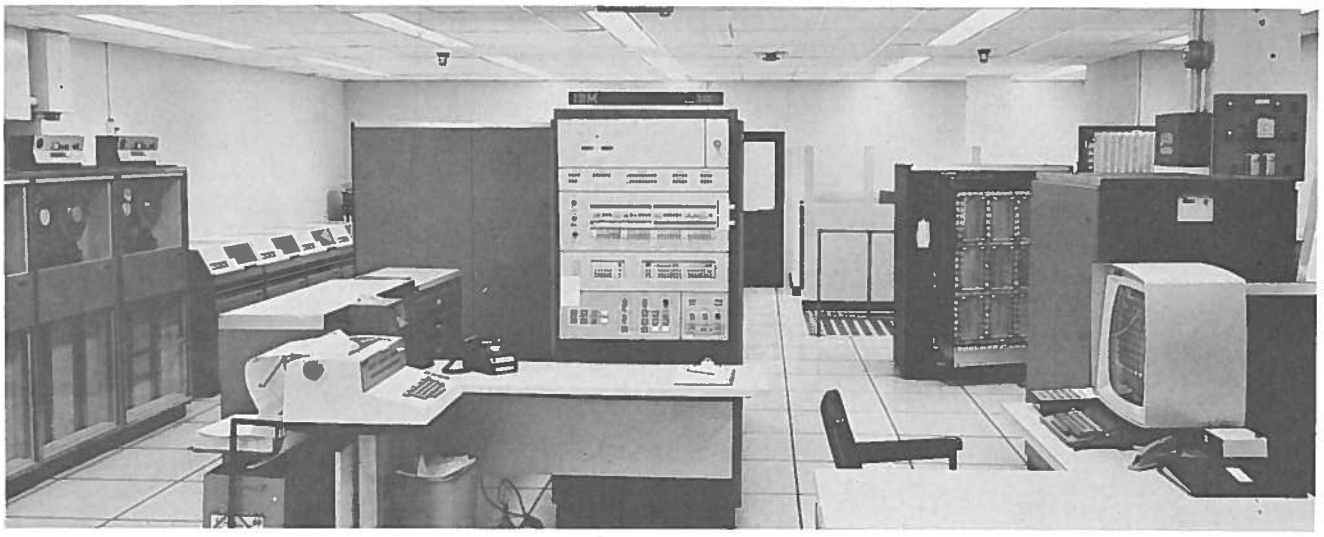


FIGURE 1. FLIGHT SOFTWARE DEVELOPMENT LABORATORY

*This work was sponsored by National Aeronautics and Space Administration Contract NAS8-14000.

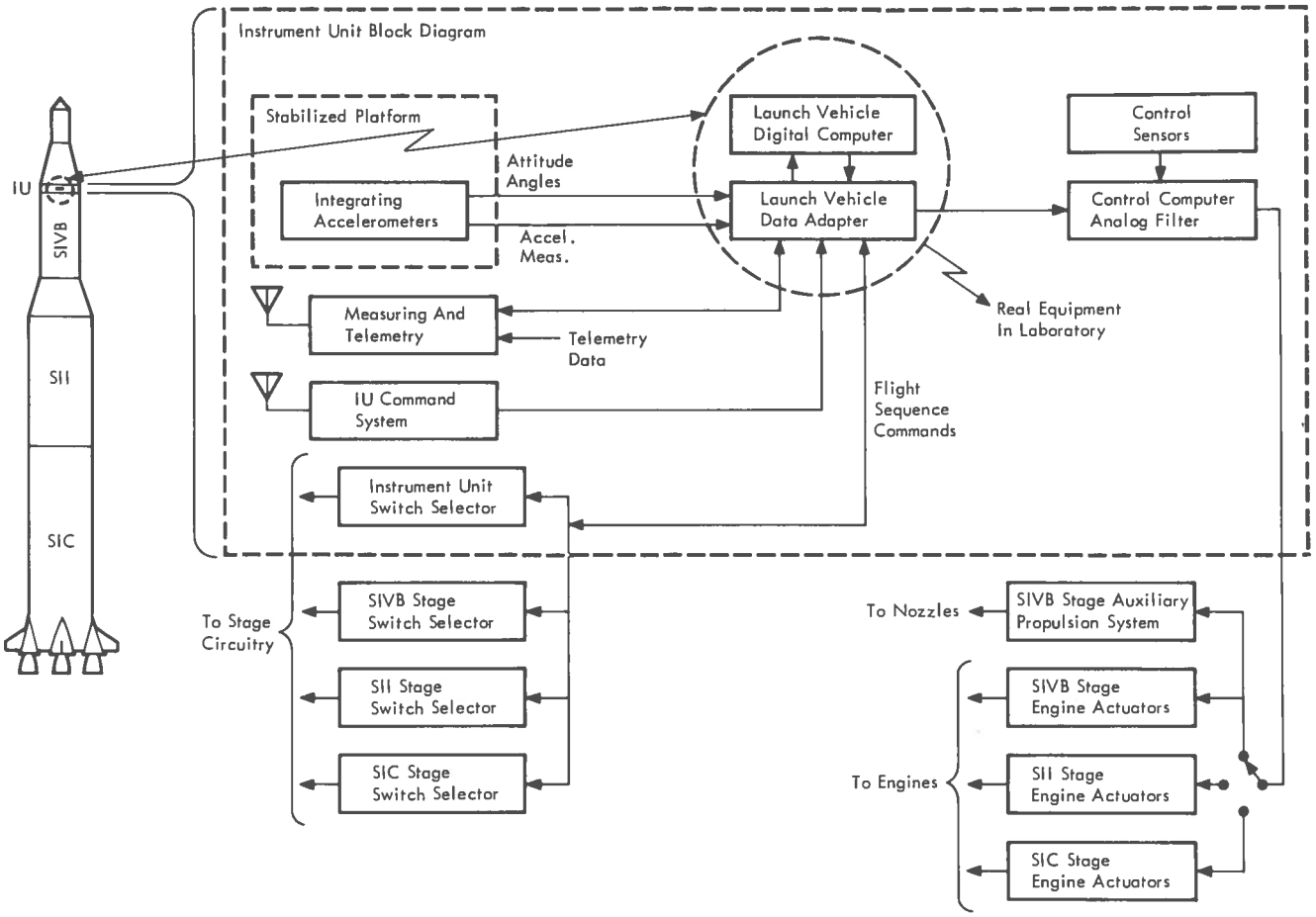


FIGURE 2. REAL AND SIMULATED FLIGHT EQUIPMENT

assembled, and checked out by the program unit or module, the flight phases are integrated and checked out. This process continues until the entire flight software has been integrated. The procedure described above requires that the programmer/engineer be able to measure and evaluate his progress in an efficient manner. The purpose of this laboratory facility is to provide the programmer/engineer with a user-oriented tool by which he is able to test and evaluate his programs in a simulated flight environment, using an actual spaceborne computer and interface hardware. This enables him to measure and evaluate flight software performance against the engineering requirements for the many vehicles and environmental variations.

The Laboratory user must produce quality software in the shortest possible time frame. Therefore, the key objective in designing the Laboratory was to provide accurate simulation models in the form of user-oriented tools. Thus, he can swiftly determine the progress and results of his work through real time man-computer interaction. The computer offers data, counsel, and guidance to the man, who in return supplies certain indispensable knowledge of the overall system. Systems reliability and effective communications between the Laboratory and user play a major roll in establishing user confidence. Operating experience in the Laboratory has clearly demonstrated that these objectives have been satisfied.

Briefly, the Laboratory consists of an IBM System/360 Model 44 linked to a Saturn Flight Computer and Data Adapter. An IBM 2250 Display Unit is used as an integral part of the Laboratory to provide good man-computer interface for communications with the system. This makes possible real time programmer/engineer participation.

The choice of equipment to be used in a simulator such as this is in many ways a subjective judgment based on the user's objectives, his experience, and available resources. To meet the system objectives, it was decided to use the actual flight hardware (Computer and Data Adapter) so as to minimize the risk of error. An alternate approach would be to simulate the Flight Computer and Data Adapter within the System/360 Model 44. An inaccurate simulation of the Flight Computer and Data Adapter would seriously impact the reliability of the software. On the other hand, to provide the required flexibility, it is highly desirable to simulate other vehicle subsystems with equation and logic models. This approach provides the ability to easily generate the numerous malfunctions and off-nominal conditions that must be introduced to measure and evaluate flight software performance.

Hardware Configuration

The Laboratory has as its main hardware components an IBM System/360 Model 44, linked through a special purpose interface to a Saturn Launch Vehicle Digital Computer and Launch Vehicle Data Adapter. An IBM 2250 Display Unit is employed as an integral part of the Laboratory, providing two-way man-computer communications. Figure 3 illustrates the organization of the hardware components and in general indicates the basic paths of information flow.

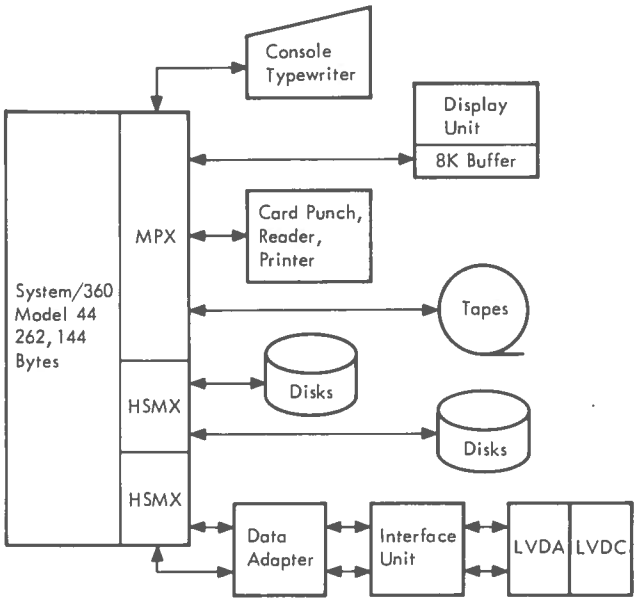


FIGURE 3. FLIGHT SOFTWARE DEVELOPMENT LABORATORY - BLOCK DIAGRAM

The Model 44, even though a member of IBM System/360, has been tailored to handle scientific data acquisition, and to process control applications. The central processing unit has a 1.0-microsecond storage cycle time with a four-byte parallel access-to-processor storage. High speed internal circuits are used to implement four-byte (word) data flow and control. Sixteen general purpose registers use high speed circuits and have a .25-microsecond read/write time. The Model 44 instruction set, including the floating point and commercial features, is the same as the System/360 universal instruction set.

One high speed multiplexer channel has been dedicated to the flight hardware interface. Each of the subchannels is likewise dedicated, as shown in Figure 3. The dedicated channel and subchannels minimize interference from other I/O activities and enable the creation of a special low overhead channel scheduler. These features incorporated with the priority interrupt scheme discussed below make the Model 44 highly responsive to the real time interface requirements. The other high speed multiplexer channel is dedicated to disks that support real time data collection and permit fast access for the display system.

The 32 levels of priority interrupt couple the computing power of the central processor to the ability to respond quickly to different external events with a minimum of central processor time.

In this particular application, six of the 32 levels are used by external hardwired equipment. The others are used by internally generated software functions for scheduling time-dependent software functions.

The Launch Vehicle Digital Computer and Launch Vehicle Data Adapter are the two flight components that have been integrated into the Laboratory.

The Flight Computer is a general purpose computer which, under control of a stored program, processes data serially, using fixed point 2's complement arithmetic. The principal storage device is a random access ferrite core memory with separate controls for data and instruction addressing. The memory can be operated in simplex or duplex mode. In duplex operation, memory modules are operated in pairs with the same data being stored in each module. Each memory module provides 4096 28-bit words of storage. A maximum capacity of eight modules provides a simplex storage of 32,768 words.

The Data Adapter serves as an input/output device for the Flight Computer and the central station for the signal flow in the Saturn Astrionics System (Figure 2). The Data Adapter accepts discrete input signals from the stage switch selectors, Instrument Unit command receiver, ground launch computer, telemetry computer interface unit, telemetry data multiplexer, control distributor, and other vehicle equipment. It has output registers to provide discrete output signals to the above-mentioned equipment. It also accepts and processes computer interrupt signals from the ground launch computer and Instrument Unit equipment. Figure 4 depicts the Flight Computer, Data Adapter, and interface equipment.

The interface unit links the Flight Computer/Data Adpater combination and the parallel data adapters contained in the IBM 2701 control unit illustrated in Figure 5. The interface was built on site, using the same solid logic technology as is used in the System/360 to maximize reliability and to control and minimize noise. The interface unit provides the necessary features to check out operational flight software, in unaltered form, using flight equipment. In addition, the interface unit contains sufficient internal logic, controls, and displays to perform extensive automatic self-check via the central processor and limited manual testing.

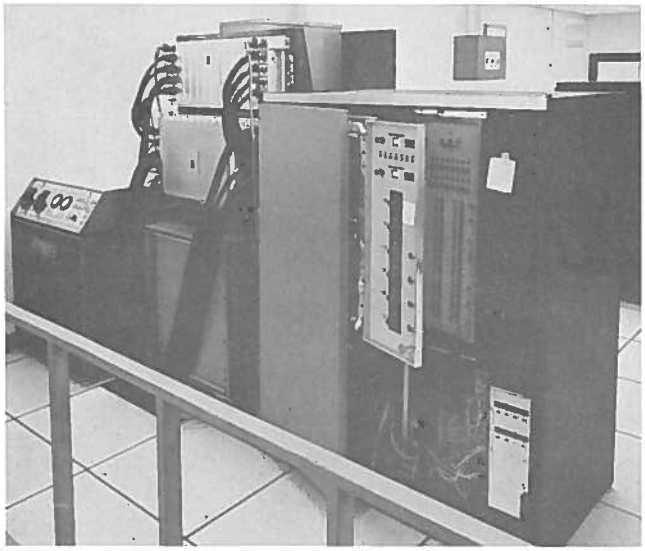


FIGURE 4. FLIGHT COMPUTER, DATA ADAPTER, AND INTERFACE EQUIPMENT

The parallel Data Adapter in the 2701 control unit allows the connection of external devices that perform parallel-by-bit, serial-by-word data transfers with the central processor. The first word of a write operation contains addressing information used to drive the multiplexers. Once a multiplexer channel has been set for an input or an output operation, the central processor is free to transmit or receive data through the parallel Data Adapter without interfering with the central processor.

The priority interrupt control scheme gathers information from many points within the interface unit and generates external priority interrupts and data for the central processor. Each of the six external interrupts is preassigned, allowing the central processor to respond in a predetermined manner.

The interface unit provides all the normal ground and flight communications paths between the flight hardware and the central processor. However, this interface was designed to go beyond

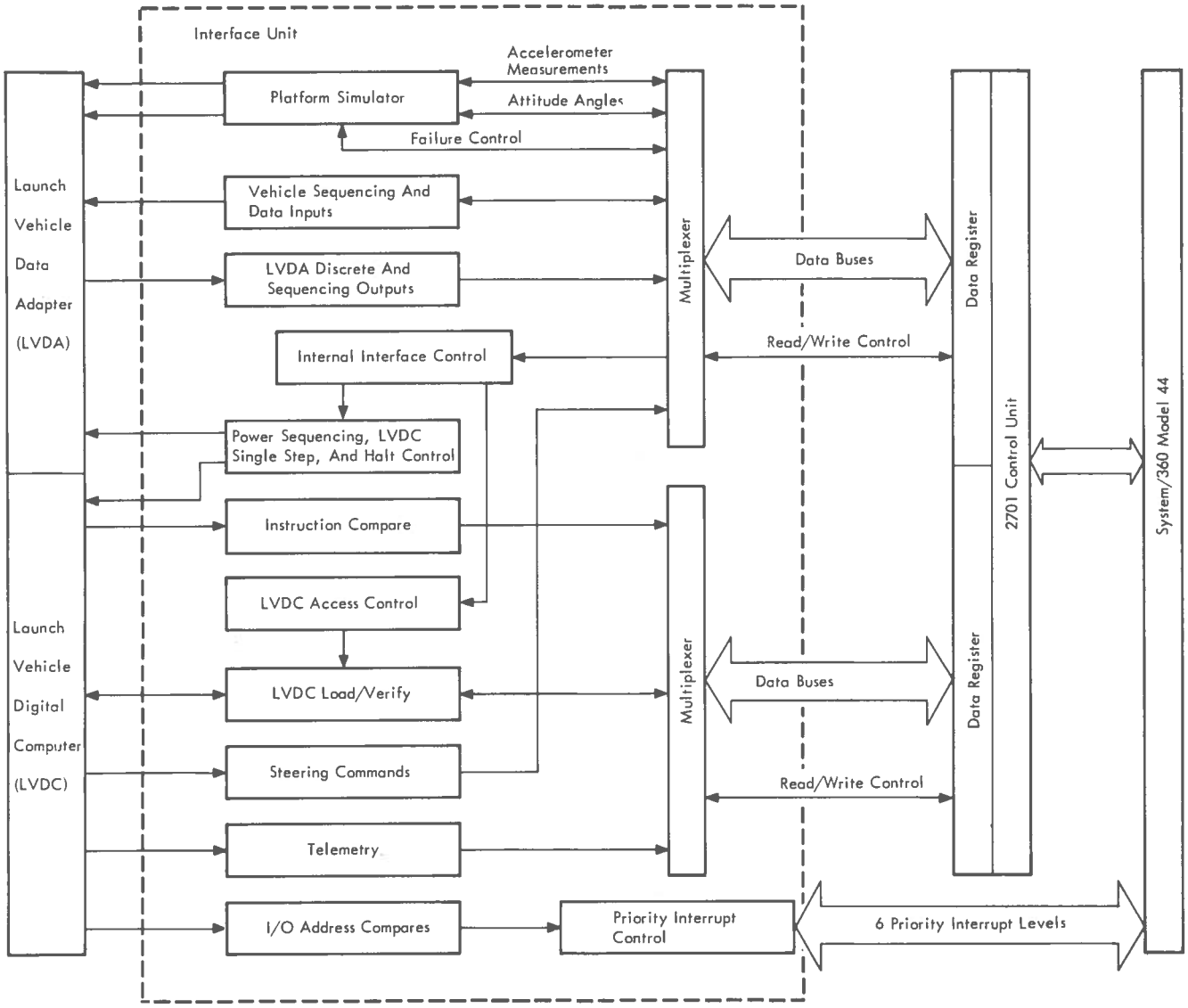


FIGURE 5. INTERFACE EQUIPMENT

these requirements. The interface is unique in that it was designed to place emphasis on (1) minimizing the central processor interface traffic and (2) maximizing user visibility by giving the user the control of internal flight hardware operations and the access to information internal to the Flight Computer. Also, the unit was designed for ease of maintainability. Specifically, three major capabilities have been incorporated into the interface unit. First, the interface unit has been designed so that it can control the internal operation and timing of the Flight Computer and Data Adapter. Through the interface unit, the central processor can single-step the Flight Computer and stop, restart, recycle, or halt the flight hardware under Model 44 software control. All such operations are transparent to the programs executing in the Flight Computer. Secondly, the interface contains special hardware, oriented toward supporting flight program debug as opposed to program verification. The interface unit has direct access to the Flight Computer accumulator, transfer register, operand, address, and memory buffer registers. With such capabilities, the central processor has direct access to the Flight Computer memory, allowing both forced loads and reads of memory that are independent of flight software. In addition, this capability provides the central processor with the ability to force any desired Data Adapter operation by directly loading I/O commands into the Flight Computer transfer register, and forcing their executions without the necessity of processing a flight computer stored program. All these features were implemented through the interface, requiring no modifications to the Flight Computer.

The Flight Computer memory and accumulator access capability is used extensively during the real time mode to allow the user to perform Flight Computer traces, snaps, and dumps. The interface is also able to perform address compares on Flight Computer instructions or data addresses during real time runs as well as compares on Flight Computer I/O addresses. These compares can be used either to collect data (snaps) and/or to control Flight Computer stops, and trace selected areas of the flight program.

Finally, the interface unit has been designed so that extensive automatic diagnostics can be run from the central processor to isolate suspected interface failures. In general, every register in the interface unit may be loaded from and read by the central processor. During the operational mode, diagnostics are automatically run before the user begins to utilize the system for software development. This ensures that the user has a fully operational interface unit before beginning his runs. In addition, the central processor software has been designed so that it is immediately made aware of internal failures in the interface and automatically notifies the user of the suspected problem.

The IBM 2250 Display Unit is organized around a cathode ray tube on which computer-programmed graphic and alphameric information is displayed at high speeds. This provides visual communication between the computer and the user. In addition, keyboards and a light pen provide the user with a versatile means of entering and modifying computer information. With the display system, the user has direct and rapid access to stored data which can be selected, processed, modified, and displayed in alphameric and graphic representation. For example, the user can display and modify memory in both the Model 44 and the Flight Computer through the display unit.

The display unit was configured to minimize central processor time and core requirements on the Model 44. A primary feature of the display unit is a buffer storage of 8,192 bytes which is used to store images for display regeneration purposes. The use of a buffer enables the display unit to operate concurrently with the computer system, freeing the main core and the channel for other functions. Additional features which greatly compress the image storage requirements are the absolute vector and character generator features.

Operating System

The operating system for the Laboratory is designated as the Checkout Control System (CCS). It is the operating system which is furnished with the IBM System/360 Model 44, with additions and modifications to convert the system from a sequential batch job processor to a real time multiprogramming processor. However, all the original functions and features have been retained. Programs not requiring the elements of a real time multiprogramming system may operate as though the additional facilities were not present.

The Model 44 Programming System (44PS) consists of a supervisor, an assembler, a FORTRAN compiler, and system support programs. It provides FORTRAN and assembler language processing and program execution in a monitored environment, with automatic job-to-job transition, interrupt handling, and input/output supervisor. The system has facilities for the creation and maintenance of libraries (program and data) and the manipulation of their contents. It also provides extensive job control and program segmentation capabilities for flexibility and versatility in the preparation of programs for execution.

The supervisor controls the entire system and provides a common interface to all processing programs. In particular, it manages the use of input/output devices, data sets, and processing programs. Program modules are loaded from the absolute program library as required by job control or requested by programs in execution. It handles input/output requirements including error recovery

procedures and the execution of input/output without reference to a particular device type by the user. All classes of interrupts are serviced with transfers to the appropriate system or user routine for processing. The use of the input/output channels is scheduled to effect overlap in these operations.

The principal area of 44PS in which additions and changes have been made is the supervisor. The required functions of CCS include the ability to support various operations of computing at precise intervals of time. These operations are selected by a priority scheme which controls the sequence of execution. Other operations are designed to execute as a result of interrupts induced outside the central processor. These are generally of such importance that their priorities are higher than operations initiated as a result of time. The function of multiprogramming through a scheme of priority interrupts and the requirement of real time operation are the principal requirements for CCS. To satisfy these requirements, capabilities in three principal areas have been added. These are multiprogram scheduling, real time input/output scheduling, and application program phasing control.

A principal element of the program scheduling facility for CCS is the timer queue (Figure 6). It consists of a string of items ordered in ascending sequence of time-to-execute. Each item of the queue contains a pointer to the routine to be executed at the corresponding time. When the timer interrupt occurs, the timer processor routine

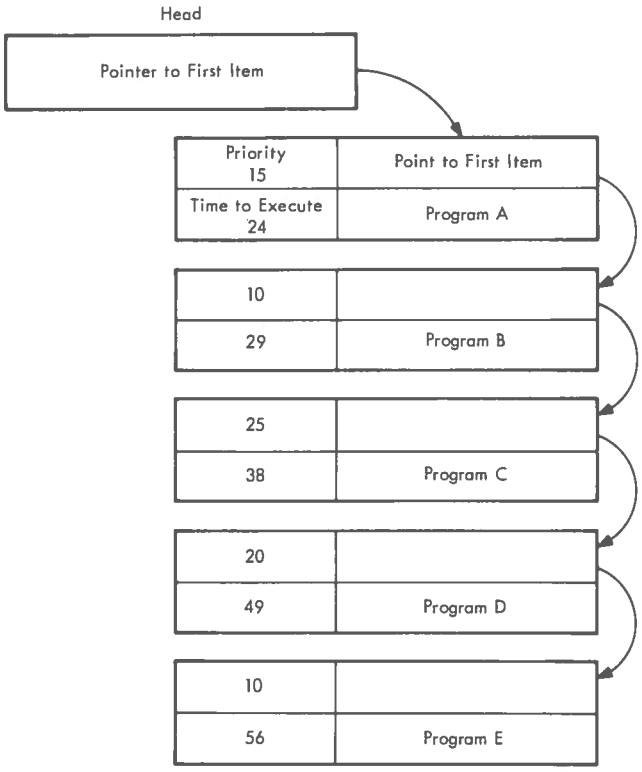


FIGURE 6. TIMER QUEUE

gains control and the routine corresponding to the timer interrupt is placed into a state of execution. Its immediate or deferred execution is a function of priority levels. When a timer interrupt occurs, a comparison is made between the priority level of the routine currently in execution and the level of the routine for which the timer interrupt has occurred. If the level of the current routine is higher than or equal to the other, it resumes execution while the execution of the lower priority routine is deferred. Conversely, if the priority level of the current routine is lower, the other is placed immediately into execution, temporarily suspending the first. This method of scheduling uses the hardware priority interrupt system and additional software of CCS.

Figure 7 illustrates some of the conditions which may occur with a typical combination of timer initiated priority routines. Notice that the execution priority level of timer interrupts is coincident with the priority level executing at that time. In addition, the figure shows how the high priority routine gains control from a lower priority routine. In this priority system, low magnitude numbers correspond to high level priority.

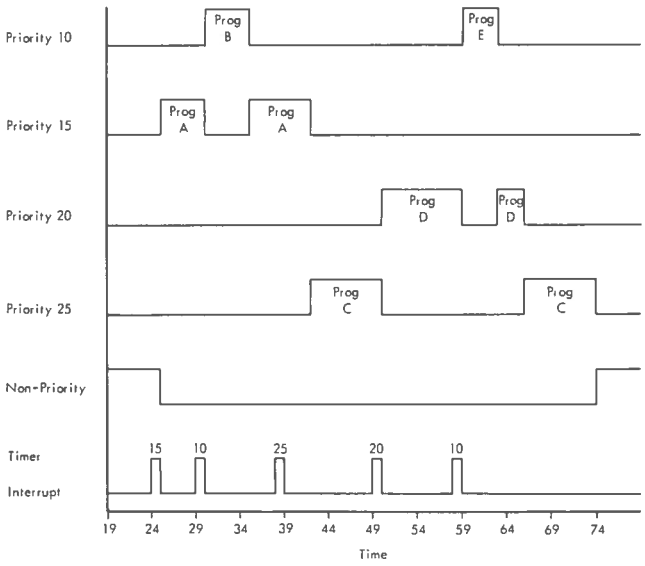


FIGURE 7. TIMER-INITIATED MULTIPROGRAMMING

Figure 6 illustrates a timer queue containing several items, which will initiate programs on various levels at different times. These items match the information illustrated in Figure 7. As each item reaches the top of the list, the internal interval timer is set to the increment of time from "now" until the program is to execute. When the timer expires, a priority level request for the program is set, the item is removed from the queue, and an interval for the next item is calculated. The program pointed to by the item which caused the timer interrupt is attached to its priority level for execution. When the queue becomes empty, the non-priority level regains control.

The second major feature of program scheduling is the supervision of priority interrupts by the priority interrupt executive. Certain 'housekeeping' functions are performed by this feature, such as register saving and restoring, as control passes up and down the priority levels. Control is automatically given to the priority interrupt executive whenever any one of the 32 levels is activated. The routine to be given control is determined, registers are saved as required, and a pointer to parameters is set. Control is then given to the priority routine. When the routine concludes its operation, it returns control to the priority interrupt executive which restores registers and causes the routine on the next highest level to resume or begin execution.

Figure 8 illustrates the overall flow of data and control in CCS. Whereas Figure 7 illustrates the effect of program scheduling, this figure illustrates the mechanics involved. A program currently executing may be interrupted by the timer (1). The timer processor selects data from the queue (2) and attaches the routine to execute (3). It sets a new interval in the timer (4) and initiates a priority interrupt (5) (assuming the routine is of a higher priority than the current program). The priority interrupt executive determines the routine to execute (6) and gives control to the routine (7) which returns control (8) when finished. The executive then returns control to the interrupted program (9). At step 5, the condition may exist that the timer-initiated routine is of lower priority than the current program. If so, the timer processor returns control directly to the current program (10).

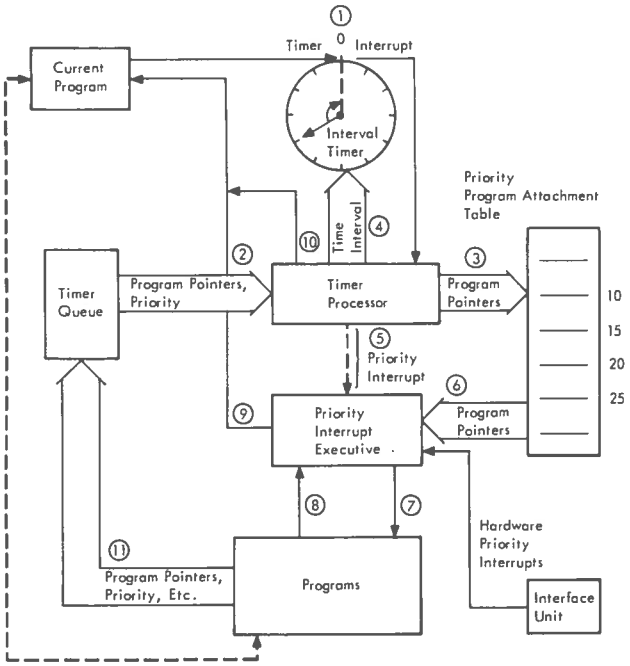


FIGURE 8. PROGRAM SCHEDULING

Both application and system programs may queue routines using the timer queue (11). The actual queueing is done by a system routine.

In Figure 8, the dash line connecting "programs" and "current program" is intended to show that the current program is merely one of many which is selected by the priority interrupt executive.

The System/360 Model 44 Program System (44PS) provides two levels of I/O program interface facilities. The higher of these, called read/write, allows the user to specify and initiate I/O activities by means of system routines. The lower level, called execute channel program (EXCP), allows the user to use some system routines in combination with user-supplied routines. In the real time environment of CCS, the user is required to initiate I/O using the EXCP level; however, FORTRAN I/O calls, all of which are initiated on the same priority level, are allowed to use the read/write level. This restriction is necessary due to the non-reentrant nature of the read/write level facilities.

The 44PS I/O channel scheduler was changed to handle a multiprogrammed environment. Among the changes were new real time, core resident I/O device routines, a gated front end to the channel scheduler and I/O termination interrupt routine, and de-queueing logic at the exit point of the channel scheduler.

Of several options available to modify the channel scheduler, one was chosen which allows only one transaction into the scheduler at a time. The effect of this method is to allow an I/O request to enter the scheduler and be serviced only if the scheduler is not currently processing a previous request. Due to multilevel program execution, an I/O request being made while another is being processed can occur only when the new request is of a higher priority level routine. Therefore, the request in process when interrupted by a higher priority level is resumed at its point of interrupt. The new higher priority level request is serviced immediately thereafter.

Certain routines are used in common by both the channel scheduler and the I/O termination interrupt program. Therefore, the I/O termination interrupts are disabled while a transaction is in the channel scheduler. Conversely, the gate is closed to new requests while an I/O termination is being processed. A posting function is also associated with I/O termination. Its purpose is to allow priority level routines to request I/O, give up control on their level, and then regain control when the I/O is completed.

To meet the demands of real time I/O for the Flight Computer, a special low overhead channel scheduler is used. The gated channel scheduler using 44PS has some features neither necessary nor required for the Flight Computer channel.

Therefore, a specialized channel control program was introduced which reduces the amount of computing overhead required to service I/O requests for the flight computer. Moreover, a much better response to I/O requests exists for the channel. As Figure 9 shows, a test point is included at the beginning of the normal channel scheduler to filter out requests for that dedicated channel. The figure is a simplified flow chart of the gated I/O scheduler.

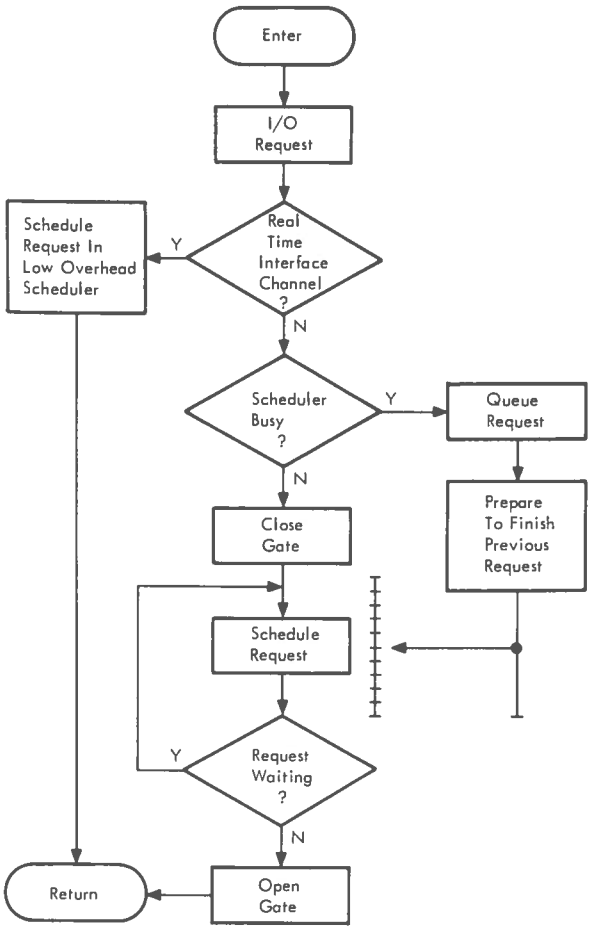


FIGURE 9. GATED I/O

The de-queueing and return logic when a waiting request has been serviced causes program control to return to the high priority routine. The control which would have allowed return to the lower priority routine is adjusted to cause return after the high priority routine relinquishes control on its high priority level.

The CCS phasing function provides the capability to load and initialize the required set of application programs under the control of operations initiated at the display console. An application core load (phase) contains all programs required in memory at the same time to perform one of the major simulation functions.

The phases are resident on the system residence disk volume and are transferred to the application program area of central processor memory when requested by the flight programmer/engineer via the display console. The transfer is implemented by CCS through standard 44PS load capabilities. After a phase has been loaded, the unused portion of memory is calculated and designated as available work space which will later be used by the phase programs. The phase is given program control at its entry point on the nonpriority level of program execution.

Selection of the phase to be loaded and executed is made from the initial tutorial display which is set up by the initial loading of CCS. The user at the display console is enabled to make his selection of the phase via the light pen instrument. The selection of a phase initiates CCS operations which result in a core load of the application program area. When the user chooses to change from the execution of one phase to another, he requests the redisplay of the initial tutorial. Upon this request, CCS executes an orderly shutdown of the activities in process for the current phase and then reloads the central processor memory with the phase requested. Figure 10 illustrates the general allocation of core for the application programs.

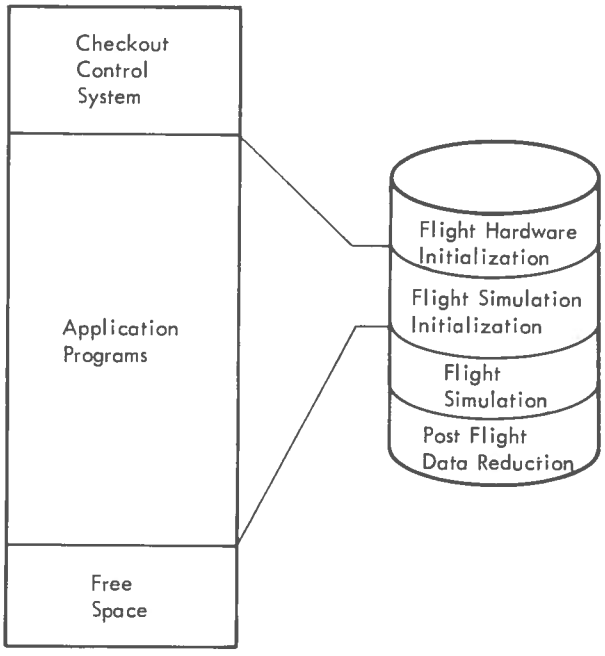


FIGURE 10. PHASE LOAD CORE MEMORY MAP

Application Software

Application software in the Laboratory is designed to perform four basic tasks: (1) hardware diagnostics, (2) flight simulation initialization, (3) flight simulation execution, and (4) post-flight data reduction. A self-contained set of software programs, called a phase, has been constructed to perform each of these tasks. At any given time only one phase resides in core with both communication region and temporary data set residing on a disk device.

The hardware diagnostics phase contains programs which perform the power-up and initialization function for the Flight Computer and its interface unit. The diagnostic programs are required by engineering for maintaining and servicing the interface unit.

The flight simulation initialization phase consists of the programs to specify the details and options of the particular flight simulation the user wishes to make. He may specify such items as loading, modifying, and accessing the flight program; digital command system orders; computer interface unit measurements; real time output quantities; flight pause points; data to be saved for post-flight analysis; the particular Saturn vehicle to be simulated; and the type of simulation run to be made.

The post-flight data reduction phase contains all software necessary to process data that was collected by real time programs operating in the simulation phase. This consists of data from the Flight Computer, the 6-DOF simulator, and the FORTRAN flight program model. The capability of generating plots on the display unit has been provided along with conversion, formatting, analysis, and outputting of data on the printer.

In these three phases, very little use is made of the priority interrupt feature on the Model 44 as an instrument of real time operation. Practically all programs are initiated by operator action at the display console and programs receiving control operate on the priority level assigned to display control. This same level is reserved for display control in each phase. The entire real time simulation phase is built around and is controlled by the priority interrupt feature.

The application programs in this phase perform the following tasks:

- 6-degree-of-freedom (6-DOF) launch vehicle simulation.
- Digital command system simulation (ground data link).
- FORTRAN equation and logic model execution of the flight program.
- Data reduction and analysis.

Figure 11 presents an overview of the major application software components required for the real time phase and their interrelationships. The

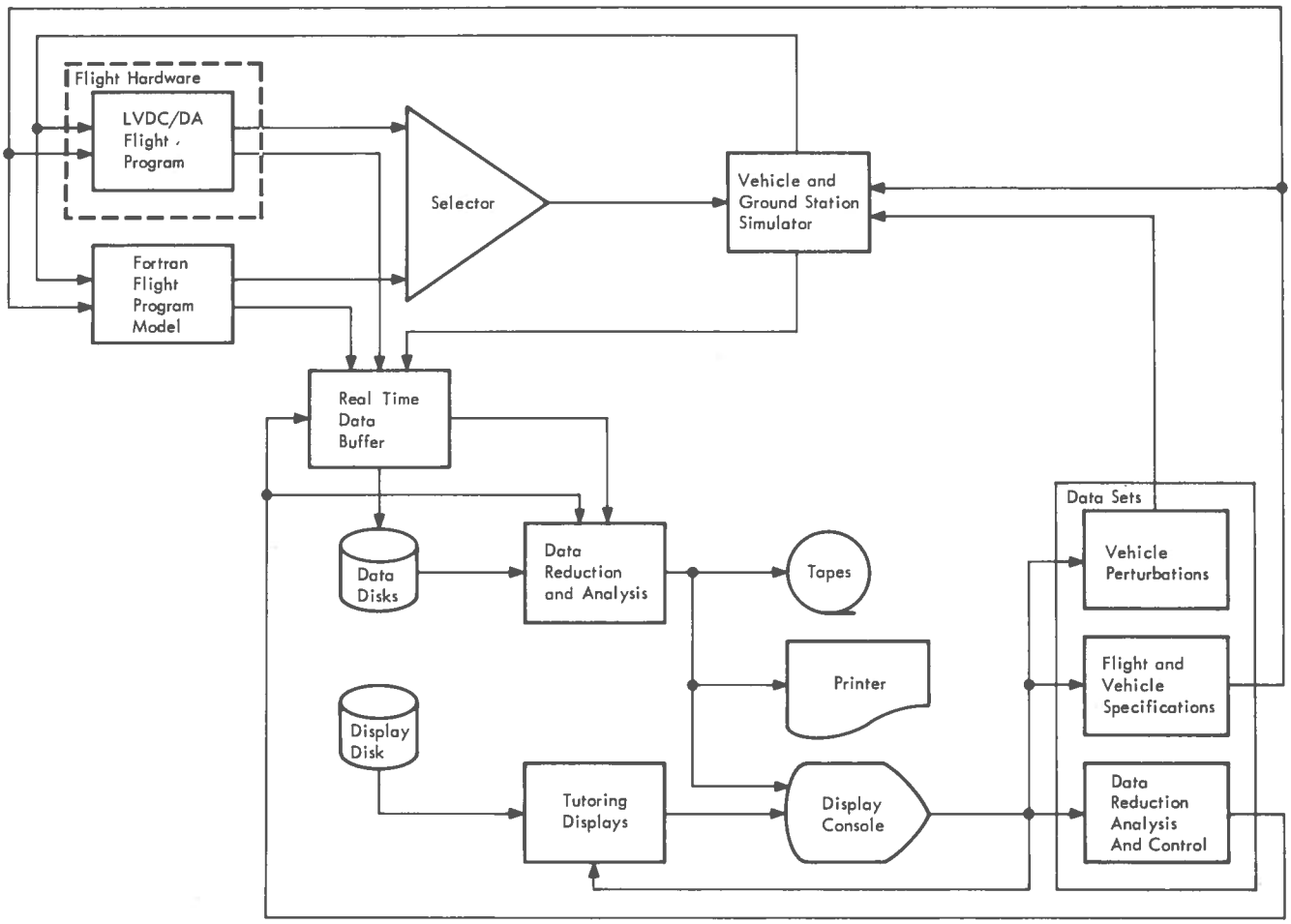


FIGURE 11. APPLICATION SOFTWARE FUNCTIONAL FLOW - REAL TIME PHASE

operator, seated at the display control unit, communicates with the system through preformatted tutorial displays. Three data sets are used as an input interface between the display unit and the real time application software. The flight and vehicle specifications data set is used to structure the vehicle and flight program skeletons to any of the many missions under development. This data set is defined and generated during the initialization phase. The two remaining data sets (vehicle perturbation and data reduction analysis and control) are accessible both during the initialization and real time simulation phases. The vehicle perturbation functions allow the operator to specify various vehicle anomalies such as thrust perturbations, command receiver failures, staging or event failures, inertial platform failures, etc., in addition to the start time and duration for each. The appropriate control information is ordered by time of occurrence and recorded in a vehicle sequencing queue until the specified activation time. A similar procedure is followed in the creation of the data reduction analysis and control data set.

The FORTRAN flight program model is an engineering representation of the flight program. It serves as an additional reference to measure and evaluate actual flight software performance.

The real time data buffer receives data from the FORTRAN flight program model and the 6-DOF vehicle simulator as well as telemetry data from the flight hardware. This entire set of data is recorded on tape for the post-flight data reduction phase. Data selected for real time observation is organized, formatted, and recorded on the disk. This particular data is accessible at any time upon request from the display console as either tabular data or graphic plots. Such displays may be generated from historical data beginning at some particular point in the past and carried up through the current values or it may start with current values. In both cases, the display is continually updated from the real time data buffer. In addition, the real time tabular data may be permanently recorded on the printer.

The 6-DOF simulator consists of both rotational and translational dynamics as well as a simulation of the vehicle subsystems involved in vehicular control, such as sequencing, digital command system, etc. This simulator may be driven by either the actual flight hardware or the FORTRAN flight program model. In turn, the 6-DOF simulator supplies inputs to both the flight hardware and the FORTRAN flight program model.

Each of these real time application programs is assigned a relative priority and an absolute priority level. Figure 12 shows groups of application programs and their priority interrupt level assignments used in this system at the present time. In general, high priority levels have short execution times. These routines respond to discrete external events or internal keying by the interval timer. Routines with longer execution times are

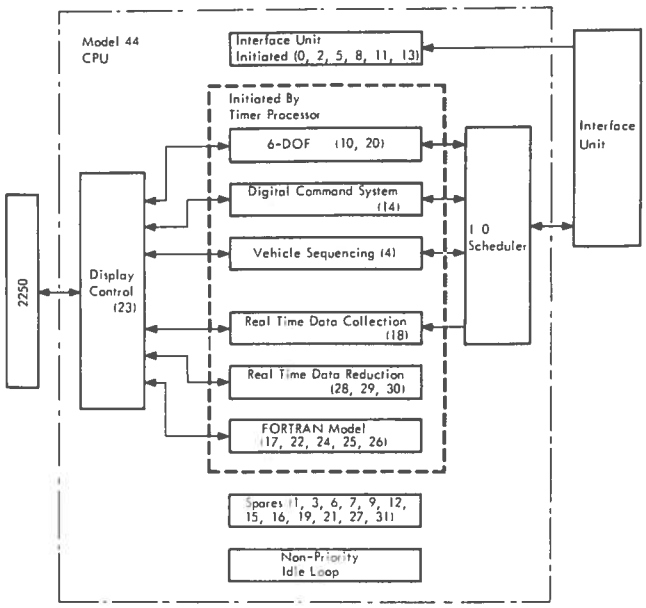


FIGURE 12. FUNCTIONAL GROUPINGS OF PRIORITY INTERRUPT LINE ASSIGNMENTS (0-31) IN REAL TIME PHASE

on lower priority levels. Among these routines are real time reduction and graphic support. The priority level assignments, both hardware and software activated, can be changed easily to optimize system performance.

The requirements of the real time application programs guided the design of the program scheduler in CCS. As a result, the priority scheduler provides the real time application programs with a highly flexible operating environment. This environment has developed as a result of these significant points. First, the System/360 Model 44 and the Flight Computer operate asynchronously with respect to one another. This condition relieves the system of several constraints in its operating environment which would otherwise be present and tend to constrict the system rather than allow it flexibility. Second, the application programs are very responsive to the information supplied from the interface unit via the priority interrupt feature. There are six high priority levels of the thirty-two which are assigned to signals from the interface unit. This structure permits immediate response to Flight Computer conditions in the Model 44 by interrupting programs operating on a lower priority level. Third, a related function to point two is that the low priority operations such as servicing display unit operations execute on a non-interference basis with the time critical functions on higher priority level assignments. Fourth, with respect to time-slicing, it is a self-adjusting system. This means that programs on lower priority levels will automatically give up time to programs operating on higher levels. For example, the solution rate on vehicle navigation can be changed by simply altering one constant so that the entire system will self-adjust to the new solution rate.

User/System Interaction

With reference to user/system interaction, the system may be said to have two primary objectives: to provide a more detailed and complete checkout of flight programs, and to ease the burden and reduce the time required of a flight programmer to check out a flight program. To meet these objectives, the following system criteria were established:

1. Minimal knowledge of the central processor required of system users.
2. The number of people required to run a simulation minimized.
3. The operator control stations centralized.
4. The users' ability to influence the flight program maximized.
5. The time required to set up runs minimized.
6. Entire simulations run by nontechnical operators.

In order to satisfy these criteria, system start-up procedures were automated, peripheral device management routines were written (to allow tapes and disks to be remounted on arbitrary drives), direct access storage was fully utilized, program overlay was used extensively, and all operator control (after initial setup) was centralized at the display unit.

The graphic display software provides the interface between the application programs and the IBM 2250 Display Unit through which the user communicates with the system. Through this software, the display console operator is in complete control of the flight program and has a very wide range of capabilities in initializing, controlling, monitoring, and analyzing the flight program performance.

The display program that provides this interface operates in a real time environment. Therefore, to reduce the core and time requirement necessary to create each individual display, all displays are preformatted by an off-line graphics program. (See Figure 13.) This program receives card images of the text and control information associated with each display and creates a 'book' of displays. This display book resides on a disk cartridge and is divided into one index and as many chapters as there are displays. Each display text is in an 'expanded' format containing embedded graphic orders and in a format ready for transmission to the display unit buffer. It requires no editing, scanning, or unpacking in real time. The keyboard and light pen pages provide control information needed by the real time display control program to respond to operator keyboard and light pen inputs. For each chapter created by the off-line program, there is a corresponding entry in the index. Each index entry

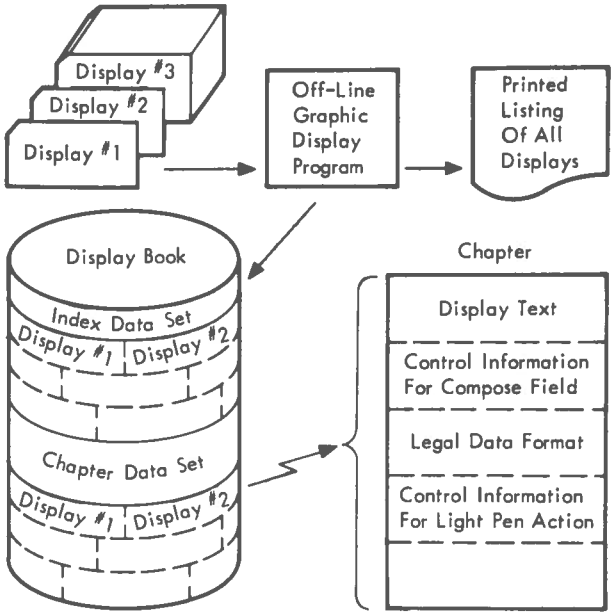


FIGURE 13. DISPLAY BOOK GENERATION AND ORGANIZATION

contains the name and disk address of its corresponding chapter.

During system initialization, the real time display control program reads the index into core and retrieves the system initial display from the display book. The initialize phase of the real time display software is complete when the initial options are displayed. Light pen or keyboard inputs from the display console operator are required to initiate the display of new texts.

The display device routine services the light pen and keyboard I/O interrupts and schedules the display control program on the timer queue for immediate execution on a predetermined priority interrupt level. When display control gets control on its priority interrupt level, the type of action taken by the operator is examined. The light pen page and the keyboard page of the current display chapter define all legal light pen inputs and keyboard entries.

The display control program displays the proper text in response to the operator's light pen actions, validates keyboard inputs, and passes control information specified in the keyboard or light pen page. Response to the user's inputs appears to be instantaneous to the operator (500 milliseconds maximum). When a longer time is needed to process the operator's request, the program to perform the operation is scheduled to operate on another priority level and normal display processing continues. The operator may initiate several tasks to be performed simultaneously.

On a light pen detect, the light pen page of the current display chapter is examined for a possible NEXT PROGRAM. If one is specified, control is passed to it and the NEXT DISPLAY is presented

when the program returns control to display control. When the NEXT PROGRAM is omitted from the light pen page, the NEXT DISPLAY is presented immediately and the display control program priority level is freed for additional operator inputs.

Display control makes legality checks on all keyboard input against the legal data in the keyboard page of the display chapter and passes the data to the designated program. When it is necessary to input a large amount of data through the display compose fields, or when many displays and light pen actions are required to initiate a procedure, the light pen options and keyboard entries may be predefined on cards or disk. The display software can initiate one option after another and each time return to the predefined option set for another option rather than waiting on the console operator for further action. This speeds the setup for procedures done repetitively and greatly reduces the possibility of operator errors.

The real time display control program accepts inputs from the display console operator and also from the application programs. While there can be many application programs providing input during a small interval of time, there can be only one display being presented to the operator. The inputs affecting future or past displays are entered in a queue and may be viewed by the operator by use of the function keyboard.

An application program on any priority interrupt level may use the display system to communicate with the user through previously defined input areas in the display text. These input areas may be defined by the user to suit his needs and to present his input data in an easy-to-read format. For example, if the input areas in a display are defined in a column format, the programmer's data will automatically be presented in a column format when the input areas are filled.

Figure 14 illustrates the display control interface with the display unit. The control information pages of a display chapter will remain in memory as long as the display text is being presented to the operator. When the operator uses the light pen or keyboard, display control will use these pages to determine the NEXT PROGRAM and NEXT DISPLAY. When the chapter for the next display is retrieved from the display book, the text page processor merges the display text with any application program data to be displayed and transfers the combined text and data to the display buffer. The new light pen and keyboard pages will remain in memory to identify the next operator action.

A function key must be lit by the function key processor before the key becomes active. An application program can direct display control to activate a function key and present a given control display when the operator uses the key.

As the user views the system, the heart is the display system. The programmed book of tutorial

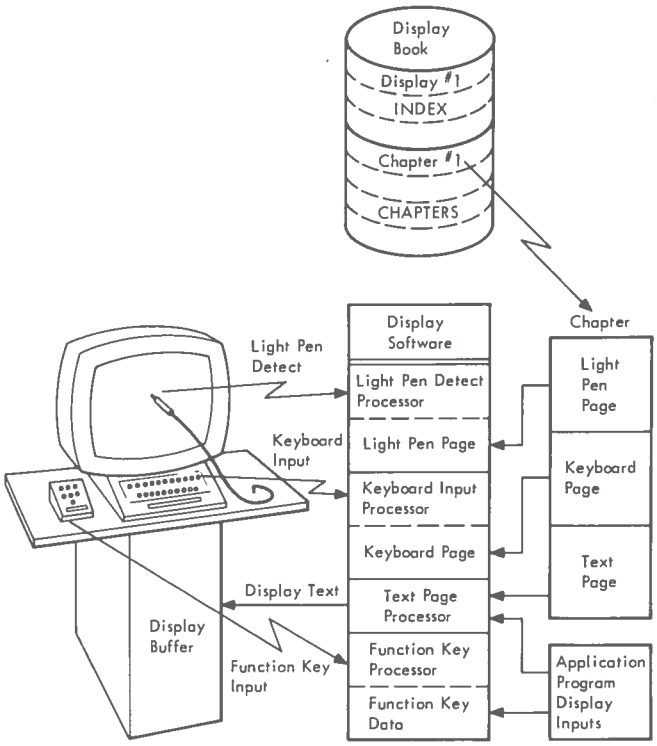


FIGURE 14. DISPLAY CONTROL INTERFACE

displays is provided to give him complete control over the Flight Computer, the interface, and the simulation itself. The book along with the use of the light pen and the display keyboard leads the user through the functions of powering up the Flight Computer and Data Adapter, loading and accessing the Flight Computer, setting up and executing the simulation, and post-processing simulation data. Each user option is carefully spelled out, and all user input is verified before it is accepted by the system. Should error conditions occur (due to incorrect input, hardware failure, or flight program failure), error messages are presented to the user with instructions as to the recovery action.

A complete history of user actions at the display unit is logged on the console typewriter for later reference.

Figures 15 and 16 demonstrate user activity at the graphic display terminal. By using the light pen, the user is able to travel through the display structure illustrated in Figure 15. Figure 16 depicts photographs of the displays represented by the structure of Figure 15. Figure 16a shows the top level display (DM00). When the light pen is applied to the keys for path 1, it leads to a display, DM33DA (Figure 16e), calling for information to be entered from the keyboard. The keyboard entry or compose field is defined by the legend on the display. Path 2 shows how the same path may be entered from the execute flight simulation display, real time phase.

Along with the capability to set up and execute complete runs through the display system, the user has the ability to monitor the execution and take

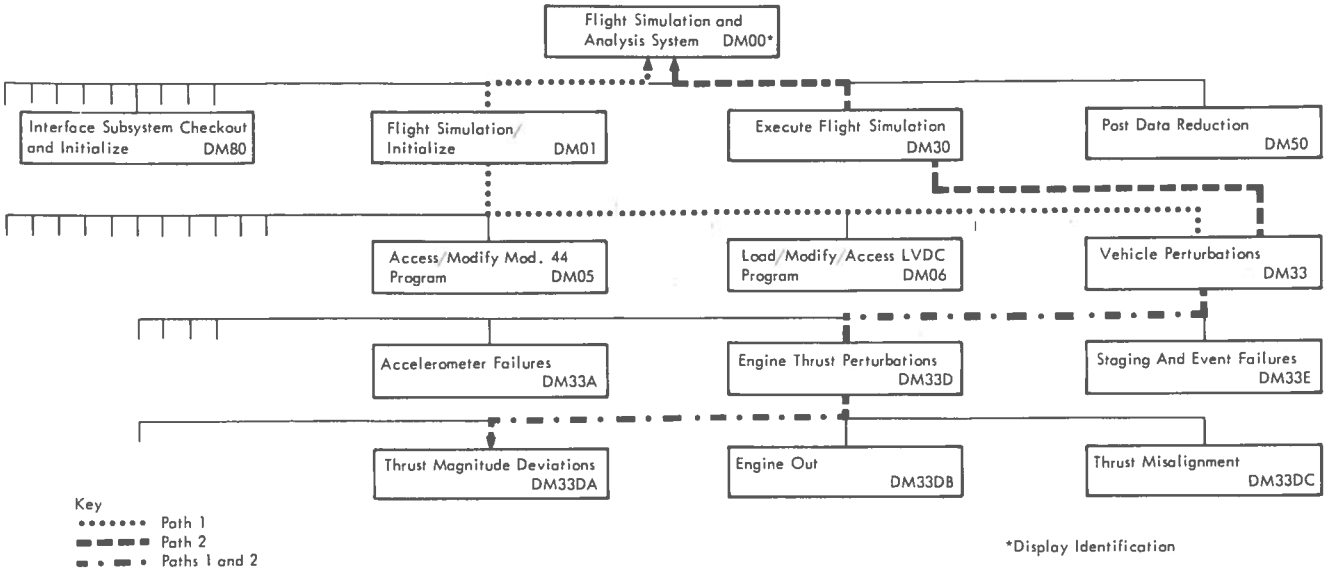


FIGURE 15. DISPLAY STRUCTURE

restart dumps. These, too, are controlled from the display console. A request may be made to print out specific quantities on the printer as they are calculated in either the Flight Computer or the central processor. At the same time he may request various status displays, tabular displays, or data plots to appear on the display unit. Should an irregularity be detected, the operator has the ability to pause the simulation, process all the data acquired thus far, and make changes or corrections. He then has the option to continue the simulation, restart the simulation from various points where restart information is available, or terminate the run entirely.

Due to the complexity of the flight programs, the flexibility of the facilities of the Laboratory, and the desire to ease the burden of job setup as much as possible, a scheme has been implemented to sequence automatically from start to finish. The sequencing and input information can be saved on cards or in data sets on direct access storage. Through the use of this scheme, it is possible for a flight programmer to set up complex runs, submit the job for running by an operator, and return later to pick up complete output from the run.

All the pertinent information flowing through the real time data buffer is collected and saved on tape (the 'Post Processor tape') for later analysis. If the flight programmer has requested SNAPS and TRACES of actual Flight Computer memory locations during instruction execution, this information is saved on the postprocessor tape. The postprocessor tape may be processed immediately or at a later date.

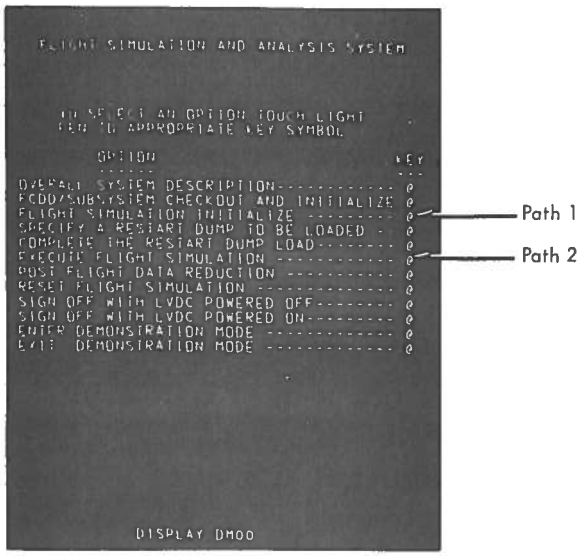
When processing the tape, the user has several options available to him through use of the display system. He may selectively dump any data on the tape and request that the data be converted to decimal form in specified units prior to printing. He may have his data printed in a tabular form or he may plot data on the display unit. Special calculations may be performed on some of the data and the results printed or displayed. He may print or plot errors between various quantities to verify that the flight program results agree with the 6-DOF simulator.

Experience

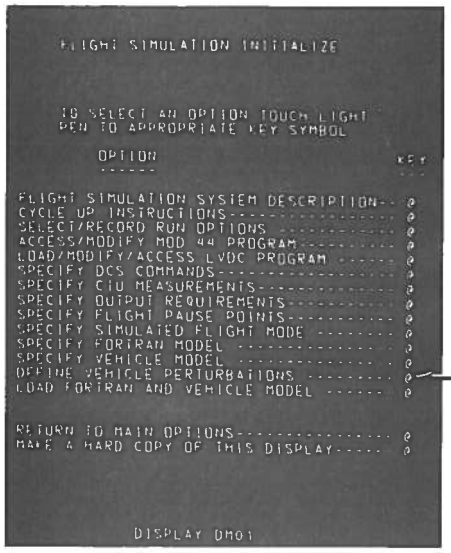
The implementation of the Laboratory from the initial planning phase through the completion of the original objectives spans a seventeen-month period. The System/360 Model 44 was installed during the sixth month; the flight hardware and interface unit were installed during the seventh month; and the system was made available for flight software development, with limited capability, during the ninth month. As of this writing, twelve months into the project, the Laboratory facility has been used extensively in support of the AS-503/C Prime (Apollo 8) flight software development activities. Although it is still too early to completely assess the total impact of this Laboratory on the future of flight programming, it has met all initial objectives thus far.

The programmers and engineers are currently using the system in excess of 40 hours per week for its intended purpose. The Laboratory has provided them with a new dimension of visibility and man-computer interraction which has had the effect of stimulating higher levels of interest and creativity.

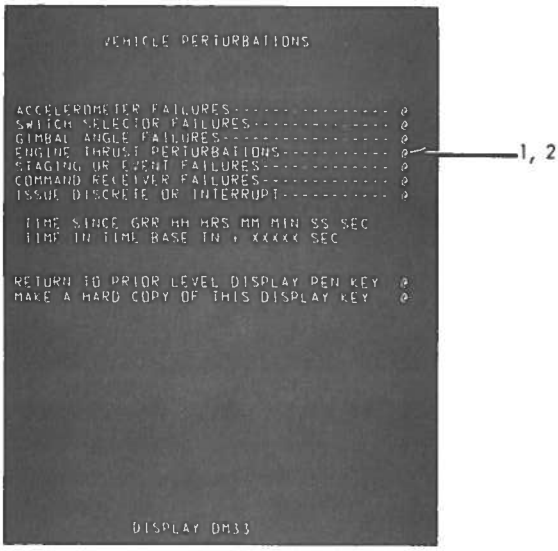
In the development of the Laboratory, certain aspects have been cause for concern. A great deal of time and effort by the most experienced and competent software designers was spent in the planning and organization of the vast amount of software needed. Such an approach necessarily makes for a slow start but is of such importance to be done properly that any undue haste in this phase is likely to result in an inferior product which will later require extensive rework. Another phase of the de-



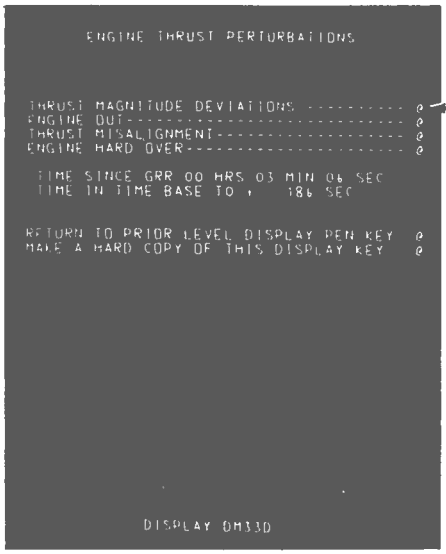
16a.



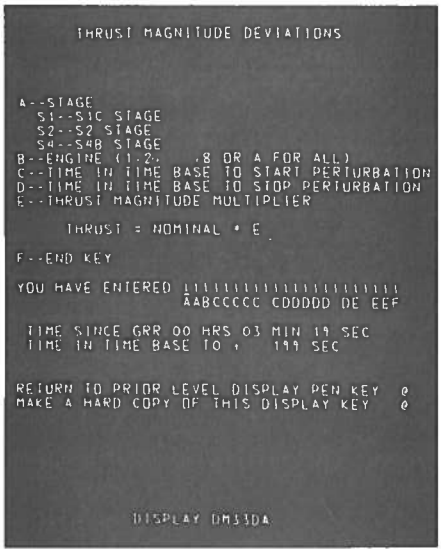
16b.



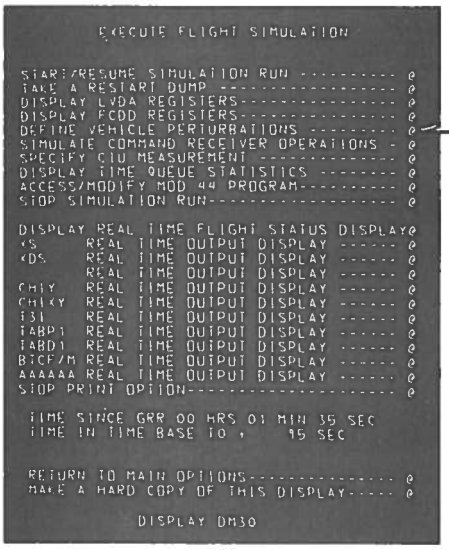
16c.



16d.



16e.



16f.

FIGURE 16. TUTORIAL DISPLAYS

velopment which was nonproductive in itself was the period in which the computer was entirely dedicated to operating system checkout. However, this period is essential and must be done not only with dispatch but with accuracy so that the working application programs which run under control of the operating system may be executed with little or no error conditions occurring in the operating system itself.

Conclusion

The problem of designing a Flight Software Development Laboratory and ensuring that the programmer/engineer has the capabilities he requires is a complex task. The techniques of simulation, the selection of equipment, and the methods employed for man-computer and computer-computer interface must be carefully weighed. The requirement for pin-point accuracy in the Laboratory resulted in a real time multiprogrammed system which is proving an invaluable tool for assisting in the development and checkout of the flight programs. It has made possible the development of flight software which can be relied upon to a much greater extent than before and has reduced the amount of time necessary to produce it. In effect, the Laboratory accomplishes the necessary aids toward producing successful flight software.

Some of the software concepts employed in the Laboratory may certainly be applied in related areas of simulation technology. The operating system and display support software have direct conceptual applications in airborne and space vehicle simulators.

Federal Systems Division, Space Systems Center, Huntsville, Alabama