A display processor design

by R. W. WATSON
Shell Development Company
Emeryville, California

T. H. MYER
Bolt Beranek and Newman, Incorporated
Cambridge, Massachusetts

I. E. SUTHERLAND
Evans and Sutherland Computer Corporation*  
Salt Lake City, Utah

and

M. K. VOSBURY
Sanders Associates, Incorporated
Nashua, New Hampshire

INTRODUCTION

This paper describes the results of a collaborative design effort aimed at development of a general purpose display system for the SDS-940 time-shared computer. The important features of the system evolved gradually from a number of separate design goals. We wanted a display system that would:

1. Contain an extensive but straightforward set of display generating commands.
2. Be able to generate pictures from highly complex data structures.
3. Allow easy access to display files from user programs in the main computer.
4. Provide some immediate feedback and interactive processing service to the display user, and be able to call upon the main computer for more extensive service.
5. Permit attachment of special purpose display generation and interactive hardware, as well as multiple display consoles.
6. Be capable of time-sharing its central resources among separate console-users.

These goals and their influence on the system design provide a framework for the detailed discussion that follows in the body of this paper. Before proceeding, however, we would like to give some orientation by presenting an overview of the system design without dwelling on our motives.

Figure 1 is a block diagram of the system. As indicated, its main components are a display processor (including computer interface) that controls the system and channels digital information among the other components, a display generator that produces ap-
Figure 1—System configuration

Figure 2—Processor

Figure 3a—Display commands

Figure 3b—Address commands

appropriate analog drive signals, and a collection of display consoles and other peripheral devices. Note that the display shares the memory of the central computer.

The display generator contains high speed vector, character and beam positioning generators. Display generators are discussed in references 1, 5, and 8, and the characteristics of the display generator for this system are discussed in reference 10. This paper is primarily concerned with the design of the display processor, and with certain aspects of the overall system design.

Figure 2 shows the display processor in more detail. One can view it as a collection of registers, each of which is connected to two main information paths—the Main Input and Main Output Busses. Other information paths provide connection to the display generator, peripheral devices, and computer interface.

Figures 3a–d describe the command set for the display processor. All commands are "immediate" in the sense that each contains its operand(s) in what is usually the address field (referred to in this paper as the "operand field"). The Display Commands (Figure 3a) supply information to the display generator via the X, Y, and character registers. The Address, Data and Miscellaneous Commands (Figures 3b–3d) affect the contents of the various display processor registers, and may also cause information to be stored in 940 memory. Most of these commands contain separate fields to specify the operation and the register to be operated on. Because the various registers serve distinct functions, the effect of a given command will vary depending on the register specified. For example, a Load of the Program Counter is equivalent to a conventional jump; a Loading of the I/O register will have an entirely different effect. The I/O commands (Figure 3d) transfer digital data and control information to and from the peripheral devices, either directly, or via the I/O register.

As shown in Figure 1, the Display Processor and Generator, taken together, control and supply information to the peripheral devices via the Analog and Digital I/O Busses. Three paths allow the 940 computer, in turn, to control and inform the Display Processor. These are a direct connection between the
A Display Processor Design 211

Display Processor and 940 Core Memory, and two connections, the I/O and interrupt lines, between the Display and the 940 Processor.

Display commands

One of our goals was to design a rich but "clean" series of display commands. In particular, we wanted to avoid a difficulty we encountered in several other display systems—the fact that word length restrictions force reliance on two word instructions or on dual operating modes in which the machine will treat all words either as display data or instructions, depending on its mode. The 24 bit word length of the 940 provided enough space (just barely) to allow all instructions to carry OPCODE and X-Y or character data in a single word.

Figure 3a shows the six Display Commands. The display generator can produce lines and characters. Lines are drawn in $2 + 3L$ microseconds, where L is the length of the line in inches. The beam can be randomly positioned anywhere on the screen in 7 microseconds maximum. Characters are drawn in from 4 to 12 microseconds, depending on size and number of strokes required. One command plots three characters in "typewriter" format; the remaining commands specify the endpoints of displayed lines.** The endpoint of a line can be specified, in two's complement, as an absolute location on the 1024 by 1024 coordinate grid of the display screen or as a relative displacement from the current beam location. One pair of commands allows endpoints to be specified in relative or absolute terms. Another pair allows mixed specifications—one coordinate absolute, the other relative. The remaining command allows three endpoints to be specified as short, relative displacements. Each X or Y component of a short displacement specification (Figure 3a) is represented, in two's complement, by one sign bit and two magnitude bits. The two magnitude bits are treated by the hardware as the two high order bits of a three bit magnitude representation. The low order bit is assumed to be 0. This allows displacements of about 0.1 inch in X and Y to be specified. Each line specification carries an unblank bit (U). If set, the line will appear, otherwise it will produce an invisible beam movement.

The appearance of displayed elements is controlled by the three fields of the display parameter register (RIO), (Figure 4). Eight intensity levels and four character sizes are available. A line can be drawn solid, in a variety of dotted and dashed formats, or as a single dot at its terminal point (point plotting). To allow independent control of the three parameters, a masking mechanism is included.† To change parameters one uses a Load Command (Figure 3c) with bits 12-14 specifying which parameters are to be affected.

* A null code can be placed in the unused character position when it is desired to plot one or two characters. In addition, the character generator has an unusually rich complement of control characters, including space and half space up, down, backwards, and, forwards. Full details are covered in reference 10.

** The string point for a line or group of characters is the current beam position. The X and Y registers always contain this value; their contents are appropriately updated as each Display Command is executed.

† A similar scheme was used in the Digital Equipment Corporation (DEC) 340 and 338.
One of our key goals was to achieve a display system that would allow us to represent pictures by means of complex data structures. Behind this goal was a desire to eliminate or minimize the separation that is necessary in many systems between a “master representation” and a “display file”. Looking at this rather general goal in more detail, we wanted the ability to:

1. Execute nested picture subroutines to arbitrary depth.
2. Create “transparent” subroutines—save and restore selected display registers such as the X and Y beam position and display parameters on entering and leaving a subroutine.
3. Pass parameters to subroutines.
4. Easily identify objects selected by light pen or stylus in terms of the picture structure.
5. Perform certain forms of general list processing.

Nested subroutines can be handled by a variety of subroutine mechanisms. The need for easy light pen selection led us to use a pushdown stack. When processing a light pen or stylus “hit” one must trace one’s path back through the subroutine hierarchy in order to relate the object selected to the drawing structure. Without a pushdown stack this requires search through the subroutine structure. With a stack system, however, the required trace is maintained compactly and automatically by the return addresses stored in the stack.

The pushdown stack

The pushdown stack is not in itself new with this design. The DEC 338 display, for example, made very successful use of a stack system. What is unique in this display is the way in which the stack was implemented. The need for save and restore information other than return addresses meant that it had to be possible to push any register into the stack. In order to get the information back into the right register, data in the stack had to be marked in some way. After considering several marking schemes, we hit on the idea of placing instructions rather than data in the stack. When a display register is “pushed” into the stack, what actually appears in memory is an instruction to reload the register in question with its original contents.

The notion of putting instructions in the stack, of course, changes one’s conception of the whole stack mechanism. The POP instruction (counterpart to PUSH), for example, becomes a special variety of “execute”, and the stack pointer a kind of auxiliary program counter. In recognition of this, we reversed the direction in which stacks usually build. As information is pushed into the stack, the stack pointer is decremented. This means that instructions in the stack are “popped” (executed) in the usual low-to-high address order.

Treating the stack pointer as an auxiliary program counter suggested that we make it accessible, as is the program counter, to certain processor instructions. By doing so, we freed the stack from a fixed location in core. Because one can load the stack pointer, one is free to start the stack where one pleases. Moreover, as we shall see below, one can even achieve a stack that occupies disjoint areas of memory by saving the old stack pointer at the beginning of each new section of stack.

With this background, we can now look at some details of the stack system. The Push, Load/Push, and Push Data commands (Figures 3b and 3c) place information in the stack, POP and POP but Skip if Jump (Figure 3d) get it back out. As mentioned above, the Push commands assemble instructions in memory; the POP commands execute these instructions. The Push operation may seem complex, but is in fact quite simple. To see this, let us examine a Push command in detail.

1. Assume “push the X Register” has been fetched into the Instruction Register (R1).
2. The register field (bits 4–7) of R1 selects the X register (R8). The contents of R8 are copied to bits 12–23 of R1.
3. Bits 8 and 9 of R1 are cleared to 0. The remainder of R1 is left unchanged.
4. R1 is copied back into the memory at the location selected by the Stack Pointer (R3).
5. The Stack Pointer is decremented.

The main use for Push is to save register contents for later restoration at the end of a subroutine. As indicated in Figures 3a and 3b, Push can be brought to bear on any register accessible to the programmer. Because the stack is marked, a single instruction restores the information regardless of where it came from.

* A variant of Push will place an Add Command in the stack.
In dealing with display structures, it is convenient to supply names or tags for the objects being presented. These may, for example, be pointers to other areas of memory that describe non-graphic properties of the objects. The No-Op command (Figure 3b) allows names to be included in a display file. It causes no action, but its operand field may contain tag information. Push Data allows names to be pushed into the stack, a further convenience when tracing back through a subroutine hierarchy. This command writes its own operand field into the stack in the form of a No-Op command.

The third Push variant—Load/Push—exchanges its operand field with the selected register before writing the original register contents into the stack. Load/Push the Program Counter provides a standard subroutine call. The current program location is stored in the stack (as a Jump instruction) while the Program Counter is simultaneously reset to the subroutine entry point specified by the Load/Push command. Load/Push can be used in a similar way to save and simultaneously reset any other register.

Load/Push the Stack Pointer deserves special attention. Because the Stack Pointer is loaded with the new value before its original contents are pushed, the old value will be pushed into the new stack. Thus, the first word put into the new stack is a pointer that links it to the old stack. It is this feature that allows one to create disjoint stacks; the saved stack pointers provide an automatic address chain back to the original stack. We have chosen to call these stored links “Stack Jumps.”

Pop, the counterpart to Push, causes the display processor to execute instructions in the stack. When the processor encounters a Pop, it increments the Stack pointer, fetches the instruction selected by the new pointer value, executes that instruction, and then returns to normal instruction execution under control of the Program Counter. Typically, the instructions executed by Pop will be Load or No-Op commands created by one of the Push instructions. However, any instruction can be executed through Pop.

With the Pop instruction in hand, we can now examine a typical subroutine linkage. Having entered the subroutine through a Load/Push Program Counter, one can use Push or Load/Push commands to save any other registers. The net result is a series of Load Commands in the stack with a Load Program Counter occupying the last (highest numbered) address. Two commands: Pop followed by a Jump to the previous instruction will restore the saved registers and provide a subroutine return. The processor loops on these two commands, reloading the saved registers, until the stored Load Program Counter removes it from the loop and returns control to the main program.

The Pop but Skip on Jump command allows one to restore saved registers without returning from a subroutine. This command behaves exactly like Pop except upon encountering a Load Program Counter in the stack. In this event the stacked instruction is ignored, the Stack Pointer decremented and the Program Counter incremented an extra time. The net result is that the processor breaks out of a loop such as the one suggested above, just before executing the return Jump.

The above discussion has suggested some conventional uses for the stack instructions. However, such features as the ability to manipulate the Stack Pointer in various ways permits the user to devise more sophisticated uses for the stack mechanism. We have made heavy use of this flexibility in the software support package. One example application is the handling of rubber band lines and other simple constraints within the display processor. We accomplish these functions by performing list processing in the display file using the stack feature.

Experience in working with the system has shown that the heavy use of multiple stacks could be more efficient if another stack pointer were available or if a 14 bit address length general purpose register were available for temporary storage of the Stack Pointer. The Shell system is being modified to add two such 14 bit general registers. The ability to execute instructions in the stack has given generality and power to the display processor at modest cost.

Memory sharing

A consequence of our desire to achieve close coupling between pictorial and other information was the need to allow easy access to display files from programs in the 940. As well as permitting advanced graphics applications, we felt that close access would simplify the general software support for the display.

To realize this goal we attached the display processor directly to the core memory of the central computer rather than relying on a separate buffer memory.* The display processor addresses the 1.75 microsecond 940 memory through its program counter and stack pointer. In operation, the display processor refreshes the display consoles by executing display commands stored in 940 memory and passing the data they contain to the display generator.

Given this close interconnection between display

---

* This connection utilizes the 940's second memory port.
and main computer, considerable care was necessary to ensure a display system that could operate effectively without degrading or endangering the supporting time-shared computer system. One potential danger—competition between display and central processors for memory access—was reduced to an acceptable level by use of dual access priorities on the second path to memory.**

A second and more serious danger—inadvertent alteration of 940 memory by a display program—was eliminated by including memory mapping and protection hardware in the display processor. This equipment is identical in function to equivalent hardware in the 940. By means of this mapping, the 16K word “virtual” memory that can be accessed by the display (and 940) instructions is mapped into 2K word physical pages that may be scattered through the 64K words of 940 core memory. At any one time only a few of these pages may be assigned to the display, and those pages that are assigned may be made accessible for reading only or for reading and writing.

Registers in the mapping hardware indicate, for each of the eight pages that the display might address, whether or not a physical page is assigned, and if assigned its status (read only or read/write). Only the 940 monitor can change the contents of the map registers. As shown in Figure 5, memory addresses transmitted by the display processor, are processed through the mapping hardware before accessing 940 memory. Any attempt to address an unassigned page or to write into a read-only page stops the display processor and sends an interrupt signal to the 940.

One consequence of mapping is that undebugged display programs are of no danger to the system or to other users. Mapping has the additional benefit of allowing users and system software designers to treat display programs in exactly the same way as 940 user programs. In fact, because mapping for a user’s 940 program can be made identical to the mapping for his display file, the two can share the same address space and thus, be merged in any way the user pleases. Thus, the user can, if he wishes, create a common data structure that represents pictorial and other properties of the objects to be viewed. In addition, he can achieve an unprecedented richness of interaction between operations performed at a display console and the underlying processing in the main computer.

Processing tasks—Display vs. 940

The issue of how much power to include in the display processor is a complicated one. This issue is discussed more fully in an earlier paper that was inspired by the difficulties we encountered on his project. We chose to include enough computing power to handle the immediate response to interactive events such as light pen “hits” or the depression of push buttons. Less than this would yield sluggish interaction; tasks requiring more power could, we felt, be relegated to the 940 processor.

With these ideas in mind, we equipped the display processor with a set of commands aimed specifically at interactive situations. As shown in Figure 3, these include bit manipulating and skip commands and an arithmetic compare operation. The bit manipulating and skip instructions include Clear, Toggle (Complement), And, Set, Skip on 0, Skip on 1, Skip on 1 and Clear, all handled under the mask in the operand field of the instruction. These commands are used to test or change status, control interrupt masking and so forth. There is also a three way arithmetic compare of a selected register with the operand giving a skip of 0, 1, or 2, depending on the result. This command allows one to branch on the X or Y location of the display beam or of a coordinate input device. Taken together with the Add, Register Exchange and General

**The 940 CPU accesses memory through the first path to memory. The display accesses memory through a second path. Devices on the second path can request access with either higher or lower priority than the first path. The display processor overlaps the drawing of a vector or character with the fetch of the next command. Memory accesses at this time are with low priority. When the display operation is completed, access is made with high priority, if not previously successful. Non-overlapped accesses are made with high priority. Using the above mechanism, reasonable assumptions on command mix and the fact that the 940 memory has 4 independent interleaved modules, it has been estimated that the 940 CPU will be blocked from immediate memory access less than 2 percent of the time.**
Register Commands † and the stack mechanism, these interactive commands have allowed us to do such things as handle light buttons, produce point rasters, and perform the work involved in light pen tracking, all without intervention from the 940. Control of the display processor is implemented with microcoding and a read-only memory. The time required per microstep is 400 nanoseconds. Command fetch, decoding, and program counter update require 6 microsteps plus a memory read time. The number of microsteps required per command execution is variable, Load requires 1, Push 3 and Pop 9, for example. The Pop and General Register Commands have the longest execution time. The read-only memory can be easily modified or inexpensively replaced. This feature will be used to modify or add commands thought to be useful from the software experience.11

In spite of its power, the display processor must call on the 940 for assistance in tasks beyond its capabilities. In addition, the 940 must, of course, have ultimate control over the display. We satisfied both needs by connecting the display processor to the I/O and interrupt systems of the 940. Through these connections the display processor can transmit service requests to the 940. The 940 processor can in turn interrogate and set the registers of the display. Together with the shared memory mechanism, these two connections yield a closeness of coupling that contributes importantly to the ability of the two machines to share their processing resources.

Through its I/O lines the 940 processor can directly access all registers of the display. Any display register can be brought into the 940 processor by a 940 Parallel Input (PIN) instruction. Conversely, the 940 processor can set any display register through a Parallel Output (POT) instruction. This feature aids the 940 in initializing the display and in processing interrupt requests. If the 940 sets the display's Instruction Register through a POT instruction, the display will treat the information as a command, execute it, and then halt. Unless directly altered by a command executed in this way, the display's Program Counter is not changed. The net result is that the 940 can, in effect, “execute” any display instruction. As well as access to the display registers, the direct I/O connection allows the 940 to stop and start the display set the display’s memory map and the “device map” described in the next section.

The interrupt system gives the display a means for requesting help from the 940. Some events in the display (irrecoverable errors, for example) can only be dealt with by the 940. Either the 940 or the display processor can cope with other situations (light pen hits, scope edge violations). In recognition of this, we grouped all interrupt as well as other control and status information into one register—the System Parameter Register (R11), shown in detail in Table I. The bottom twelve bits of this register are accessible both to the bit manipulation commands of the display and, via the POT/PIN instructions, to the 940. The top seven bits are accessible only to the POT/PIN instructions because only the 940 can deal with the information they contain.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>These two bits assist the 940 in interpreting certain interrupt events.</td>
</tr>
<tr>
<td>6</td>
<td>Parity Error Flag.</td>
</tr>
<tr>
<td>7</td>
<td>Memory Map Violation Flag.</td>
</tr>
<tr>
<td>8</td>
<td>Time-Out Flag (the display has a built-in down-counting clock).</td>
</tr>
<tr>
<td>9</td>
<td>Halt Mask.</td>
</tr>
<tr>
<td>10</td>
<td>Halt Flag.</td>
</tr>
<tr>
<td>12</td>
<td>Unused.</td>
</tr>
<tr>
<td>13</td>
<td>X Edge Overflow Flag.</td>
</tr>
<tr>
<td>14</td>
<td>Y Edge Overflow Flag.</td>
</tr>
<tr>
<td>15</td>
<td>Edge Overflow Mask.</td>
</tr>
<tr>
<td>16</td>
<td>Synchronous Hit Flag (e.g., light pen).</td>
</tr>
<tr>
<td>17</td>
<td>Synchronous Hit Mask.</td>
</tr>
<tr>
<td>18</td>
<td>Asynchronous Hit Flag (e.g., pushbutton or keyboard).</td>
</tr>
<tr>
<td>19</td>
<td>Asynchronous Hit Mask.</td>
</tr>
<tr>
<td>20</td>
<td>Blink (toggles continuously at blink rate).</td>
</tr>
<tr>
<td>21</td>
<td>Blink Control.</td>
</tr>
<tr>
<td>22</td>
<td>Slow Mode Control (for storage tube consoles).</td>
</tr>
<tr>
<td>23</td>
<td>Master Unblank (if 0 unconditionally blanks the display).</td>
</tr>
</tbody>
</table>

† Though not directed at any particular interactive function, our implementation of the processor design allowed us to include these commands at little cost. They have proven more than worth the price. The Add (Figures 3b, 3c) and Register Exchange (Figure 3d) generates a new processor instruction in which OP₂ operates on Rₚ using the contents of Rₚ as operand. This allows one, for example, to add or compare two registers.

* Nineteen of the possible 24 bits in this register were implemented.
The lower bits in R11 handle several kinds of events, for each of which there is a flag bit and a mask bit. The flag bit is set whenever the event occurs; the setting of the mask bit determines whether or not an interrupt signal is sent to the 940. This arrangement allows the programmer to cope with events through the bit-oriented instructions of the display processor, or ignoring them in his display program, to pass them on as interrupt signals to the 940. In addition, a display program can request service from the 940 by executing a Halt and Interrupt instruction (Figure 3d).

Because the 940 must assist the display processor in certain situations, it was necessary to allow display users to write real-time 940 programs. The problem of preventing real-time programs from degrading the time sharing performance of the 940 was handled by setting limits on a display user's CPU usage during each refresh cycle of the display.

**Consoles and other I/O devices**

So far, we have considered the display processor and its relationship to the parent computer. We were also concerned with display consoles and other peripheral devices, and their relationship, in turn, to the display processor and generator. Our main goal in this area was flexibility. We wanted the ability to attach a variety of display consoles, differing in some cases in their equipment complements, as well as other non-display devices including graphic input tablets, and specialized analog equipment, such as circle or raster generators. We met this need by dissociating from the display processor design any consideration of individual consoles or other devices. Instead, we elected to treat these as I/O devices, and to handle their control and the transmission of information to and from them by means of a very general I/O bus system.

The digital portion of this bus system is similar in nature to the bussing schemes used on several general purpose computers. Devices are selected by an address field in the I/O instructions; all devices are treated homogeneously as collections of registers; and a given register may contain control or status information, input or output data, or a mixture of these.

Figure 3d shows the Input/Output commands. Two of these permit the user to transmit information between the I/O register (R15) and the registers of external devices. Incoming data and status information can then be examined by the Display Processor, through the test and skip instructions described in the last section, or dealt with by the 940 through the POT/PIN commands. The remaining two commands permit somewhat faster direct output of key commands and direct testing of key device status bits. As mentioned in the last section, another component in the digital I/O bus system is the channeling, through OR gates, of synchronous and asynchronous events in the peripheral devices into the H1 and H2 bits of the System Parameter Register.

Corresponding to this treatment of digital information, the transmission of analog signals within the system was also handled through a bussing scheme, which allows input of analog signals to summing points within the display generator as well as output of display drive signals.* Because of this treatment of peripheral devices, one can view the display processor and generator taken together as a specialized hybrid computer whose main job is to handle a series of I/O devices through a combined analog/digital bus system.

Just as the 940 processor is time-shared, we wanted the ability to time-share the display processor and generator among a number of user consoles without danger of interference between them. This was achieved by giving the 940 processor the ability to control and thus schedule, usage of the display processor, and by allowing for device protection hardware in the display's I/O bus design. This hardware utilizes a mapping scheme similar to the memory mapping and protection hardware in the 940 and has the additional advantage of allowing a user to refer to peripheral devices through "virtual" addresses that can remain constant even though he may be assigned a different console at different times.

**CONCLUSION**

The stack mechanism in this design is the most significant departure from previous machine design practice. The features of a marked stack, and the ability to create disjoint stacks (through the "stack-jump" linkage) are both easy to implement and useful. As is by now well known, the stack feature in a display processor is essential for orderly treatment of "hits" detected by the light pen or other stylus devices.

Close coupling between display information and 940 programs has been achieved by the mechanism of shared memory. Other general purpose display systems seem to be relying more and more on small local computers for interactive service and to shield the main computer from the display. By contrast, we deliberately set out to achieve a rich interaction between display and parent computer, and the extremely close coupling

* Whether a device generates or responds to analog signals depends upon bit settings in its control register.
of the two machines reflects this goal. Our experience so far indicates that this coupling can be achieved without serious degradation of the 940 time-sharing system.

Until now most displays have been treated strictly as I/O equipment. As displays have grown in complexity over the years, however, we have come to recognize that display processors have many of the attributes of general purpose computers. In recognition of this, we deliberately approached the design problem with a processor-oriented rather than I/O device-oriented approach. This thinking is reflected in the display's extensive instruction set, in the use of memory and device mapping, in the uniform treatment of consoles as peripheral devices, and finally, in the microcoding and uniform bussing scheme that dominate the display processor design.

ACKNOWLEDGMENT

The authors would like to acknowledge the contributions of their co-workers at Shell Development, Bolt Beranek and Newman, and Sanders Associates, during the various design and implementation phases of this project. The Bolt Beranek and Newman effort was supported by the Advanced Research Projects Agency of the Defense Department under contract F 19628–68–C–0125.

REFERENCES

1 L C HOBBISS

Display application and technology
Proc IEEE 59 12 1870-1884 1966
2 N A BALL H Q FOSTER W H LONG
I E SUTHERLAND R L WIGINGTON
A shared memory computer display system
3 K H KONKLE
An analog comparator as a pseudo-light pen for computer displays
IEEE Trans on Computers C-17 1 54-55 1968
4 W W LICHTENBERGER M W PIRTLE
A facility for experimentation in man-machine interaction
AFIPS Proc 27 589-598 1965
5 C MACHOVER
Graphic CRT terminals—characteristics of commercially available equipment
AFIPS Proc 31 149-160 1967
6 T H MYER I E SUTHERLAND
On the design of display processors
Com ACM 11 6 410-414 1968
7 M W PIRTLE
Intercommunication of processors and memory
AFIPS Proc 31 621-634 1967
8 H H POOLE
Fundamentals of display systems
Spartan Books Washington D C 1966
9 C SEITZ G F PFISTER
A display processor for a small computer
AFIPS Proc This issue 1969
10 R W WATSON
The design of a general purpose graphic terminal for a time-sharing system
11 R W WATSON et al
Paper in preparation describing the design philosophy of the software for use with the display system reported here.