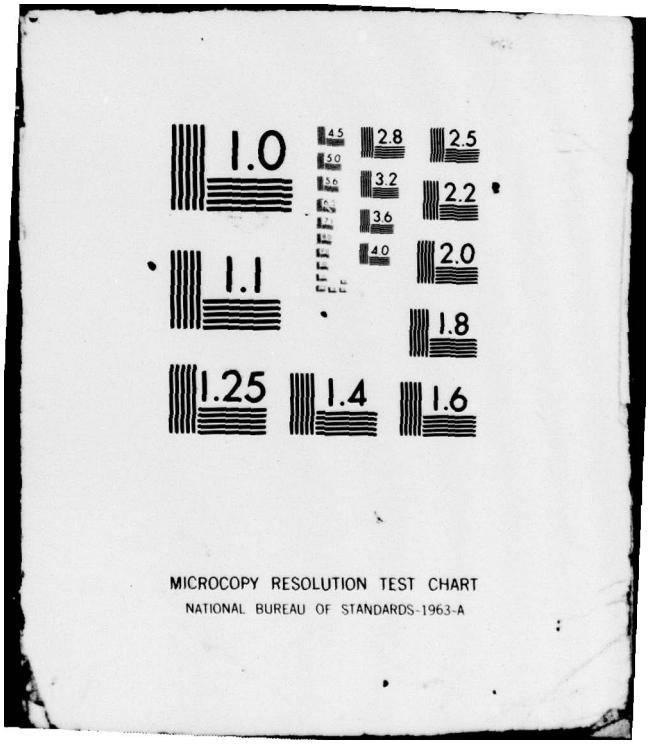
AD-A074 704 SCHOOL OF AEROSPACE MEDICINE BROOKS AFB TX F/G 9/2
WIZARD: A STRING-ORIENTED MICROPROCESSOR OPERATING SYSTEM.(U)
AUG 79 S D WILSON, M E WOMBLE, M L TWOREK
UNCLASSIFIED SAM-TR-79-14 NL

END
DATE FILMED
6-81
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Report SAM-TR-79-14

# WIZARD: A STRING-ORIENTED MICROPROCESSOR OPERATING SYSTEM

Shelburne D. Wilson, Major, USAF, MC
M. Edward Womble, Ph.D.
Michael L. Tworek, Captain, USAF

August 1979

Final Report for Period October 1976 - September 1978

Approved for public release; distribution unlimited.

USAF SCHOOL OF AEROSPACE MEDICINE
Aerospace Medical Division (AFSC)
Brooks Air Force Base, Texas 78235

## NOTICES

This final report was submitted by personnel of the Clinical Sciences Division, USAF School of Aerospace Medicine, Aerospace Medical Division, AFSC, Brooks Air Force Base, Texas, under job order 7755-20-10.

When U.S. Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

M. Edward Womble

M. EDWARD WOMBLE, Ph.D.
Project Scientist

Clarence F. Watson Jr.

CLARENCE F. WATSON, JR., Colonel, USAF, MC
Supervisor

L J Enders

LAWRENCE J. ENDERS
Colonel, USAF, MC
Commander

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

| | |
|---|---|
| 1. REPORT NUMBER: SAM-TR-79-14 | 2. GOVT ACCESSION NO. |
| 3. RECIPIENT'S CATALOG NUMBER | |
| 4. TITLE *(and Subtitle)*: WIZARD: A STRING-ORIENTED MICROPROCESSOR OPERATING SYSTEM | 5. TYPE OF REPORT & PERIOD COVERED: Final report, Oct 76 — Sep 78 |
| 6. PERFORMING ORG. REPORT NUMBER | |
| 7. AUTHOR(s): Shelburne D. Wilson, Major, USAF, MC; M. Edward Womble, Ph.D.; Michael L. Tworek, Captain, USAF | 8. CONTRACT OR GRANT NUMBER(s): N/A |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: USAF School of Aerospace Medicine (NGC), Aerospace Medical Division (AFSC), Brooks Air Force Base, Texas 78235 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62202F, 7755-20-10 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: USAF School of Aerospace Medicine (NGC), Aerospace Medical Division (AFSC), Brooks Air Force Base, Texas 78235 | 12. REPORT DATE: Aug 79 |
| | 13. NUMBER OF PAGES: 11 |
| 14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* | 15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

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

Microcomputers, Operating systems

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

A string-oriented operating system for Intel-8080-based microcomputers is described. The system consists of a hierarchy of virtual machines. The lowest level virtual machines extend the instruction set of the 8080 to include additional 16-bit arithmetic and logical instructions, new data types, and operators. The data types include strings and string operators derived from the SNOBOL programming language. A table data type is constructed from strings, and table-manipulation operators are provided. A bit-map data type and associated

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

operators are also included. An Input/Output Control System (IOCS) supports device-independent IO to multiple devices and diskette files. File name aliases permit many logical IO streams to be dynamically mapped onto a restricted set of physical IO units. Pseudo device handlers expand the capabilities of IO devices and are transparent to application programs. Distributed command decoders interpret IO command strings. Once communication is established with a logical device, a low-overhead IO Vector mechanism may be used for further access. A keyboard monitor provides interactive debugging facilities to application programmers. System resource allocation is implementation dependent and is not embedded in the system nucleus. Multiple implementations over a range of system sizes have demonstrated the utility and adaptability of WIZARD.

# WIZARD: A STRING-ORIENTED MICROPROCESSOR OPERATING SYSTEM

## DESIGN PHILOSOPHY

Because of ever-increasing requests from various disciplines for computer assistance, we set out to design an operating system (given the name WIZARD) that would meet a variety of requirements. Recognizing that WIZARD should be constructed as a hierarchical series of independent abstractions as described by Dijkstra (1), we examined the design of several well-constructed operating systems (2-5). Typically, the nucleus of these systems supervised resource allocation, including memory and central processing unit (CPU) cycles. Our approach, however, needed to be somewhat different.

Our applications are implemented on highly dissimilar hardware configurations with radically different interrupt structures and do not have a requirement for dynamic memory allocation. Virtual memory and paging schemes are particularly alien to our usual environment. What we did require was a family of systems adapted to configurations from very small, with 4 to 6 kilobytes of memory and 1 or 2 input/output (IO) devices, to medium-scale microprocessor applications with 64 kilobytes of memory, random access files, and perhaps a dozen IO devices. Even the smallest of the applications required high-level operators and data types as well as flexible IO. Character string manipulation was particularly important. The design methodology chosen was patterned after Parnas' suggestions for the construction of software families designed for ease of expansion and contraction (6-10).

## OVERVIEW

In the first approximation, WIZARD may be considered to consist of a hierarchically arranged set of virtual machines (Fig. 1). At the lowest level of this hierarchy is the hardware. The currently implemented version of WIZARD operates on several microcomputer hardware configurations that use the Intel 8080 central processing unit (CPU), and is also compatible with other CPUs (such as the Intel 8085 or Zilog Z80) that execute the 8080 instruction set. Although the 8080 instruction set is adaptable to our applications, it does not include the data types and operations upon them required for significant string-processing applications.

At the time we began work on WIZARD, we were restricted to the use of assembly language and did not have a macro capability. Our development effort was further constrained by a deadline for delivering a

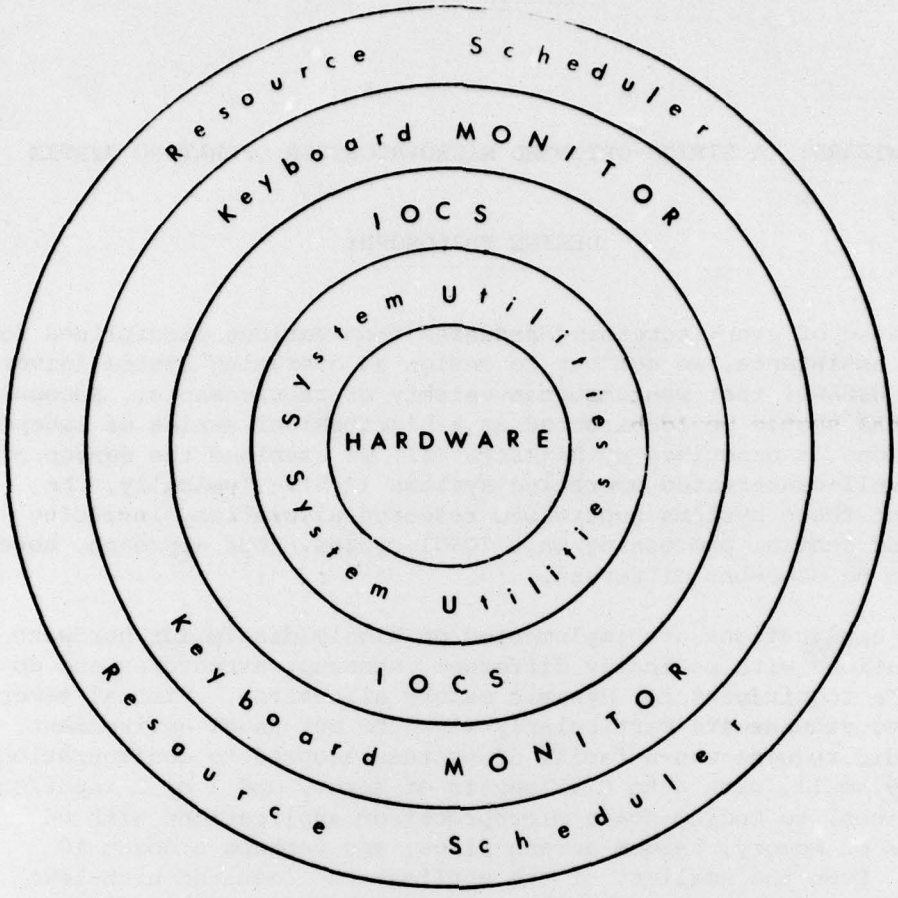



Figure 1. WIZARD overview.

high-visibility applications product. Our solution was to expand the capabilities of our system by constructing a series of virtual machines with characteristics more suited to our needs than the 8080 itself. The first of these virtual machines was implemented as hardware and a collection of system utility subroutines. This package extends the ability of the 8080 to perform arithmetic and logical operations upon 16-bit quantities. In addition, it implements three new data types and high-level primitives for their manipulation. These new resources are made available to both application programs and to the remainder of the system. These first virtual machines were then used to construct a higher level virtual machine incorporating a highly flexible input/output control system (IOCS) and a variety of other system functions. This triad (hardware, system utilities, and IOCS) is used by application programs to accomplish their tasks. A monitor program operating at the same hierarchical level as the application programs permits programmers and other sophisticated operators to direct the system from the console for debugging and for conventional computer operations. In our typical dedicated-application environment, the existence of this monitor is unknown to the

system operator. The final segment of the WIZARD system is the system scheduler which establishes the initial system environment, services interrupts, and allocates system resources. Each of these elements will now be discussed in somewhat more detail.

## FACILITIES TRIAD

### Hardware

In microprocessor systems, many elements of the hardware which in a conventional system would be constant across all implementations, are in fact highly variable. A particularly appropriate example is the interrupt system whose characteristics are largely determined not by the microprocessor CPU but rather by the various support chips used in a particular hardware implementation. These differences are accentuated by the varying interrupt-handling capabilities of different CPUs that utilize the same basic instruction set. One of the goals of WIZARD is to isolate both user programs and the WIZARD nucleus from implementation-specific characteristics of the hardware. WIZARD encourages programmers to use directly only those features of the hardware common to all implementations. It permits implementation-specific system software to deal with the variable elements of the hardware.

### System Utilities

The system utilities package, which occupies somewhat less than 2 kilobytes of read-only memory, substantially expands the set of primitive operations and data types of the 8080 CPU. Except for a few commonly used string patterns that are not executable code, the utility package consists of reentrant subroutines that share data representation and register usage conventions. The limited ability of the 8080 to manipulate 16-bit integer quantities is augmented with subtraction, multiplication, value testing, and shift and rotate instructions. Three new data types are implemented. These are strings, tables, and bit maps. A WIZARD string consists of zero or more characters followed by an end-of-record character. WIZARD uses 7-bit ASCII characters with the high-order bit set to Ø. User programs may augment the standard character set by utilizing characters with the high-order bit set to 1. End-of-record characters are the Carriage Return (CR), Escape (ESC), Unit Separator (US), and Record Separator (RS). Except during input/output operations, WIZARD software considers these four end-of-record characters to be equivalent. User programs may select different record terminators to define various record types. In input/output operations, typical WIZARD device handlers treat CR and ESC as being equivalent record terminators of one type, while US and RS are treated as equivalent record terminators of a second type. Output (or input echo) of CR or ESC causes execution of a carriage return and zero or more line feeds. The number of line feeds is a user-settable handler parameter with a default value of 1.

Output (or input echo) of US or RS terminates the input/output operation without causing carriage/cursor movement.

The use of multiple equivalent end-of-record characters greatly increases user program flexibility and allows for several different user-defined record types. Strings may be concatenated to construct higher order structures such as groups or files. Certain WIZARD routines utilize these higher order structures. A group is terminated by a Group Separator character (GS) immediately preceding an end-of-record character. An end-of-file mark is defined as a File Separator character (FS) immediately preceding an end-of-record character. In some contexts, strings are considered to be divided into fields separated by one or more break characters (space, comma, tab). This convention, closely approximating conventional English usage, lends itself to very natural man/machine communication. Although WIZARD routines normally manipulate strings in an unidirectional fashion from left to right, strings may be freely manipulated in a bidirectional fashion if an end-of-record character immediately precedes the first character of a record. The WIZARD utilities package contains a number of routines that implement string manipulation primitives. These functions are largely based on the primitives of the language SNOBOL (Fig. 2).

| | |
|---|---|
| ANY | SNOBOL function ANY |
| NOTANY | SNOBOL function NOTANY |
| SPAN | SNOBOL function SPAN |
| BREAK | SNOBOL function BREAK |
| MATCH | Perform character match (pattern terminated by EOR) |
| BRKMAT | Perform character match (pattern terminated by break character or EOR) |
| MOVSTR | Move string |
| LEN | SNOBOL function LEN |
| REM | SNOBOL function REM |
| NXTARG | Move string pointer to start of next field |
| HEXIT | Convert binary to ASCII hexadecimal character |
| DEHEX | Convert ASCII hexadecimal digit to binary |
| DEDEC | Convert ASCII decimal digit to binary |
| DEOCT | Convert ASCII octal digit to binary |
| GETHEX | Convert string of ASCII hexadecimal digits to binary |
| GETDEC | Convert string of ASCII decimal digits to binary |
| DETOCT | Convert string of ASCII octal digits to binary |
| TODEC | Convert 16-bit binary to 5 ASCII decimal digits (leading Ø's) |
| SPADEC | Convert 16-bit binary to 5 ASCII decimal digits (leading blanks) |
| COMDEC | Convert 16-bit binary to 1-5 ASCII decimal digits |

Figure 2. String operators.

The concept of a failure flag is also borrowed from SNOBOL. Since our base machine does not include a failure flag in its complement of registers, we chose for this purpose to use the CARRY flag (which the hardware does in fact use as a failure flag in arithmetic operations). Both system routines and user routines set the failure flag to indicate that they were unsuccessful in carrying out a requested function. Often a routine is called in such a fashion that failure may be expected. The failure flag is also used to indicate error conditions and other unanticipated program behavior. The use of this flag throughout the system has been invaluable in facilitating the creation of highly modularized systems. It permits a low-level module to convey an indication of anomalous behavior upwards, through several levels of program hierarchy, without requiring any routine to be cognizant of the characteristics of its callers.

In addition to the basic string-manipulation routines, a collection of numerical conversion routines exist to provide conversation between string representation of numeric quantities in any of three number systems and 16-bit binary values (with conversion in both directions).

The WIZARD operating system includes tables as a data type. A table is constructed as a string--composed of zero or more table entries and terminated by a Carriage Return. Tables are single keyed and unordered. CR indicates the end of a table, while the end of available table space can be indicated by GS. Table entries consist of fields separated by an augmented set of break characters--comma, space, tab, and the commercial @ sign. Each table entry is bounded at beginning and end by an exclamation point. Fields within a table entry can contain any ASCII character except the control characters 00H through 1FH and the three characters (!, =, and @) that have special significance within tables. Table-entry fields are of arbitrary length. A table entry can have only one key but any number of parameters. Parameter names are separated from their values by an equal sign (=). Address fields consist of four ASCII encoded hexadecimal characters preceded by an @. A table entry normally has a single address field; however, this convention is not enforced and under certain circumstances multiple address fields may be desirable. Operators are provided for inserting and deleting table entries, for looking up table entries based on their key values, for identifying and obtaining parameters within a table entry, and for generalized table housekeeping functions. The table data type is extensively used by the WIZARD system software.

The WIZARD operating system incorporates a bit-map data type. A bit map may contain up to 64K individual bit entries. Operations are available to set/reset/test any individual bit within a bit map as well as to initialize an entire bit map to either 1 or Ø.

## Input/Output Control System

The next level in the hierarchy of WIZARD virtual machines incorporates input/output functions. It is collectively referred to as the Input/Output Control System (IOCS), but it also incorporates a number of other system functions, such as error handling, which depend on an IO capability. All WIZARD IO is accomplished through logical devices which may be assigned to specific diskette files or physical IO devices either at system generation or at run time. Run-time assignments may be made by application programs or by the operator. IO devices and diskette files are logically equivalent; any subsequent reference to devices should be considered equally applicable to files.

The nature of microprocessor-based systems implies a close relationship between application programs and the physical characteristics of IO devices. With WIZARD, however, every effort was made to minimize application-program awareness of the characteristics of the devices controlled by the program. At the same time, IO device-handler conventions were established to facilitate the writing of new device handlers by relatively unskilled programmers. This was considered mandatory since many microprocessor applications are tied to unique IO hardware. All IO device handlers are required to accommodate 14 basic IO functions (Fig. 3).

| | |
|---|---|
| INIT | Hardware initiation |
| IOCOM (IO command) | Device-specific actions, parameter setting |
| OPEN | Software initiation |
| CLOSE | Software wrapup |
| CHRIN (character in) | Input 1 ASCII character |
| CHROUT (character out) | Output 1 ASCII character |
| STRIN (string in) | Input an ASCII character string |
| STROUT (string out) | Output an ASCII character string |
| BININ (byte in) | Input one 8-bit byte of binary information |
| BINOUT (byte out) | Output one 8-bit byte of binary information |
| BLKIN (block in) | Input a variable number of bytes |
| BLKOUT (block out) | Output a variable number of bytes |
| TEST | Test for operation completion |
| WAIT | Wait for operation completion |

Figure 3. Required handler function.

In many cases some of these functions may be inapplicable for a given device. In such a case the device handler must respond to the inappropriate request in a manner that will maintain system integrity. For example, a request for input from a line printer would result in an error message and the setting of the failure flag.

The INIT function is a basic hardware initialization required by many programmable IO support chips and by some external IO devices. The IOCOM function is used to direct a device to perform auxillary functions such as forms control or rewind, or to set device parameters such as line length or tab settings. Many of these IO commands are device specific. The OPEN and CLOSE functions are relatively conventional. The CHRIN and CHROUT and the STRIN and STROUT routines are used to communicate ASCII encoded character data between the program environment and the outside world. In our typical applications, these routines are used almost exclusively. The binary input/output routines provide both single-byte (BININ/BINOUT) and block (BLKIN/BLKOUT) transfer capability for 8-bit binary data. The TEST and WAIT functions provide IO synchronization in an interrupt-driven environment.

ASCII command strings are used for normal communications with the IOCS and for all IOCOM requests to device handlers. The WIZARD system incorporates a number of command decoders at each software level. All WIZARD command strings use a simple syntax which was chosen for maximum ease in parsing while retaining adequate power to express any required request. Parsing is done strictly from left to right, and a single command string may be passed sequentially through a hierarchy of command decoders. At each command-decoder level, the first field remaining in the command string must be a command name that can be used to vector to action routines, which in turn can continue parsing the remainder of the string. The action routine parsers can extract command parameters or can themselves be command decoders. This interpretive analysis of operating system requests by a hierarchy of command decoders has been extremely useful and in particular offers essentially unlimited ability to expand system functions. A significant penalty may be paid, of course, in the overhead required to parse the command strings. In practice this has not been significant and is greatly outweighed by the gain in system adaptability.

WIZARD provides two methods of access to input/output facilities. The more general of these is the Input/Output Control System call. This IOCS call passes a command string to the first level IOCS command decoder. Although the WIZARD IOCS call provides great flexibility in accessing IO devices and diskette files using logical file names, it does so at the expense of significant overhead. The overhead required to access an open device can be greatly reduced by using IO Vector calls to the basic input/output functions. Each of the basic IO functions has an associated IO Vector call. When any logical device is opened, it returns to the calling program its device-access code which may be used by the IO Vector routine to pass control back to the same logical-device handler. In so doing, the table manipulation and command decoding required to access a device handler through a standard IOCS call are bypassed. Thus, applications programmers are offered both the generality of access to IO facilities using logical file names and low overhead that would be encountered if access were via fixed-address handlers.

The WIZARD IOCS recognized three distinct types of logical file names, although the distinction is largely transparent to the application programmer. Names that refer directly to diskette files have the type parameter FC (File) and are serviced by the file-handler routine. Names that refer directly to device handlers have the type parameter D (Device) and are serviced by device handlers or pseudohandlers. Names that also refer directly to other file names have the type parameter A (Alias). Names for logical IO streams are normally aliases and may be assigned to device handlers or to other aliases. In many applications, several logical IO streams must be multiplexed onto a small number of physical devices. The use of aliases facilitates rerouting logical IO streams as configurations change, without affecting application programs.

Another feature that materially enhances the growth potential of WIZARD, and which strongly encourages separation of device dependencies from application programs, is the use of pseudohandlers. With the WIZARD device-handler interface, it is very practical to write routines that appear to the system to be device handlers but which in fact are data-transformation routines that communicate with other logical devices. As an example, in one of our operational systems, an application program communicates with a data-base management system on a remote host computer. The actual telecommunications IO from the microcomputer system is controlled by a communications device handler. This handler is aware of the characteristics of the telecommunications hardware to which it is connected, but is indifferent to the characteristics of a specific host. A pseudohandler, which is ignorant of the characteristics of the telecommunications hardware, is cognizant of the sign-on protocol and other characteristics of the specific host machine. The application program, which is ignorant of the characteristics of both the host system and the telecommunications equipment, communicates with the pseudohandler. The pseudohandler communicates with the host operating system, using the communications handler. Two very different telecommunications systems are used interchangeably by exchanging communications hardware and handler. A different pseudohandler would adapt the system to a different host. In both cases changes are completely localized to a single module. IO data may pass through several virtual IO devices before arriving at the real-world interface.

## IMPLEMENTATION

### Application Programs

The triad of hardware machine instructions, system utility extensions, and IOCS functions is used by the applications programmer in creating his software product. Current applications programs have been written in assembly language, and the programmer has been responsible for using the proper register conventions. An expanding macro library is reducing programmer responsibility for register loading. An interface program has been written to permit PLM programs to communicate with

WIZARD, and a compiler for a high-level language, designed to fully utilize the capabilities of WIZARD, is being constructed. Current applications include telecommunications interface to a data-base management system, local word processing, real-time electrocardiogram digitization, and an intelligent audiometer.

## Monitor

The monitor is at the same hierarchical level as the user (application) program and has a relationship with the program similar to that of a coroutine. A conventional microprocessor keyboard monitor is currently used with WIZARD. The primary function of the monitor is to permit sophisticated users, especially applications programmers, to control the system directly while testing and debugging their programs. In some conventional general-purpose computer applications, the monitor can be made accessible to users. Linkage conventions are provided so that any application program can pass system control to the monitor, with subsequent return of control to the application program. In our typical dedicated-application environment, however, users are not aware of the existence of the monitor, although a monitor-access facility (not discussed in documentation supplied to users) is normally included for the convenience of software-maintenance personnel. In limited-memory hardware configurations, the monitor software is stored in read-only memory on a separate diagnostic board which is inserted by maintenance personnel when needed.

## Scheduler

In most operating systems, scheduling of system resources is a primary activity. In WIZARD, input/output resources are allocated by the IOCS. Other resources are allocated either by the scheduling subsystem or by the applications program. This is rather unlike more typical operating systems in which resources are scheduled at a much lower level in the hierarchy of virtual machines. This removal of resource allocation from the central portion of the operating system in WIZARD is not accidental. It is a result of the range of environments to which WIZARD is adapted. Different microcomputer implementations, even though using the same central processing unit, may differ dramatically in such important areas as basic interrupt structure and memory management capability.

WIZARD was intended to be usable with an absolute minimum of modification and with essentially complete user-program transparency on hardware configurations ranging from small, dedicated, intelligent controllers to time-shared multiprogramming systems supporting multiple independent users. For WIZARD to be useful in the typical small-scale dedicated-application environment was considered to be particularly important. In this environment, there is usually a single application task, perhaps with concurrent interrupt-driven IO. Those portions of WIZARD subordinate

to the application program are adapted to operate in this environment. In our typical implementation, only a vestigial scheduler is required to initialize the system and transfer control to the single-application task. When interrupts are used, a very straightforward control routine may be written to service the implementation-dependent interrupt system and to transfer control to the IO driver routines that are unaffected by the details of the interrupt structure.

## Implementation with Experience

The implementation of WIZARD, from initial design to operational status, was accomplished in 18 calendar months and required approximately 10 man-months of effort. The initial WIZARD implementation consisted of 8K lines of assembly code (excluding comments). Subsequently, at least half a dozen variants have been made operational. These range from a minimal device controller, consisting of the system utility package and one IO handler on a single printed-circuit board, to general-purpose data processing systems. Adaptation of the software for a dedicated standalone application (retaining 60-80% of the total WIZARD code) requires 1 or 2 man-days for modification and checkout.

## CONCLUSION

WIZARD is a highly modularized, string-oriented microprocessor operating system which has been designed and implemented to facilitate the construction of application software for use at the man/machine interface. It is constructed as a sequence of nested virtual machines, isolated to the greatest extent possible from idiosyncrasies of the hardware environment, and usable within a broad range of applications. Operational experience with several applications has proven WIZARD to be a highly adaptable software package that meets all of its design goals.

## REFERENCES


1. Dijkstra, E. W. The structure of the "THE" multiprogramming system. CACM 11(5):341-346 (1967).

2. Hansen, P. Brinch. The nucleus of a multiprogramming system. CACM 13(4):238-250 (1970).

3. Liskov, B. H. The design of the Venus operating system. CACM 15(3):144-149 (1972).

4. Wulf, W., et al. HYDRA: The kernel of a multiprocessor operating system. CACM 17(6):337-345 (1974).

5. Parnas, D. L., et al. Design and specification of the minimal subset of an operating system family. IEEE Trans Software Eng SE-2(4):301-307 (1976).

6. Parnas, D. L. A technique for software module specification with examples. CACM 15(5):330-336 (1972).

7. Parnas, D. L. On the criteria to be used in decomposing systems into modules. CACM 15(12):1053-1058 (1972).

8. Parnas, D. L., and D. P. Siewiorek. Use of the concept of transparency in the design of hierarchically structured systems. CACM 18(7):401-408 (1975).

9. Parnas, D. L. On the design and development of program families. IEEE Trans Software Eng SE-2(1):1-9 (1976).

10. Parnas, D. L. Designing software for ease of extension and contraction. Proc 3d Int Conf Software Eng, Atlanta, Georgia: IEEE Catalog Number 78CH1317-7C, pp. 264-277, May 1978.