A hierarchical distributed real-time computing system, LABS/7, provides facilities for attaching multiple IBM System/7s to a host System/360 or System/370. LABS/7 consists of a multiprogramming and multitasking supervisor for the System/7, a host communication facility that supports multiple satellite System/7s, and a high-level real-time language for user application development.

LABS/7 is operational in a variety of environments, including research, development, manufacturing, and clinical. The functional characteristics of the system are reviewed, performance is stated for well-defined situations, and experience with the system is reviewed with reference to some typical applications.

LABS/7—a distributed real-time operating system

by D. L. Raimondi, H. M. Gladney, G. Hochweller, R. W. Martin, and L. L. Spencer

Many systems designed to meet the requirements of real-time event-driven applications have been described.[1] However, none of the systems of which the present authors are aware has demonstrated utility in a broad range of environments. For example, many process control systems cannot be used for laboratory automation, and vice versa. In this paper we describe the Laboratory Applications Based System, or LABS/7,[2,3,4,5] a system that supports a wide variety of applications in research laboratories, development and test laboratories, and manufacturing and production environments.

In our experience, except for the most trivial applications, a computer hierarchy is the most economical means of collecting and processing data.[6,7] The rapid trends toward data collection at the original source and toward interactive computing favor distribution of function among several processors coupled together. Real-time event-driven (sensor-based) computing is special in this respect only because the nature of the source data may impose special timing requirements, or because a very large number of asynchronous events may have to be processed within predefined intervals.

It is not at all obvious, in a computer hierarchy intended for a wide variety of applications, how best to distribute function among the available computing elements, or how to design future systems to minimize overall cost. Generally, large central installations tend to have more effective facilities for program preparation, data reduction, and long term data storage than small computers assigned to a limited set of applications. On the

other hand, adequate responsiveness and availability may well be easier to achieve in a small, local computer. The system described in this paper illustrates a successful distribution of function among multiple IBM System/7 local, or satellite, computers and a System/360 host.

Once a certain distribution of function has been chosen, of course, the design of a successful system must consider each element as part of the overall system, not as an independent entity. The quality of the communications facilities, both hardware and software, is critically important. Particular attention is given to communications facilities in the discussion that follows.

Usually it is possible to share each local processor among several applications, since the processor's capabilities generally far exceed the average requirements of any single application. The decreasing cost of computer hardware is sometimes given as a reason for dedicating a minicomputer to each application, but this reasoning ignores the relationship of program preparation, testing, and alteration costs to the quality and cost of peripheral devices and communications links between computers. LABS/7 enables each satellite System/7 to be shared effectively among unrelated applications, and it also provides for easy and inexpensive preparation of application programs.

It is necessary to have a control program at each satellite in a distributed system. Control of any single real-time application usually involves several asynchronously running tasks, and the usual mechanism for regulating contention for resources is a resource manager which allocates to the various tasks the serially reusable resources of the system. Granted that such multitasking is required, extension of the resource manager to permit multiprogramming is straightforward.

Another important consideration is that the costs of computer hardware and supervisory programs are continuing to decrease in relation to the cost of preparing application programs. This consideration has been explored quantitatively for laboratory automation in a study which found that the development of application programs accounts for 10 to 50 percent of the total cost of automation, depending on the particular experiment under consideration.[8] To minimize this cost, the application programmer must be provided with both high-level languages and interactive procedures.

System objectives

The overall objective of LABS/7 is to manage the entire set of resources and provide the user with guidelines by which the ap-

plication should be designed, including the most cost-effective balance of function between host and satellite.

If we assume that overall costs can be reduced by having several applications share each local computer, then multiple independent processes must be supported concurrently in such a manner that each application can be designed, programmed, and tested with minimal reference to other applications or to the specific set of hardware on which the application will be executed. And if application development costs are to be minimized, each user must be able to write his own application programs, with minimal consultation with support programmers, in a language that provides detailed control of system resources and that is easy to learn.

The specific objectives of LABS/7 are to achieve these broad goals with a sensor-based system in which each of several satellites provides event-driven support for several independent applications, and in which a central host system provides for program preparation and large-scale data reduction and storage. It is helpful to consider three categories of requirements: those associated with real-time control and data acquisition; those associated with data reduction, storage, and reporting; and those associated with program preparation and testing. Each of these categories is discussed below.

Real-time control and data acquisition in LABS/7 are supported by a control program that manages the resources of the satellite computer and by application programs that are executed on the satellite. The support provided by LABS/7 includes:

- A command language for real-time application programs;
- The ability to initiate any application program from a terminal or another application program, and to pass parameters to the new program;
- Multitasking in each application program, with preemptive task switching;
- Interval timing, with timing precision tolerances that can be deduced from the system specification for each mix of applications;
- Multiple terminal support, so that a terminal can be assigned to every application that requires one;
- A relocating loader, so that an application program can use any area of main storage that is available at the time of invocation;

- The ability to operate independently of the host computer, except when an application program explicitly invokes interprocessor communication.

Data reduction, storage, and reporting functions in LABS/7 are supported on both host and satellite. They are made convenient and flexible by:

- The ability to transmit data to and from the host, to initiate program execution at the host from the satellite, and to synchronize program events on the host and satellite;
- FORTRAN support on the satellite as well as the host;
- Support for several satellite computers on a single host;
- A range of communications options, so that the cost of the network, and its speed and type, can be tailored to the user's needs.

To give the user as much flexibility and simplicity as possible in preparing application programs, the system includes:

- Symbolic addresses in application source code, including the addresses of hardware devices and data files;
- Procedures that minimize what the programmer needs to know about the host environment;
- Source code editing on either host or satellite.

System overview

LABS/7 is based on an architecture for a hierarchical, distributed, real-time computing system which could be implemented on any central computer and any process control computer. Our implementation, on System/360 and System/7, is an evolving set of software and hardware components which address the general requirements described above. Several variations of LABS/7 have been used in laboratories and plants both inside and outside of IBM. Each of these systems has had slightly different characteristics of appearance, performance, and function. The most obvious differences have occurred because each installation has had different requirements for interprocessor communications. In the discussion below, when it is necessary to be specific about some variable detail, we will refer to the version in operation at the IBM San Jose Research Laboratory.

The hardware for LABS/7 consists of:

- A host computer (System/360 or System/370);
- One or more event-driven satellite computers (System/7);
- Communications lines (telephone or coaxial cables and appropriate interface hardware).

The software consists of:

- OS/360 or VS/370 on the host computer;
- The LABS/7 supervisor on the System/7;
- LABS/7 communications facilities on the host and satellite.

The new operating system, LABS/7, consists of the aggregate of:

- A multiprogramming, multitasking supervisor for System/7;
- A high-level, real-time language for application development;
- An operating environment that includes communications facilities between host and controller, program preparation facilities, interactive procedure aids for application development, and a set of System/7 utility programs.

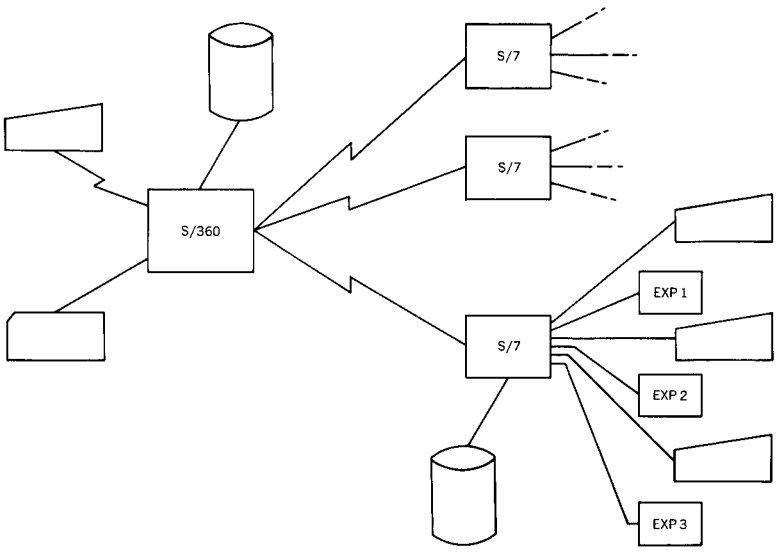
In the following sections we describe the functional characteristics of each of these items. Performance data are given for well defined situations, and experience is reviewed with reference to a few typical applications. Discussion of host (System/360 or System/370) and satellite (System/7) hardware, and of host software (OS, VS) is limited to references or specific performance characteristics that relate to the real-time distributed functions of the system.

Figure 1 illustrates schematically the hardware configuration in our laboratory. A System/360 is used for program preparation and large-scale data analysis, and several System/7s are used for real-time data acquisition and control. Each System/7 has three to ten attached applications.

implementing new applications

Implementation of a new application begins with program preparation, either on the System/7 or on the host. The user writes the instructions that define the sequence of functions to be performed on the System/7 to read his data and control his apparatus. His source code is then assembled on the host into a machine language load module, which is sent to the appropriate System/7 and stored in a direct access program library.

Figure 1 Configuration of IBM System/360 and System/7s in the San Jose Research Laboratory



The next step is performed at the System/7. To begin an experimental run, the user at his terminal may communicate with the LABS/7 supervisor to request that a specific application program be loaded into main storage. Thereafter he may communicate, through the same terminal, with the application program to enter initialization parameters and to start data acquisition.

Once started, the user's program will perform the event-driven I/O operations and timing and logical functions that he has defined. Usually, the application program will store output data on a System/7 disk file, although it could transmit the data directly to the host. The availability of disk storage for both programs and data means that data acquisition, experimental control, and preliminary data analysis can be performed independently of the host facility. We have found this capability essential in providing the necessary reliability and availability for most experiments.

The next implementation step involves data reduction and the output of results, again on either the System/7 or the host. Most users prefer to use the host, which generally has more varied and powerful facilities. For some applications, of course, this step may be just the beginning of a series of operations of far greater complexity. It is important to recognize that this scenario serves only to illustrate a typical mode of operation; specific techniques vary widely, as dictated by different requirements.

Essential to the viability of a distributed computing system in the real-time event-driven environment is a communication facil-

ity which, in a straightforward and responsive manner, provides for bidirectional data transfer and batch job submission. Bidirectional data transfer enables a satellite application program to transmit data to the host computer and retrieve data from the host; and through batch job submission, a satellite application program can cause another program to be executed on the host.

A key requirement of the communication facility is that it have a high degree of availability. Ideally, it would be available always. It must also be responsive in terms of human reaction time and convenience (but not necessarily in real time—after all, if the host computer had a real-time capability, there might be no need for the satellites and the distributed system). Therefore it is necessary to focus on providing function and capability in a facility whose resource requirements are small, so that it can be available for long periods.

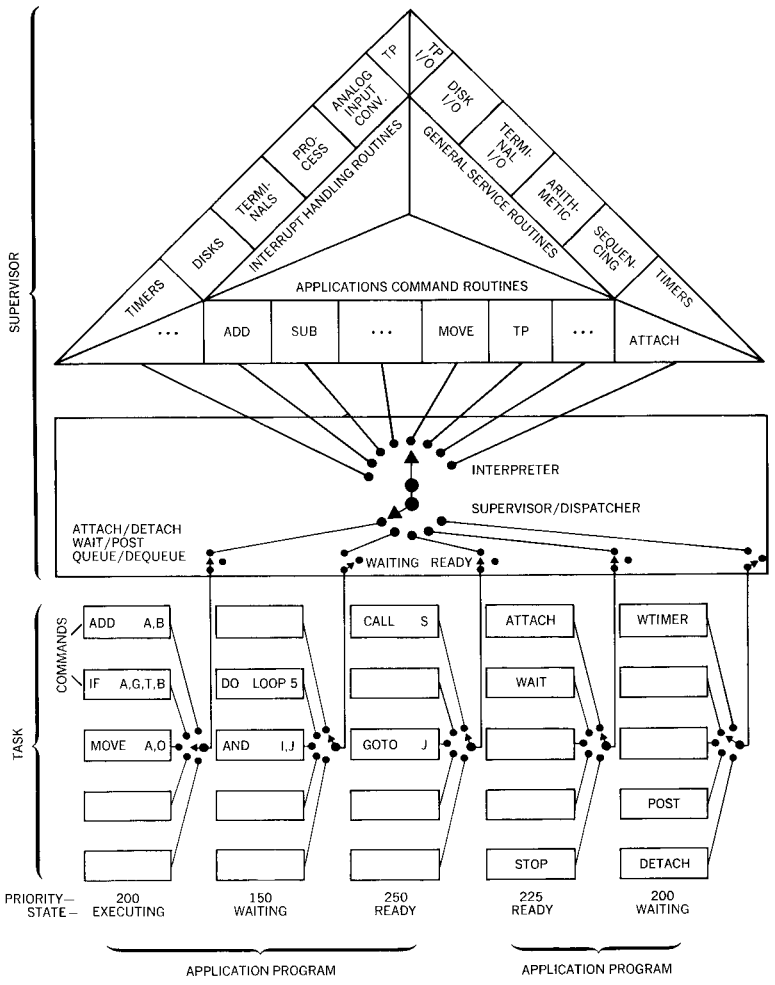
Host Communication Facility

Such a facility, called the Host Communication Facility (HCF), has been developed for the LABS/7 environment. HCF is composed of two blocks of code: a program that executes as a system task on the host computer, and matching functional modules in the LABS/7 supervisor. The interface between these two blocks of code is a well defined set of software protocols that are not dependent on particular satellite hardware, such as the various interfaces available for System/7. In the following discussion, the term HCF refers to the functions and characteristics of the host portion of the program.

For an application program running on a System/7, HCF provides access to the host data base, including sequential read or write access to any on-line, cataloged, direct access data set. The data set can be either sequential or a member of a partitioned data set, and it can have blocked or unblocked records with fixed or variable lengths. The only record size restriction is the amount of main storage an installation is willing to allocate to HCF for buffer areas. The user can create and delete data sets at will without affecting the running HCF program, since data sets are located through the catalog facility and then only when referenced by the user. The main storage requirements are minimal because each satellite system uses only one set of control blocks, and those are reused with each data set reference.

As currently constituted, this program requires approximately 34K bytes of storage for all executable instructions, including device I/O and data management routines. Approximately 1.5K bytes are required for each attached System/7. The remaining required storage is used as I/O buffer space and can be adjusted to suit installation requirements. At our installation, 82K bytes provide service for 11 System/7s, with 32K bytes devoted to I/O buffers.

Figure 2 Overview of the LABS/7 software on the IBM System/7



Data processing can be initiated on the host by submitting JCL (Job Control Language) statements into the batch job stream. In effect, the satellite becomes a remote job entry station. In the implementation to date, jobs have been submitted in either of two ways: in HASP[9] systems, by "punching" JCL to a HASP "pseudo-device," or, in systems without HASP, by the conventional OS START READER function.

HCF includes additional functions. Through a common data set, an interprocessor communication mechanism allows communication between programs on different machines (e.g., System/7 and System/360 or System/7 and System/7), making it unnecessary to make special modifications to the host control program. HCF also has the ability to fetch time and date information from the host computer, and it can query the host about the sta-

tus of a job that has been submitted. Although the user could handle these functions himself, using bidirectional data transfer and batch job submission, they are included in HCF to increase the convenience and responsiveness of the system.

System/7 hardware

For readers who are unfamiliar with the functional characteristics of System/7, the following summary is provided:

System/7 is an event-driven computer with 2K to 64K words of storage (1 word = 16 bits) and a 400-nanosecond cycle time. Interfaces provide analog and digital I/O and priority interrupt signals. There are four hardware priority levels. The system switches from one level to the next (requiring two cycles) whenever a priority interrupt occurs at a level higher than the one on which the processor is currently executing. For each priority level there are seven index registers, an instruction address register, and an accumulator. An operator station provides for keyboard entry, printed output, and paper tape input and output. The processor has two interval timers—separately controlled binary counters which are decremented every 50 microseconds.

System/7 can be coupled to another computer in any of four ways: via start/stop telecommunication facilities at speeds ranging up to 50,000 baud, via coaxial cable attached to a Sensor Based Control Unit on a System/370 at speeds of up to 277,000 bytes per second,[10,11] via binary synchronous telecommunication facilities, or via a data channel. Direct access storage is provided by one to 16 disks, each with a capacity of 1,228,000 words. For further details, see the *System Summary*.[12]

System/7 software

LABS/7 supervisor

Support software has been developed for System/7 to meet the requirements reviewed above. The LABS/7 supervisor is designed to provide multiprogramming, the main objective being to support multiple, independent, time-dependent processes with minimal interaction in such a manner that each application can be designed and programmed without reference to other application programs. The LABS/7 supervisor contains a multitasking supervisor, an interpreter, direct access support, communication support, and multiple console support. These components are the heart of the system, and many additional facilities have been developed from them. The most important are source program preparation, program debugging, system error recovery, FORTRAN support, and system activity measurement.

Figure 2 provides an overview of the system, including two user programs. The supervisor includes interrupt handling routines,

general system services, and the application instruction interface. The supervisor dispatcher allocates the CPU to the highest-priority ready task—the left-hand task in the figure. The interpreter links the user instruction to the appropriate system routine. Each task is in one of three states: executing, ready, or waiting. The state of the task depends on its priority and the occurrence of external events such as process interrupts, timer completions, and I/O completions.

The supervisor manages resources such as CPU time, storage, I/O devices, timers, and process interrupts. The elementary units of work for the supervisor are instructions, which are combined to form larger units of work called tasks. Each task is assigned a priority, which is a measure of the relative criticality (usually time dependence) of its function. Priorities are used by the supervisor to allocate execution time. The number of tasks is limited only by storage requirements. An application program consists of one or more tasks related by shared variables. The System/7 hardware interrupt levels can be used also for determining priorities. The convention in our installation is that all time-critical tasks are assigned to the second lowest interrupt level, and that noncritical tasks are assigned to the lowest level, with the dispatcher exercising control within each level. The two highest interrupt levels are used for hardware device and timer control.

The supervisor manages storage in the System/7 dynamically. The resident supervisor code, or nucleus, occupies 3000 to 6000 words of storage, depending on the optional modules included and on the size of the sensor I/O interface. The remaining storage is allocated to application programs in contiguous blocks when the programs are loaded. When the programs terminate, the storage again becomes available. An application program can be assigned to any available storage.

LABS/7 can control multiple processes concurrently. The number of processes or applications that can run concurrently is a function of the availability of and aggregate requirements for CPU storage, sensor I/O points, and CPU time. It takes about 30 microseconds for the supervisor to switch control from one user task to another with a higher priority. In special applications requiring very high data rates, however, normal supervisor routines can be bypassed easily, so that response time is limited only by the System/7 hardware. The maximum data rate that one can expect to sustain with LABS/7 in a time-shared environment is 50,000 I/O operations a second.

The supervisor also provides routines that support all sensor I/O devices as well as interval timing, process interrupt functions, and many commonly used arithmetic and logical functions.

Table 1 Examples of real-time commands

Category	*Command name*	*Function*
System configuration	SYSTEM7	Defines the size and I/O configuration of the System/7 hardware
	IODEF	Relates a symbolic sensor I/O address to System/7 hardware addresses
Task control	ATTACH	Defines and starts a new task within a program
	WAIT	Waits for completion of a named event
	QUEUE	Enqueues on a named (serially reusable) resource, such as the printer
Program flow	CALLFORT	Calls a FORTRAN subroutine and passes parameters
	GOTO	Unconditional or calculated branch
Timing control	INTIME	Returns the time elapsed, to one millisecond, since the last execution of INTIME
	WTIMER	Waits until a previously set time interval expires
Data definition	BUFFER	Defines a buffer and a pointer, which is automatically indexed when certain I/O commands are transmitted to or from the buffer
Data manipulation	ADD	Adds a single-precision constant or vector to a single- or double-precision constant or vector
	CONVERT	Translates ASCII to EBCDIC characters or vice versa
	ADDINDEX	Increments the contents of an index
Teleprocessing	TP OPENOUT	Initializes communication with a host processor data set
	TP SUBMIT	Submits a job to the host processor job stream
Console support	YESNO	Transmits a query to a console, and branches if the answer is not yes
	WRITE	Enqueues for service a table of console output commands (when the console is free)

LABS/7 language interpreter

The command language for application program development is intended to be sophisticated enough to make the writing of application programs relatively easy, yet elemental enough so the application programmer will have as much flexibility and control of machine resources as possible. The command language provides considerable freedom in application design, and it allows for variation in operation sequences, simple real-time calculations, real-time decision making, and calls to subroutines. It is recognized that all possible user requirements will not be covered by

the currently available LABS/7 instructions, so users can follow certain conventions to code, in assembler language, their own operations as "user exit routines."

There are 70 instructions in the present implementation. Some examples are shown in Table 1, with brief explanations.

Source programs are written using the LABS/7 instruction set and are assembled on the host computer. The resulting load module is a table in which each source instruction, with its operands, is represented by a few machine words. The supervisor contains an interpreter which analyzes each instruction and provides linkage to the system subroutine required to execute the function called for, and provides the mechanism for stepping through each program instruction. Since each instruction contains only a reference to the required system subroutine, user programs remain small. The overhead for interpretation of each LABS/7 instruction is 10 to 15 microseconds.

Following the execution or interpretation of each LABS/7 instruction, the supervisor examines the queue of ready tasks. If a higher priority task has become ready, it will be given control of the CPU. An application program can have more than one associated task. These tasks run independently, although they can communicate with each other via common storage locations or event control blocks.

LABS/7 instructions are designed to be similar in appearance to FORTRAN wherever possible while maintaining the flexibility inherent in assembler language. Many instructions have vector operands with automatic indexing. For example, one may add, subtract, multiply, or divide two vectors with a single instruction. In event-driven I/O instructions and in disk I/O, the user refers to the specific hardware address by symbolic name. This feature makes application programs somewhat hardware independent and considerably easier to code. Timing instructions are provided where repetitive and reproducible timing sequences are required. The minimum and maximum time intervals that can be requested are one millisecond and 60,000 milliseconds (one minute). Application programs are stored in 128-word records, referred to as pages. Data acquisition programs are typically five to ten pages in length, including data storage areas.

direct access support

Direct access support is included for programs and data. Programs are stored in a library from which they can be loaded by means of a command from a console or loaded and started from another program by means of a LABS/7 instruction.

Data are stored in 128-word records in direct access data sets, which are referenced by symbolic name. Up to nine data sets

can be defined for each application program. Utility programs are provided for both user and system housekeeping functions. These functions include defining new or deleting old data sets, initializing and compressing the program and data libraries, and transmitting programs and data between host and satellite. LABS/7 provides instructions for reading and writing data in a sequential manner or randomly by relative record number, and these modes can be intermixed.

host communications support

For communication with the host computer, LABS/7 has facilities for bidirectional transmission of data using start/stop or high-speed communication hardware (the Sensor Based Control Adapter and Sensor Based Control Unit).[10,11]

LABS/7 instructions for bidirectional host communication are provided for input or output at the record or buffer level. Utility programs are provided for transfer of data sets to and from the host. The user need not be concerned with the line protocol required. Data transfer rates depend on several factors; rates in excess of 100,000 (16-bit) words per second are attainable. Programs generally are transferred in considerably less than a second, and even large data sets take only a few seconds.

Primarily, programs are transferred *from* the host, and data sets *to* the host. But data can also be transferred from the host, and jobs can be submitted for execution in the host. Also, small amounts of data can be transferred directly between programs that are active on both machines simultaneously. This facility, in conjunction with the interactive terminal support on the host (TSO and APL), has been used to explore new techniques for computer aided scientific work.

multiple console support

Several LABS/7 instructions are used for communication between the experimentalist and his application program or the system itself. The standard System/7 configuration includes a teletypewriter operator station. LABS/7 provides for additional alphanumeric I/O devices, however, because when several experiments share the same processor and are remote from it, it is desirable that each have its own terminal. The terminal to be used by a program is assigned dynamically by the system. The assigned terminal is the one used initially to invoke the program, and it can be changed from one invocation to the next without program change.

Various kinds of terminals have been used with LABS/7. Most desirable are display terminals, which are connected to the digital I/O interface of the System/7. As many as eight can share one DI/DO pair. The LABS/7 console support enables any device with an alphanumeric keyboard and display to be substituted with a modification only to the basic device level I/O code.

source program preparation support

The application program library of the satellite System/7 includes an interactive text editing facility which allows the user to develop application source code from a terminal. This facility is patterned after, and provides a subset of, the facilities of TSO, the interactive terminal support on the host.[13] Commands are provided for transferring source program modules to and from the host, job submission to the host, and job status inquiry. A full set of text editing subcommands permits interactive program preparation, giving the user an alternative to a terminal system on the host.

program debugging facilities

LABS/7 has extensive program debugging facilities. During program assembly all commands are checked extensively for syntax, accuracy, and consistency of use. For example, one user is prevented from accidentally referencing another user's sensor I/O addresses, and all references to I/O are checked against the explicit system configuration. Once the user's program is apparently error free, it is formatted for transfer to the System/7. This step provides further program validation, insuring that proper procedures and conventions have been followed.

These debugging facilities, which are interactive, provide a large measure of protection and isolation for production applications when new applications are being developed. Once a program has been stored in the System/7 program library, the LABS/7 execution debugging facilities are available to the user. Program flow can be traced, data areas dumped, programs suspended, interrupts simulated, and data and program statements changed, all while productive applications continue to run.

error recovery procedures

There are error recovery procedures for all system I/O services. In the event of a nonrecoverable error, a message is printed at the central operator station to aid in explicit problem determination. When a user's program terminates abnormally, a system message notes the key items in the user's program. Programs can be cancelled at any time from any operating console attached to the system.

FORTRAN support

One or more FORTRAN programs or subroutines can be included in a LABS/7 application program, and FORTRAN programs can call LABS/7 subroutines—for example to invoke sensor I/O commands. All features of the IBM System/7 FORTRAN IV language are available, with a few minor exceptions (e.g., PAUSE, REWIND), including formatted and unformatted input to or output from direct access files or any operator console.[14]

system activity measurement

An optional System Activity Measurement Facility is available with LABS/7 to aid in measuring the use of the System/7. This feature logs the daily usage of the CPU, storage, host communication facility, and sensor based hardware, and it provides reli-

ability data as well. A utility program prints either summary or detailed daily reports on the collected system data, which can be used to aid in understanding the operating characteristics of a machine or to compare the usage of two or more machines. This kind of information can be helpful in making decisions on new hardware requirements or selecting a machine on which to install a new application.

Timing and responsiveness

In the discussion above, we have attempted to describe the major functional features of LABS/7. Many details of the system have been omitted. For example, we have minimized reference to timing because the precise operating characteristics of the system are dependent on many variables. Optimal responsiveness and reliability are the primary goals in the design and programming of the system. It must be recognized, however, that if precise, unbending limits are placed on the parameters of a real-time system, the generality of the system will be restricted. A guaranteed response time to an external interrupt can be provided only by defining the exact characteristics of all processes operating on the system.

In LABS/7 the generality of the system is limited only by the hardware, and system services are performed in the most efficient manner consistent with the stated goals. An installation can tailor a set of applications so that timing requirements will be met when predetermined constraints are met. Where precise requirements exist, an installation can impose restrictions upon its users, but the restrictions are not built into the system.

software limits

The execution time for a single non-I/O LABS/7 instruction, including interpretation, ranges from 13 to 150 microseconds; the median is 25 microseconds. Assuming that all tasks are in a wait state, a task can be made ready (posted) in 30 to 40 microseconds after the receipt of an external interrupt. A task can be put into the wait state in approximately 30 microseconds. Similarly, a ready task can be activated (put into execution) in 30 microseconds. Thus, using the LABS/7 instruction WAIT, about 100 microseconds are required to execute the instruction, service the interrupt, post the task, and dispatch the task.

If a task is executing and an external interrupt occurs which makes ready a task of higher priority on the same hardware interrupt level, the maximum time required to activate the higher priority task after the interrupt is approximately 200 microseconds. The average time should be much less, with a minimum of 60 microseconds.

On ease of use

interactive orientation

The LABS/7 system attempts to separate system functions from application functions, allowing users to readily learn the procedures required to develop an application, concentrating solely on the application. Because the LABS/7 supervisor and utility programs are designed for interactive use, only three steps are required for most users to initiate service at the application level:

- The terminal's *attention* or *request* key is depressed to get the system's attention.
- When the system responds by printing the character ">", the characters "$L" are entered to indicate that the user is ready to load a program.
- When the system responds with a prompting message ("PGM NAME = "), the user enters the name of the program he wishes to load.

The several utility programs are designed to assist the user interactively in defining the work to be done, so that he does not have to enter a complicated string of commands. For example, assume that a user has loaded the utility program $DISKUT1 by following the procedure outlined above, and he wishes to allocate 100 sectors of direct access storage for a data set to be called MYDATA. Once loaded, the program would prompt him with a "COMMAND (??):" message. The user would respond with the command "AL" to request allocation of a new data set. The ensuing terminal sequence would be as follows, with prompting messages in lightface type for purposes of this example, and the user's responses in boldface:

```
COMMAND (??): AL
DS NAME = MYDATA
NO. SECTORS = 100
MYDATA CREATED
```

The terminal would then request the next command by repeating the "COMMAND (??):" prompting message. The "??" in the message is to remind the user that if he forgets the code for a command, he can enter "??" to request a display of the defined functions and their command codes.

symbolic I/O references

Because symbolic I/O references are incorporated in LABS/7, the application programmer does not have to think in hardware terms when referencing event-driven I/O functions. Assuming that I/O points had been assigned, wires connected, and the assignments stored in a data set, the application programmer would simply refer to, say, the first Digital Output (DO1), or the

third Analog Input (AI3), or the second Process Interrupt signal (PI2), and the assembler would automatically bind the references to physical addresses. This facility not only makes I/O specification simple, it makes application programs easily transferable from one system to another without requiring source statement modifications.

Data sets are also referred to symbolically (as DS1, DS2, etc.) within application program source statements. They can be named fully in the header of the program, or the naming can be deferred until the program is loaded. This feature makes programs more flexible and more easily transferable from one system to another.

a simple powerful command language

The simplicity of the LABS/7 command language is illustrated here by means of a brief example. Assume that upon receipt of a start signal, 100 digital readings are taken from a scanning device. Each reading must be preceded by the setting of a digital latch to initiate a digital readout. Because a single reading is subject to noise, it is necessary to repeat the scan of 100 readings 50 times and average the results. The following LABS/7 statements illustrate how these steps can be accomplished:

```
                .
                .
                .
                WAIT        PI1                 WAIT FOR START SIGNAL
                DO          AVGLOOP, 50         BEGIN AVERAGING LOOP
                DO          SCANLOOP, 100       BEGIN DIGITAL SCAN LOOP
                SBIO        DO1                 SET DIGITAL LATCH
                SBIO        DI1, BUFR, INDEX
SCANLOOP        CONTINUE                        END OF SCANNING LOOP
*
*   ADD 100 READINGS FROM 1 SCAN INTO AVERAGING BUFFER
                ADD         AVGBUFR, BUFR, 100, PREC = D
                MOVE        I, 0                REST BUFFER INDEX
AVGLOOP         CONTINUE                        END OF AVERAGING LOOP
*
*   DIVIDE DATA FROM 50 SCANS TO GET AVERAGE
                DIVIDE      AVGBUFR, 50, 100, BUFR, PREC = D
                .
                .
BUFR            BUFFER      100, INDEX = I
AVGBUFR         BUFFER      200
                .
                .
```

The start signal is associated with the process interrupt PI1. When the interrupt occurs, the program starts the data collection process. An outside loop (AVGLOOP) of length 50 is required to scan the digital data 50 times. An inside loop (SCANLOOP) of length 100 is used to read each digital input point. The read latch is set with an SBIO (sensor based input/output) digital output (DO1) command, and the data are read with an SBIO digi-

tal input (DI1) command. The data are read into the buffer (BUFR) using the automatic indexing feature, which puts each successive reading into the next available position of the buffer. After 100 readings, the data are added into a double-precision averaging buffer. One ADD statement adds all 100 single-precision values in BUFR to the 100 double-precision values in AVGBUFR. The program then proceeds to scan the data again, continuing this process until 50 scans are complete. The accumulated data are then averaged by dividing each double-precision total by 50. The DIVIDE statement divides the entire double-precision vector AVGBUFR by 50 and stores the results back in the data collection buffer, BUFR.

This sequence, of course, represents only part of a total application. The data collected can be processed on the System/7 to produce a report, or can be sent to a host computer. To send the data to a host, only the following additional steps are required:

```
         TP      OPENOUT, DSNAME     OPEN HOST DATA SET
         TP      WRITE, BUFR         WRITE DATA TO HOST
         TP      CLOSE               CLOSE HOST DATA SET
          .
          .
          .
DSNAME   TEXT    'SYS7. TESTDATA'    NAME OF HOST DATA SET
```

The data set named SYS7.TESTDATA is opened for output, the data stored in BUFR are written to the host, and the data set is closed.

simplified host programming

Because the use of a host computer, and therefore JCL, is both necessary and desirable in the LABS/7 environment, several procedures have been established to make the process easier for the user. The most frequently used host function is program preparation, including the assembling and compilation of program source statements, possibly linkage editing of object modules, and final formatting to produce a load module ready for transfer to a System/7. The simplest program preparation phase is the assemble/format process, which ordinarily would require at least 20 JCL statements. But through an OS Cataloged Procedure in LABS/7, this process has been reduced, from the user standpoint, to two JCL statements and four symbolic parameters. The TSO terminal facility is available at our installation, and we have taken advantage of it to give the user an interactive facility for JCL development.

An example of a JCL development procedure is given below. The user has created a program in source form and desires to inject it into the OS/360 batch job stream for the program preparation phase. System queries and responses are shown in upper case, and user responses in lower case.

```
labs7
ENTER 'SYS7ID'
k02
ENTER PROGRAM NAME
myprog
ASSEMBLE RWM4968.MYPROG.ASM? (CR = 'YES' or 'NEWNAME')

IODEF = MYPROG? (CR = 'YES', OR 'NONE', OR 'NEWNAME')
none
ENTER NAME FOR OUTPUT PROGRAM IF DIFFERENT FROM 'MYPROG' OR CR

THE DEFAULT JOB CLASS IS 'O'. ENTER NEW CLASS LETTER OR CR

DO YOU WISH TO SUBMIT THIS JOB NOW? (CR = 'YES' or 'NO')
yes

ASSEMBLE(RWM4968.MYPROG.ASM) SYS7ID(K02) IODEF(NONE) FORMAT(MYPROG)
JOBCLASS(O) SUBMIT(YES) KEEPJCL(YES)

ARE ALL PARAMETERS CORRECT? (CR = 'YES' OR 'NO')

MYPROG.CNTL
SAVED
JOB MYPROG SUBMITTED
AT 14.53.40 ON 75.094
THE JCL WAS KEPT
READY
```

Experience

Although LABS/7 originally was intended for laboratory automation support, it has proved effective in many environments. Eight IBM plants currently are using LABS/7 for a variety of applications, including research, manufacturing assembly test, engineering development, quality control, and process control. Approximately 40 System/7s at these installations are attached to host systems ranging from a System/370 Model 145 to a System/360 Model 195. Some of the applications are discussed below as an indication of what can be achieved.

In an analytical chemistry laboratory, one System/7 supports four gel-permeation chromatographs, a vapor-phase chromatograph, two differential scanning calorimeters, a nuclear magnetic resonance spectrometer, and a polymer light scattering experiment. The System/7 is used primarily for data acquisition and instrument control because previously prepared data reduction programs were most easily converted to run on the host computer. Therefore the computational load on this particular System/7 is light, so a relatively large number of instruments can be supported.

In contrast is an application in which photoconductive properties of various materials are being evaluated for office copying machines. This application has been described previously.[15] Up to 15 samples are measured simultaneously in a one- or two-minute performance test. Voltages on four independent electrometers are measured at rates of up to 100 points per second each, and each measure is performed by a separate LABS/7 task. Programs for graphic output, for reduction of voltage measurements to photosensitivity curves, and for comparison of different samples have been segmented to run with 3K bytes of storage on the System/7. The data can be reduced, displayed, and manipulated rapidly, regardless of the availability of the host.

Another application concerns two phases of the production of masks for the photolithography of integrated circuits. A program on the host computer first converts an encoded description of a mask into a two-dimensional map described by over 10^5 numbers, then the map pattern is transmitted to a System/7. There it is displayed on a storage oscilloscope and inspected. After the mask pattern is verified, it is retransmitted from the host to the System/7 in the form of commands to drive a programmed light table, which reproduces the pattern by moving a photographic plate under a point light source and exposing each selected region through one of a set of apertures. Since the exposure process has no critical timing constraints, it is run at low priority so the System/7 can be shared with a crystal grower and an electron spectroscopy instrument for chemical analysis. Since a disruption in any of these applications would be a serious inconvenience, none of them requires communication with the host while executing.

Conclusions

This paper describes a system that demonstrates the feasibility and practicality of a hierarchical set of computers for real-time event-driven applications. The system currently is operational in a wide variety of environments and has been generally well received. When new functions are required, they have proved easy to include.

LABS/7 demonstrates an effective distribution of function between a large central host facility, which is able to provide a range of function and power not economically available in a small computer, and a local satellite, which is able to provide a degree of responsiveness and stability that is difficult to achieve in a central installation. The communications link between the host and the satellite can be said to be loosely coupled in that each system is relatively insensitive to unavailability of the other. In LABS/7, the satellite generally acts as master relative to the host.

Further, LABS/7 demonstrates an effective way of sharing a process control computer among several independent applications. User isolation has proved to be good, and resource conflicts among applications have proved to be few and easy to resolve.

ACKNOWLEDGMENTS

The authors wish to acknowledge R. Aylsworth, N. Hien, S. Kupka, and D. Thompson, who have made significant direct contributions to LABS/7. We would also like to thank those users whose suggestions and requests have led to improvements and increased reliability. Finally, we express our appreciation to R. H. Kay for his continued support of the project.

CITED REFERENCES

1. See, for example, S. P. Perone, "Computer applications in the chemistry laboratory—a survey," *Analytical Chemistry* **43,** 10, 1288–1299 (August 1971).
2. *Laboratory Applications Based System—Program Description/Operations Manual,* IBM Systems Library, order number SH20-1363.
3. *Laboratory Applications Based Communications Utilities—Program Description/Operations Manual,* IBM Systems Library, order number SH20-1364.
4. H. Cole, "System/7 in a hierarchical laboratory automation system," *IBM Systems Journal* **13,** 4, 307–324 (1974).
5. H. Hultzsch, A. A. Guido, and H. Cole, *Laboratory Automation in a Novel Computer Hierarchy,* IBM Research Report RC 4714 (1974).
6. H. M. Gladney, "A simple time-sharing monitor system for laboratory automation," *Journal of Computational Physics* **2,** 3, 255–273 (February 1968).
7. D. L. Raimondi, *Multiprogramming Monitor for Laboratory Automation,* IBM Research Report RJ 1075 (1972).
8. R. H. Kay, H. D. Plötzeneder, and R. J. Gritter, "Cost effectiveness of computerized laboratory automation." *IEEE Proceedings* **63,** 10, 1495–1502 (October 1975).
9. *OS/VS2 HASP II Version 4—Systems Programmer's Guide,* IBM Systems Library, order number GC27-6992.
10. *IBM System/7 Sensor-Based Control Adapter (SBCA)—General Information Manual,* IBM Systems Library, order number GA34-1512.
11. *IBM System/7 Sensor-Based Control Unit (SBCU)—Planning Guide,* IBM Systems Library, order number GC34-1522.
12. *IBM System/7—System Summary,* IBM Systems Library, order number GA34-0002.
13. *IBM System/360 Operating System: Time Sharing Option Command Language Reference,* IBM Systems Library, order number GC28-6732.
14. *IBM System/7 FORTRAN IV Language,* IBM Systems Library, order number GC28-6876.
15. B. H. Schechtman, S. S. So, and E. W. Luttman, "Interactive use of a time-shared process control computer in electrophotographic sensitometry," *IBM Journal of Research and Development* **17,** 6, 500–508 (November 1973).

Reprint Order No. G321-5028