System Design of a Computer for Time Sharing Applications

E. L. Glaser
Massachusetts Institute of Technology
Cambridge, Massachusetts
and
J. F. Couleur and G. A. Oliver
General Electric Computer Division
Phoenix, Arizona

Introduction

In the late spring and early summer of 1964 it became obvious that greater facility in the computing system was required if time-sharing techniques were to move from the state of an interesting pilot experiment into that of a useful prototype for remote access computer systems. Investigation proved computers that were immediately available could not be adapted readily to meet the difficult set of requirements time-sharing places on any machine. However, there was one system that appeared to be extendible into what was desired. This machine was the General Electric 635. The 635 is a single address stored program computer with a word length of 36 bits. It possessed many of the characteristics that were deemed necessary for the application of a computer to time-sharing. The three most important characteristics are:

1. A clean and comprehensive order code,
2. a multiprocessor capability, and
3. nonsynchronous design.

The first of these requirements stems from the quantity of software to be written for the machine. The size of the operating system demands it be written in some higher level language. An orderly instruction set is essential to permit the use of good code selection algorithms in the compiler. The multiprocessor characteristic was desired to permit a feasible fail-soft characteristic in the system and to allow system growth without major increments equivalent to entire system duplication. The third characteristic, non-synchronous communication between major components, was deemed desirable because of the flexibility afforded in the significant modification needed to achieve the time-sharing system that we had in mind. The nonsynchronous characteristic allows a system to become large without suffering measurable degradation. These modifications in a fully synchronous system could result in degradation of performance. (Degradation of course will take place in any system if the delay time of a signal through a cable is a significant part of a basic operating time. However the nonsynchronous approach permits this time to be minimized.)

The Scope Of Extension

The design of the extensions to the 635 began in early May 1964 and the end result is what is now known as the **GE 645**. The changes are in several areas. First, a totally new I/O control unit has been designed to integrate the control of standard peripheral devices and various types of communications lines. The latter are necessary in the time-sharing environment. A large movable head disc, the DS-25, was available as a standard 635 peripheral. This unit appeared suited for the type of use we envisioned. A high-speed drum (DS-300) was also available, but its performance was not sufficient for the purposes of the highest speed secondary store in the projected timesharing system. Therefore, a new high-speed drum system was designed for this function (MS-32). The introduction of a new form of addressing logic incorporating segments and pages is a significant change to the system. This, with its concomitant changes in interrupt logic and related portions of the machine affected by it, was by far the major change to the system.

The Segments And Pages

The concept of the **paged memory** has appeared in the literature for the past several years and has been implemented on at least one machine[1,2] The purpose of paging is to make the allocation of physical memory easier. One wan think of paging as the intermediate ground between a fully associative memory, having each word addressed by means of some part of its contents, and a normal memory, having each memory location addressed by a specific integer forever fixed to that physical location. In paging, blocks of memory are assigned differing base addresses. Addressing within a block is relative to the beginning of the block. Thus if association and relative addressing are handled with a break occurring within a normal break of the word (viz. in a binary machine block size is a power of 2), then a number of noncontiguous blocks of memory can be made to look contiguous through proper association. The association between a block and a specific base address can be dynamically changed by program during the execution of appropriate parts of the executive routine.

Segments on the other hand are used not for the allocation of physical memory, but for the allocation of address space. The concept of segments has received little attention in the literature until recently.[3,4,5,6] A segment defines some object such as a data area, a procedure (program) or the like. In a sense, each segment corresponds to a virtual memory whose size is whatever size, up to a maximum limit, that is required. In theory, as many such segments can be available to a programmer as necessary. In the case of the 645, the practical limit is 2**18 segments, each one of which can obtain up to 2**18 words. Observe that although segments and pages are two distinctly different entities, they work together to facilitate the allocation of physical memory and virtual memory. Although a large number of segments may be defined, each one having a large number of words, only the currently referenced pages of pertinent segments need to be in memory at any time. A very limited concept of segments has been used in computers previously.[7] Recently other system designs have employed a similar segment and paging technique.[8]

Descriptors

A **descriptor** is a word that is used to define and locate in physical memory either a page or a segment. Hence there are two kinds of descriptors: page descriptors and segment descriptors. The difference between them is the table in which they are found. Each has slightly different functions as will become obvious.

A segment descriptor contains among other things the location of either the segment itself or, if the segment is paged, the location of the table in which its pages are defined. Each page descriptor corresponds to one of the pages of the segment. A page descriptor contains the location of the base of the block of memory in which this page is to be found. All of the page descriptors for a given segment must lie in contiguous locations of the page table for that segment.

Both segment and page descriptors also contain certain access control information known as the descriptor control field. These fields define the nature of access permitted to a particular piece of information. An example of such a control might be the Write Permit Bit. This bit determines whether this segment or this particular page of a segment can be written into or only read. Alternatively, we may think of the segment descriptor as defining a certain set of restrictions on accessing the entire segment. Specific page descriptors may add additional restrictions; however, they may not take any away. For example, the segment is defined as being a data segment with writing permitted. The result is that control can not be transferred to this segment, but the words of the segment can be used as data with either reading or writing possible. If for some reason there is one specific page of data that we wish to protect, that page can be marked as "Read Only" (the write permit bit is set to zero). As a consequence, this page is defined not only as data but it is now "Read Only" data. The page descriptor has applied an additional constraint above and beyond those constraints contained in the segment descriptor. In addition to the control and address information the segment descriptor contains a bounds field. This bounds field defines the number of pages that the segment contains. In the case of unpaged segments this bounds field defines the total number of words in the segment.

The Descriptor Segment And Base Registers

The segment descriptors associated with a given process are all contained in a single segment known as the descriptor segment. The descriptor segment has a distinguished role in the operation of the system in that the processor uses this segment as the sole means of relating program references to memory location. The descriptor segment may be paged in the same manner as any other segment. Its location in memory is defined by a special processor register known as the descriptor base register. This register defines either the base of an unpaged descriptor segment or the base of the segment page table of a paged descriptor segment. This register can only be loaded or stored by privileged instructions not available to slave mode programs. Note that all user programs and most of the executive system are written in slave mode.

A segment can now be identified by an ordinal number which locates its descriptor relative to the base of the descriptor segment. This number is known as the segment number. The address of a location in memory is specified in terms of a segment number and a location within that segment. In the 645 both quantities are expressed as 18 bit numbers. During all addressing in the 645 both parts are necessary except under very unusual circumstances. Both parts are usually supplied explicitly. In some cases they are implied by certain conditions of the machine.

There are several provisions for forming these two-part addresses. The first is by means of the instruction word. An address may refer to a location within the current procedure segment or alternatively to some other segment. A control field in the instruction specifies the choice. If the reference is within the current procedure segment the segment number is found in the procedure base register. This is an internal processor register and not directly accessible to the user. If a reference is to some other segment the segment number is located in one of eight address base registers. The three most significant bits of the instruction address field are used to specify this selection.

A set of commands is available to load, store and modify the contents of the address base registers. Additional flexibility is provided by allowing these base registers to operate as index registers (internal bases). When employed in this manner the indexing base register is coupled with a base register that holds a segment number. This base pair in effect defines a base location internal to the segment. The association of bases, together with marking bases as internal, and "locking" certain bases so that they cannot be changed except in a privileged "master" mode, are all contained in a control register for the bases.

For high-speed storing and reestablishing of status, it is possible to store or load the eight base registers with a single command. Any bases which are locked, on a load bases command are passed over and remain unchanged. The use of these various forms of addressing would be difficult to discuss at this time and the reader is referred to one of the subsequent papers in this session describing the software for the new Multics system.[9]

A second form of segment addressing is made available by means of the indirection facilities in the 645. Two variants are provided. The first of these is known as INDIRECT TO SEGMENT (ITS). This form of modifier requires two words, the first of which gives the segment number together with the tag that indicates it is an ITS word. The second word appears as a normal indirect word in either the 635 or 645. It contains an address together with an address tag indicating whether further indirection is to take place and if so what type, and additionally if indexing is to take place before using this address. The second variant is by means of the indirect word pair which is called ITB, that is, INDIRECT THROUGH BASE. This form of indirection is identical to the ITS form with the exception that the first word of the pair contains the ITB modifier and a number from 0 through 7 which indicates which of the 8 address base registers contains the segment number of this address. When either form is encountered during indirection, the segment name then in effect is canceled and replaced with that given by the pair. The indirection will continue as prescribed by the pair.

The Three Modes Of Procedure Execution

In the 635 there were two modes for program execution: namely, master and slave. In slave mode only a restricted set of processor instructions are executable. Certain instructions such as I/O connect, those instructions dealing with the loading of the elapsed-time register and instructions affecting the relocation register were trapped. In master mode these privileged instructions could be executed and the relocation feature disabled.

In the 645 three distinct modes of execution are defined. These are absolute, master and slave. Slave mode is considered to be the normal mode of instruction execution. In this mode no privileged instructions may be executed. Further the relocation logic for segments and pages is fully operative. Master mode uses the segment and paging hardware identically to slave mode with the exception that privileged instructions may be executed and certain constraints on the access to segments are removed. Absolute mode is superior to the other two modes of operation. In the absolute mode, the segmentation and paging hardware is disabled and all instructions in the machine may be executed. Additionally, none of the segment access restrictions apply. Absolute mode is entered only by the occurrence of an interrupt. The machine enters temporarily into absolute mode to record system status but can be caused to remain in this mode if desired. The segmentation hardware can be temporarily enabled on any instruction merely by the use of the instruction word control bit used to indicate base register selection. Further, encountering either an ITS or ITB modifier will cause the segmentation hardware to be turned on for this instruction execution. When in absolute, the mode of the program can be turned back to either master or slave by the execution of an appropriate instruction. This instruction is a branch instruction that defines a segment number which indicates what form of procedure shall be executed, that is, master or slave.

Because it was felt desirable to make it possible to branch easily between various programs including between slave and master programs, a certain degree of insurance has to be built into the hardware to guarantee that spurious branches would not take place into the middle of master mode programs from slave programs. As a consequence, a master mode procedure when viewed from a slave mode procedure appears to be a segment which can neither be written nor read. Further, the only method of addressing this segment that is permitted is a branch to the 0th location. Any attempt to get at other locations by branch, execute, return or any other instructions will result in an improper procedure fault causing an appropriate interrupt. A special form of procedure called EXECUTE ONLY has also been defined which is similar to MASTER PROCEDURE in terms of entry restrictions imposed on slave mode. Once entered, this procedure has all of the execution characteristics of slave mode.

The Associative Memory

The addressing system as now defined would be very unsatisfactory if employed for each instruction or operand reference. If both the descriptor segment and the data segment are paged and if the procedure being executed is paged, a large number of memory cycles might be required to develop a memory address. To overcome this an associative memory is incorporated in the processor. This memory "captures" a compounded descriptor, derived from the segment and page descriptors. The resultant working descriptor represents a particular page of a particular segment. This can be either a data segment or the procedure segment. As a consequence, if a particular page of a segment is being used quite heavily, its descriptor will always be in this associative memory and no additional references to main memory are required to develop the memory address. The associative memory is "invisible" to the user. Its only effect is to greatly speed up the execution of the programs. Whenever a new working descriptor is created, it is placed in the associative memory. If the associative memory is already full, a wired algorithm selects a memory position to be used for the new descriptor and causes an older descriptor to be discarded. A set of commands permit storing selected words from the associative memory and clearing the associative memory. All of these instructions are privileged.

Additional Aids To Memory Allocation

The environment in which this system is to work places a high premium on efficient management of memory resources. Paging of itself simplifies the allocation process. A further gain is possible if one appreciates the effect of frequency and duration of usage. Two distinct mechanisms are employed to supply this information. The first of these involves the page descriptors. A record is made in the descriptor if the page is accessed for any reason. Once this bit has been set to a one, it is unaffected by subsequent page references. A supervisor program will periodically reset these "use bits" to zero and at the same time determine which pages have been accessed since the last entry into this procedure. A second bit in each of the page descriptors is set to one if the contents of the page is altered in any way.

The second mechanism involves the associative memory. The privileged instructions that store the contents of the entire associative memory or store the contents of the cell whose contents are "the oldest" descriptor provide the supervisory program with a measure of the extent and frequency of page usage.

Interrupt Considerations

In the 645 interrupts are generated by external stimuli while faults are generated by processor conditions. The occurrence of either an interrupt or fault causes the execution of two commands located at specific "wired" addresses. On most computers, interrupt can only take place between command executions. In the case of the 645, certain aspects of segmentation made desirable the interruption of the computer at many points during execution of a specific command. After the interrupt is serviced, execution is resumed. Resumption at the precise point of interruption, rather than restart, is mandatory because of the nature of the indirection in the 645.

The features of the segmentation system which first made it mandatory to add this more generalized interrupt capability were associated with the various control checks implied by the descriptors. Examples of these control checks are the bounds check, attempting to write in a Read Only segment, etc. It is advantageous in a segmented environment to cause an interrupt as a result of accessing an appropriately coded segment or page descriptor. This type of interrupt or fault is called a Directed Fault. Its name is derived from the fact that this descriptor directs control to a specific function based on which one of 8-bit configurations is found in the segment or page descriptor. The encoding and placement of this type of descriptor is done by the supervisor. The use of these descriptors for marking missing pages or missing segments will be discussed in a subsequent paper.

The 645 interrupt handling mechanism has been called the "snapshot" register. This is a set of flip-flops which, although used by other functions of the machine, are primarily available for storing the machine state. A trap, be it either an interrupt or a fault, causes the state of the machine to be stored in this snapshot register. The contents of essential registers and a history of control states comprise the snapshot. The first instruction in the interrupt handling routine normally will be a store control unit instruction. This privileged instruction stores the contents of the snapshot register into six memory locations. The subsequent execution of a restore control unit instruction takes the contents of the six words and reestablishes the control unit.

Configuration Controls

Because of the highly on-line nature of this system it is necessary to reconfigure the system relatively easily. As a consequence, those switches required for reconfiguration control are remotely located from the units they control to allow rapid setting from a central point. At first glance it might seem desirable to make program configuration possible; however, if the system is malfunctioning it would be necessary in any event for the program to notify and probably obtain permission from the floor supervisor. As a consequence, it was decided initially to make reconfiguration controllable by manual switches and to allow the computer to instruct the operator during reconfiguration.

Reconfiguration is used for two prime purposes: to remove a unit from the system for service or because of malfunction, or to reconfigure the system either because of the malfunction of one of the units or to "partition" the system so as to have two or more independent systems. In this last case, partitioning would be used either to debug a new system supervisor or perhaps to aid in the diagnostic analysis of a hardware malfunction where more than a single system component were needed.

The effectiveness of rapid reconfiguration is difficult to determine in a paper simulation and efficiency of the system chosen will only be proved or disproved after a number of months of practical use in the on-line environment.

Conclusions

We have attempted in this paper to describe some of the considerations that led to the unique design of the GE 645 as a processor for a multi-access, remote user, information processing system. We have already learned much in the process of designing this machine and feel that within the next two to three years much more will be learned. It is difficult at this time to make any statements about what the future of such processors should be beyond a few tentative conclusions.

First, additional speed both in arithmetic capability and in the memory hierarchy is desirable still; but even more, increased channel capacity of the main store of the machine is required. At present it appears that the principal limitation and expansion of the system will be the channel capacity of core memory. Obviously, this will be alleviated to a great extent by the advent of higher speed central memories for computers. However, this is but one answer. We feel that superior facilities can be gained by closer attention to the system functions that we have emphasized with the 645; namely efficient interrupt handling capability, and comprehensive addressing logic to improve the allocation and protection of physical and logical memory.

Finally, it is felt that the designer of future timesharing systems must remember that a main part of the system is not in the computing center. Rather it is composed of the communications lines, the terminals and the various users, be they human beings, experiments or the like at these terminals. Therefore, the designer must keep in mind that he is engaged in a communications activity as well as an information processing activity and that proper attention must be paid to both aspects. We have done this to the best of our ability in the present system, although we are sure that some several years from now we will be able to return with the description of a machine that will be as great a step over the 645 as the 645 is over previous designs when applied to this new emerging field of time-shared computation.

References

1. F. H. Sumner, "The Central Control Unit of the Atlas Computer," *Proc. IFIP Congress*, 1962, pp. 291-296.
2. J. Fotheringham, "Dynamic Storage Allocation in the Atlas Computer," *Comm. ACM, vol.* 4, no. 10, Oct. 1961, pp. 435-436.
3. A. W. Holt, "Program Organization and Record Keeping for Dynamic Storage Allocation," *Comm. ACM*, vol. 4, no. 10, Oct. 1961, pp. 422-431.

4. M. N. Greenfield, "Fact Segmentation,"*Proc. SJCC*, vol. 21, May 1962, pp. 307-315.
5. J. B. Dennis and E. B. Van Horn, "Nesting and Recursion of Procedures in a Segmented Memory," Project MAC Memo, M-187, M.I.T., Oct. 1964.
6. J. B. Dennis and E. L. Glaser, "The Structure of On-Line Information Processing System," *Proc. Of the Second Congress on Information System Sciences,* Spartan Books, Inc., Washington, D.C., 1965.
7. "The Descriptor," Burroughs Corp., 1961.
8. "IBM 360, Model 67, Computing Report for the Scientist and Engineer," 1, 1 (May 1965) p. 8, Data Processing Division, I.B.M. Corporation.
9. V. A. Vyssotsky, F. J. Corbató, R. M. Graham, "Structure of the Multics Supervisor", this volume.

** Work reported herein was supported (in part) by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01).*

1965 Fall Joint Computer Conference