CONSIDERATIONS FOR MICROPROCESSOR-BASED TERMINAL DESIGN

Reid G. Smith

Heuristic Programming Project
Department of Computer Science
Stanford University
Stanford, California, 94305.

Tom M. Mitchell

Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, New Jersey, 08903.

1 Introduction

We discuss the design of hardware and software for inexpensive microprocessor-based terminal/microcomputers.[1] Such devices are fundamentally microcomputers that have been adapted, with specialized software, to operate as remote terminals for a host computer.

The discussion centers on a specific video terminal designed and constructed by the authors. This terminal is based on the **Intel 8080** microprocessor and is equipped with software sufficient to emulate the characteristics of standard video terminals required by several available *screen-oriented* text editors in common use at sites throughout the **ARPAnet** (such as **E [Samuel, 1978]** and **TV-Edit [Kanerva, 1975]).**

Screen-oriented editors[2] differ from other editors in their use of high-speed video terminals to display the contents of large sections of a file being edited. As editing operations are performed, the display is revised to indicate their effects on the file (i.e., editing operates in *a What you see is what you get* mode). Such editors require terminals capable of primitive text-processing operations, such as inserting a character in a line of text by shifting the existing characters. In addition to such capabilities, the terminal is typically expected to support 8-bit transmission (instead of the usual 7 bits plus parity), selectable modes for displaying characters (e.g., normal or inverse video, blinking, or dual intensity), and an 80-character line width.

Because of the high percentage of both user and computer time invested in interactive editing of manuscripts and programs at many computation sites, the careful design of editing systems and terminals can have a strong influence on the efficiency of both the user and the computer. We believe that the capabilities of microprocessor-based terminals will make them a central component of future editing systems.[3]

We have found that the microprocessor adequately serves as the controller for such terminals, and that a *software-based* approach to the design of such terminals offers substantial advantages in capabilities, flexibility, and cost over the *hardware-based* approach. We suggest guidelines for future designs of microprocessor-based terminals on the basis of our experience designing and using the terminal described here.

[1] The authors wish to acknowledge a number of helpful discussions with R. David Arnold and Pentti Kanerva.

[2] This paper was edited with the terminal and the screen-oriented editors **TV-Edit** and **E** (and transferred between coasts via the **ARPAnet** several times during its creation). The final camera-ready copy was produced on a **Xerox Graphics Printer.**

[3] We do not consider graphics here. See **[Baskett, 1976]** and **[Flexer, 1978]** for relevant discussions.

In order to take full advantage of the flexibility afforded by microprocessor-based designs, we have implemented the capability to *download* and execute **8080** programs written and assembled on a host computer (as can be done, for example, by the **DEC GT-40** terminal). This allows the user to customize and extend the features of his terminal. At the same time, it provides access to the **8080** as a microcomputer with the software development tools and mass storage provided by the host computer. The terminal is thus a complete, stand-alone microcomputer system specially configured for its role as a terminal.

Most of the terminal functions are implemented in software, using an **8080** as a terminal controller, and EPROM to hold the display software. The terminal is connected to a standard black-and-white television set or monitor at the video amplifier. We have discovered that our own commercial television sets have sufficient bandwidth in the video amplifier to support lines of 80 characters, even though they are of the older, tube variety. The terminal parts cost was approximately $650 (small quantity prices, late 1976, not including the television and the modem).

2 A Design for a Microprocessor-based Terminal

2.1 Hardware

The system is constructed on four **S-100** Bus-compatible circuit boards. The first houses an **8080** microprocessor and associated circuitry, 2048 bytes of EPROM (**2708**), an 8-level priority interrupt circuit, and the keyboard and serial line interfaces. The video display interface is placed on the second board. It includes 2048 bytes of static RAM (**2102-1**), the memory-addressing and control circuitry, and the screen-formatting circuitry. The third board houses 4096 bytes of static RAM (**2102-1**). A 1200/150 bps. split-speed modem is placed on the fourth board. It is also possible to couple to an external modem at speeds ranging from 150 to 9600 bps.

2.2 Video Display

2.2.1 Display Memory

The video display board contains 2048 bytes of memory, used by the processor to store the **ASCII** codes of characters to be displayed in the 25-line 80-character display. This memory is addressed as part of the address space of the microprocessor. The memory is constantly scanned by the display circuitry that generates the raster scan video signal. Memory conflicts between the processor and display circuitry are resolved in favor of the processor, and precautions are taken to minimize screen noise that arises when the display circuitry is denied access to memory.

The top line of the display is used to display terminal status information (on-line/local, full/half duplex, etc.). The remaining 24 lines comprise the *usable* screen, which displays

data transmitted from the host. Characters on the top line are displayed white on black to set the line apart from the usable screen, where characters are displayed black on white.

2.2.2 Display Mode Control

The terminal can display characters in any combination of normal/inverse video, blink/no blink, and normal/double intensity.

Each 80-character line in the display uses 81 bytes of memory. The first of the 81 bytes (called the display mode byte) defines the display modes for characters in the line. The remaining 80 bytes contain the **ASCII** codes of the characters to be displayed in the line.

The display mode byte specifies two display modes (a mode is any combination of normal/inverse video, blink, and dual intensity). The eighth bit of each of the 80 character-bytes is used to determine which of the two display modes associated with the current line is to be used for the current character.

This method allows a display of characters in any of the combinations but allows only two distinct display modes for characters on the same line. Because 8-bit memory has been used, there are not enough bits per word to completely specify the display mode for each character individually. The method described above has served as a useful compromise between having only two possible modes for the entire screen and having eight possible modes for each character. We discuss this issue further in Section 4.3.

3 Software

Within the hardware design described above, a range of alternate terminals can be defined in software. In this section, we describe the set of terminal features that we have implemented in microcomputer software. The collection of these features makes up the specifications of the terminal as viewed by the host computer. We describe, as well, a small amount of software written for the host computer that eases the task of software development for the microcomputer and eliminates the need for local mass storage.

3.1 Terminal Software

We have written software for operating the microcomputer as a video terminal capable of executing several specialized functions: insert and delete characters and lines, erase to end of line, blank screen, set display mode, and address cursor--both relative (up, down, left, and right) and absolute (to a specified row and column). As these are software controlled (in the microprocessor), any similar set of features is possible. The size of the **8080** code that defines these terminal features is approximately 1.34 K bytes.[1]

3.2 Microcomputer Software

We have written software to provide terminal capabilities beyond those required to support screen-oriented editors. In particular, we have written a monitor that runs in the microprocessor and accepts commands from the keyboard to execute specified tasks. The user may invoke this local monitor at any time by typing the two control character sequence, ↑U ↑V.[1] When this is done, software control is passed from the section of code that emulates a terminal to the monitor, which awaits a command. Some of the commands currently available in the local monitor are shown in Figure 3.2. The **8080** code that defines the local monitor resides in 1.3 K bytes of memory.

Store Screen: *store the current screen contents in local memory.*
Recall Screen: *display the screen stored in local memory.*
Transmit Screen: *transmit the currently displayed screen of text to the host in the same format as if it had been typed from the keyboard (trailing spaces are not transmitted).*
Select Display Mode: *set the mode for displaying characters (normal/inverse video, etc.) as desired.*
Logout Job: *logout the host job (transmits the correct characters to logout a job from the* **DEC-10 TENEX** *or the* **DEC-20 TOPS-20** *operating systems).*
Go To Address: *execute microcomputer program beginning at specified address.*
Exit: *return to terminal mode of operation.*

Figure 3.2. Local Monitor Commands.

3.3 Using the Host to Develop Microcomputer Software

One interesting aspect of microprocessor-based terminals is the possibility that they can provide inexpensive, stand-alone processing capabilities. The main expense associated with current microcomputers is support-peripherals such as floppy disks and input-output devices. Microprocessor-based terminals connected to a host can be configured with substantial savings if they rely upon the host for mass storage and other support.

We have therefore included a capability to transfer data from the host directly to the local memory of the terminal. This capability to *download* data and code from the host computer to any section of the memory of the local microcomputer considerably eases the task of microcomputer software development. Needing only the addition of memory to the bare hardware to implement the basic microprocessor-based terminal, one can take advantage of the full processing capabilities of the microprocessor.[2]

3.3.1 Cross Assembler

The cross-assembler (called **Micro-SYMBOL**) runs on the **DEC-10** and **DEC-20.** It enables us to write software for the terminal processor using the flexible time-sharing facilities of the host.[3] The **Micro-SYMBOL** cross-assembler is similar to the **MAC 80** assembler **[INTEL, 1974]** and produces a machine code file in the same format. It can produce machine code for the **Intel 8080**, the **Motorola 6800,** the **M. O. S. 6500** series, and the **Zilog Z-80.**

3.3.2 Downloader

The downloader (also written in **SAIL**) enables programs assembled on the host computer (or any data in the correct format) to be transferred directly to the local memory of the microprocessor via the terminal connection. Downloaded data

1 The terminal emulates the functions of the **Datamedia Elite 2500** terminal **[DATAMEDIA, 1976]** necessary to run the editors **E** and **TV-Edit.**

[1] ↑U is an *escape* character. In order to transmit a *real* ↑U, it must be typed twice in sequence.

[2] This feature is transparent to a user who does not wish to make use of it.

[3] **Micro-SYMBOL** was written in **SAIL** (an **ALGOL-**like language **[Reiser, 1976]**) for the TOPS-10 operating system by Bill Weiher at the Stanford Artificial Intelligence Laboratory. It was modified for the **TENEX and TOPS-20** operating systems by Reid Smith.

are distinguished from transmitted characters by prefacing the data with the two control characters ↑U ↑D (NAK EOT). Downloaded data is formatted in records. Each record includes information concerning the number of bytes to be stored, the starting address for the record, a checksum byte, and a control byte (that specifies whether or not the microprocessor is to proceed to the starting address after the record has been correctly loaded). The download receiver in the terminal software acknowledges receipt of each record with ↑F (ACK) if the checksum is consistent with the received record; or otherwise, with ↑U (NAK). The downloader will attempt up to three retransmissions of an improperly received record before aborting the download.

3.3.3 Execution of Local Programs

The downloaded program will be executed immediately by the microprocessor if the control byte of the transmitted record (see above) is set. Alternately, a program can be downloaded without immediate execution. In this case it is executed by entering the local monitor (described in Section 3.2) and executing the **Go To Address** command with the starting address of the program.

3.3.4 Use of Local Terminal Software

Because the terminal software provides a natural means for microprocessor input and output, and because it contains several useful subroutines, we have found it convenient to make the software readily available to downloaded programs. We therefore maintain a table of addresses of useful routines at a fixed place in memory so that downloaded programs may access these routines given the location of the table.

In addition we have found it extremely useful to allow downloaded programs to treat the terminal software as a *virtual terminal*, calling it with a sequence of characters to be interpreted as though they had originated at the host. Thus, downloaded programs may call the subroutine **TERMINAL** with a queue of characters to be treated as characters input to the terminal. By using this virtual terminal routine, it is possible to write programs *in the language of the virtual terminal*, using the control characters that specify the special terminal functions (insert/delete line, etc.) as commands interspersed with text. This capability allows very compact local display processing programs to be written.

4 Microprocessor-based Terminals

4.1 The System as a Computer Terminal

In its role as a terminal the microprocessor-based design offers several advantages in terms of cost, capabilities, and flexibility for the system.

The primitive text-processing operations (insert and delete line, etc.) of the terminal described above are typically implemented in hardware in other terminals. They require special circuitry, and thus increased cost. Implementing such features in software requires no additional hardware other than a small amount of memory. We have found that the cost of building a terminal with only these primitive text-processing operations is less for software-based design ($650, small quantity prices, late 1976, not including the television and modem) than for the corresponding hardware-based design.

In addition to cost, microprocessor-based designs for terminals offer more capabilities than hardware-based designs. The complexity of the features that can be implemented in software (e.g., a local monitor) is limited only by available memory; whereas for hardware-based designs, the complexity of features that can be implemented is strongly limited by the cost of additional hardware. In the near future we expect to add capabilities such as *macro* command definition--dynamic association of a string of characters to be transmitted to the host with a single keystroke. Thus, a user can define macros for sets of commands that he frequently uses (e.g., logout job, or run a given utility program with commonly entered input parameters). It is likely that as experience with microprocessor-based terminals grows, ideas for new capabilities will quickly escalate (see, for example, the efforts of the RITA project **[Waterman, 1978]**).

Possibly the most important feature of microprocessor-based terminals is the increased flexibility that they provide. General-purpose microprocessor-based terminals can be mass produced at low cost and then customized for individual uses by altering or extending the terminal software. In addition, a single terminal can easily be redefined dynamically by downloading new terminal software from the host. Thus, a single design can be redefined for use in a range of applications. For example, software for the terminal described above is currently being designed (by Greg Cooper at Stanford) that will extend the terminal features to simplify and speed up interaction between a physician and an established medical diagnosis aid program (**MYCIN [Shortliffe, 1976]**). It is to be done by storing (in the terminal) quick-response menus for normal interactions and complete response choices that can be called up if necessary by the physician. These two alterations will also have the advantage of decreasing the load on the remote host (such consultation programs are typically quite large).

One advantage that hardware-based designs for terminals have over software-based designs is speed. For example, primitive text-processing operations can typically be executed more rapidly in hardware than in software. Because some editing operations require more time than the time between received characters, our terminal buffers incoming characters. With a 1000 character buffer we are able to operate the terminal regularly on a high-speed line (4800 bps.) without noticeable delays in processing characters, even when using the screen-oriented editors that rely upon high transmission rates and extensive use of terminal editing primitives. The lengthy execution times of some of the terminal functions--scrolling is the slowest at 47 msec.--are primarily a result of the way in which we have organized the display memory--as a normal portion of the address space of the processor. Thus, 1840 characters must be moved, and 80 characters must be entered for each scroll. It is possible, however, to reduce the scroll time to of the order of 1 msec. if a *line-oriented* organization is adopted. A table of starting addresses for individual lines of characters is stored in a separate portion of the display memory. The remainder of the memory holds the characters to be displayed on individual lines of the display. The display circuitry then scans the table to pick up the start addresses of lines to be displayed and retrieves the appropriate bytes from addresses following the start addresses for the lines (the processor must be locked out during the access time for each of these addresses). For a 24-line display, the new scroll function entails entering 80 characters and altering 23 addresses. The trend toward faster memories and microprocessors indicates that speed will be even less a problem for future designs.

4.2 The System as a Microcomputer

An important byproduct of basing a terminal on a microprocessor is the possibility of using the microprocessor for independent computation, Since any terminal must be connected to a high quality display, and must be connected to a host computer, the major expenses typically associated with stand-alone microcomputers (i.e., an associated terminal and mass storage) pose no additional cost to the microprocessor configured as a terminal controller. The capability to download and upload data between the host and terminal allows the host to provide mass storage facilities for the microcomputer. Even large microcomputer programs represent only a small burden in terms of disk use on a typical host.

Thus, the microprocessor configured as a terminal controller is ideally suited for use as a general-purpose microcomputer with mass storage and software development facilities (text editors, cross assembler, and downloader) available on the host. The use of the microprocessor for stand-alone computation could also reduce the computing load on the host by performing computations that it would normally perform. Mass storage facilities on the host allow maintenance of several sets of terminal software, corresponding to alternate definitions of terminal characteristics available to the user. At the same time, the use of mass storage on the host for storing microcomputer software allows easy sharing of programs developed by different users with terminals connected to a common host.

There is a great deal of interest in using the microprocessor-based terminal for *local* text editing to decrease the load on the host computer and free the user from the frustrations of high-load editing (e.g., long and unpredictable delays). There is no doubt that current microprocessors are sufficiently powerful to handle the editing function, *given a suitable local file system*. However, the local file system must be carefully designed. The file systems of most large-scale host computers are designed to enable recovery in case of failure and are periodically backed-up. Thus, the user operates in a relatively *safe* and *forgiving* environment. It follows that he will expect such features in the local file systems of future terminals.

Another possibility that may be explored is *shared* editing, wherein most editing is done by the terminal, in close contact with the host; that is, the host is used for periodic storage and retrieval of pages. This approach is complex and warrants further experimental study to determine whether it can reduce the load on the host while remaining transparent to the user.

4.3 Suggestions for Future Designs

We believe that the following features are *essential* for video terminals, especially for those that are to be used for text editing.

1. The video display must support lines with (at least) 80 characters. The more easily implemented 64 is simply not sufficient for text editing; most text uses 72 or more characters. A shorter line length only leads to awkwardness for the user.
2. The video display must support (at least) 24 lines. This number is currently a standard of sorts, but more lines are always useful. Even when screen-oriented editors use fewer lines, the extra lines on the screen can be used by local programs (or for display of terminal status information as has been done in the terminal designed by the authors).
3. Variable display modes, such as underline, normal/inverse video, blinking, and dual intensity, are essential. Editors can make effective use of different display modes to indicate different editing modes or type styles. **TV-Edit**, for example, uses a double intensity I to indicate that *insert* mode is in effect. **E** uses double intensity to indicate lines that have been *attached* for movement elsewhere in a file.
4. Because of the utility of different display modes for editing functions, the most useful cursor is one that *underlines* the character and blinks. It should be XORed with the character video signal so that it shows up even for characters displayed in inverse video, and it should blink at a different rate than the character blinking rate so that it is also easily located for characters displayed in that mode.
5. A high-quality, properly human-engineered keyboard is essential.
6. The keyboard must afford a convenient method for specifying editing commands. One common method is to have a special *edit* key which, when depressed, turns on the eighth bit of any character typed, thus signaling an editing command.
7. An *alpha-lock* is similarly essential for interaction with many programming systems (e.g., **Interlisp**).
8. A local test mode is essential for isolating faults that originate with the terminal, as opposed to those that are due to the line or the host.

9, There should also be an audio indicator, preferably with a pleasant (perhaps interesting) sound and a volume control.

These features have all been essential *hardware* features. The following are essential *software* features.

10. Character and line insert and delete.
11. Erasure to end of line.
12. Blank the screen.
13. Cursor addressing, both relative (up/down, left/right), and absolute (to a specific position).

The following are *desirable* features. First, the *hardware* features.

1. The display mode should be selectable for individual characters.
2. Variable type styles, as well as display modes, should be accessible, especially with the increased availability of output devices capable of producing such type styles (such as the graphics printer on which this paper was produced).
3. Multiple cursors (or an underline display mode) are desirable to indicate editing of different portions of text on the same screen.
4. There should be the ability to locally store and recall multiple lines or screens (for later transmission to the host).
5. The capability to mix separately generated characters and graphics on the screen would be very useful.
6. The ability to accept remotely generated video (perhaps to superimpose), which is used effectively at the Stanford Artificial Intelligence Laboratory to allow multiple users to share screens of data **[McCarthy, 1967]**, is another useful feature.
7. An input device (e.g., the SRI mouse) can be used effectively for editing (as a pointer to locations on the screen).
8. Extra keys on the keyboard are a useful feature. These can be associated, for example, with the customized *macros* previously discussed.

9. The keyboard should be separated from the display, logically as well as physically.
10. Although speed is not a major problem, all primitive text-processing operations should require less than 1 msec. (approximately 1 character time at 9600 bps.) for completion. This feature would reduce or eliminate the need for buffering characters received from the host computer (see Section 4.1).
11. The speeds of the transmit and receive lines should be variable and independent (e.g., to support the 1200/150 bps. format in common use at **ARPAnet** sites).
12. If local editing is to be pursued (see Section 4.2), then local mass storage (e.g., floppy disks) is required.

The following are desirable *software* features.

13. Home cursor (top left of screen). ,
14. Erasure to end of screen (by lines).
15. Split screen modes and protected fields (e.g., for editing multiple files, or one file in different places)
16. Functions typically associated with the front panel of a computer--e.g., load/examine/deposit, and low-level debugging (an extension to the local test mode mentioned earlier).
17. Local editing: line editing and file editing (see the discussion in Section 4.2).
18. Macro definition (see the discussion in Section 4.1).

These lists of essential and desirable features indicate some trends in the underlying hardware required to support them. The remainder of this section discusses our thesis that future microprocessor-based terminals will use processors that have longer words than the 8 bits common today. We argue that 16 bit processors have a useful role in future designs, and we demonstrate some ways in which the added length can be effectively utilized.

Aside from the increased processing power of 16-bit microprocessors, the extra 8 bits per word are desirable for the display memory. Remembering that selectable *display modes* for individual characters was one feature where compromise was forced because of the 8-bit word length, we see a clear need for longer word-lengths. A 16-bit word, for example, makes such modes possible, as well as the other *desirable* features listed above (for the video display),

Figure 4.1 shows one possible organization for a 16-bit word in a video terminal. We assume that the host still transmits 8-bit characters to the terminal, occasionally transmitting 2-byte sequences per displayed character.

Bits	Function
0-6	*character code*
7-9	*type style*
10-12	*display mode*
13-14	*matrix position*
15	*cursor*

Figure 4.1. 16-bit Word Organization.

Bits 0-6 are used for the character code. Bits 7-9 indicate one of 8 possible type styles (i.e., fonts or character sets) for the character (8 is a compromise that could be adjusted either way). Variable type styles can be implemented with different character generators (for a predetermined selection of styles) and/or software-programmable memories (for dynamically alterable styles). Note that we are assuming that all type styles are such that the characters fit in the same display matrix (e.g., 7 x 16). We assume that if styles of different sizes are desired then a *bit-map* video display is a more appropriate way to implement them (see, for example, **[Baskett, 1976]**).

Bits 10-12 encode the display mode (e.g., one of the 8 combinations of normal/inverse video, blink, and dual intensity). Bits 13-14 encode the position of the character within the display matrix. The intent here is to allow subscripts and superscripts. We assume that the display matrix is larger than that required for any character that is to be shifted in this way (as it is in our current terminal design). Bit 15 indicates a cursor is to be displayed under the character--multiple characters can be underlined in this way.

5 Summary

We have discussed basic considerations for microprocessor-based terminals and illustrated our points with references to a terminal of our own design. We have further discussed the advantages and disadvantages of the *software-based* approach to terminal design and highlighted essential and useful features for inclusion in future designs.

Finally, we have discussed the advantages of close interaction between a remote host computer and a terminal/microcomputer that this approach makes possible: for customizing the terminal and simplifying software development for the microcomputer.

References

[Baskett, 1976]
F. Baskett and L. Shustek, *The Design of a Low Cost Video Graphics Terminal.* STAN-CS-76-546, Dept. of Computer Science, Stanford University, February 1976.

[DATAMEDIA, 1976]
DATAMEDIA, *ELITE 2500 Technical Manual.* Pennsauken, N.J.: DATAMEDIA Corporation, 1976.

[Flexer, 1978]
J. R. Flexer and G. Wiederhold, A Building Block Approach To High Quality Graphics. *Computer,* 1978, in press.

[INTEL, 1974]
INTEL, *MAC 80 Reference Specification: 8080 Macro-Assembler.* March 1974.

[Kanerva, 1975]
P. Kanerva, *A User's Guide To Tec/Dalamedia TV-Edit* (revised edition). Institute for Mathematical Studies in the Social Sciences, Stanford University, October 1975.

[McCarthy, 1967]
J. McCarthy, D. Brian, G. Feldman, and J. Allen, THOR--a display based operating system. *SJCC Proceedings,* Vol. 30, Montvale, N. J.: AFIPS Press, 1967. Pp. 623-633.

[Reiser, 1976]
J. F. Reiser (Ed.), *SAIL.* STAN-CS-76-574 (AIM-289), Dept. of Computer Science, Stanford University, August 1976.

[Samuel, 1978]
A. L. Samuel, and B. Harvey, *E Manual* (rev. ed.). Stanford Artificial Intelligence Laboratory, Stanford University, 1978.

[Shortliffe, 1976]
E. H. Shortliffe, *MYCIN: Computer-Based Medical Consultations.* New York: American-Elsevier, 1976.

[Waterman, 1978]
D. A. Waterman, Exemplary Programming. In D. A. Waterman and F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems.* New York: Academic Press, 1978. Pp. 261-279.

The Computer Science Department at Stanford University in Stanford, California, is a leading school for computer science. It was founded in 1965 and has consistently been ranked as one of the top computer science programs in the world. Its location in Silicon Valley makes it unique among computer science programs. The Stanford University Computer Science Department was founded in 1965 by George Forsythe. The CS department grants B.S., M.S., and Ph.D. degrees. Stanford University was founded in 1885 by a former governor of California and an entrepreneur Leland Stanford. The businessman moved to the West during the Gold Rush and made his fortune as a railroad tycoon. The university was founded and named in honor of Leland Stanford Jr., who died suddenly from typhus at the age of 15 in 1884.Â Stanford University has more than 5000 research and scientific projects sponsored by investors and patrons of art. In 2,013-2,014 USD 1.35 billion was allocated for university projects.Â There are programs for future engineers DOE Computational Science Graduate Fellowship, Ford Fellowship, Fulbright-Hays and many others at Stanford University. The scholarships range from 10,000 USD to 38,000 USD . Stanford's Department of Computer Science is one of the top computer science departments in the world. The CS department grants B.S., M.S., and Ph.D. degrees. This is an outline of the requirements for the B.S. as outlined in the Stanford Bulletin. Mathematics (23 units minimum). CS 103X, or CS 103A and CS 103B. Discrete Structures. MATH 41, MATH 42. Calculus. STATS 116 or MS&E 120 or CME 106. Probability. Mathematics electives. 6 units from. Science (11 units minimum). PHYSICS 41 Mechanics. Work in artificial intelligence and computer science at Stanford University. Home. Browse.Â 4. Milestones for Stanford Heuristic Programming Project research with DARPA Support. Physical Description: 1 document. 5. Ventilator Manager : A Program to Provide On-Line Consultative Advice in the Intensive Care Unit (Stanford Heuristic Programming Project Memo HPP-78-16) 1978. Physical Description: 1 document. Paris France Bruce G. Buchanan, Ph.D. Professor of Computer Science (Research) Department of Computer Science Stanford University Stanford, California 94305 A. Bruce Campbell, M.D., Ph.D. Practice of Hematology/Oncology 9834 Genesee Avenue, Suite 311 La Jolla, California 92037 William J. Clancey, Ph.D. Research Associate Department of Computer Science Stanford University Stanford, California 94305 Jan E. Clayton, M.S. Knowledge.Â MYCIN as History MYCIN comes out of the Stanford Heuristic Programming Project (HPP), the laboratory that without doubt has had the most impact in setting the expert-system transformation in motion and determining its initial character. I said that MYCIN is the granddaddy of expert systems.