LOSS OF DIGNITY:

SOCIAL DANGERS OF A COMPUTERIZED SOCIETY

by Jay Yablon

Submitted in Partial Fulfillment

of the Requirements for the

Degree of Bachelor of Science

at the

Massachusetts Institute of Technology

May, 1976

Signature redacted

Signature of Author _____
Depts. of Comp. Sci., Poli. Sci., Date

Signature redacted

Certified by _____
Thesis Supervisors

Signature redacted

Accepted by _____
Chairmen, Departmental Committees on Theses

Contents

Figures

Abstract

Loss of Dignity: Social Dangers of a Computerized Society

by Jay Yablon

Supervised by Dr. Joseph Weizenbaum, Professor of Computer Science

and

Dr. Louis Menand III, Professor of Political Science, Assistant to the Provost

This thesis deals with some of the important social consequences related to computer use. Although it examines a number of different areas, the primary proposition presented here is that the data banks of today, as serious as their implications may be, provide only a glimpse of the dangers that may one day occur if these data banks are linked up to artificial intelligence programs designed to make decisions based upon data in these data banks.

In addition to discussing data banks and artificial intelligence at great lengths, we devote considerable time to dispelling some of the "mythology" that has often surrounded computers in the minds of the general public.

In the final chapter, we offer short discussions of a number of other computer related issues which are of social concern.

Finally, we include an appendix which offers a description of how computers work and how they are commonly used.

Introduction-

The Side Effects of Technology

The past few centuries have seen changes in our world on an almost unbelievable scale. If one were to take a look at our world today and compare it to the world of a few centuries ago, he would find few things that have not changed. Our modes of transportation and communication have changed completely. The jobs that the average person typically hold have changed. Our physical environment has changed from fields and forests to skyscrapers and concrete. Our methods of waging war have changed. We could go on and on, but in short, it seems as though we are today living in a [redacted] ~~comp~~letely different world than ever before.

If we were to examine these changes in an attempt to find a common root among them, we would most certainly find that that root is technology. Although the tendency has always been to extol these technological changes and those who had the wisdom and imagination to create them, [redacted] the recent past has beackoned man to take a second look at his technology. We have seen that technological change is often accompanied by many other undesired and unforseen changes. And often, these undesired changes may be so bad as to negate many of the technological benefits that were originally sought.

One needn't look far to find examples of what we are talking about. The automobile for instance, brought about much

change beyond a simple increase in man's traveling speed. In fact, it [redacted] brought about major changes in man's everyday lifestyles. The auto allowed people to live farther from where they work, which in turn allowed the creation of suburbs. It created a dependence upon gasoline and upon those who produce the materials necessary to make that gasoline. It is this dependence which in turn has helped to bring about the "energy crisis" which occupies the attention of so many people nowadays. The exhaust emitted from automobiles has also become a major concern. Pollution from cars and other sources is now perceived by many as a major threat to our continued healthful existence on this planet. And we certainly cannot ignore the countless violent deaths and injuries that occur every year in auto accidents. Such is the nature of the sacrifices we have made to achieve faster travel.

To cite another example, we turn to as seemingly benign an invention as television. Today, our society suffers from an ever increasing crime rate. Although it would be ludicrous to attribute this rise in crime to any single source, a large body of respectable opinion has concluded that television contributes to at least some of the increased violence because of the example which violent television programs set for the people who watch them. Certainly, there was nothing inherent in television per se which said that it had to evolve to have many violent programs, yet a lack of foresight, among other things, did allow television to evolve in the way that it has.

In years past, the long term storage of food products was rather difficult. In response to this problem, many chemicals were developed which, when added to certain foods, could preserve those foods for a longer period of time. Today however, there are many who claim that the long term ingestion of certain of these chemicals can help to promote a number of major diseases, cancer in particular.

Another place we can turn our attention to is the nuclear engineering industry. Hopefully, we need not even begin to describe the enormous and ominous implications that the invention of nuclear weapons brought about for mankind. Nuclear power plants too, have become the subject of much controversy recently because of the possibility of radioactive leaks and the buildup of nuclear wastes. Once again, the introduction of a new technology has brought with it many dangers which may ultimately counterbalance any of the intitial gains brought about by the introduction of that technology.

We could continue to go through countless other examples of how certain technological advances have been accompanied by a melee of other problems, often with extremely dangerous implications. What is more important however, is that we begin to pick out a pattern which seems to accompany most, if not all of our technological advances. In particular, it seems that our technological advances often seem to give us more than we originally bargained for. And it is often too late before we come to realize this fact.

In all of the cases that we have mentioned, and in many others, it seems that the approach initially taken was to simply introduce a new technology into the market, wait until that technology caused problems, and then first attempt to deal with those problems. And often, we have tried to deal with those problems either by legislation or by the introduction of a new technology designed to eliminate the negative side effects of the first technology. But this approach simply hasn't worked. It operates upon the assumption that regardless of what problems a particular technology might cause, future technology and future legislation will be able to somehow "fix" those problems. It ignores however, the fact that the technology designed to "fix" a particular technological problem is likely to introduce its own negative side effects. And it ignores the fact that there are certain technological problems that cannot simply be legislated away. For instance, after we discover that chemicals in foods have negative side effects on human health, we can't simply pass legislation to restore good health to those who's health has already been damaged by these chemicals.

Also, the introduction of a new technology often means the creation of a powerful new interest group, namely those who market that technology. After a particular technology has been on the market for some time, the people who sell it will have often acquired enough capital and influence to effectively resist any attempts to legislatively change the fundamental nature of their technology. All of this makes it clear that we must concentrate

on the prevention of technological problems before they occur rather than on a cure for them after they occur.

If this discussion teaches us anything, it should teach us that when a new technology is first introduced, the question we must ask is not "will this technology have any negative side effects?" but "what negative side effects will this technology have?" This implies a shift of the burden of proof. No longer should it be up to the opponents of a new technology to prove that the technolgy in question will cause problems. It should be up to the people who wish to introduce the new technology to prove that their technology won't cause problems. Such a strategy would require an intensive effort to uncover and study the potential hazards involved in a new technology at the time that it is first proposed. If it is determined that the hazards outweigh the gains, then we must be willing and able to say "no" to the introduction of that technology.

In this thesis, we shall be taking a look at one such technology, namely computer technology. Computer technology has expanded rapidly over the past few decades, and it is quickly becoming an underpinning to much of our way of life. Yet, we have thus far only scratched the surface of what computer technology can ultimately be made to do. If the more extreme proponents of computer technology have their way, there isn't a thing that people do today which won't one day be done by computers. In many ways, the computer is our ultimate technological invention. It is, if you will, a kind of "do it yourself" technology which can

be applied to just about anything we can think of. The introduction of computer technology in many ways was not the introduction of a single new technology, but rather it was the introduction of a limitless number of new technologies for which computers can serve as the foundation.

Because the computer offers so many new technological possibilities, it is important that we try to asess some of these possibilities to determine what possible detrimental social consequences may accompany them. We must learn at the outset to distinguish between whether we <u>can</u> find a new use for computers and whether we <u>should</u> use computers in this new way. Too often in the past, the question of whether or not we are able to do something has been synonymous with the question of whether or not we should do it. What we shall be adressing throughout this thesis is the question of whether or not computers should be used to do certain things regardless of whether or not they can be made to do these things.

The first chapter, entitled "Computer Technology: The Myths and the Limitations" deals with many of the misconceptions about computers which are commonly held by the general public. It is our feeling that before any intelligent discussion about the use of computers can occur the reader should have a knowledge that is based upon facts rather than upon rumors and misconceptions.

Chapter two is about the "Computerized Data Banks" that have begun to attract the concern of many people in recent years.

In particular, we attempt in this chapter to discover some of the ways in which the use of computers to retain and disseminate personal information about individual people has served and may serve to severly impinge upon our privacy and our civil liberties.

The third chapter, entitled "The Use of Artificial Intelligence" deals with what may well be the one of the most important social consequences of computer use in the years to come. There are a number of people in the artificial intelligence field who feel that computers have the ability to "think." They feel that this "thinking" ability will one day surpass man's thinking ability, and as a result, they feel that computers should ultimately replace man in many or most of his thinking tasks. The social implications of such a situation are simply overwhelming, and we shall be dealing with them in great depth.

The fourth and final chapter, "Other Consequences of Computer Use," is simply a collection of one or two page essays on various other implications of computer use which, because of limitations in time and space, we have been unable to deal with in greater depth. Hopefully, our discussions in this chapter will spurn others to begin thinking about and studying these problems in greater depth. The author hopes to ultimately expand this thesis into a book in which many of the issues raised in this chapter will be dealt with more fully.

Finally, the Appendix contains a textbook-like description of how computers work and some of the ways in which

they are commonly used. At this time the Appendix is incomplete and some sections of it repeat much of what is said in chapter one. Although it is still in its rough draft stage we include it for those readers who are interested. Even though the Appendix requires no technical computer background whatsoever, it is written in a more technical tone than the rest of the thesis. In particular, the section entitled "How a Computer Works" will probably require a good deal of time and concentration on the part of the reader as compared to the other sections of this thesis. For those who do choose to read through this section, I strongly suggest that they make good use of the diagrams that are provided throughout the text. If the reader tries to read the Appendix without looking at and studying these diagrams, he will have a great deal of trouble understanding what we are talking about.

In many cases, statements made in this thesis will be derived from other sources, and this fact will be indicated through the use of footnotes. In other cases, statements will be made which simply reflect feelings or impressions about computer technology that the author has developed throughout four years of education in M.I.T.'s computer science department, although these statements cannot be pinned to any single specific source. At the end of the thesis, we have included an annotative bibliography which provides a short description of the pieces of literature upon which much of this thesis was based. Hopefully, this will provide some pointers to other sources for those who

wish to do further research into the social consequences of computer use.

Finally, it is important for me, as the author of this thesis, to acknowledge the responsibility that I assume by writing this thesis. One of the very crucial problems presented not just by computer technology, but by all technology is the so called "information gap." Generally speaking, the knowledge of the inner workings of any particular technology is posessed by a very limited number of people. In many cases, these people are the same people who design and market that technology. As a result, they are the ones who stand to benefit the most if the public is given a favorable picture of that technology. The rest of the general public remains largely ignorant of the principles behind that technology. Because of this situation, it is often much too easy for these few people who have knowledge of a particular technology to play upon the ignorance of those who don't have that knowledge, by presenting a distorted picture of that technology whenever they talk or write about it. Often, this distorted picture is presented with the self serving goal of creating public support for that technology, rather than with the goal of objectively educating the public about both the benifits and the drawbacks of that technology. For various reasons ranging from "classified company information" to "national security," it may be impossible for the general public to ever get an accurate knowledge of that technology because that knowledge is never released. The approach taken by the few people

with knowledge of that technology often comes across to the public as "you don't know what this technology is all about and we do, so you should leave all the decisions about this technology up to us." This type of "we know what's good for you" attitude is the source of more technological problems and more public misconceptions about technology than we could ever hope to enumerate.

From the very beginning, I have been fully and continually aware of the fact that many of the people who will read this thesis have little or no prior knowledge about computer technology. As a result, they may read this thesis and come away with the impression that everything we have said here about computer technology is an absolute fact. Although I have worked hard to give a more unbiased picture of computer technology than would be given by someone who has a direct financial interest in computer technology, and although I certainly hope that the reader will agree with most of what I have to say, let me caution against the danger of interpreting this thesis, or for that matter any piece of writing about any technology, too literally. At best, all this thesis can do is represent one person's point of view on an issue that is exceedingly complex, certainly at a level of complexity well beyond the limited understanding or experience of a single person. More than anything else, my goal is to get the general public to start thinking about some of the issues that will be raised here, for effective public action on any particular issue must always be preceded by a sound public

understanding of that issue.

With this in mind, let us go foreward and begin to discuss some of the social consequences of computer use.

Chapter one-

Computer Technology: The Myths and the Limitations

The Myth of "Perfection"

Some of the biggest obstacles which one has to overcome in order to discuss computers with the layman are the misconceptions which have often surrounded computers in many people's minds. People don't have any idea of what is going on inside a computer; they simply know that it outputs all sorts of nice neat looking results, almost as if by magic. A detailed discussion of the inner workings of a computer might help to take some of the "magic" out of them, but our purpose here is not to go through the technical details of how a computer works. For those readers who are interested in such a description, the appendix will prove to be informative. We are concerned here, more than anything else, with simply eliminating some of the commonly held misconceptions about computers which, when held strongly enough by enough people, help to contribute greatly toward many of the social problems surrounding computer use.

The first and foremost myth that needs debunking is the one that computers are somehow "perfect." Right here and now, let us state that the only thing that computers can ever do perfectly is follow instructions and store information given to them by imperfect humans. And even in this respect, there are limitations. There are many circumstances in which the data

inside a computer may become damaged or altered in some way, and hardware and other problems may even cause a computer to make a mistake in interpreting instructions. But as a rule, if one stores some data into a computer, he can reasonably expect that data to look the same way when he gets it out as it did when he put it in. And if he gives a set of instructions to a computer to follow, he can expect those instructions to be interpreted precisely by the computer. The problem occurs when people go too far in attributing "perfection" to computers. Dr. Joseph Weizenbaum, in his book "Computer Power and Human Reason," illustrates the problem well.

He wrote a program to which he gave the name "Eliza." This program, when used by people, interacted with them as though it were a psychiatrist. Although the program was not written with any intent whatsoever to psychoanalyze people, Weizenbaum soon found that many people, including his own secretary, were using the program as though it were a real psychiatrist. This type of situation is a direct result of this strange awe with which people often view computers. The feeling that computers are in some way "perfect" causes them to be taken in time and time again by all sorts of computer programs just like "Eliza." What must be mentioned over and over is that the only thing that computers will ever do "perfectly" is follow instructions and store data, subject to the limitations mentioned earlier. But people can give a computer "bad" data or "bad" instructions, and it will store that data and follow these instructions just as perfectly as it

would have stored "good" data or followed "good" instructions. The significance of this point must be fully comprehended. In particular, it must be fully understood that a "bad" method of doing something will be carried out more perfectly through the use of a computer than through any other method.

Today, there are people writing computer programs to do all sorts of things. In particular, people are using computer programs more and more to aid them in making or to even make certain decisions. But the method for making any of these decisions must be specified by a program, and that program must be written by computer programmers. And often, these programmers know little or nothing about the decision making process that they are trying to put into their program. This lack of understanding will be fully reflected in a program which will be dilligently followed by a computer. And so the decisions which are ultimately reached by the computer will also reflect fully and completely whatever misunderstandings the programmers had of the decision making process that they were trying to program.

Much more will be said on these points later on, but they are crucial enough to be brought up early and often during our discussion.

Undetectable Errors

One of the problems which is frequently encountered when using a computer is programming errors. There isn't a

programmer alive who can write an errorless program first time every time. As a matter of fact, a substantial portion of any programmer's programming time is spent finding and removing errors from his programs. During this "debugging" process, it is usually rather easy for a programmer to detect that there is an error in his program, because the program simply won't work and that fact will present itself in rather obvious form. But hopefully, sooner or later, after going through the testing and debugging process, the programmer will finally have a program that appears to be doing what it is supposed to do. It is at this point that the program will finally begin to be used to do whatever it was designed to do. But there is an important problem here that must be pointed out.

Let's say that somebody is writing a program which, for instance, can take a list of words and alphabetize that list. After writing the program and testing it for a while, the programmer gives his program the words "boy," "cat" and "apple" and it alpabetizes them correctly. He gives it the words "man," "woman" and "car" and it again alphabetizes them correctly. And after a few more similar tests, all of which produce a correctly alphabetized list, the programmer concludes that his program works. So he tells whoever he is writing the program for that it works, and that person begins using the program to alphabetize words. But when that person gives the program the words "bat," "boy" and "ball," all of which begin with the same letter, it suddenly doesn't alphabetize them correctly because the

programmer wrote a program which only looks at the first letter of the words that it is attempting to alphabetize. Then, he tested the program by giving it only words beginning with different letters. As a result he became convinced that his program always worked. We point this out because it brings out something very important about the nature of the errors that one often finds in a program or in a computer system. Many times, a program will be written which appears to work all the time, even though it really doesn't. Instead it works almost all of the time, yet it doesn't work when some special condition occurs which has been overlooked in both the writing and the testing of the program. And it is not an error resulting from faulty instructions. It is an error due to the fact that the programmer did not have a full understanding of the problem that he was trying to solve when he wrote his program.

The "special case" of two words beginning with the same letter is hopefully not the type of case that one would overlook, but it shows how in a more complicated program something might indeed be overlooked. If a program just plain doesn't work under any circumstances it is easy enough to detect that fact when the program is being tested, and nobody would use the program to do anything important unless they were looking for trouble. But if the program does appear to work when it is tested, yet in reality it contains an error that occurs only in an obscure situation which was overlooked, then we have a problem. People will use the program under the assumption that it works all the time, when in

fact it doesn't. And often, the process of finding the rare cases for which a program doesn't work involves simply waiting around until somebody stumbles onto one of these rare cases while actually using the program. Sometimes, even this isn't good enough. Often an error will occur and the person using the program won't realize it. He may thus accept an incorrect answer from the computer, thinking that it is correct. Herein we find the crux of one of the problems that will be important in our discussions later on: that computer programs frequently contain errors which can result in undetectable mistakes. If these errors were at least detectable, it would be okay, since nobody would use the erroneous results. But if a mistake is undetectable, as it often is, we then have the very serious danger of people relying upon "wrong" answers from a computer. These wrong answers may in turn have serious effects. There are many cases where such a situation has caused untold misery for the people and organizations involved, and it deserves much serious consideration.

The Security of Information

To many people, a computer is simply a big mysterious machine with a "typewriter" attached to it which people can use to put things in and get things out. Although this is partially true, it is not totally true.

If we were to take a computer and attach a single one

of these "typewriters" (which are actually called "teletypes") to it, then only one person would be able to use this computer at a time. But the fact is that in general, a single person using a computer will only make use of a small fraction of that computer's total "computing ability." In other words, for a small fraction of its total time, the computer would be doing work for the person who is using it. During the rest of its time, it would be idle.

Because a computer is generally so expensive, it seems rather uneconomical to have it work for only a small fraction of the time that it is capable of working. As a result, many of the computers on the market today have more than one of these "teletypes" attached to them. This way, a number of people can use the same computer at the same time. Since each person only needs a small fraction of the computer's time, the computer can go back and forth among the various "users" and give each one a little bit of its computing time when he needs it. In this way, the computer comes closer to working up to its full capacity.

Of course, if one of these computer "users" required all or most of the computer's computing time, than we wouldn't be able to divide up the computer's time in this fashion. But since in general, each user only requires a small fraction of the computer's time, we are able to get away with such a "timesharing" scheme.

Because of their ability to accomodate more than one user at a time, many of these "timesharing" computers have the

ability to share programs and information among a number of users. But often people will have things inside a computer that they only want to share with a limited number of people. For instance, a group of people working together on a program might want to be able to share copies of that program among themselves so that they can all work on it at the same time. But they wouldn't want other people using the same computer system to have access to that program until the program is finished and working. And even then, they may still not want others to see that program. For instance, they might only want others to be able to use that program without being able to look at it to see how it works. Or they may want to simply deny any form of access to that program. Sometimes, people store sensitive or private information on a computer and only want a few people to have access to that information. For instance, if some state government had a computer which was shared by both the "Payroll Department" and the "Motor Vehicle Department," and if the Payroll Department's program printed out paychecks for all state employees every Friday at 3:00, we might want the person using the Payroll Department's program to have access to the salaries of all city employees. Yet, we certainly wouldn't want the person running the Motor Vehicle Department's program to have access to the salaries. If he did have access to them, he might try to change the part of the computer with his salary in it from reading "$250 per week" to reading "$10,000 per week" every Friday at 2:55. Then at 3:00, the Payroll Department program

would print out paychecks for all city workers, including a $10,000 check for the person who changed his salary. Then, at 3:05, this person can change his salary back to $250, so that nobody will ever know the difference. Although such a ripoff technique seems too simple to be true, many people have in fact committed computer aided crimes using very comparable techniques. "What bothers many is the simplicity of the (computer) fraud and embezzlement schemes that heve come to light so far. 'One can't help but wonder what the really clever people are doing,' says (Brandt Allen, a professor from the University of Virginia)."-1 As a result, some computer systems are being designed today which allow a person or organization to specify exactly who has access to the various information it stores on the computer, and how much access he has. (Can he only read that information or can he also change it. Can he make his own copy? etc.) But very few computers on the market today are really secure. "Joe Wasserman, (a pioneer in the computer security) field, says, 'Computer security in general stinks. Computer centers tend to be either completely secure or completely insecure.' He says most of the secure ones are companies dealing with classified government work- 'and that's a pretty small number.'"-2 There are in fact many organizations which, for precisely this lack of security, have decided to buy a computer all for themselves so that nobody else can get at restricted information and cause problems.

As we can see, the sharing of information in a computer brings out a lot of social problems that must be dealt with. We

must remember that, in a way, the computer has taken the place of the file cabinet. Information that would have been put in a locked file cabinet or a desk 10 years ago is today put in a computer. Certainly, no employer would have let his employees into his file cabinet ten years ago to change their salaries. A person writing a book would not want someone to break into his desk and steal copies of that book. A doctor hopefully wouldn't want an outsider to get at sensitive medical information in his files. But nowadays, businesses keep salaries in computers, writers write books using computers, and doctors keep records on computers. It seems reasonable for them to expect at least the same amount of security for their property when it is stored on a computer as they would have expected if it had still been kept in a file cabinet. The problem is this: In the initial rush for computer companies to produce working computers and get them on the market, the technological push was directed almost entirely toward simply getting the computers to do useful things. According to "John Weil, who heads Honeywell's advanced systems and technology unit... 'the (computer) industry has tended to neglect the security problem in its haste to develop basic computer technology.'"-3 The boom in the popularity of computers saw more and more people and organizations computerizing their operations, thus putting more and more information of various sorts on computers. Gradually, these people began to realize something. Although many technological advances had been made in the area of "getting the computer to do something," very few

advances were made which could adequately protect the information which people were putting into the computers. This lack of protection encouraged many people to begin stealing or illegally changing information stored by others on the same computer. Some of the computerized ripoffs that have occurred in recent years boggle the imagination in their scope and simplicity. Computer technologists have begun to respond to the problem, and there is a growing field in information protection. But the fact is that very few of the computers on today's market adequately protect the information they contain. And crime is not the only concern here. Because computers often contain sensitive material on individuals, this lack of security creates a number of problems in the area of personal privacy.

Nowadays, a large amount of computerized information is sent over telephone lines. This provides an excellent way for someone to tap into a computer illegally to do things that he shouldn't do. The computer designers have started to respond to this problem by producing "scramblers" to scramble computerized information that is transmitted over telephone lines. But the scramblers provide no guarantee that someone won't find a way around them, and the fact is that today, much computerized information is still not scrambled when it is transmitted over telephone lines.

There are also physical hazards to be reackoned with by any computer installation. Some of these hazards are common to all industries; others are unique to the computer industry.

Fires, physical destruction (intentional or not) and most other day to day hazards certainly face any computer installation. But the unique hazards of the computer industry provide even more potential for problems. For instance, "One west coast data processing manager lost his job when a group of Boy Scouts touring the computer center happened to have some magnets with them that erased most of the company's records stored on tapes. In another case, a repairman stuck his magnetic flashlight to the nearest support- which happened to be a storage drum. The result: 80,000 scrambled customer credit records."-4 Other obscure things, such as various types of radiation, also have the potential to damage a computer system.

"(In 1970) millions of dollars worth of computer equipment and data were damaged and destroyed by sabotage alone. Add to that increasing instances of computer misuse- such as fraud and embezzlement- and serious accidental disruptions, and it becomes clear why some observers see trouble ahead in the computer age."-5

The Need To "Trust" the Computer

To many people, the computer is nothing more than a magical black box. They put things into that box and they get things out of it, yet they have no real idea of how the things they put into it were used to determine the things they got out of it. As a result, they must often accept on faith that what the

computer is telling them is really accurate. This "faith" which people must place in a computer before they use it underlies an important problem, and so we offer an example to illustrate exactly what we mean by "trusting" a computer.

Let us say that we wanted to write a program to simulate a game of five card poker. How might we approach the writing of such a program?

Certainly, it seems that at some point in our program, we would have to write a routine that simulates the dealing of five cards from a card deck. Because there are 52 cards in a deck, let's take each card and give it a number from 1 to 52. The number 1 might represent the two of clubs, the number 2 might represent the two of diamonds, the number 52 might represent the ace of spades and so on. If we assume that there is some way to get the computer to pick random numbers from 1 to 52, then the process of dealing out five cards would be fairly simple. All we would have to do is have the computer pick five different random numbers from 1 to 52. Then, we can simply have the computer figure out which cards are represented by the numbers that were picked, and those cards will be the cards that have been "dealt" out. But there is a problem here: it is very difficult to write a good program to produce random numbers. Because of this, the person trying to write the section of the poker program which deals out 5 cards would end up spending virtually all of his time writing the section of the program which picks random numbers. And before he even begins to write this random number section, he

would probably have to spend a lot of time reading through a few books or papers to learn all about generating random numbers. But the person writing the poker program doesn't want to spend most of his time researching and writing random number programs; he just wants to write his poker program. He wants to use the random numbers once they are generated, but he could care less about writing a program to generate them. Certainly, this is a reasonable desire. An auto repairman doesn't want to worry about building a muffler before he puts it into a car; he simply wants to have a complete muffler on hand, ready for use when it is needed. He lets someone else, namely the people at the muffler factory, worry about actually building the muffler. Of course, there is always the chance that the mufflers sent from the factory will be faulty. In this case, even though the auto repairman might install the muffler properly, the car will not perform the way it should. If the random number subroutine that we use for our poker program is faulty, (for instance, if it picks the number 52 more than the other numbers, causing the ace of spades to be dealt out more than the other cards) then the whole poker program will be faulty, even if we use the random number subroutine properly. And if the writer of the poker program doesn't know that the random number program which he is using doesn't work properly, then he will think that his own program works properly, even though it doesn't.

Today, most programmers often make use of many such "pre-packaged" subroutines. And since these subroutines may not

always behave in the way that they are expected to, this adds a strong element of doubt to the correctness of a program.

Another example of a way in which people must "trust" a computer is in the use of what are called "high level languages." Before discussing high level languages however, an understanding of a few technical concepts is in order.

Anything and everything that we put into a computer must ultimately be represented by a bunch of what are called "bits." A bit is simply a piece of electronics inside the computer which is either on or off. Just like your televisions, radios, lights, cars and lots of other things are either on or off, so is a bit. Probably the best way of thinking of a bit is as a little light switch which is turned on and off by the electronics of a computer rather than by a human. A computer has a tremendous number of these on-off bits, and everything that it does is ultimately determined by which bits are on and which bits are off.

Since we want to get the computer to do certain things for us (arithmetic, bookkeeping, handling telephone calls etc.) we must find some way of taking whatever it is we want to do, and translating it into some sequence of these on-off bits.

Let's now begin to take a look at how one might go about organizing these little on-off bits to get a computer to actually do whatever we want it to do.

As we have said before, a single bit can be either on or off. Let's say that we were to take two of these bits and put

them side by side. Now, we have four possible ways of arranging these two bits. They are 1) off-off, 2) off-on, 3) on-off and 4) on-on. Since it becomes rather tedious to write the words "on" and "off" everytime we want to talk about a bit, we'll use the digit "1" to represent the word "on" and the digit "0" to represent the word "off." So our new way of representing these four on-off patterns is 1) 00, 2) 01, 3) 10 and 4) 11. If we have three bits, then we can come up with eight possible ways of arranging them, and four bits provide us with sixteen ways. As we can see, it seems that everytime we add another bit we double the number of ways of arranging them. This is in fact the case, so five bits can give us 32 arrangements. If we want to represent, say, the 26 letters of the alphabet, we could simply take five bits and use one of the 32 possible bit arrangements to represent each letter. One such assignment might be as follows:

A 00000	J 01001	S 10010	UNUSED
B 00001	K 01010	T 10011	11010
C 00010	L 01011	U 10100	11011
D 00011	M 01100	V 10101	11100
E 00100	N 01101	W 10110	11101
F 00101	O 01110	X 10111	11110
G 00110	P 01111	Y 11000	11111
H 00111	Q 10000	Z 11001	
I 01000	R 10001		

Figure 1

Now, anytime we want to represent say, the letter "L" in our computer, all we have to do is find a chunk of five bits somewhere inside the computer and put them in a "01011" or off-on-off-on-on sequence, as shown in the chart.

The same procedure of arranging bits in various on-off patterns is also used to represent numbers, instructions and virtually anything else that we wish to put in a computer. For instance, the bit sequence "0101101011011011110" might just happen to be an instruction to the computer telling it to add two numbers which are stored somewhere inside the computer. These two numbers would also be represented by a bunch of bits in a similar fashion. For instance, the bit sequence "0111101100010110110" might happen to represent one of the two numbers and the bit sequence "1001101000111000101" might happen to represent the other number.

But, as the reader can probably imagine, it is a tremendous inconvenience to have to think of everything inside a computer in terms of these bits. As a result, much of the early research in the computer field centered around finding ways to use a computer without having to think of everything in the computer in terms of these bits. In particular, people found it much more convenient to give the computer "englishlike" instructions such as "add" or "subtract" than to give it "bit form" instructions such as "0101101011011011110." And it is likewise easier to give the computer real numbers instead of these tedious bit strings. So the computer industry developed programs which would enable people to give the computer instructions, in numbers and letters this easier to understand "englishlike" form. These programs act as translators in a sense; they take instructions which are written in "englishlike" form and

translate them into instructions which are written in "bit form." And it is these translated "bit form" instructions which are ultimately used by the computer to do something.

These "translator" programs are generally referred to as "assemblers" or "compilers." The easier to use, more "englishlike" instructions comprise what are known as "assembly languages" or "higher level languages."

Now, as we said before, the purpose of designing these translator programs was to free the programmer from worrying about the exact form of "1's" and "0's" that his program would ultimately take on inside the computer. Instead, he can leave the details up to these "translator" programs. As a matter of fact, almost all programming done nowadays utilizes some higher level or assembly language of the form described here. But high level languages often have so many different features that it is impossible for a person to always know exactly how his high level language instructions will look once they are translated into bit language instructions. He must often trust the compiler to translate his statements into a bit language program that will do what he really wants it to do. But if we are writing a program that will, for instance, be used to make important decisions about people's lives, it may not always be wise to leave so many details up to the computer. This is similar to the problem mentioned earlier where a person might use a defective random number generator without knowing that it is defective, and have his program work improperly up as a result. In a more general

sense, we are dealing with the problem that people often allow computers to do things for them without really knowing how it is doing them or if it is doing them correctly. We shall spend a lot of time later on discussing some of the social implications of this situation.

The Construction of Large Scale Computer Programs

The recent years have seen the production of many gigantic computer programs; programs which require a massive effort by many people in order to be produced. Because such large scale programs are becoming more and more common by the day, it might be illustrative if we discussed the process by which one of these large programs is often produced and some of the problems encountered along the way.

Earlier, we showed how the use of pre-packaged "subroutines" can help to make programming easier for those who use them correctly. But the notion of a subroutine has much impact beyond this. Consider for instance, the task of writing a baseball game on a computer. For simplicity, all games will last nine innings regardless of score. Before starting however, we can make a few observations about the structure of a baseball game. For instance, if we can write a subroutine which we shall call "Playinning" which is capable of playing a single inning of baseball, we can use it nine times in a row to produce a full baseball game. So instead of concentrating on the whole game,

let us concentrate on [redacted] one inning.

If we can write a subroutine which we shall call "Teambat" which allows a single team to bat until it has three outs, then we simply have to use the "Teambat" program twice, once for the visiting team and once for the home team, in order to play a full inning. So instead of concentrating on a full inning, let's take it a half inning at a time. Well, what is involved in having a single team bat for a half inning? All we really have to do is write a program called "personbat" which allows a single person to bat. Then, by using this routine over and over until a team has 3 outs, we can produce a half inning. Notice what we have done. We have taken the task of writing a full baseball game and reduced it to the task of writing a subroutine which enables a single person to bat.

Now, how might we write this subroutine that allows a single person to bat? One approach might be as follows: Let's say a player has a total of 500 at bats with 40 home runs, 5 triples, 30 doubles and 75 singles. We could use a pre-packaged random number routine to pick a number from 1 to 500. Then, if the number picked is from 1 to 40, we can call for a subroutine called "Homerun" which will give the batter a homerun. If the number is from 41 to 45, we can call for a subroutine called "Triple." 46 to 75 will cause us to call for a subroutine called "Double," and 76 to 150 will invoke a subroutine called "Single." Any other numbers picked will send us to a subroutine called

"Out." Each of these subroutines will in turn call on even more subroutines. For instance, when the "Double" subroutine is called, that subroutine might itself call another subroutine called "Baserunning" which will figure out what happens to all the baserunners. And that baserunning subroutine might even call another subroutine called "Keepscore" which will update the score if the baserunning routine determines that someone has scored. This technique of breaking a program down into smaller, and thus more manageable parts is known as "structured programming" and is perhaps the most powerful programming technique which has been developed to date. Its obvious advantage is that it limits the scope of what we have to think about at any one time to a very small section of the overall program. Then, once each of these smaller subroutines is written, all we have to do is piece them together into a full program. In a sense, we are breaking down our program into "ready made parts" which are comparable to the subroutines discussed earlier. This approach results in programming that is much faster and much easier than it would otherwise be if the person writing the program had to worry about every detail of the whole program all at once. And if the program isn't doing the right thing when a person gets a triple, the author of the program merely has to look at the triple subroutine to find his error, rather than to search through the whole program. And finally, if more than one person is working on the program, this technique provides a natural way to divide the work. One person can write the single and double subroutines, one

person can write the triple and home run subroutines, and so on. These subroutines can then be put together to form the whole program, even though each person does not have to have any idea of how the sections written by the other people actually work. It is this particular feature of structured programming that led us to a discussion of the topic in the first place.

Most of the large programs that we have been and will be talking about cannot possibly be designed by one or even a few people because of the sheer size of these programs. Because we need to have so many people working on a large program, we are forced to divide up the work in such a way that each person can work on his own individual subsection without ever having to know how the rest of the program works. As we have seen, structured programming accomplishes this task rather well, and is thus an invaluable technique for large scale program design.

This is not to say that structured programming techniques have been used in the past to design large programs; in fact, they generally haven't, since they have only been developed recently. But then again, it has been very difficult in the past to turn out very large programs that really work. The advent of structured programming however, promises to make large scale programming efforts much easier in the future.

But even so, the design of a large scale program or any program for that matter remains an exacting exercise. One can not make a mistake in writing a program and expect the computer to figure out what he is really trying to do. The computer is a

merciless judge of program correctness. Getting a large number of people to design such exacting pieces which in turn must fit together in a very exacting way is far from a simple task, and good management and good communication are as essential as good programming.

One phenomena that is often found in large programming ventures is that the structure of a programming organization often takes on the structure of the program that the organization is writing. If the program they are writing is initially broken down into five smaller parts, then the organization will itself split into five groups, one to work on each part. Each of these groups will probably have its own boss, as will the entire project. Let us say that one of these five main parts of the program is itself broken down into four smaller parts. Then the group working on this part will itself be broken down into four smaller groups, each with its own boss and so on. This will keep happening until we reach the level of "one person groups."

The rule in almost all programming efforts, be they large or small, is that the design of a program frequently changes as soon as people start working on the nitty gritty and discover that there is an easier approach than the one that was originally proposed. If it were suddenly discovered that the above program could be better written with four main parts instead of five, we would have to change the top level of our organization from five main groups to four main groups. This means that the boss of the fifth main group loses his boss

status, and the members of that fifth group must either be relocated or laid off. Because such a situation arises so frequently in programming, such an organization must be able to change its structure radically and frequently. If the boss who was moved in the reorganization doesn't like the fact that he is no longer a boss, we can either tell him "tough luck" or we can decide to stay with the original five part breakdown of the program, even though the program might not turn out as well with five parts as it would have with four. What we have assumed here is that the change to four main parts will produce a program to do exactly the same thing that the program with five parts would have done, except more efficiently. It often is the case however that the overall goals of a program which is being designed are changed to meet the programming needs. If a certain feature which was originally included in the programming goals turns out to be difficult to program, that feature may simply be dropped from the overall goals. Thus, there is often a great difference between what a program is originally intended to do, and what it actually does once it is finally procuced. This doesn't mean that the final program doesn't work. It simply means that it doesn't do what people had originally hoped it would do, in the way that they had hoped it would do it. This [redacted] leads us into another major problem. When a large program is designed for use by some organization, the program simply does not do things in the same way that that organization did them before. As a result, an organization is often forced to redesign itself to fit a

computer, rather than the other way around. This phenomena of an organization restructuring itself around its new computer has occurred time and time again, and it is thoroughly documented in the literature. If a program is being designed with some social purpose in mind, social goals in the area with which the program deals may be changed not because they are invalid social goals, but because they are simply not easy to program. This situation can cause very severe problems, and will be discussed at great lengths when we get into artificial intellegence. There is a strong desire among many artificial intellegence researchers to eventually use computers for making important decisions. Certainly, we don't want the process of making complex social decisions to be determined by how easily these decision making processes can be programmed.

Another thing that we might notice here is that there is no one person who really knows what these large programs are doing. There are people who know what certain sections of the program are doing, but it is simply impossible for a single person to be aware of all the details of the whole program. After all, the whole reason that we split up the program among so many people in the first place was that it was simply too complex for one person to comprehend. Also, the computer industry has few, if any standards for program documentation. Program documentation is simply the process of writing, in addition to a program, a description of how that program works. The lack of a good description of how a program works makes it difficult if not

impossible for anyone to ever figure out how that program works. Add this to the fact that shortly after the completion of a programming project, the people who worked on it generally are not available or don't remember in detail how their own sections worked, and we see that our large scale program is really nothing more than a mysterious black box. People put information in and get answers out, but they have no idea how the information they put in was used to arrive at the answers they got out. Thus, a person using a computer must be willing to place blind faith in the program that he is using. If he wants to find out how it is arriving at its answers, he can't, since nobody really knows. And if he did know how the answers were being arrived at, he might just decide to never use the program again!

Since it is difficult to examine someone else's program in detail, it is also fairly easy for someone writing a section of a program to change around a few instructions to purposely subvert the entire program. As we have said before and will say again, computers provide a means for crime that is unparalelled anywhere else.

What all of this results in is a reliance upon a program that nobody fully understands, which may or may not solve the problems it was designed to solve in the best way possible, and which may have accidental or malicious flaws in it. The original goals of the program may have been changed simply to make it easier to write the program. And even if the program does solve the problem that it is attempting to solve in the best

way possible, (which we will have no way of knowing for sure) that method of solving the problem may become obsolete within a short time when some better method comes along. And of course, we will have no way of detecting this obsolescence, because we don't know for sure what the program is doing to begin with.

Hopefully, this discussion has helped to refute the myth that "somebody somewhere must know what a computer program is really doing because somebody somewhere had to write the program." The fact is, that many computer systems are entering the market today which are not understood by any one person and which, in fact, may never be understood by anybody.

At this point in the discussion we hope that the reader has more of a realistic view of what a computer is and what its limitations are. In particular, we hope that he understands some of the problems that often lie behind the practical production of any program. He should begin to see by now that computers are only human creations, and as such, are subject to the [redacted] flaws and limitations, and even the corruptions of the people who create and use them.

With this in mind, we shall move on to a discussion of some of the important social questions surrounding computer use.

Chapter Two-

Computerized Data Banks

"What Do I Have To Hide?"

Nowadays, when one speaks of the social issues involved in computer use, the issue that most likely comes to mind is the databank issue. Although it is only one of many social issues related to computer use, it is one that has begun to receive considerable attention in recent years. A data bank is, essentially, any organized collection of data pertaining to individual people or organizations. A number of important studies on the use of computers to collect and disseminate personal information have been done in recent years. Among them are "Databanks In a Free Society" by the National Academy of Sciences, "Records, Computers and the Rights of Citizens" by the Department of Health, Education and Welfare, and "The Assault on Privacy" by Arthur Miller. Recent revelations about the nature and scope of computerized files about individual American citizens which are maintained by the F.B.I., the C.I.A. and the Army, as well as by other governmental and private bodies have served to fan the fires under the data bank issue.

Although the data bank question has to date received more public attention than most of the other social questions related to computer use, that is only because it is one of the first computer abuses to have become pervasive enough to attract

such attention. But if one tries to "look into the future" to get a feel for some of the computer related social problems that may one day be confronting us, the data bank question, as serious as it is now, seems dwarfed by comparison.

Yet, the data bank issue is currently a very important issue, and it is one that will persist for a while to come. So let us take a closer look at some of the important problems that have developed as a result of the computerization of data banks, and see if we can come up with some feasable approaches for dealing with these problems.

Before we even start our discussion of data banks, it is very important to clear up a major misconception which a lot of people seem to have. When one talks with others about the databank issue, a remark that is heard all too often is "you can't get hurt by your record if you've got nothing to hide." This statement just is not true! In case after case, people have been hurt by the misuse of data about them when applying for jobs, schools, government services and just about anything else that one applies for these days. And in many cases, the people who were hurt were people who had absolutely "nothing to hide."

"In one case, a communtity tutoring project's secretary called a school to find out what grade a child was in. The principal responded to the request and offered the additional information from the record that the child was a bedwetter and his mother an alcoholic with many boyfriends. A mother of a junior high school boy sneaked a look at another school record.

She found that a teacher in second grade had said her son had "exhibitionist tendencies". After considerable effort the woman tracked down the teacher who had, by then, left the school system. The "exhibitionist tendencies" label had been pinned on her son because of a single incident in which he had rushed out of a lavatory unzippered."-6

In another case, a teenage diabetic had a diabetic siezure in a grocery store. "He grabbed something sweet to eat and, as a result, found himself under arrest. When the reason for his "theft" became known, the charges were dropped."-7 Yet, this "theft" was maintained on his arrest record, and when he brought suit to have the record of that arrest destroyed, he lost. And as a result, that arrest will follow him for the rest of his life and will be shown to prospective employers and many other people with whom he will have to deal.

In yet another case, a report sent to an insurance company from a credit bureau said that a particular couple earned "only $5000 a year between them. Their 18 year old son, a "hippie type youth" is "active in various anti-establishment concerns." The son, who would be driving the car, is "suspected of using marijuana on occasion." ...And so, with that information" this family's insurace policy was cancelled.-8

"There was only one problem. The information... was all wrong." The husband "was then an Oldsmobile salesman and his wife is a secretary. They make a lot more than $5000 a year. Their son is described by his principal as a "model student, a

straight kid,' whose 'anti-establishment' activity consisted of participating in a couple of protests against the Vietnam War."-9

Another case in which very many people were hurt by their records was with the use by the armed forces of SDN and SPN discharge code numbers. "SDN is the air force's acronym for Seperation Designation Number. The army and navy equivalent is SPN, which stands for Seperatior Program Number. Every veteran discharged between 1955 and March 1974 has an SPN or SDN number on his discharge papers. Most of the 530 numbers used were free of stigma. But in 1973 alone, 35,640 men who got out with honorable discharges had 'unsuitable' SPN numbers; 21,000 were coded as 'character or behavior disorders'; another 10,000 were branded: 'defective attitudes, and an inability to expend effort constructively.' Other secret codes said a veteran had 'homosexual tendencies' or was a 'shirker' or was guilty of 'disloyalty or subversion' or 'unacceptable conduct.'"-10

"In all, about a million veterans have been given derogatory discharge codes. And, although the veterans themselves did not know the meaning of the codes, the personnel departments of Firestone, Boeing, Chrysler, Standard Oil of California and many other major employers did. They knew what to look for on a veteran's discharge papers."-11

"Before the meaning of some of these codes was first publicized in March 1973, victimized veterans had no idea what they meant. Publicity did not reach everyone affected. Many honorably discharged veterans with stigmatizing numbers are

probably still mystified by the difficulties they have had in getting jobs."-13

But the problem was not simply the secrecy of the codes but the labels themselves."-14 "Evidence is emerging that the code labels have sometimes been parting gifts from vindictive officers or simply clerical errors."-15

Hopefully, the reader has begun to realize by now that one really does not have to have "something to hide" in order to be hurt badly by the misuse of data maintained about him. If however, the reader is still unconvinced of this fact, I refer him to Aryeh Neir's book "Dossier: The Secret Files They Keep on You" from which these examples came and in which one can find countless other examples like them.

Of course, many of the examples cited above simply illuminate problems that can result from the misuse of any type of data, be it computerized or not. For instance, the discharge codes which we mentioned were put right onto a person's discharge papers rather than into a computer, yet they did untold harm to those who were unfortunate enough to get hit with a derogatory code number. In fact, two of the "largest and most harmful data banks... -The F.B.I.'s Identification Division and the Retail Credit company- are both old fashioned manually operated systems. They are being computerized in order to speed the process with which they furnish information."-16

What we have tried to get across here is that any data, be it computerized, manual or otherwise has the potential to do

great harm to someone if that data is misused. What we shall attempt to do henceforth is examine some of the ways in which the introduction of the computer into the data bank business has helped or might help to aggrevate this problem far beyond its present proportions by introducing new factors and capabilities into the process of data maintainance and use.

Misuse of and Inaccuracies in Computerized Data

Gathering the Information

As we have stated previously, the source of many computer related problems is the fact that many people accept the highly false notion that "if something comes out of a computer, it must be accurate." Often people who use computerized data rely upon that data to make important decisions about the people who the data pertains to; whether to give them jobs, whether to admit them to schools and whatnot. But there are, in fact, numerous ways in which data stored inside a computer can be inaccurate or misleading and so it is necessary to explore some of them.

The time that any piece of data first comes into being is with the initial collection of that piece of data. This data collection process can take on any number of forms. It may or may not be done with the knowledge of the person whom that data is about. And even if a person does know that certain data is being collected about him, he may not be too thrilled with the

idea. Questionairres or various subjects are widely distributed throughout our society. Nowadays, one has to furnish information about himself just about every time he applies for anything, and the questions asked can be pretty sensitive at times. "Consider the documented case of an 18 year old college coed applying for a summer secreterial position with a federal agency. She was asked, regarding a boy she was dating: 'Did he do anything unnatural with you? You didn't get pregnant, did you? There's kissing, petting and intercourse, and after that, did he force you to do anything to him, or did he do anything to you?'"-16

The testing of children also provides lots of juicy data for the data mongers. Personality tests, in particular, offer a striking example of just how far some people have gone. Many of these tests contain questions like "Are you troubled by the idea that people on the street are watching you?," "do you think something is wrong with your sex organs?" or "do you think that Jesus Christ was greater than Lincoln or Washington?"-17

"In one particularly insensitive experiment, University of North Carolina sociologists imposed a thirty one page questionnaire on seventh and ninth graders in Durham. Inquiries included the following:

Was the home your parents made for you ever broken up?

If the home your parents made for you was broken up, whose fault was it?

How do your parents feel about white (black) people?"-18

Such surveys are often compulsive, and even if they are not, the people who are asked to respond to them are often given the strong impression that are. Often, people may be misled into thinking that they are responding annonymously to surveys when in fact they are not. It was recently revealed in several national newspapers that many major U.S. magazines were sending out questionairres to their readers asking fairly probing questions about social and political attitudes. The people responding were not asked to put their names anywhere on the questionaires, and the format of the questionaires would be such that these people would think that their responses were annonymous. Actually however, there would be a code number printed in invisible ink on each questionairre through which each person could be identified, although the existence of this code number would not in any way be mentioned or implied in the questionnaire.

Many times, the source of information about a person is not the person himself, but other people who know him in some way. Often, this practice can take on some very shady forms. For instance, one professional information gatherer said of his work "You go to a neighbor and establish rapport... Then you ask 'What's your opinion of X's home life; how do you think of him as a family man?' This will usually illicit some hint. ...Then you start digging. You press them as far as they will go, and if they become recalcitrant, you go somewhere else."-19 There are also "several politically orientec groups, such as the right-wing Church League of America, who make their investigative

talents available to employers with the claim that they can weed out 'undesireables' and 'troublesome individuals.' A more blatent form of blacklisting is difficult to imagine."-20

With all the problems that can occur when information is first gathered, it becomes clear that we must establish unambiguous standards for the collection of information. We are, of course, speaking of all information, whether or not that information is to be computerized. However, we must be particularly careful with the collection of computerized information because of the way in which many people often view computer output as gospel.

Entering Information Into a Computer

Once information about a person has been collected for use by a computer, that information must actually be put into the computer. This process will often involve typing that information directly into the computer or typing it onto punch cards for use by the computer. At this stage in the process, a number of things may happen. Of course, everyday human error on the part of the person typing the information into the computer can cause inaccurate information to be entered into the computer. Nowadays, it is rare indeed to find someone who hasn't hac a bill or an order for merchendise which was fouled up by a computer. In reality, there is a pretty good chance that the error occurred when that information was first entered into the computer.

Another problem is that raw information is often changed into a form that will fit the program which is going to use that information. We have all had the experience, for instance, of receiving questionairres that restrict our answers to only a few multiple choice possibilities, even though the answer that we might really wish to give doesn't in any way fit the choices that we are allowed to make. So we are forced to use the answer that comes the closest to what we would really like to say, even though it may be a pretty poor representation of the answer we would actually give if we were given the choice. When this "standardization" process is performed on personal data by a person whose job it is to type that information into a computer, vast distortions of that information are likely to occur. The "standardization" process may involve rewording the original information, or even adding to or deleting parts from it. This too, must be considered in determining appropriate policy to deal with the computerized data banks.

Information Inside the Computer

Once a piece of information is inside a computer, there is still a chance that it will be altered in some way. That alteration could be the result of an error in the program that uses the data, and we must remember that errors in programs are not always accidental.

As we have stated before, there are very few computer

systems on the market today that provide any real measure of security for the information which they contain. This means that there is always the chance that sensitive or important information will be seen or even changed by someone who should not have access to that information. And, unfortunately, there are no existing standards for the protection of information in computer systems. Finally, the technical field of information protection is still far from advanced, as we have stated earlier.

Many of these problems relate closely to the ones that we discussed in chapter one in our section on "The Security of Information," and they are of crucial significance.

The Ultimate [redacted] Use of Computerized Information

Finally, we come to the question of how information which is stored inside a computer is ultimately used. This particular stage of the record keeping process is possibly the one that is subject to the greatest abuse. In many cases, raw information about people is given out to just about anybody who desires it. These include prospective employers, creditors, and many other people who have some interest in the person who they are requesting the information about. In fact, in many cases, the only person who cannot see certain information is the person who that information is about. But the introduction of the computer into information handling has allowed for some very unique ways to distribute this information.

One of the big differences between computerized data banks and manual ones is the speed with which information can be retrieved and the amount of information subject to this fast retrieval. Where it might take an appreciable amount of time to locate some data in a manual data keeping system, it generally takes far less time to locate that same piece of data in a computerized system. "For example, there is not the slightest doubt that it is technologically possible today, especially with recent advances in mass storage memories, to build a computerized, on-line file containing the compacted equivalent of 20 pages of typed information about the personal history and selected activities of every man, woman and child in the United States, arranging the system so that any single record could be retrieved in about 30 seconds."-21 We might add that with the advances that are steadily being made in computer hardware technology this capability will be vastly increased in the very near future. These tremendous new capabilities which are offered to data collectors by computers have a number of very important consequences which we must discuss.

The National Academy of Sciences in its study "Databanks in a Free Society" visited 55 organizations who used computers for at least some of their information handling and concluded that "the organizations that (they) visited have not extended the scope of their information collection about individuals as a result of computerization."-22 However, they "did observe the emergence through computerization and rapid

communications systems of regional and national data systems that are ...giving rise to some new patterns of information handling and use."-23 In particular, what has happened is that various organizations which had collected data specifically for their own use before the advent of computers have now begun to collect data for the purpose of sharing it with other organizations. "Perhaps the most striking examples... are in the field of law enforcement."-24 Local police organizations, during the era of manual record keeping, would generally keep records only on people who had caused problems within their own jurisdiction. Although one local organization would occasionally share some of its information with other organizations, this would only be done in exceptional cases because of the time and effort involved. Today, these limitaions no longer exist. "The F.B.I.'s National Crime Information Center and the new computerized Criminal History System together comprise a national computerized network, with more than 40 law enforcement and criminal justice agencies having computer-to-computer links into the system, and over 4000 local agencies able to enquire through these state and local agencies."-25 In short, the problem is not that individual organizations keep more data about people in their own files. Rather, the problem is that these organizations tend to pool their data, resulting in, essentially, one large centralized data bank. And this data bank contains the sum total of all the data maintained in each of the individual data banks that contribute to it. The use of everyday telephone lines has made all of this

possible by allowing a large number of computers to all be hooked together into one.

So our concern with the ways in which computerized data can be abused must not focus only on the "pieces," namely the individual organizations which maintain data banks, but also on the "whole," that is, the ways in which data maintained by individual organizations can be put together to produce a much more powerful type of data bank which was not possible until computers came along.

Although it can be argued, and quite rightly in some cases, that this criminal information network has increased the efficiency of law enforcement efforts, there are others who argue that the enormous amount of money which is poured into collecting and maintaining this data would be more effective if spent on other law enforcement efforts. But our purpose here is not to debate the pros and cons of this particular computer network. Its mention here was simply to illustrate for the reader the nature of the trend toward information sharing that has been brought about by computers. This trend, if it continues and spreads into other areas of data use, may prove to be very dangerous for our individual liberties, for it will result in many enormous data banks with tremendous amounts of sensitive information on a very large number of people.

By now, we have all probably had the experience at one time or another of receiving "personalized" mail from a computer. By "personalized," we are talking about the situation in which a

person's name and a few other tidbits of information about him are sprinkled throughout the letter to make it appear as if somebody really knows and cares all about his personal life. Of course, the reason that such "personalized" information appears in the letter is because it was easy enough for the computer to fill in a few pre determined blank spaces with information from its data bank, and not because anybody took the time and concern to write a personalized letter. But such practices have enormous potential for getting out of hand. Consider the following letter that was sent to a number of people in the Washington area in late 1969 by a company that was trying to convince these people to join a commercial venture. The names and adresses of the people mentioned have been changed to protect their privacy.

"Dear Mr. Zurkowski:

I'm amazed at the number of my friends who have dramatically increased their incomes in just the past few months! John and Joanna Q. Public of 325 Orchard Way in Suburbia tell me their August income in a new business they created from what had been a part time job was $2050. That's a big jump from John's previous $1380/mo. at NASA.

Bob Babbitt of 225 Main Street in Anytown quit managing a fleet of trucks for the Icicle Ice Company in December 1968 to start his own business. By August of '69 he had reached a monthly income of $3750.

Three years ago you and Mrs. Zurkowski bought your present home. The Publics and the Babbitts were homeowners too.

It was when their ownership responsibilities caused money problems, that they sought a way to make more.

Most (people) started out with no more money than the few hundred dollars you have in the bank right now. Few of them had two cars like the Zurkowskis do. Usually they had a car less desirable than your '67, or no car at all, when they decided to 'rise above it.'

....

However, after making a careful household-by-household study of Washington residents with incomes in the critical $12,500 to $19,500 range, I have selected you and Mrs. Zurkowski as possibly being among the few who will take positive action if given the opportunity.

....

I look forewarc to telling you how I've doubled my income since retiring from the Air Force (full colonel) in July.

Sincerely,"-26

"This letter is but one by-product of the ccuntless computerized lists that are now commercially available in the United States and which are conservatively estimated to contain over five hundred million names."-27

For those people who are under the misccnception that computerized information is disseminated only with the utmost discretion, let this letter speak for itself. Can we envision receiving from, say, a psychiatrist a letter that opens with the statement:

"Dear Mr. and Mrs. So and So,

We've heard that you have some psychiatric problems. But you are not alone. Let me cite just a few other cases where people with problems just like your's have come to us for help..." The difference between this letter and the previous one is only one of degree. Experience has shown that people who collect and maintain data just love to give it out. And this abuse is not confined to private commercial organizations. For instance, questions such as "Does your TV set have UHF," "Do you have a flush toilet?" and "Do you have a bathtub or shower?" were asked of millions of U.S. citizens on the 1970 census.-28 Questions like these "have been asked at the request not only of social planners from both governmental and private institutions, but also of industry groups desirous of securing information that might aid in making product design and marketing decisions."-29 And the American citizenry is required, under penalty of law, to answer these questions, even though these questions may ultimately do nothing more than to bring a deluge of annoying salesmen to their door and junk mail to their mailbox. But even so, this is merely an annoyance.

However, "the army maintains files on the membership, ideology, program and practices of virtually every activist political group in the country. These include not only such violence prone organizations as the Minutemen and the Revolutionary Action Movement, but such non-violent groups as the Southern Christian Leadership Conference, Clergy and Laymen

United Against the War in Vietnam, the American Civil Liberties Union, Women Strike for Peace and the N.A.A.C.P."-30 Although it might feel comfortable to deceive ourselves into thinking that such information will not be misused, "I see no reason to assume that the government will be any more resistent to the pressures of the moment in the future than it has been in the past. Sending Japanese-American citizens to concentration camps would have been immensely speeded by having a National Identity and Data File, and McCarthy could have destroyed many more careers if he'd had computer records of security investigations. Protestors of... Viet Nam policy could be easily marked 'politically unreliable for shipment off to the Tulelake Relocation Center after we bomb China.'"-31

And we needn't say what a computerized file of all the Jews in Europe might have done for Adolph Hitler during the holocoust.

So far our discussion of the misuse of computerized information has dealt only with raw data. But there are other ways to misuse computerized personal data without anybody ever having to see that data. For instance, if somebody is using the raw data stored inside a computer directly to aid in screening job applicants, there is a considerable difference between that situation and the situation where the computer itself sifts through and evaluates the raw data, makes a yes/no decision about the applicant, and simply prints out "yes" or "no." Although most current systems still print out raw data which people can

then use directly to make their decisions, the notion of the computer itself actually sorting through the data, evaluating it, and making decisions based upon it is not too far off in the future, particularly with the advent of the artificial intelligence field. People in this field hope to produce programs which they claim will be "intelligent," and which, as a result, can be used to make decisions for us. This presents some very important social questions which we shall be dealing with in great depth later on.

How to Deal With the Databanks

All of these examples of data bank misuse lead us to the conclusion that although it is often necessary to gather and maintain records on individual people, this process is subject to excessive abuse if it is not carefully limited. For this reason, we feel that it is necessary to sharply define limits upon those organizations that choose to collect, store, disseminate or use personal data. Of course, the limits will be different for different types of organizations, depending in many ways upon the type of data which that organization truly needs to function effectively.

Regarding the point in time where data is first collected, there are a number of questions which we must adress. What types of information can a given organization gather? How may that information be gathered? Will it be gathered from

questions asked directly of the person who the information is about or will it be gathered by somebody sneaking around and asking his neighbors for juicy tidbits of gossip? What knowledge, if any, will the person who the information is about have of the existence of or the contents of that information? In what ways can he contest the accuracy of that information or even the need to keep such information?

Regarding the point where information is first put into a computer, we must adress the following questions: Who is allowed to put information into a computer and what safeguards must be met to insure that the information is being accurately entered? In what ways, if any, can the initial information be altered before it is put into the computer? Perhaps certain information should be entered into the computer along with a notation telling who supplied that information. Such a provision would certainly cause the people who gather information to use a little more discretion before they go and put that information into a computer.

Once information has been put into a computer, what standards of protection and security for that information must be met? After all, the stealing or altering of information in a computer is often a simple task for anybody with some technical knowhow and a little bit of spare time on his hands. Once again, many of the problems about information security which we discussed in the first chapter are of crucial concern.

And, perhaps most importantly, we must define the ways

in which information can anc cannot be used. To what extent can this information be used to make decisions about the people it pertains to? Who else, if anybody, can this information be shared with? To what extent is the person who the information is about entitled to know how that information is being used and who it is being shown to? To what extent does he have tc give consent to such uses? Must the informatior be used in "raw" form or can the computer "interpret" some of the information and simply output its interpretation?

As we can probably sense, many of these questiors are very difficult to answer. Most of them involve a tradeoff between the need for people to occasionally maintain information about others and the right of people to keep certain information about themselves private.

Pursuing the actual answers to many of these questions is beyond the scope of this thesis, although in future sectiors we shall attempt to adress at least some of them. But at the least, it is hoped that these questions will provide some challenging food for thought for those who are concerned with formulating specific policy for controlling information abuse.

Of all the questions which we have raised here, the one question which we shall explore ir considerable detail is the question of the extent to which computers themselves can actually interpret the data that they contain and reach decisions based on that data. Although this particular question has not received widespread public attention as of the moment, it promises to be

one of the most ominous questions about computer use in the future, particularly with some of the advances that people in the artificial intelligence field hope to make.

Possibly the best current indicator of this problem is the credit industry. Anybody who has ever made a purchase on credit has had the experience of standing around for ten or fifteen seconds at the sales counter while the salesperson makes a call to a central computer data bank to check the customer's credit. If the customer's charge account is overdrawn, a message is sent back stating that fact, and credit will generally not be granted. And because the same credit service is subscribed to by a large number of commercial credit granting enterprises, if a person isn't granted credit in one store, he won't be granted credit in any store. And it is not the salesperson at an individual store who gives the "go/no go" for the granting of credit; it is the computer.

There have been many uses proposed for the computer in the field of education. Many of these proposals envision students sitting at computer terminals where the computer types out various things to the student and the student is asked to respond to what the computer has typed out. It is argued that this will allow for individualized instruction, and that it will be easier to chart a child's progress by recording all of his responses to the computer's inquiries. Of course, the possibility of recording the student's responses raises some very important implications for the privacy of that student. Certainly, we must assume that a

child would spend a fair portion of his early years communicating with the computer. If this is the case, then the computer will contain a very large record of the student's responses to and attitudes about a wide variety of subjects. Such information, if abused in some of the ways that we have been discussing earlier, can do a great deal of harm to the student. But to add to the problems posed by the presence of so much sensitive information about the student, just suppose that somebody were to come along and propose that a program be added to the computer to sort through the student's responses on various subjects and evaluate him. Thus, a computer can label a student as "brilliant," "uncooperative," "retarded" or just about anything else. And these computer evaluations would undoubtedly follow the student for the rest of his life and would determine to a great extent what he will be able to make of his life. In this situation, as in the previous situation, it is not other human beings that are making decisions and judgements about people; it is a computer. Hopefully, we needn't tell the reader what would happen if a person writing a section of this "student evaluation" program decided to give undesirable evaluations, say, to all students with a particular political persuasion which the programmer personally dislikes. And what is even more dangerous is that there would be no way to find out that this bias exists in the program.

As another example, envision the situation in which some computer software company comes up with a program which it

claims should be used by employers to decide who to hire for jobs. The prospective employee need [redacted] answer only a few questions, his answers would be put into the computer, and the computer would respond with either a "hire" or a "don't hire." Imagine too, if this program came to be used by a large number of employers in various fields. In the same way that a person who is rejected for credit at one store will be rejected for credit by all stores, a person who is denied a job by one company which uses this "job applicant screening" program might be denied a job by every company which uses that program. Once again, it is a computer program, not a person, which would be making the judgement.

Finally, consider the case of an 82 year old woman who was found dead two weeks after her gas had been shut off. "The Allegheny County Coroner's Office said the cause of death was freezing...

During a coronor's inquest... (the) assistant district attorney tried to determine who, if anyone, was responsible for (the lady's) death.

While questioning (the credit manager of the company involved, the Assistant D.A.)asked, 'You mean the order for cutting off the gas came from the computer?'

'That's correct,' (the credit manager) replied."-32

In all of these cases, the pattern is the same. No longer would people be making decisions for and about themselves. Instead, they would have computers making those decisions.

Today, this type of situation is not yet a pervasive problem, and there are many people who take the attitude that "nobody would ever do such a thing." But like it or not, the current trend in computer use is headed in this direction. At this moment, the artificial intelligence community is busily at work trying to produce programs which they claim are "intelligent." And so, they reason, these programs can be used to make the same types of decisions that any other "intelligent" beings might make. So let us further examine the notion of "intelligence" and see if we can identify some of the problems that one might encounter if he wanted to produce a computer program capable of making "intelligent" decisions.

Chapter three-

The Use of Artificial Intelligence

The Regurgitating Intelligence

If we are even to think about discussing artificial intelligence, we must first ask the question "what constitutes intelligence?" In recent years many people, particularly test designers, have claimed for themselves the ability to measure intelligence, although whether or not they really are able to measure intelligence is certainly open to serious question. At this point in time, the status quo maintains that intelligence can indeed be measured, and it supports that belief wholeheartedly through its actions. Nowadays, one has to face numerous attempts to measure his intelligence which often have a profound effect upon his life. I.Q. tests in the early grades often determine the academic level of the classes that a student is placed in and the expectations that others will have of him. Various other tests which in one way or another claim to measure intelligence or "aptitude" are used to decide who gets into certain colleges and who gets certain jobs, among other things. By the time the youngster of today has finished his education, he will have faced countless attempts to asses his intelligence in various ways. Of course, many of the people who are found intelligent by these tests are labeled as such because they are good at taking these tests and not necessarily because they are

"intelligent."

One cannot doubt the fact that what a teacher expects of a child often influences heavily what that child will expect of himself, and thus the level at which that child will perform. These "self fulfilling prophesies" are a major by product of our obsession with testing, and have often served to hamper the lives of children throughout their formative years and on into adulthood. "Pygmalion in the Classroom," a study done a number of years back, shows just how much certain tests can influence a teacher's expectations about and reactions to a child.

Many of the testing procedures that we use have a strong built in cultural bias. For instance, "the Wechsler Intelligence Scale for Children (I.Q. test), constructed in 1950 by testing 2,200 people, all white, is valid only for whites, according to the test's author, David Wechsler. It has undergone no significant revisions since 1950 and is administered only in English... in practice, the tests alone usually determine which children are committed to 'Children with Retarded Mental Development' classes."-33 It is a strong condemnation of our educational system that these tests have been given such weight in determining a child's future, when one considers that children who are not white and who do not have English as a native language are still given these tests, which are "valid only for whites," in English. Then, when these children do poorly, which is all that one can reasonably expect, they are categorized as "mentally retarded," a categorization which will

hurt them for the rest of their lives. Besides the cultural biases which underly intelligence testing, there is an underlying assumption that intelligence can indeed be quantified. And in fact, there is a further assumption that intelligence is static; if a person posesses a certain amount of intelligence, he will always have that same amount of intelligence. "If John is more intelligent than Bill, then he will always be more intelligent than Bill, and, what is more important, John will 'do better' on every task than will Bill."-34 These assumptions about the nature of intelligence provide an even stronger indication of just how much damage has been done by intelligence testing. Besides saying to the student "you are mentally retarded because you couldn't pass this test" we are saying to him "you will always remain mentally retarded because intelligence doesn't change." Not only are we classifying someone as mentally retarded on the most narrow of evidence, but we are removing all hope for his recovery from this condition. If ever there was a way to destroy a young child, this is it!

Finally, we must realize that the tests of which we speak really only measure a person's ability to agree with those values held by the designer of the test. The questions on these test have only one "right" answer; that which its designer has designated as the right answer. This is another problem with the current standards for measuring intelligence. Intelligence today is measured almost exclusively by one's ability to memorize and regurgitate "right" answers, rather than by his ability to think.

"What students mostly do in class is guess what the teacher wants them to say. Constantly, they must supply 'The Right Answer.'"-35 This situation in turn, illuminates what may well be the biggest flaw in our contemporary educational system; the fact that it brainwashes students into thinking that there is a definite right and wrong answer to everything; an answer which has been predetermined by somebody else and which they must passively accept. And often, the questions and answers found in the clasroom might well be of the form "Why did we drop the bomb on Hiroshima?" Answer: "To save hundreds of thousands of american lives." Or "Who assasinated President Kennedy?" Answer: "Lee Harvey Oswald." Or "Why is America the greatest country in the world?" Answer: "Because we have liberty and justice for all." Certainly, there are significant schools of thought which would vehemently oppose these "right" answers.

We forget that "fact" is a subjective judgement; that what one person holds to be fact may be held to be totally false by someone else. And we forget that "fact" is subject to change; what is fact now may not have been fact a few years ago or may no longer be fact a few years from now. We encourage a static view of the world that is founded upon unchanging maxims which one is taught in school. The questions that we presented earlier might best be asked in the form "Who do many people at the moment maintain assasinated Kennedy?" or "What does popular opinion give at this moment as the reason for bombing Hiroshima?" yet questions are rarely, if ever, asked in this form. In many ways,

the idea that educating somebody involves filling his head up with facts rather than making him think plays a strong role in maintaining popular belief.

If one were to try to devise a true measure of intelligence, if such a measure exists, I would suggest that he look more at a person's ability to formulate questions for himself and to actively pursue their answers, rather than his ability to take somebody else's questions and arrive at somebody else's answers. The greatest breakthroughs in mankind's knowledge have come not from those who simply went along with the "right" answers all the time, but from those who dared to ask where the "right" answers came from, and who challenged the "right" answers when they found them questionable. There was a time when people thought the earth was flat, and it was only when Columbus decided to challenge that "fact" that we discovered that the earth really wasn't flat at all. Had Copernicus not decided to challenge the "right" answer that the sun revolves around the earth, people might have gone on for years thinking that this was so. And virtually anybody who has ever been recognized as a "creative genius" in his or her field achieved that status precisely through the creation of new and original modes of expression within that field. Nobody achieves such status by simply copying everybody else.

If it is the asking of new questions rather than the regurgitating of old answers that has been responsible for many of our greatest advances in all fields, then why are we still

obsessed with getting people to regurgitate the same old answers? Certainly, the status quo is maintained more easily if the young people are inculcated with traditional values, rather than being encouraged to challenge them. Also, it is simply easier to grade a person by asking him questions with predetermined answers. Many teachers wouldn't know where to begin if they were asked to design a test that measures a person's ability to ask questions or to think. Certainly, it would be more difficult to test people along these lines, even though it might be a better indicator of their "intelligence." So what have we done? We have copped out. Since it is more difficult to evaluate people along these lines, we have simply stuck to the answer regurgitation method. And because of our inability to design tests which measure a person's ability to think or to formulate questions, we have simply redefined intelligence to fit the tests that we are able to design. In particular, the recent use of the computer to grade tests has greatly encouraged the use of multiple choice tests for evaluating intelligence, because a computer is simply unable to evaluate written responses to a question. I was particularly offended by the "writing ability" section of the Law School Admissions Test which claimed to measure my writing ability not by asking me to write something, but rather by asking me to answer a bunch of multiple choice questions.

Many of the ideas expressed in this section are discussed in great depth in the book "Teaching as a Subversive Activity," by Neil Postman and Charles Weingartner. I strongly

recommend this excellent book for anybody who is concerned with our current methods of ecucating people and who feels that there must be a better alternative to answer regurgitation.

I have brought all of these points up because they show the problems that result when we start trying to quantify such abstract notions as "intelligence." It shows, in particular, how we have redefined intelligence to fit our tests, rather than redesigning our tests to fit intelligence.

In the area of computer science, we have often seen companies that have had to change their way of doing things to fit their new computer when they first computerize their operations. Now that people are starting to talk in terms of "intelligent" computers, might we start redefining intelligence to fit the capabilities and limitations of our computers? If we stick to our current notion of intelligence as being an ability to remember pre-chosen "right" answers, then computers will dwarf us in intelligence simply because they have a much greater capacity than humans to retain information. But if we go beyond this overly simplistic notion of intelligence and include other factors in determining a person's "intelligence," then the idea of an "intelligent" computer raises serious social questions. These are the questions that we shall adress throughout the remainder of this section.

What To Put In and What To Leave Out

As we said earlier, one of the things that the artificial intelligensia hopes to do with its "intelligent" computers is put them to work making human decisions. After all, they reason, if they can produce a computer which is more "intelligent" than a person, than why not use it to make people's decisions? Of course, we have already seen the difficulties involved in defining intelligence and some of the problems that result when we try to use a narrow, one dimensional definition of intelligence. Herbert Simon and Allen Newell, two of the foremost leaders of the Artificial Intelligence field as early as 1958 said that "There are now in the world machines that think, that learn and that create. Moreover, their ability to do these things is going to increase rapidly until -in the visible future- the range of problems they can handle will be coextensive with the range to which the human mind has been applied."-36 And psychologist George A. Miller once said 'I am very optimistic about the eventual outcome of the work on machine solution of intellectual problems. Within our lifetime, machines may surpass us in general intelligence.'"-37 This strong desire on the part of these people to go ahead and produce an "intelligent" computer and apply it to solving man's problems and making his decisions raises some very important questions. Can a computer really be made more "intelligent" than a person? What does "more intelligent" mean and who will define what it means? Will they be the same people who have already dehumanized us by giving us a single number, namely our I.Q., to completely characterize our

intelligence? If we are able to determine what attributes of a person make him intelligent, can we necessarily put those attributes in the form of a computer program? And finally, even if we could in theory design an intelligent computer, which we shall maintain that we cannot do without using a very simpleminded definition of intelligence, to what extent would practical programming considerations make it at best, a risky endeavor?

Because we are trying to gain some insight into the problems that might result if we used computers to make decisions for us, it might be good from time to time to envision a specific situation in which a computer might be used to decide something. The use of a computer to make judicial decisions might provide a good example. We shall not concern ourselves with the constitutional questions that would ultimately arise if someone really proposed using a computer as a judge, although these are certainly important questions. Our discussion of a computerized judge will be used more to illustrate certain general problems that one would encounter anytime he wished to write a program to make decisions, regardless of the specific situation. We also choose a computerized judge because we don't have to devote any time to convincing the reader that the decisions made by such a computer program would have profound effects upon the lives of those people whom it judges. The effects are obvious. And besides, with the ever increasing case loads and the inability of judges to keep up with them, it is not at all difficult to

envision a proposal from the Artificial Intelligence community to actually use computers in this way. "In fact, Professor John McCarthy of Stamford once said "'What do judges know that we cannot tell a computer?' His answer to the question (was)... 'nothing'"-38

If we were put into the position of having to design this computerized judge, what problems would we have to adress before we could begin? First of all, we would have to ask "how should our program interpret the law?" Should laws be interpreted to their absolute letter without allowing any other considerations to enter into the decision? Or should the laws be interpreted more loosely; by their spirit rather than by their letter? If we should decide to interpret laws in their "spirit" what factors should enter into the spirit? Should we include the transcripts of every word that was said about each law involved in a particular case at the time that that law first became a law? If so, how much weight should these things be given? Also, how much of the "human" factor should enter into the decision? Should a man who killed another man because the other man raped his wife be treated in the same way as the cold blooded killer who plotted a murder for weeks before committing it? Or should things like "compassion" enter into the decision? After all, a human judge has a wife and loved ones, and he can probably understand how he might feel if one of them had been raped. And if we decide that our computer should "feel" for the man who's wife was raped, is it really possible for us to program such

feelings into the computer when the computer has never had any loved ones that can be raped? The question of whether we can actually program a computer to have such feelings is in fact a very pertinent one which we shall adress later on. But many of these questions, and others like them do not have any definite "right" or "wrong" answer. Many of them involve opinions and value judgements that have been subject to debate throughout history. No two judges hold precisely the same view of how laws should be interpreted, and no one can say what the "right" way to interpret laws is. All anyone can do is make suggestions. Yet, before we can ever hope to produce a program to decide judicial cases, we must come up with a very definite method for interpreting laws. The determination of this method involves many value judgements which no one person should ever be in a position to make.

In thinking about a program, I often liken it to a book or essay. After all, a program is essentially nothing more than a written description of how to do something. Instead of being written in a natural language which most people can understand, such as English, it is written in a computer language which very few people can understand.

Now think about this: what would happen if a person came to us and said "I have written a book that describes precisely and unambiguously a method for reaching a verdict in all court cases. I propose that henceforth we abolish all of our courts, get rid of all our judges, get rid of all the opinions

and precedents that the legal field has produced throughout its history and simply use the method described in my book whenever we decide future cases. Furthermore, this method will be followed to the letter and I shall not allow anyone to look at this method or try to understand how this method works. Everybody should simply be willing to use this method on faith?" We would probably wonder where this person got the unmitigated gall to even make such a suggestion. What about the hundreds and thousands of other people who have ever expressed an opinion about how to judge a case? What is so great about this one person's method that we should use it and nothing else? Besides, even though someone has developed a method for deciding court cases, that doesn't necessarily imply that he has developed a good method. Simply flipping a coin is a method which could decide all court cases, but that doesn't make it a good method.

Although the case that we have just cited may seem totally absurd, it is highly analagous to what people in the artificial intelligence field are asking us to do by using their programs to make decisions for us. If a person were to write a book describing how to make judicial decisions, at least that book would be open to inspection. (We ignore for the moment the stipulation made earlier that nobody is allowed to read or try to understand the method proposed in the book.) Judges, and the public at large would be able to read this book and see for themselves the method that it is proposing for use in all judicial cases. People can agree with parts of the method, and

judges may even allow some parts of it to influence their decisions. And people can disagree with other parts of it and consequently not consider them when making their decisions.

Keeping in mind that a computer program is nothing more than a written description of how to do something, but in computer language rather than natural language, what would we do if someone came before us and proposed that a program of his be used to make all judicial decisions? Although some people might dismiss such a proposal on the same grounds that they would dismiss the person with the book there are others who would be intrigued by the idea, and who would actually consider it. Why? Because there is something about a computer that mystifies people. People view a computer as some sort of magical black box which is capable of solving any problem that they give it to solve. They have the highly mistaken notion in their heads that computers are perfect. And since computers are perfect, they reason, why not put them to work making important decisions for us? After all, a perfect computer can do a better job than an imperfect human. What people fail to realize time and time again is the fact that the only thing that computers can do perfectly is store data and follow instructions given to them by imperfect humans, and even then, there is an occasional problem. If a computer is given "good" instructions, it will follow them perfectly. If it is given "bad" instructions, it will follow them just as perfectly. This is something which the general public claims to know, and it often uses the phrase "garbage in, garbage

out" to flaunt this knowledge, yet the public does not seem to realize the full implications of this fact. If the reader gains nothing more from this book than an understanding of the fact that all computers can do "perfectly" is follow instructiors and regurgitate data, I shall have accomplished a major objective of this thesis. The reason that we place such emphasis on this point is because the most dangerous feature of a computer might very well be the "perfection" for which it is so widely applauded. If we decide to accept a program to make decisions for us, we are forced to accept the whole method described by that program down to the very last minute detail. What is so dangerous about the computer's ability to follow instructions without error is that it follows bad instructions just as perfectly as it follows good ones. The computer does not have the discretion to decide what the good points and bad points of a particular method are. Although we may sound repetetive, this point is a very important one which must be repeated and emphasized over and over again until it is fully understood, for it is one of the most important sources of misunderstanding about computers. Even more will be said on this later.

Another question raised earlier that we should adress is "what exactly can and can't we express in the form of a program?" Can we really take human emotions and put them in the form of a program so that a computer can have these emotions too? Let us take a look at this question, because it will give us a lot more insight regarding the extent to which computers may be

able to make decisions for us.

The Limits On Computer "Intelligence"

At the moment, there is not a single programming language in existence that even comes close to having the number and variety of expressions that natural language provides us with. In many ways, the difference between programming language and natural language is one of precision; in natural language, we often have the option of expressing a particular idea in a number of different ways, all of which will get our message across to the listener. We can often rely on the fact that a person to whom we speak will be able to figure out what we are trying to say, whether or not we actually say it in the right way. We can even go so far as to make statements that completely violate the grammatical structure of our language and still get our point across. Inflections and accents in our presentation may change the meaning of a phrase without a change in its words. Even the hand and body motions that we make while speaking often help to get our point across. We cannot do such things with a computer. Although there may be a number of different ways of writing a particular program, whichever way is finally chosen must be expressed in extremely precise terms. There is no chance for us to say "you know what I meant" to the computer. If we do not express our ideas clearly in a program that we write, the computer will not clarify them for us. It will simply follow our

mistakes. There is, at the moment, considerable work going on to develop prorams that allow people to program in natural language. But even if someone does come up with a way to program in natural language, there is still a problem. Any program designed to interpret natural language simply reflects one person's way of interpreting natural language, namely that of the person who designed the program. It describes, if you will, "one person's way of interpreting what other people say." No matter what type of language understanding programs are designed in the future, those who use them must accept on faith that the program is interpreting what they say in the way that they wish it to be interpreted. It is not at all clear that we can ever design a language understanding program that can always understand everything that anybody says in the way that it was meant to be understood. This is simply because people must ultimately write the program, and people often misinterpret what other people say.

One can get a really good idea of exactly how far people are willing to go with their computer programs by looking at some of the statements made by Dr. Adam V. Reed during a symposium at the 142nd annual meeting of the American Association for the Advancement of Science. At that symposium, Dr. Reed said "Ideally, the computer of the future should be an extension of the natural brain, functioning in parallel with some of the existing structures and using the same program and data languages." He went on to say that "Once the natural language of human thought and memory has been decoded, it will be possible to

program a computer in it" and transfer programs directly to the computer from the brain. -39 What is being completely ignored here is the fact that any program which is capable of "understanding" our thoughts and transferring them into program form must still interpret the meaning of our thoughts. If a language understanding program may not always interpret what we say properly, then what is to say that it will interpret our thoughts properly? People may start to find that they are misinterpreting themselves all the time! And since we really won't have any way of knowing how the language understanding program is translating our thoughts, we may actually have programs "running" our mind which we have no control over and no understanding of. People might have to start constraining themselves to think in a pattern compatable with the computer in much the same way that companies adjust their way of doing things to achieve compatability with their computers. Such a prospect is absolutely absurd, yet it is frightening when we realize the seriousness of the people who support it. And most dangerous of all is the fact that such a situation would provide absolutely unparalled opportunities for social control. It would allow ill motivated people to literally control the very thoughts of other people. How much trouble are we looking for?

Getting back to our previous line of discussion, if a program is simply a written description of something, then the question of what can and can't be expressed in a program might well be answered by discussing what can and can't be expressed in

natural language. When we say "natural language" we include not only the languages used everyday by people, but also the technical and highly specialized languages that have been developed for specific areas of human involvement. For instance, many of the concepts involved in high energy physics are not expressable in everyday natural language, yet they can be put into a program because they are expressable in the specialized and often mathematical language of the physicist. Thus, our definition of natural language will cover all languages that have been developed by man to describe the world around him.

As we have said earlier, the programming languages that have been developed to date do not even come close to providing the expressive power provided by natural language. This might lead us to believe that there are some things which can be expressed in natural language that cannot be expressed in a programming language. Although this is a possibility, we shall choose to ignore it and shall assume that anything that can be expressed in natural language can also be expressed in a programming language, since this will not hurt the argument that we are about to present in any way.

The next question which we wish to ask is "if some technique is expressable as a program, does that imply that the same technique can be expressed in natural language?" Since a program is ultimately nothing more than a series of instructions, each of which has, in fact, a very precise natural language description of exactly what it does, we can certainly take any

program and translate it into natural language by simply translating each instruction individually into a natural language description of what it is doing for the overall program. If the reader will grant that anything that can be expressed in a program can also be expressed in natural language, then he must also grant the contrapositive; that anything which cannot be expressed in natural language cannot be expressed in the form of a program. This is a crucial point that we shall use as the basis for many of our future arguments. In particular, we now know that anything which we cannot express in natural language cannot be expressed in a program.

Language, as many people have observed, is nothing more than a collection of metaphors; that is, words or phrases to which people assign certain meanings in their heads. The word "metaphor" has itself become a much used metaphor in recent years through the works of McLuhan and others, but since it is an effective metaphor, we shall use it

When we hear the word "car," for instance, the first thing that comes to mind may be a picture of that physical entity used for transporting people from place to place to which we have assigned the word "car." But even then, the word "car" will not always paint exactly the same picture in everybody's mind. Some people will visualize a Cadillac, others a Volkswagon. To a teenager, the word "car" may represent something that is important for picking up girls, and may be seen as a sex symbol of sorts. To others, the word "car" may represent the noxious

smell of pollution that one must put up with everytime he is near a traffic jam. But the word "car" has no meaning in and of itself. The only meaning it has is the meaning that we give to it when we hear it; the pictures, thoughts and emotions that we experience internally when we hear the word "car." All human interpretations are made within the context of the belief structure of the individual who is making them. Yet, with all the potentially different interpretations that one might give to the word car, there is still little fundamental disagreement among people as to what a car is. But let us take a look at some of the more ambiguous metaphors in our language and see how people might react to them.

If we were to ask the average person on the street what, say, love is, we'd probably expect to receive quite a variety of answers. Some people may be completely tongue tied. Some might say that love is a very positive, very special feeling between two people. Others may attempt to distinguish between different types of love; the love one feels for a spouse as opposed to the love one feels for a parent, child or friend. And even if we were to narrow our discussion of love down to, say, the relationship between a husband and wife, we would still find fundamental differences in people's perceptions of love. There are many marriages where love stems from a deep emotional attachment between the two people involved. Other relationships may involve a love that is based upon financial arrangements. Still, there are other relationships where the partners are

simply afraid to look for someone else for fear of rejection and so they stick together. We would also find fundamental disagreement as to which of the preceding examples would even qualify as "love." Is the relationship that is based upon one person's money called "love" or should it perhaps be given another name? And two people who have been married for many years must certainly have a special type of love derived simply through many years of mutual experience; a love that nobody else could ever feel. The point is that there is simply no description of love that we could offer where we can state with finality "this is what love is." Love is a multifaceted experience which each person must discover for himself. It is something that has, in fact, defied adequate description by the greatest literary geniuses in our history. A writer may be able to get his reader to experience "love" emotions by offering a moving description of two people deeply in love, but the emotions that the reader experiences must be based upon his own previous love experiences. How can someone truly know what love is without having ever experienced it? Note the distinction that we have made here. We are saying that a writer can indeed get his readers to feel the emotions of love by describing some love experience which the reader can associate with his own previous love experiences. But the writer cannot simply come right out and say "the following is a description of love" and then proceed to objectively describe what love is. And it is precisely because the feelings which constitute love are so difficult to describe that we have

invented a catch-all metaphor, namely the word "love," to describe them. Then, whenever a writer wishes to convey the feelings that constitute love to his reader, he can use the word "love" and hope that the reader will associate that word with the feelings that the writer wishes to convey. But the feelings themselves cannot be written down; they must be conveyed to the reader through the metaphor "love."

The point of all this is as follows: a computer cannot interpret a metaphor such as love, because that metaphor is designed to trigger within the person who reads it certain feelings which are based on previous experience. A computer does not have the "previous experience" with which to interpret such metaphors as love. The computer must simply rely upon programs which tell it how to interpret things. If we wanted our computer to understand the word "love," we would essentially have to write a program which objectively describes "love" and put it into the computer. But since we cannot construct a precise natural language description of the word "love," we cannot construct a precise program description of it either.

There are countless other feelings which we [redacted] experience in our lives that must also be conveyed in metaphorical terms because it is simply beyond our ability to express them fully in natural language. Can one write out an accurate and precise description of "compassion?" Or "empathy?" Or the feelings that one experiences from the death of a loved one or the birth of a child? The reason that we can use the word

"indescribable" to describe such feelings is precisely because the feelings themselves cannot be put into words. If these feelings cannot be described by our language, then they cannot be described by a program either.

This brings us back to the earlier question of using computers to make decisions for us. If one feels that there are certain feelings, thoughts or emotions which cannot be adequately described in natural language, but which nevertheless play an important if not decisive role in making certain decisions, then it is clear that a computer cannot be programmed to employ these emotions to make decisions the same way that a human might employ them. This is not to say that we cannot construct a program to make decisions. We certainly can, for instance, write a program to which we can feed certain aspects of a judicial case and from which we will ultimately receive a verdict of "guilty" or "not guilty." But that program will not and can not reflect any of the unexpressable "human" considerations that would be reflected in the verdict of a human judge.

Here, we come to the crux of the problem. If we put programs which are incapable of experiencing human emotions to work making decisions for us, then we are making a very strong value judgement. We are saying, in effect, that "better" decisions are made when we leave out the subjective aspects of decision making than when we include them. But there are many people who would strongly disagree with such a judgement, and who may not want decisions about their lives made on such strictly

logical terms. One of the highly negative consequences of the scientific revolution has been a severe devaluation of human emotions in the decision making process. People are always asked to come up with purely logical reasons for reaching certain decisions, as though pure conscious logic is the only mental process that they are justified in employing to make these decisions. But this just is not so. For instance, I, and countless others too, have had the experience of working on a difficult problem late at night for a few hours. After reaching the conclusion that further concentration will yield little more, I would go to sleep only to wake up the next morning and find the answer thrust firmly into my conscious mind. In this type of situation, the subconscious mind plays a crucial role in arriving at an answer or a decision, while pure conscious logic has had little or nothing to do with it. Although such phenomenon have been observed often, we have little knowledge of how the subconscious is actually able to do such a thing. And if we only have a very sketchy idea of how the subconscious works, then we certainly cannot write a program that can even come close to simulating the activities of the subconscious. If we decide to ignore this problem and simply leave the subconscious out of a decision making program, we shall have removed from the decision making process in one fell swoop a crucial and very powerful part of the human mind which we are only beginning to understand.

How often do we make decisions and wher asked how we made them do we say "I just 'felt' like it was the right

decision?" Often, the "gut" feeling is a crucial factor in reaching a decision. But if we can't even begin to describe this aspect of decison making, then how can we ever expect to put it into a program?

We must remember that a person is shaped to a large extent by his environment and the problems that he has to face. All men are faced with a finite lifetime, and that fact certainly has a great effect upon how each person conducts himself from day to day. How can a computer feel "mortal" when in fact it isn't? A computer does not have to feed and clothe a family. If a computer decides to drop an atomic bomb and wipes out millions of lives it doesn't have to sit around for the rest of its life and face the guilt and remorse that a person making a similiar decision would have to face. All of these things point to the conclusion that there are simply certain feelings which people have that computers cannot be made to have. If a computer is programmed to make decisions without employing these feelings, as it must be, then the decisions reached will not in any way reflect any human feelings. And we would be supporting the very strong value judgement that human feelings should play no role in decision making.

Also, if the ability to ask new and original questions contributes to one's intelligence, then we have a further problem, since it is unclear that computers can formulate any significant questions other than questions that they have already been programmed to ask.

Finally, if we intend to ever place computers in a decision making position, we must realize that the computer's way of reaching these decisions may become in a sense, a "standard" for reaching those decisions. In much the same way that we have already redefined intelligence to fit our I.Q. tests, we might redefine intelligence to fit the logical and emotionless way in which a computer might reach a decision. In other words, we will redefine our intelligence to fit our computers because we can't redesign our computers to fit our intelligence. Such a situation strongly devalues much of what makes us human and is, so far as I am concerned, an affront to our humanity!

"Prediction Making" Instead of "Decision Making"

Up to this point, we have actually been giving the people in the artificial intelligence field more credit than they deserve. We have been discussing decision making programs as though the writer of such a program would at least attempt to have his program reach its decisions by taking into account the same things that a human decision maker would take into account to reach his decisions. Actually, the approach which at the moment is being taken by the artificial intelligence people is to write programs that try to "predict" what a decision maker would decide in a given situation. That "prediction" is then used as the actual decision. That is, if someone were writing our computerized judge program, he wouldn't try to write his program

so that it makes decisions in the same way that a human judge might make those decisions. Instead, he would simply write a program to "predict" what decision a particular judge or group of judges would reach in a given situation, and that "prediction" would become the program's decision.

With such an approach, the reasons for which a certain decision is reached mean nothing. All that matters is what the ultimate decision turned out to be. In other words, these programs would be built on the assumption that if a judge finds a person guilty in a given situation, then the program merely has to predict that verdict accurately in order to replace the judge. It doesn't make a difference if the judge reached a guilty verdict because the defendent was really guilty, because the defendent had long hair or because the judge was bribed. All that matters is that the program ultimately reaches the same decision that the judge would have reached. Such an assumption is thoroughly absurd. It is saying, that the "ends" are all that count and that the "means" by which those "ends" were reached are unimportant.

Supporting such an assumption would be like saying "if a computer can predict the results of an election, then we needn't have an election" or "if a computer can predict a supreme court vote, then we needn't have a supreme court." Since when do people reach important decisions by trying to "predict" what someone else would do in the same situation? The President of the United States doesn't reach decisions by trying to predict what

his predecessor would have done in the same situation. In fact, our current President is making a conscious effort to avoid doing what his predecessor would have done in many situations.

Besides, it is the unique extraordinary decisions; the decisions which nobody expected, which often have the most significance for the people that they effect. The use of a computer to "predict" decisions would produce nothing but a bunch of mundane, uncreative decisions.

Using a computer to make predictions about the outcome of some decision making process and then using that prediction as the actual decision is the most straightforward form of a "self fulfilling prophecy that can be envisioned. If a computer predicts that an atomic war will destroy the world tomorrow, does that mean that we should start dropping atomic bombs to make that prediction come true?

And we have not even touched upon the way in which fraud and all sorts of other problems related to computer security would also come into play in this type of situation.

This form of precicting the future and using the prediction to make itself come true presents a great threat to mankind, and it should make us even more careful about using artificial intelligence programs.

Practical Programming Obstacles and Secret Programs

Up to this point, much of our discussion has been in

largely theoretical terms. We have tried to advance the hypothesis that a computer cannot be made to feel man's emotions and feelings in the same way man does, and thus cannot make man's decisions in the same way that man would. We have discussed how at the current time, the approach in artificial intelligence has been to write programs that simply "predict" certain outcomes rather than to write programs that really take into account the factors that determine these outcomes. However, many people in the artificial intelligence community feel that at some time in the future we will have enough of an understanding of these intangable feelings that are involved in decision making to be able to express them in the form of a program. Although I am skeptical of such a claim, it cannot be ignored. So, we shall not ignore it. In fact, we shall even go so far as to assume that at some point in the future computers will be able to make decisions in the same way man can, with emotions and feelings and all sorts of subjective factors included in the decision making process. Even if this were to be the case, which we maintain it most certainly won't be, there are still many problems to be dealt with.

One question that would inevitably come up is "what type of personality do we give the computer?" Certainly, we could design our computer to operate within the belief structure of anbody from Richard Nixon to Thomas Jefferson to Adolph Hitler. Who do we pick? Not only that, who does the picking? This decision about what belief structure a computer should operate

within is a very crucial one. After all, we must remember that the "perfection" which people so often associate with computers lies in part in the ability of computers to follow instructions that people give them. The computer will not ignore sections of a program if it doesn't like them or if it considers them dangerous. It will not go to the library to find out what other people think about the decision it is being asked to make. It will simply employ a single method precisely; that which is given to it by the program. Dr. Joseph Weizenbaum, in his book "Computer Power and Human Reason" describes this phenomena as a "destruction" [redacted] of history. That is, once we put a program to work making decisions, it takes nothing into account but its own instructions. Any other thoughts or feelings which people may have had throughout history about the particular decison making process are simply ignored, unless those ideas were originally incorporated into the program.

And we must realize how dangerous it is that computers have such an ability to follow instructions "to the letter." When Hitler devised his plans for an Aryan race, at least there was a chance that somewhere along the line, members of the Nazi party would be able to soften his radicalism by interjecting their own judgement. But if Hitler had been able to program his plans directly into a computer and had used that computer to direct everything, his desires would have been carried out with a precision and perfection undreamed of. This type of situation illuminates what may well be one of the greatest of all possible

dangers of computer use; the fact that a computer allows a single person's instructions to be carried out with absolute precision. If such a computer program was used to run significant aspects of a society, the social control possibilities would be absolutely frightening. And remembering the Neuremborg trials where many of Hitler's higher ups were called to judgement for their faithful execution of his orders, we must ask what would have happened if it were a computer, instead of a bunch of people, which was seeing to it that Hitler's orders were faithfully executed. It might well have been possible to commit the same autrocities with nobody to blame but a computer. In this way, the computer provides an incomparable scapegoat for evil doers. They can simply program a computer to do certain things, and then say "its not my fault, its the computer's fault" when undesirable things start happening. This disavowal of responsibility has already become very apparant in today's world where companies with computerized operations will often blame a mishap that befalls a customer on the company computer.

Another thing that we must realize about writing a program to make make judicial, or in fact any other type of fairly complex decisions is that if a single person were to attempt to write such a program, it would have to be such a simplified version of the decision making process involved that we could not even begin to consider using such a program. The program would have to at least be complex enough to take into account all the things that a human judge would take into

account, at least if we want to be able to claim that this program does a "better" job of deciding cases than does a human. The program would have to be so large in fact, that we would need a sizeable number of people to write it. And this brings us headlong into the problems we have discussed in pervious chapters which are a part of any large programming effort. Management and communication problems begin to interfere. When the program is finally completed, nobody really knows how the whole thing works. After all, the whole point of using so many people to write the program in the first place is to allow each person work on a small section without having to know how the overall program works. And we certainly can't discount the fact that somebody might try to corrupt a section of the program, especially when we consider how easy it is to do such a thing. In a day where the very secrecy that has surrounded many major decisions has itself come into question, the computer provides the capability for a new brand of secrecy which is much more dangerous than any we have now. At least when a person makes a decision in secret, he is often held to account for that decision sooner or later. But when a large computer program makes a decision, there is simply no way to bring the secret of how that decision was reached out into the open. Since such a program is bound to be well beyond the comprehension of any single person, the decisior making process is, essentially, a secret which not a single person knows. One of the most striking examples of the type of secrecy that computers can help to provide occurred during our war in

Vietnam. When President Nixon "decided to bomb Cambodia and keep that decision secret from the American congress, the computers in the pentagon were "fixed" to transform the genuine strike reports coming in from the field into false reports to which government leaders were given access... And the high government leaders who felt themselves privileged to be allowed to read the secret reports that actually emerged from the Pentagon's computers of course believed them. After all, the computer itself had spoken."-40 And as a result, President Nixon was able to carry on his secret bombing of Cambodia for a substantial time before anybody found out about it. Given today's problems with secrecy, can we afford to provide even greater opportunities for making secret decisions? I think not.

Although up to this point we may have given the reader the impression that all of the "intelligent" programs which we have discussed would be written by a person or by a group of people, this may not necessarily be the case. The approach which is actually being taken by many artificial intelligence researchers toward designing their "intelligent" programs is to design "learning" programs which will enable a computer to "teach" itself. What they hope to do is to give the computer a comparatively small base of initial knowledge and then set the learning program into action so that the computer can increase its knowledge. Such an approach, from a technical standpoint, essentially involves writing a program that can generate its own new programs in the hope that these new programs will in some way

add to the computer's overall "intelligence." At this point in time however, we have only a very limited knowledge of the process by which people learn, and there has been very little progress toward producing programs which can in turn generate their own programs. But even if it were possible to produce such self generating programs, these programs would only serve to add to the secrecy which already surrounds large programs. It is bad enough today that there are so many large programs which nobody understands because they were written by so many people. But at least they were written by people. In the situation where a computer is generating its own programs we would soon find that we were using programs which had never been written or seen by anybody to make our decisions. This would create a situation even further beyond our control than do today's complex computer systems.

Another thing that we must remember, which has been stated before, is that in the course of writing a large program, the ultimate goals of that program may be changed along the way to suit programming needs. So even if somebody were able to devise a "sound" method for making certain types of decisions, there is no guarantee that the "soundness" of that process won't disappear by the time the programmers get done with it.

We should be aware too, of the fact that the people who actually write these computer programs may not know anything whatsoever about the decision making processes that they are trying to program. Their job, as they see it, is to simply write an efficient program that works in some fashion. But the quality

of the final program will ultimately depend upon how well these computer programmers really understand the decision making process that they are attempting to program. We reemphasize the point that just because a program works, it isn't necessarily a good program.

And we cannot overemphasize the curious unconcern that many computer programmers have for social problems. This lack of concern is reflected time and time again in their desire to simply to produce a program that works without thinking about whether or not the way in which it works will actually do justice to the process that it will be used to perform. Weizenbaum's chapter on "Science and The Compulsive Programmer" provides a fine illustration of what we are talking about here.

So the program that we finally end up with is based upon very definite value judgements about how the decision to which it is being applied should be made and it is incapable of employing any human feelings in making that decision. It is a program that has probably had some of its major goals modified because some methods of programming were found easier than others. It may contain errors, be they intentional or not, and it is probably so large that nobody really understands how it works. And as a result, the decision making process cannot possibly be challenged by anybody, no matter how poor that process may be. And what do some people want to do with such a program? They want to use it to make decisions about people that may effect them for the rest of their lives. To give the reader an idea of just how

serious this can become, we need only point out some of the ways in which computers are already making our decisions and some of the uses that have been proposed for them in the future. "For example, a planning paper circulated to the faculty and staff by the director of a major computer laboratory of a major university speaks as follows: 'Most of our research has been supported, and probably will continue to be supported, by the Government of the United States, the Department of Defense in particular. The Department of Defense, as well as other agencies of our government, is engaged in the development and operation of complex systems that have a very great destructive potential and that, increasingly, are commanded and controlled through digital computers. These systems are responsible, in large part, for the maintenance of what peace and stability there is in the world, and at the same time they are capable of unleashing destruction of a scale that is almost impossible for man to comprehend.'"-41 "On September 11, 1971, a computer programming error caused the simultaneous destruction of 117 high altitude weather balloons whose instruments were being monitered by an earth satellite. A similar error in a military command and control system could launch a fleet of nuclear tipped missles. Only censorship prevents us from knowing how many events involving non nuclear weapons have already occurred."-42 In other words, a computer error may very well determine the ultimate fate of every single person and of all future generations on this planet. And what is worse, if anything could be worse, is that the context within

which the earlier statement was made was one of full support for such a situation, and for any other situations in which computers, rather than people are put in charge of our fate. These people want computers to make decisions about how to run our economy. It is forseeable that people may use the decision of a computer in hiring job applicants. The proposals to use computers as teachers may find children who have their lives determined by the academic grades given to them by a computer. It has been proposed to build computers for psychiatric counselling. Can we imagine the possibility of a person being stigmatized for life because a computer has judged him insane? We could go on and on imagining case after case where someone might conceivably propose the use of an artificial intelligence program to make our decisions, but that is not our purpose. Our purpose is to give the reader a feeling for just how dangerous such a situation can become before that level of danger is reached.

Recently, Herbert Simon, one of artificial intelligence's most renowned leaders gave a lecture at which I was present. When I asked him what he thought artificial intelligence programs could ultimately be applied to, he said "... anything you can think of. The sky's the limit." If we allow the trend which is being encouraged by such ideas to continue, we shall soon find that we have given complete control of our destinies to computers. It sounds like something out of a science fiction movie, but science fiction is rapidly becoming

reality. Unless we realize the full consequences of the trend toward using computers to make decisions for us, we will find ourselves totally out of control of our world. We cannot wait until people start doing these things with computers to start worrying about them. We must worry about them right now, before they have a chance to become reality. There are many people who feel that these technological trends are inevitable and that there really isn't anything we can do about them. If we say to ourselves "these things are inevitable," then we will simply give up and not try to stop them from occurring. And when nobody tries to stop them, then they will indeed become inevitable.

So What Do We Do?

The question that now sits before us is "what do we want computers to do and what don't we want them to do?" To my way of thinking, one of the ultimate goals of computer science should be to enable man to make more of his decisions, not less of them. There are many jobs which people perform that can best be described by the statement "it doesn't matter how they get done; it only matters that they get done." Here, computers provide tremendous potential. Because there are many jobs that fit this description, we can use computers to do them. Man will then no longer have to worry about trivial things just because they have to get done. This will provide more and more people with more and more time to do important, non trivial tasks. All

people would be free to pursue whatever areas of human endeavor are relevent to their own fulfillment. In this way, computers can help to maximize the contribution that each person on this earth can make toward human knowledge and culture. Everybody can be an artist, a poet, a decision maker, a philosopher or anything else that turns him or her on. Man will have more time to contemplate the complexities of his world and make better decisions about how to run that world. Today, only a privileged few have the opportunity to participate in such activities. But the goal of the computer at all times must be to allow us to make more of the decisions about our world, not less of them. After all, if computers are making all of our important decisions for us, then what do we as humans have left to do? Do we want to turn ourselves into an obsolete species that exists only to serve our computers? Again, these statements sound like science fiction, but I repeat, we are no longer dealing with science fiction! We are dealing with horest to goodness problems that are confronting us now and will be confronting us for a long time to come.

It is up to us, right now, while there is still time left, to say what type of world we want for ourselves and our offspring. If we do not act now, the decision will slip from our hands and into the hands of our computers!

Chapter four-

Other Consequences of Computer Use

Thus far, we have attempted to asess some of the ways in which computers are currently being used and some of the uses that have been proposed for them in the future. We have seen how the computerization of large amounts of data has brought with it many problems, particularly when that data reflects certain aspects about the private lives of individuals. Yet, as we have said before, the question of damaging or inaccurate data only scratches the surface of the problems that we might expect to see in the future, particularly if computers are used not only to maintain data, but also to assess that data and to make decisions based upon that data. That brought us to the more general question of what types of decisions, if any, a computer can and should be allowed to make for us.

But all of the questions that we have dealt with so far have been questions of what tasks computers should and shouldn't be doing. What we shall do from here is look at some of the indirect effects that the current trend toward computerization is likely to have on our society and our way of life, at least if it is allowed to continue at its present rate. In many ways, we shall attempt to develop at least a partial picture of what a fully computerized and mechanized society might look like for the average person who has to live in it and some of the problems that such a society might face.

Many of the questions that we shall bring up in this section deserve a great deal of consideration in a depth that we shall be unable to provide here. Our purpose, more that anything else, is to simply enumerate some of these problems in the hope that others will be able to begin talking about them and exploring them in the greater depth that they deserve.

The Computerized Beurocracy

One characteristic of large computer programs is the fact that it usually requires a good deal of time and effort to make changes in them. Given the fact that many organizations employ computers for various administrative and beurocratic tasks, we may well find that computers are becoming the foundation of an even more stagnent and unchanging beurocracy than we have today. Without computers, if an organization wants to make a change in some of its beurocratic procedures, it is not usually too difficult to make that change, at least if it is a minor change. However, with a computer handling administrative functions, it is not so easy. Any beurocratic change, no matter how minor, must be made through a costly alteration of the computer program involved. And this added cost and effort needed to change a program may act as a powerful deterrent to any form of beurocratic change.

Already today, people complain of a monstrous unresponsive beurocracy. They want to find ways to make

organizations more flexible and responsive to various needs. All computers promise to do is make them less responsive. This is something which must be taken into account by any organization which is contemplating computerizing its operations. In particular, if that organization wants to computerize certain of its functions, and yet it wants to maintain a large degree of flexibility in those functions, perhaps it should take a second look at that desire to computerize. We must remember that computerization often destroys flexibility by placing too high a price on that flexibility.

Unless we want computers to help create a new beurocracy which is far more rigid than any that exists today, we must be very careful in deciding which functions we wish to computerize.

Responsibility for Errors and Loss of "Human" Factors

The recent advent of the computer has opened up a whole new can of worms for lawyers and legislators. In particular, they have found out that it is very difficult to figure out who should be responsible when a computer makes a mistake which has a damaging effect upon somebody. Already, incidents have occurred where, say, a computer charges someone for a gas bill which he has already paid. That person, upon writing to the gas company, might simply receive another computerized notice telling him to pay up. And upon complaining again, he might just receive a third

notice, again telling him to pay up. This type of situation can be very frustrating, and it provides only a small glimpse of the types of hassles and headaches which computers promise to provide us with in the future. In a nutshell, the computer has served to remove the "human" factor from many of our dealings with others. When a person has a complaint about something that a computer has done, he often finds that ther are no established areas through which he can redress that complaint. As we have stated earlier, computers provide an excellent scapegoat for people. It is all too easy for a company which handles its transactions by computer to say to a disgruntled customer "What can we do? We can only go by what the computer says." It is high time that we put a stop to these shenannigans by creating legislation which clearly defines where the responsibility lies when an organization's computer fails to function properly and causes some damage to somebody as a result. Among other things, any organizatior which uses a computer must provide some well defined human mechanism through which complaints about its computer can be heard and acted upon in an expedient manner.

The process of defining who is responsible for a computer error however, is not always that simple. If the error occurred because of the computer hardware, we might ordinarily expect the hardware manufacturer to assume the ultimate liability. If the error was the result of a faulty program, we might expect the writers of the program to be responsible. If somebody simply entered faulty information into the computer,

then the fault would lie there. If a plane flew overhead and caused some electronic interference at just the right moment to produce an error (and such seemingly ridiculous things do happen), then it is really difficult to figure out where the fault lies. And if this situation doesn't seem complex enough already, add it to the fact that it is generally very difficult if not impossible, and certainly very costly to pinpoint the source of a computer error, and we have got an extremely tangled mess. This mess is one that hopefully will be untangled sooner or later, but for the moment at least, it promises to be an important problem.

Copyrighting a Program

One interesting situation which has developed within the computer field is that it is rather difficult to give the author of a program an exclusive copyright to his program in the same way that we can give the author of a book an exclusive copyright to his book. When two people each write a book, it is easy enough to tell if the two books are the same or not; one simply has to read them. The two books might be on the same topic and they may even try to get the same points across to their readers, yet we can still tell the difference between them.

With programs however, this is simply not the case. Two people can write two totally different programs, each of which does the same thing when it is actually used by someone. So what

do we do? In one sense, we have two different programs. If we were to look at the instructions that constitute each program, they would look totally different. But in another sense, these programs are the same, because if we simply used them without looking at their instructions, they would appear to be doing the exact same thing. So the question is "how do we decide if two programs are the same for copyright purposes?" This is a sticky question which has been encountered a great deal in practice and which must sooner or later be dealt with in some fashion.

The Effects on Employment

Because of the fact that we are taking so many tasks these days and turning them over to computers, we might expect some major long range changes to occur in the types of jobs that people will be called upon to perform. Precisely what jobs people will be doing in the future depends in great measure on what jobs we give computers to do in the future. In many ways, this question relates to the questions of the preceding chapter, namely those of what jobs we want to do ourselves and what jobs we would rather we didn't have to do. We must remember that everytime we give a job to a computer, we are taking a job away from the person who used to do it. And at the same time, we are creating the new job of producing and maintaining that computer. Each time that we contemplate giving a new job to a computer, we must ask ourselves the question "do we prefer the new jobs, if

any, that we are creating to the ones that we are giving to the computer?" And we must also take into consideration how many people the new computer application might put out of work. These are very crucial considerations, for there seems to be a trend nowadays whereby we blindly give any and every task that we possibly can to a computer. And unless we want the human species to have nothing to do except to keep its computers running, we are just going to have to use a little more discretion about the types of jobs that we give computers to do. And in particular, we must always reserve the important decisions about our world and about our lives for ourselves.

Computer Predictions: The Self-Fulfilling Prophecy

Every year in early November, the American public goes to the polls to vote for candidates for public office. And afterwards, they go home and settle back in front of their television sets to see which candidates have won their respective elections. But in recent years, something very interesting has happened. Although people used to have to wait until all the votes were counted before they could figure out who won, nowadays they can find out who will probably win before most of the votes are even counted. Where do they find this out from? They find this out from a computer.

Let us set up a hypothetical situation which will hopefully get a few points across. In this hypothetical

situation, we will use the Democratic and Republican parties as our examples, however we have no intent whatsoever of making any implications about either party. We employ their names solely because people have a great deal of familiarity with these two parties, although we could just as easily label them "party X" and "party Y".

Suppose we are in the midst of electing our President, and, with a few early returns from the states on the east coast the computer predicts that the Republican candidate will win the election. At this time however, many people on the west coast and in Alaska and Hawaii have not yet even gone to the polls. Suppose now that many of the west coast supporters of the Democratic candidate decide not to go to the polls to vote because they feel that their candidate has already lost. (Of course, there might also be some Republicans who don't go to the polls because they feel that their candidate has already won, and in fact, one can come up with all sorts of reactions that various people might have in such a situation, but that is not the point.) Suppose now, when the election is completely over and all the votes have been tallied, it turns out that the Republican candidate won the election by an extremely narrow margin and that that margin was provided to him by some very narrow victories on the west coast, so narrow in fact, that if the west coast supporters of the Democratic candidate had not given up hope, the election would have turned the other way. And just to complete the story, suppose it is found out a few days later that a member

of the committee to elect the Republican candidate paid off somebody to fix the computer so that it would project his candidate as the winner. We needn't describe the uproar that would undoubtedly follow.

We bring up this whole situation because it seems that computers have recently found their way into the "prediction" business. And unfortunately, when a computer makes a prediction of some sort, people are often very willing to accept that prediction without question. They start assuming that what the computer said would happen is really going to happen, and that there is nothing that they can do about it. They may even abandon all efforts to stop it from happening. And once these opposing efforts have been abandoned, then the prediction of the computer may very well come to pass. Such is the nature of a self fulfilling prophecy.

The point is this: we must be scrupulously careful when we use a computer to model a situation and to try to predict what will happen in that situation. Although computer modeling and forecasting certainly has its value in providing insights into various situations, we must remember at all times that these models are still nothing more than programs which are written by people. And if these people do not have a proper understanding of the situation that they are trying to model, then that lack of understanding will be reflected very thoroughly in the answers that the model comes up with. And certainly, as we discussed in the previous chapter, we can never allow the "prediction" of

certain decisions to become the decisions themselves.

The Giant Computer Monopoly

Nowadays, we are seeing an increasing dissatisfaction in this country with big business. Many huge monopolies or oligopolies have grown up out of our past technological advances to a point where they exert considerable control over our everyday lives. The oil industry is certainly a good current example. Because oil is vital ir many areas of our lives, the centralization of that industry into the hands of a few has created many problems in recent years. Right now, it appears as if the computer industry is also coming under the control of only a handful of people and, if anything, the situation is getting worse.

Today, many new technologies have been made possible as a result of our computer technology. In addition, many of our old pre computer technologies have become more effective through computerization.

At the rate things are going, computer technology may one day be an essential building block for most of our other technologies in much the same way that power, for instance, is essential for much of our technology today. Because the future will more than likely see this increasing reliance upon computer technology in many areas of our lives, we may well find ourselves subject to the whims and motives of the few people who will be

controlling the computer industry. If we do not want to have the same types of problems in the future with the computer industry that we are having today with a number of other industries, now is the time to act. We cannot wait until one company runs away with the computer market, because by that time that company will have lots of money, plenty of lobbyists in congress and all the other things that make it exceedingly difficult to deal with a large monopoly.

Reinforcing the Power Structure

Another thing which we must keep in mind is that technology often means power. Those people who have access to computer systems will be able to do certain things much more efficiently and at much lower cost than people who must do the same things without acess to a computer. But, as it stands at the moment, the people who are likely to have access to to computers in the future are those who already have the money to afford them. This means that the people who have the most money today will have the most capability for using computers to expand that wealth in the future. Certainly, in this way, the computer promises to reinforce the old addage that "the rich get richer and the poor get poorer". Care must be taken to see to it that all people who need computer capabilities can gain easy access to them, and not just a priviledged few. Otherwise, we will only aggrevate the current imbalance that exists in the distribution

of wealth and power.

Computers In the Home

In the past few decades since computers were first introduced, we have seen an almost amazing drop in the cost of producing and using computers, and we are rapidly approaching the point where the cost of a computer will be within a financially feasable range for many people. Once this point is reached, there is a good possibility that people will begin to have computers in their homes and offices in much the same way that they now have televisions, radios, refrigerators, dishwashers and many other appliances. Many applications have been proposed for computers in the home which, in the long run, may hold some very vast and important implications for the way in which we run our everyday lives.

For instance, it has been envisioned that people will one day have in their homes a computer terminal which they can use to do all of their shopping. All that the individual would have to do is type a request for a certain piece of merchandise into the computer, and that request would be relayed to the appropriate merchant. The merchandise would then be delivered to the person's home in some fashion. As another example, envision the following situation: You want to have an important business meeting with a dozen other people. So you walk up to your picturphone and dial the phone numbers of the twelve people who

you want to meet with. Then, via a picturphone screen which might be able to display pictures of up to, say, twelve people at once, you can simply sit in your own home and have the meeting. Each person involved in the meeting will be able to see and talk to the other twelve using his own picturephone. Today, such a communications system would be very complex, but computers can make it fairly easy to accomplish such a thing. Another proposal which has been made often is to educate children in their homes through the use of computers. Once again, the trend is toward staying in the home. And we could go on and on mentioning all sorts of other ways that computers can enable us to do all the things we do today without our ever having to walk out of our front door. But such a situation would represent a monumental change in our living habits. People might go through their whole lives without ever coming into direct contact with people other than those who live in their immediate proximity. This situation would undoubtedly cause many other changes in the nature of human existence which we can only speculate about today, but which would certainly be overwhelming. Some of these changes may be good and others may be bad, but it is our responsibility to try to search them out now, before they happen, so that we can exercise some choice in the matter.

Building a Reliance on Computers

Finally, we must remember that whenever we add any new

technology to our society, we will proceed to develop many new ways of doing things which are based upon the new capabilities provided for us by that technology. A society, after the introduction of a new technology, is not simply the same old society plus the new technology. Rather, it is a whole new society. We cannot simply pull an established technology out from under ourselves and expect to automatically go back to where we were before that technology was first introduced. If we took away everybody's automobiles, for instance, people in the suburbs who work in the cities wouldn't be able to go to work any more. The reason that suburbs were able to spring up in the first place was because people were given the capability, via automobiles, to commute into the cities from larger distances. The same type of situation is true of virtually any other technology that is well established within our society. And the point is, that as we use computers to do more and more things for us, we will become more and more reliant upon these computers. We all know the chaotic situation that results when there is, say, a power failure. This is because of the heavy reliance that we have developed upon electrical power. We all remember the state of panic that was induced a couple of winters ago when the oil which we so desperately needed to run our cars and heat our homes was suddenly cut down by the Arab oil embargo. And even today, the energy crisis, which is a major issue of our time, represents a perfect example of the types of problems that can result when we come to rely too heavily upon a particlar technology to get

things done for us. In many ways, the ability of our nation to survive in its current technological form will depend not upon the technological power that we posess, but rather upon our ability to fulfill the dependencies that our technology has created. The oil crisis may only be the first of many situations in which our tecnnological "achilles heels" may prove disastrous for us. Imagaine, for instance, if all commercial transactions were handled by one central computer and something suddenly went wrong with that computer. We would find ourselves in the midst of a "financial blackout". Imagine too, if a computer was used to keep track of everybody's money so that we wouldn't have to use paper currency, (which, by the way, has already been proposed by many people) and that computer for some reason (sabotage or otherwise) lost all of its information about how much money everybody had. The chaos would be unbelievable! As we come to rely upon computers in more and more situations, we must consider the consequences of a computer failure in these situations. If the consequences of a computer failure in any given situation would counterbalance the benefits derived from the application of a computer in that situation, then perhaps we shouldn't be using a computer there in the first place.

Conclusion

In conclusion, let me say that I think that computers, if used properly, offer a great deal of potential for mankind. Concelvably, we could write computer programs to do anything and everything that our little hearts desire. But that doesn't mean that we should write programs to do all these things. We must make a long range determination (and by long range, I don't mean a year or five years but rather fifty or a hundred years and even longer) right now as to the types of tasks that we consider appropriate for computers to handle and those which we consider inappropriate for computers to handle. And we must take firm steps to avoid the inappropriate uses. We simply should not use computers to do everything that pops into our heads. In those cases where we do decide to use computers, we must see to it that the information that they contain and the way in which that information is used is subject to the utmost care and scrutiny. We must also be continually aware of the fact that the continued expansion of computer applications will bring about many significant changes in the world we now know. We must make a continuous effort to anticipate those changes, and we must decide if we want them or not. And if we don't want those changes, then we must have the courage and foresight to say "no" to ourselves. If we simply let our computer technology evolve in whatever direction it happens to evolve, we will soon find that our computer technology is simply one more technology over which we

have lost control. But if we make a concerted attempt to visualize the type of world that we want for ourselves and our children in the future and to apply computers in that direction, while keeping the computers out of the areas where they will be a deterrent to reaching that world which we visualize, then computers may be of tremendous service to us and all future generations.

In this thesis, we have purposely chosen to pay attention to some of the more ominous possibilities which computers offer to man. There are a number of reasons for this. Much of the current literature about computers has tended to give them glowing reviews, while it has often ignored the possibilities of misuse. Often, this stems from the fact that the people who write the literature about computers are often the people who stand to benefit the most from a positive public attitude toward computers. Also, the general public does not have sufficient knowledge of the limitations of computers to be able to argue effectively against certain computer applications, except on the grounds of "gut" feel or of plain moral outrage against the dehumanization which often accompanies computers. Computers at the moment are held in exaggerated esteem by much of the general public, and there are too many people who have the mistaken idea that computers can solve any problem we give them to solve better than can any other medium. But most of all, the reason that we have tended to concentrate on pitfalls rather than on benefits has been to jolt people out of their romance with computers and

into a realization of some very real dangers that computers may one day confront us with if we don't actively seek to avoid these dangers.

Armed with a vision of beneficial computer applications and a realistic appraisal of the detrimental ones, we will hopefully be better equipped to make intelligent decisions about computers in the future.

Appendix-

How a Computer Works

Bits and Data

The lowest level of the computer that we shall deal with is the bit. We do not go any lower because the concept behind the bit is a simple one that needs little further clarification. A bit is simply a piece of electronics that is either on or off. Just like your televisions, radios, lights, cars and lots of other things are on or off, so is a bit. Probably the best way of thinking of a bit is as a little light switch which is turned on and off by the electronics of a computer rather than by a human. A computer has a tremendous number of these on-off bits, and everything that it does is ultimately determined by which bits are on and which bits are off.

Since we want to get the computer to do certain things for us (arithmetic, bookkeeping, handling telephone calls etc.) we must find some way of taking whatever it is we want to do, and translating it into some sequence of these on-off bits.

Let's now begin to take a look at how one might go about organizing these little on-off bits to get a computer to actually do something.

We'll assume for the moment that we only want our computer to be able to deal with letters and numbers; if

we want our computer to have letters and numbers we shall have to find some way to represent them inside our computer by using these bits.

As we said before, a single bit can be either on or off. Let's say that we were to take two of these bits and put them side by side. Now, we have four possible ways of lighting these two bits. They are 1) off-off, 2) off-on, 3) on-off, 4) on-on. Since it becomes rather tedious to write the words "on" and "off" everytime we want to talk about a bit, we'll use the digit "1" to represent the word "on" and the digit "0" to represent the word "off". So our new way of representing these four lighting patterns is 1-00, 2-01, 3-10, 4-11. If we have three bits, then we can come up with eight possible ways of arranging them, and four bits provide us with sixteen ways. As we can see, it seems that everytime we add another bit we double the number of ways of arranging them. In fact, this is the case, thus five bits can give us 32 arrangements. If we want to represent the 26 letters of the alphabet, we can simply take five bits and use one of the 32 possible lighting arrangements to represent each letter. One such assignment might be as follows:

A 00000	J 01001	S 10010	UNUSED
B 00001	K 01010	T 10011	11010
C 00010	L 01011	U 10100	11011
D 00011	M 01100	V 10101	11100
E 00100	N 01101	W 10110	11101
F 00101	O 01110	X 10111	11110
G 00110	P 01111	Y 11000	11111
H 00111	Q 10000	Z 11001	
I 01000	R 10001		

Figure 2

Now, anytime we want to represent say, the letter "I" in our computer, we find a chunk of five bits somewhere and put them in a "01011" or off-on-off-on-on sequence, as shown in the chart.

If we don't want the six unused bit arrangements to go to waste, we might use them to represent typewriter characters, such as periods, commas etc. In reality, many computers use more than Five bits to represent these characters because there are generally more keys on a typewriter than can be represented by Five bits. But for our purposes here, this fact really doesn't matter. In the future we shall refer to both the letters and the special Typewriter characters simply as "characters".

For numbers we can do the same thing. Say we wanted our machine to contain all the integers from 0 through 31. We might represent the numbers in the following way:

0 00000	8 01000	16 10000	24 11000
1 00001	9 01001	17 10001	25 11001
2 00010	10 01010	18 10010	25 11010
3 00011	11 01011	19 10011	26 11011
4 00100	12 01100	20 10100	27 11100
5 00101	13 01101	21 10101	29 11101
6 00110	14 01110	22 10110	30 11110
7 00111	15 01111	23 10111	31 11111

Figure 3

In order to distinguish the numbers from the letters, we will simply keep them in seperate sections of our computer. In practice, this is rarely done, but it will help to make things

simpler to understand. Thus whenever we encounter the bit sequence 01100 in the number section of our computer, we will know that it represents the number "12". If this sequence were in the letter section of the machine, it would represent the letter "M".

Those who have ever worked with the base 2 number system will notice that the bit sequence corresponding to each number is actually the base 2 representation of that number. The idea behind base two is fairly simple, and if the reader looks closely at figure 2 starting with "0" and ending with "31", he will probably notice a pattern. To represent a wider range of numbers, we simply add more bits, but follow the same general pattern of representing numbers as was shown in figure 2. For instance, we would find that "1011010111" is the ten bit representation for the number "725". Thus to represent the number "725" in our computer, we would have to take ten bits and put them in a "1011010111" arrangement.

Beacuse numbers in base two are represented by zeros and ones in the same fashion as are the bits of a computer, it becomes very convenient to represent all numbers in a computer in base 2.

In a computer, bits are rarely thought of on an individual basis. Instead they are put into groups of a (usually) fixed size, and a bit is thought of as being a part of this group of bits. The size of one of these groups of bits varies from machine to machine, and 8, 12, 16, 18, 32 and 64 are not uncommon

group sizes. These groups are generally referred to as bytes, words, or locations, the terminology also differing from one machine to the next. In order to provide a concrete example for future discussion, we shall assume that we have a machine which groups it bits into chunks of 18, and we shall refer to each of these groups as a "location". Let's see how we might go about putting the numbers and characters that we talked about earlier into one of these locations. We start with numbers.

It just so happens that 18 bits can be arranged in 262,144 different ways. Thus, in a single location we can represent any number from 0 to 262,143. Of course, we might also want to have negative numbers in our computer. Instead of using the 18 bits to represent a number from 0 to 262,143, we can use 17 of the bits (17 bits can be arranged in 131,072 different ways) to represent any number from 0 to 131,071, and use the 18th bit to tell us whether the number is to be interpreted as a positive or negative number. We'll say that if the 18th bit is a "1", then the number is negative, and if the bit is a "0", the number is positive. Thus a number in our machine will have this general form to it:

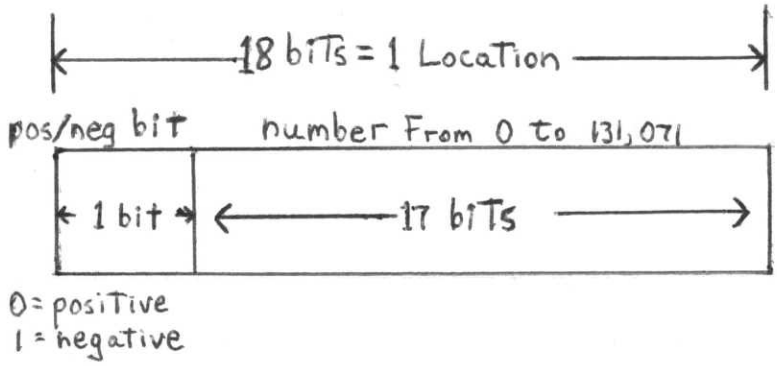

Figure 4

Now we can represent any number from -131,071 to +131,071. This is actually not the method used to represent negative numbers in most computers, rather it is a simplification made for the purposes of our understanding. The actual method used in most computers is called two's complement, but we shall not discuss it here.

We saw before that we could represent any letter of the alphabet by using five bits. If we wish to represent both upper case and lower case letters, we can add an extra bit to each letter and make this bit a "1" if the associated letter is capital, and "0" if it is small. Now, our method of interpreting letters is to look at the original five bits to see what the letter is, and then to look at the sixth bit to see whether that letter is capital or small. Since we have 18 bits in a location and we only need six bits to represent a letter, we can put three letters into each location. Thus, if we wanted to represent the letters "T", "h", and "e" in a single location, that location would look like this:

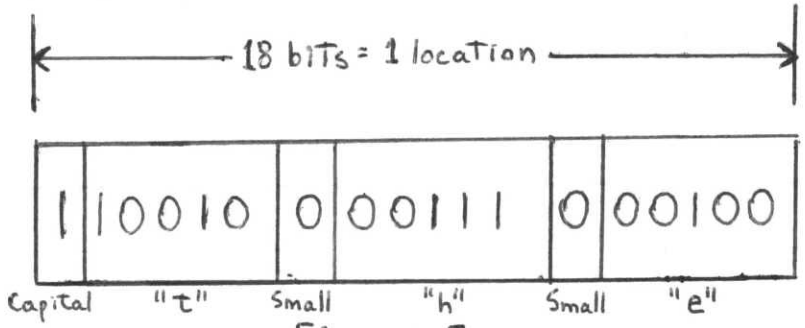

Figure 5

The reader will notice that in effect, we have put the word "The" into a single location. The obvious extension is to put a bunch of locations side by side to form even larger words. And this way we can build into sentences, paragraphs, books, etc. Since we can get 3 characters into a single location, if we wanted to be able to represet people's last names of up to, say, 12 letters, we simply have to take four of these [redacted] locations and put three [redacted] letters into each.

From this point on, we can avoid the hassle of thinking of the computer as a bunch of bits, and we can instead think of it as a bunch of locations, each containing bits organized in the manner that we have just described. Now we shall concentrate on trying to put a bunch of these locations together in order to produce a computer.

Before we go on however, the reader is reminded that we are only trying to convey to him a general feeling for some of the concepts that have enabled us to turn a bunch of on-off switches into a computer. Simply having a feeling for how we can represent real world things such as words and numbers is much more important than thoroughly understanding the details of the examples provided. They are intended only to provide a concrete

reference for understanding the general concepts.

Locations

Suppose that we were to take 4096 of these locations and pack them into a single computer. In order to be able to talk about any one of these 4096 locations, let us give each of these locations a number from 0 to 4095. Thus we shall distinguish locations by referring to them as "location 257" or "location 3967" etc. We shall refer to this whole group of 4096 locations as the computer "memory". Finally, in addition to these 4096 locations we shall create one special location of 18 bits. This location will not have a number; instead it will be referred to as the "accumulator". This accumulator is to be thought of as an auxiliary location whose function is to act as a sort of "scratch pad" for the computer. That is, the computer will use it to "jot down" things as it goes along doing whatever it is doing. The reader will be able to develop a better feel for exactly what we mean here once we show an example of how the accumulator might be used. Although we can only put one accumulator in our computer, there are many computers with more than one accumulator, and some without any. The IBM 360/370 computer has 16 of these accumulators, and in that machine they are given numbers from 0 to 15. These accumulators are also referred to as registers on many computers.

Thus far, we have designed a computer that can hold a

bunch of letters and numbers in it. But it doesn't do us much good if these letters and numbers just sit around in the computer without doing anything. Thus we would be wise to attack the problem of getting the computer to actually do something useful with all these letters and numbers.

Suppose that we wanted to get the computer to perform the calculation (3*4)+7. In order to do this we would first want to multiply "3" times "4" to get "12". Then we would want to add a "7" to our "12" to get "19", which would be our final answer. Let us assume for the moment that somewhere in the computer there are three locations, and it just so happens that one of these three locations has a "3" in it, one of them has a "4" in it, and the other one has a "7" in it. In addition we will have a fourth location into which we want to put the final answer of "19" once we calculate it. We should also remember that the accumulator was given to us as a scratch pad, so let's use it as such. If we wanted to tell the computer how to go about calculating (3*4)+7, we might tell it the following:

1. Find the location with the "3" in it, then take that "3" and copy it down in the accumulator.

2. Find the "4" and multiply it by the "3" that we already have in the accumulator. Leave the result (12) in the accumulator.

3. Find the location with the "7" in it, get the "7", and add it to the 12 that is already in the accumulator, leaving

this new result (19) in the accumulator.

4. Take the contents of the accumulator(19) and copy them down in the location that has been designated to recieve our final answer.

5. Halt because we are now done.

The way in which we are using the accumulator as a type of "scratch pad" should be a little clearer from this example. This is in fact, our first example of a program. Unfortunately however, a program written in the above fashion is not generally acceptable to a computer. What we have simply done is outlined the steps that we might tell the computer to follow in order to perform the calculation (3*4)+7.

This example establishes two important points. It shows first of all, that to get the computer to perform a specific task we have to break that task down into a bunch of very specific instructions. When a bunch of these instructions are put together in some workable fashion they constitute a program. Secondly, it establishes these instructions as the basic units of a program. Thus all tasks that the computer can perform must ultimately be described in terms of these instructions in the same way that anything that we as humans might want to say must be ultimately be put in terms of words. Let's now take a closer look at these instructions and see if we can find a way to fit them into our computer.

Programs and Instructions

The first thing that we will do is create a third section of the computer for all of our instructions. Thus we have the "character" section, the "number" section and the "instruction" section in our computer.

As we saw before, the first step that we wanted the computer to follow in order to calculate (3*4)+7 was:

"1. Find the location with the "3" in it, then take that "3" and copy it down in the accumulator."

We might have good reason to believe that copying the contents of some location in memory into the accumulator is something that people using our computer will want to do frequently, even when they are not performing the calculation (3*4)+7. Thus, we will set out to design a general purpose instruction that can be used to copy the contents of one of our 4096 locations into the accumulator. The instruction that we will create will be called "load the accumulator", and it will be accompanied by the number of the location whose contents we wish to load into the accumulator. Thus, if we say "load the accumulator 297" it means that we want to take whatever we find in location number 297 and copy it into the accumulator. In order to avoid writing out the words "load the accumulator" all the time, we shall abbreviate them as "lac". Thus "lac 297" is what we will actually say if we want the computer to load the contents of location 297 into the accumulator.

It is important for the reader to note that we are not putting the number "297" into the accumulator. Rather we are putting whatever we find in location number 297 into the accumulator. This distinction is important to remember.

If we look back again, we see that the second thing that we wanted the computer to do in order to calculate (3*4)+7was:

"2. Find the "4" and multiply it by the "3" that we already have in the accumulator. Leave the result (12) in the accumulator."

Once again, we have an operation that it seems we might want to perform frequently, namely multiplication. So we shall create another instruction and call it "multiply". This instruction too, will be accompanied by

the number of the location that we are talking about. Thus if we want to multiply whatever is in the accumulator by whatever is in location 3167 and leave the result in the accumulator, we simply say "multiply 3167." An appropriate abbreviation for multiply might be "mpl", and thus we will use it. This means that we would really say "mpl 3167" to multiply whatever is in the accumulator by whatever is in location 3167. Note again, that we are not multiplying the contents of the accumulator by the number "3167". Rather we are multiplying the contents of the accumulator by whatever happens to be in location 3167 at the time. It is important for the reader to catch the distinction between the number of a location and what is actually contained in that

location.

Step 3 said:

"3. Find the location with the "7" in it, get the "7", and add it to the "12" that is already in the accumulator, leaving this new result (19) in the accumulator."

Addition too, is the type of thing that we will probably want to do often. So we will create a third instruction called "add", and it will work in the same fashion as the other instructions, except it will add whatever is found in the specified location to whatever is found in the accumulator. Since the word "add" is short enough already, we do not have to abbreviate it.

The fourth thing that we wanted to do was:

"4. Take the contents of the accumulator (19) and copy them down in the location that has been designated to receive our final answer."

Again it seems that we have a task which we will want to perform often; namely copying whatever is in the accumulator into a location in our memory. So we'll invent another instruction called "store", which we shall abbreviate "sto". Thus "sto 1485" will take whatever is in the accumulator and copy it into location 1485.

Finally, we said:

"5. Halt because we are now done."

To take care of this situation we will create one final instruction called "halt", abbreviated "hlt". Note that we don't

have to specify a location number with the halt instruction as we did with the other four instructions. This is because halt doesn't do anything with any of our locations; it simply stops the computer when we are done with what we want to do.

At this point, we have developed for our computer a set of five instructions that we can use to get it to do something, namely lac, mpl, add, sto, and hlt.

But we can't simply stand outside the computer and shout these instructions at it. We must find a way of actually putting these instructions inside the computer. We'll approach this problem in much the same way that we approached the problem of putting in letters and numbers earlier; by seeing if there is some way to represent them using a bunch of bits.

As we said earlier, a location in our sample machine has 18 bits in it, So let's see if we can find a way to represent an instruction using 18 bits. If we can, then we can put a whole instruction into a single location.

If we look at the instructions that we have developed, we find that each one of them except "hlt" has a common two part structure. The first part is the operation that we want the instruction to perform (such as add, mpl, sto, lac) and the second part tells us which location we want that operation to be performed on. We shall refer to the first part of an instruction as its "operation" part, and the second part as its "location" part.

As we said before, our computer has 4096 locations

which are numbered from 0 to 4095. It just so happens that 12 bits can be arranged in 4096 different ways. As a result, we can use 12 bits to refer to any one of the 4096 locations in the computer. So if we wanted to talk about location 18, we would use the 12 bit pattern "000000010010", which is simply the representation for number "18" as shown in figure 2, except with some zeros padding it on the left. This means that we can use 12 bits to form the "location part" of our instructions, because 12 bits can tell us which of the 4096 locations the instruction is referring to. Since we have only used 12 of our 18 bits for the "location" part of the instruction, we can use the other 6 bits for the "operation" part.

Six bits can be arranged in 64 different ways, thus we can have up to 64 different types of instructions. We might (although we don't necessarily have to) assign the following bit patterns to the instructions that we have created so far:

lac: 000000

add: 000001

mpl: 000010

sto: 000011

hlt: 000100

and at this point we still have room for 59 more instruction types. Finally, we shall have to adopt a standard way of putting these instructions into memory, so we'll put the six "operation"

bits on the left side of a location and the 12 "location" bits on the right side. So for instance, the representation inside the computer for the instruction "sto 19" would be:

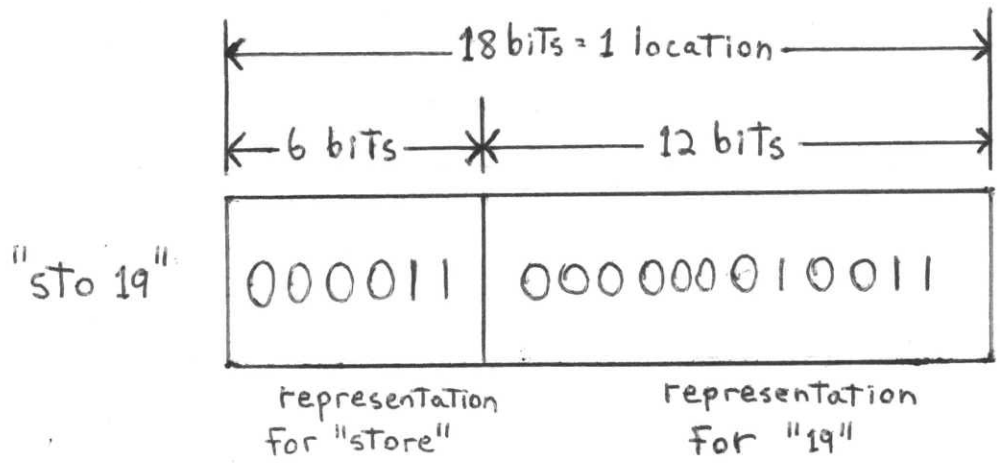

Figure 6

and as we see, this fits nicely into 18 bits. Now, whenever we encounter the bit sequence "000011000000010011" anywhere in the instruction section of the computer, we know that it means "sto 19" and will cause the computer to store whatever is in the accumulator into location 19. For an instruction such as "hlt" which doesn't refer to any location and thus doesn't require a "location" part, we can simply set the location part to all 0's. Thus "hlt" will be represented by the bit pattern "000100000000000000".

Up to this time, we have assumed that our computer is divided up into three distinct parts, one for characters, one for numbers and one for instructions. The primary reason behind doing

this was to drive home the point that the same sequence of bits can mean different things, depending upon how that sequence is interpreted. In our machine, a single bit sequence can be interpreted as a character, a number or an instruction, depending upon which section of the machine we find it in.

Let's go back now to the program that performed the calculation (3*4)+7 and see what we can do with it now that we have found a way to put instructions into our computer. Let's say that we take locations 0 through 10 of our computer and set them up in the following way:

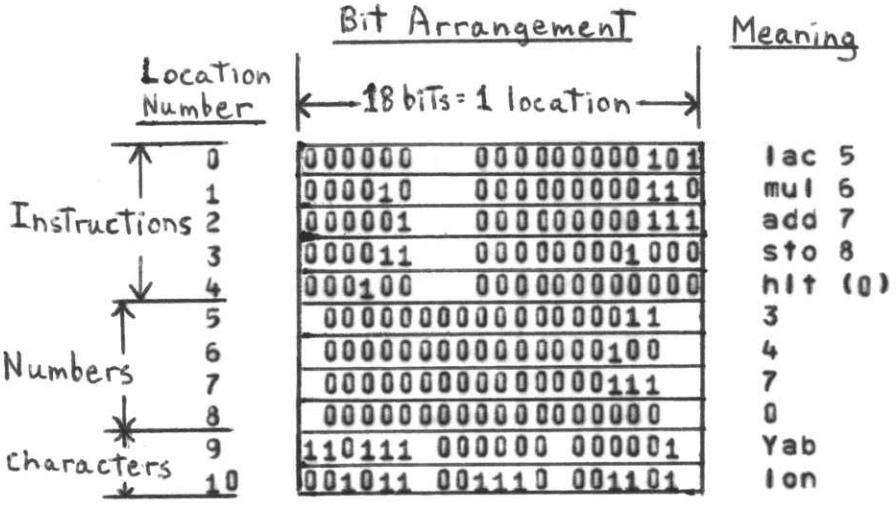

Figure 7

As we can see, we have put some instructions into locations 0 through 4. We have put the number "3" into location 5, the number "4" into location 6, and the number "7" into location 7. Location 8 has the number "0" in it, and this is the location that will receive the final answer to the (3*4)+7 calculation. Finally, locations 9 and 10 respectively have the letters "Yab" and "lon"

in them. Lets see what would happen if we told the computer to begin following the instructions at location 0 and continue until it runs into a halt instruction.

In location 0, we find the instruction "lac 5", symbolized by the bit sequence "0000000000000101". This is telling the computer to take whatever is in location 5 and put it into the accumulator. As we can see, location 5 has the number "3" in it, thus the computer will copy that "3" into the accumulator. The next instruction is in location 1 and there we find the bit sequence "0000100000000110" which really means "mul 6". Thus the computer will get whatever is in location 6 and multiply it by whatever is in the accumulator. Since location 6 has a "4" in it and the accumulator has a "3" in it, the result of the multiplication will be "12". This "12" will then be left in the accumulator, and we are then ready for the next instruction.

The next instruction, found in location 2, says "add 7" and is represented by the bit sequence "0000100000000111" This will cause the computer will take whatever is found in location 7 (which just so happens to be the number "7") and add it to whatever is in the accumulator at the current time. Since the accumulator has a "12" in it at this point, when we add "7" we will get a total of "19". This "19" will then be left in the accumulator, and we are ready for the next instruction.

Going to location 3 for the next instruction, we find the bit string "0001100000000001000" which stands for "sto 8".

Thus the computer will take whatever is in the accumulator (now a 19), and put it into location 8. Since location 8, which is our answer location, now has the answer in it, we are done with our task. And as we can see, the next instruction, found in location 4, is a "hlt" instruction.

With this example, a number of things begin to fall into place. First, we have finally gotten our computer to do something. In a strong sense, the procedure followed by the computer in executing a program is like a treasure hunt. The computer is given instructions which tell it which locations to look into and what to do with what it finds in these locations. Then it goes back for another instruction, follows it, and comes back for still another instruction until one of the instructions tells it to stop.

Secondly, it shows that we don't really need seperate sections of the computer for characters, numbers and instructions. All we really have to do is keep track of which locations contain numbers, which contain characters and which contain instructions. For instance, in the above example we had the computer look for its first instruction in location 0 because we knew that we wanted the bit string in location 0 to be taken as an instruction. If we had told the computer to look for its first instruction in, say, location

6, it would have taken the number "4" in location six and tried instead to interpret it as an instruction. Of course, we don't

want the contents of location 6 to be interpreted as an instruction; we want them interpreted as a number. Having told the computer to look for its first instruction in location 6 would have been an error on our part. Pursuing this point further, let's say that location 2 had an "add 9" in it instead of an "add 7". The computer would have gone to location 9, found the bit string "1110000000000001" which is supposed to represent the letters "Yab", and added it to whatever was in the accumulator at the time. In short, it would have interpreted the three letters "Yab" as a number instead. Certainly, this was not our intent. So we can see that the computer really doesn't worry about the difference between characters, numbers and instructions. Instead, it leaves it up to the person writing the program to make sure that he uses everything in the right way. If he doesn't, his program simply won't work as he wants it to. From now on we won't think of our computer as having an instruction part, a character part and a number part. Instead, we shall think of it as having programs, which contain instructions and data. The instructions tell the computer what to do. Locations 0-4 of Figure 6 certainly qualify as instructions. Data will simply be anything that is manipulated in some way by these instructions. Thus, everything in locations 5-10 of figure 6 will be called data. A list of stockholders on a corporation computer will be thought of as data, while the things telling the computer what to do with that list are the instructions.

Another thing that the reader might notice is that this

program is capable of not only multiplying "4" times "3" and then adding "7", but it is capable of multiplying any two numbers and then adding a third. For instance, if we were to put a "25" instead of a "3" into location 5, a "287" instead of a "4" into location 6 and a "3987" instead of a "7" into location 7, we would be able to use the program in locations 0-4 to perform the calculation (25*287)+3987. It is nice to know that we have to simply change the numbers in the appropriate locations rather than write a new program everytime we want to perform the same calculation with different numbers. This general adaptability of programs is one of their nicest features.

As we go along trying to improve our computer, we shall come across the need for some new instructions beyond our original five, and we shall add them at that time.

One of the most useful properties of a computer is its ability to make decisions. In the (3*4)+7 example, we simply gave the computer a bunch of instructions to execute, and it executed them. But often the case arises where we do not know in advance exactly which instructions we want the computer to execute and which ones we don't. What we would like to be able to do is let the computer itself decide which instructions to execute and which ones to ignore. For instance, if we are writing a program to guide a rocket to the moon, the computer may run through some section of the program which figures out whether or not the rocket is on course. If the program figures out that the rocket is on course, then we want to leave the rocket alone. If

not, we want to start executing a bunch of instructions designed to put the rocket back on course. Thus we need some way for the computer to decide whether or not to call on the routine which is designed to put the rocket on course again. In fact, most tasks which a computer might be called upon to perform do involve at least some amount of decision making on the part of the computer. If we give the computer a social security number and ask it to find the name of the person with that number, the program that we give the number to will search through a whole list of social security numbers and names. Each time it looks at a social security number on the list, it has to decide whether or not it matches the number it was asked to look for. If it does match, then we simply send out as the answer the name that is associated with that number on the list. If it isn't a match, then we have to continue searching through the list. There is, as we can see, the need for the computer to make a decision here.

We shall now introduce a new type of instruction designed to allow the computer to make such decisions. Before we do that however, we shall first introduce a "subtract" instruction, abbreviated "sub". This instruction works in the same way as did "add" and "mul", except it subtracts. We are adding this instruction only because we need it for our next example. Let's say that we want to write a program to figure out which is the greater of two numbers, and let us also say that these two numbers are in locations 100 and 101. Once the computer figures out which of the two numbers is greater, it copies the

greater number into location 102. Thus, we can think of location 102 as our answer location. Let us first look at the case where the number in location 100 is greater than the number in location 101. This means that we want to copy the number from location 100 into location 102. The program that we would use to do such a thing might be as follows:

1. lac 100 -take the number found in location 100 and put it into the accumulator.

2. sto 102 -take the number from the accumulator and put it into location 102

3. hlt -we are now done

We shall henceforth refer to these 3 instructions as "program 1". Thus, wherever location 100 contains a number larger than the one in location 101, we want to follow program 1.

If however, the number in location 101 is greater than the number in location 100, then we want to copy the number in location 101 into location 102. One program which would accomplish such a task is the following:

1. lac 101 -take the number found in location 101 and copy it into the accumulator.

2. sto 102 -store the contents of the accumulator into location 102

3. hlt -we're done!

In this program we are doing almost the same thing as before, except we are putting the contents of location 101 into the accumulator rather than the contents of location 100. This program will be referred to as "program 2". It becomes apparant now that we must provide the computer with some way of deciding whether to execute program 1 or program 2, and that this choice must be based on which of the two locations has the greater number in it. To do this, we introduce a new instruction called "jump if accumulator greater than zero" which will be abbreviated "jgt". What exactly does this instruction do? Let us say that the computer is busy running a program and at some point it were to discover the instruction "jgt 1000". The computer would first look at the accumulator and see if what it contains is greater than 0. If the number in the accumulator is indeed greater than zero, the computer goes to location 1000 to look for its next instruction, and continues executing instructions from there. If, on the other hand the number found in the accumulator is not greater than zero, then the computer will do nothing. It will simply continue executing instructions without jumping to location 100. The reader may at this point be wondering how such an instruction can be used effectively, and the next example will hopefully answer that question.

This program shows one way that we might decide whether location 100 or location 101 has the greater number. The reader should notice a few things, however, before he dives into the program and

tries to figure it out.

We see first of all, that the instructions in locations 3, 4 and 5 are simply the three instructions that constitute "program 2", which is the program that we want to use if location 101 has the greater number. We will also note that locations 6, 7 and 8 contain "program 1". This is the program that we want to use if location 100 has the greater of the two numbers. As we shall see in a moment, locations 0, 1 and 2 contain instructions that are being used to decide whether to use program 1 or program 2. As the reader will see, locations 0 and 1 cause us to perform the subtraction "contents of location 100 - contents of location 101". If the result of this subtraction is greater than zero, that means that location 100 has the greater number and so we want to follow program 1. If the result of the subtraction is less than zero, then location 101 has the greater number and we want to follow program 2. Finally we show locations 100, 101 and 102, which are the three locations that we are interested in manipulating. At the start we shall assume that location 100 has a 6927 in it and that location 101 has a 2 in it, so location 100 quite ooviously has the greater number. The reader should ignore the numbers in parenthesis for the moment. Of course, the computer doesn't actually have, say, a "lac 100" in location 0. Rather it has the strirg of bits used to represent "lac 100" in the way that we discussed earlier.

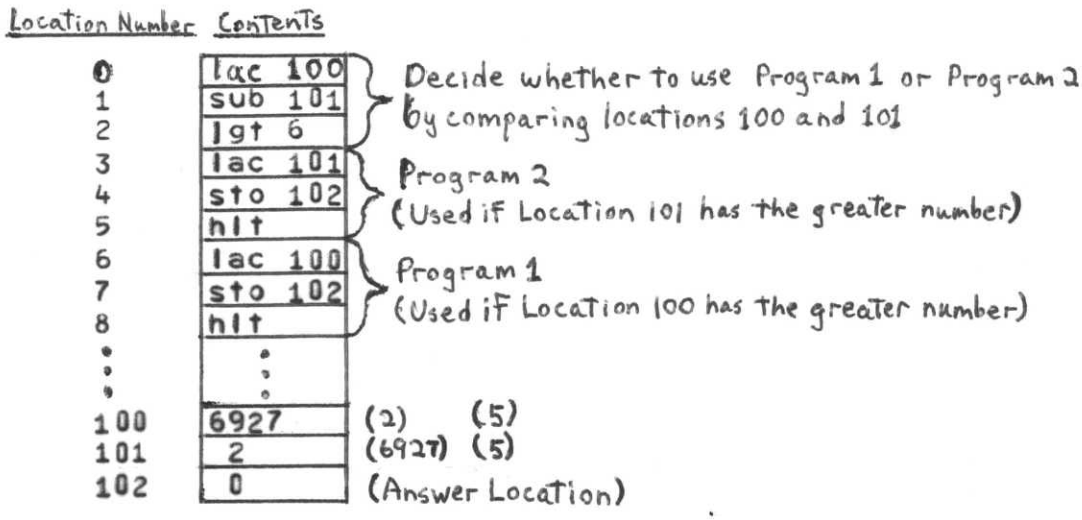

Figure 8

If location 100 has the greater number, that means that we should be following program 1, which resides in locations 6, 7 and 8. Lets see what does happen.

We start executing our program in location 0. There we find the instruction "lac 100". This will cause us to copy whatever in in location 100 (in this case a "6927") into the accumulator. Once we have a 6927 in the accumulator, we go on to the instruction in location 1. There we find an instruction to subtract whatever is in location 101 (in this case a "2") from the 6927 in the accumulator. Well, since 6927-2 =6925, we now have a "6925" in the accumulator. Finally, we go to location 2 which has the instruction "jgt 6". In other words, if whatever is in the accumulator is greater than 0, we will go to location 6 for our next instruction. Certainly the 6295 in the accumulator is greater then zero, so we do go to location 6. And as we said before, location 6 is simply the start of "program 1", which is indeed the program that we want to use in this case. If we follow

along from location 6, we first see a "lac 100". This takes the "6927" in loction 100 and puts it into the accumulator. Next we go go to location 7. This says "sto 102" which causes us to store the "6927" from the accumulator into location 102, which is our answer location. Finally, we halt in location 8, and our program has done what we want it to do. Now, let us look at the other case; where location 101 has the greater number. We shall simply switch the 2 and the 6927 and see what happens. If everything goes well, we should end up following program 2 this time around.

Starting our program at location 0 again, the first thing to happen is that the "lac 100" causes us to load the "2" in location 100 into the accumulator. Then, the "sub 101" in location 1 causes us to subtract the "6927" in location 101 from the "2" in the accumulator. 2-6927=-6925, and certainly this number is less than zero. After this "-6925" is put into the accumulator, we find the "jgt 6" in location 3 once again, which says "if whatever is in the accumulator is greater than zero, then go to location 6." But the number in the accumulator certainly isn't greater than zero, so we simply ignore this instruction. Thus, we do not go to location 6, rather we continue along at location 3. But lo and behold, location 3 is simply the start of program 2. Location 3 says "lac 101", so the "6927" from location 101 is put into the accumulator. The "sto 102" in location 4 is executed next and ccauses us to dump the "6927" from the accumulator into location 102. Once again, our answer is safely tucked away in location 102. Location 5 says "hlt" and

so we are done.

There is one special case however that we ignored; the case where locations 100 and 101 each have the same number. Let us quickly see what would happen if these two locations contained, say, the number 5. The "lac 100" in location 0 causes the "5" in location 100 to be put into the accumulator. Then, the "sub 101" in location 1 causes us to look in location 101 and subtract whatever is found there from the "5" in the accumulator. We find a "5" in location 101, and since 5-5=0, the accumulator will now contain a "0". The "jgt 6" in location 2 will only cause a jump to location six if the accumulator has a number greater than zero in it. But the accumulator has exactly zero in it, so no jump will be made. So we simply continue on to location 3. We know from before that the net effect of the instructions in locations 3, 4 and 5 is to move whatever is in location 101 into location 102. Thus, a 5 will end up in location 102 as our answer. This seems like a reasonable result for the situation, so it looks as if our program really works in all cases. (When location 100 has the larger number, when location 101 has the larger number and when they each have the same number.) We should mention that having checked for the special case where each location had the same number was essential before we could claim that our program fully worked. If we hadn't checked this case, then we would only be able to claim that our program works for most cases, namely where the locations have different numbers in them. But we could not have claimed that it worked for all cases,

since we had not checked for the case where the locations contained the same number. We point this out because it brings out something very important about the nature of the errors that one often finds in a program or in a computer system. Many programs are written which appear to work all the time, yet don't. Instead, they work almost all of the time, yet they don't work when some special condition occurs which has been overlooked in both the writing and the testing of the program. And it is not an error resulting from faulty instructions. It is an error due to the fact that the programmer did not have a full understanding of the problem that he was trying to solve by writing the program. The "special case" of two numbers being equal is not one that would be frequently overlooked (hopefully), but it shows how in a much more complex program, there might just be something that is overlooked. If a program just plain doesn't work under any circumstances because simply wrote the program wrong, it would be easy to see that the program isn't working properly when it is tested, and nobody would use the program to do anything serious unless they were looking for trouble. But if the program does appear to work when it is first tested, yet in reality it contains an error that occurs only in an obscure situation which was overlooked, then we have a problem, because people will use the program thinking that it works all the time, when in fact it doesn't. And often, the process of finding the rare cases for which a program doesn't work involves simply waiting around until somebody stumbles onto one of these rare cases while actually

using the program. Sometimes, even this isn't good enough. Often an error will occur and the person using the program won't realize it. He may thus accept an incorrect answer from the computer, thinking that it is correct. Herein we find the crux of one of the problems that we shall discuss later on: that computer programs frequently contain errors that can result in undetectable mistakes. If these errors were at least detectable, it would be okay, since nobody would use the erronious results. But if a mistake is undetectable, as it often is, we then have the very serious danger of people relying upon wrong answers from a computer to make decisions. These faulty decisions may in turn cause serious problems. We shall look at many case studies where such a situation has caused untold misery for the people and organizations effected.

So far, we have made statements such as "the computer takes a number from a location" and "the computer looks at the next instruction" and "the computer does this" and "the computer does that". But we have never really adressed the problem of how the computer does all the things that we are talking about. How does the computer add two numbers or read the next instructior? Is there a little man running around inside the computer doing all these things? One would suspect not. All of these things are actually done by what is generally referred to as the computer "hardware". The hardware of a computer is simply the collection of wires, transistors and whatnot that actually constitutes the computer. This is in contrast to the programs that we have been

talking about which are generally thought of as the "software" of the computer. Thus, it is the "software" of a computer, namely those programs that are written for it, which tells a computer how to perform a specific task. And it is the "hardware" of the computer that can actually look at the instructions of a program and do what they say to do by moving all sorts of data back and forth from one place to the next. At this point, we shall begin to look at the "hardware" of a computer to get a feel for how it is constructed.

Computer Hardware

(At this point, there will be a section about computer hardware which has not yet been written.)

* * *

At this point, we have come a long way from the simple on/off bit. We have shown how we can put together a bunch of these bits to represent letters and numbers, and we decided thereafter to organize these bits into groups of 18 called locations. But we needed some way to manipulate the data that we put into these locations; we certainly didn't want it to just sit there. So, we developed the notion of an instruction which would be able to manipulate the data in these locations in various ways. We found that if we put a bunch of these instructions together in the right way, that we could produce a program which was capable of doing something. We found that these instructions not only manipulated the data in the locations, but that they

also made decisions about which instructions to follow. As we said before, the ability to make cecisions is one of the most important features of any computer. And we can always introduce more decision making instructions to expand the range of the types of cecisions that we are able tc make. At this pcint in fact, we have already ceveloped a very limited version of a small scale computer.

A computer comparable to the type that we have developed here would generally be referred to as a mini computer. The number of locations in this computer is fairly small compared to the number of locations in the computers that we shall be dealing with later on, yet a computer of this sort provides the capability to perform most small scale programming tasks. The cost of mini computers has dropped drastically since computers first came into existence, and it is reaching a level at which many small businesses and individual people can purchase mini computers rather easily. It is this this type of computer that may one day be as common an appliance as a refrigerator, a dishwasher or a television. To get a sense for exactly what such a computer is capable of, I provide a few exemples.

Many people have undoubtedly seen the new generation of computer games that has hit the market recently. One of the most predominant of these games is computer ping pong. In the interest of saving the quarters that I might otherwise throw into these machines, I decided to write my own ping pong game on a computer which had a tv type screen similar to the one found on

the commercial version of the ping pong game. Although I changed the game slightly, the program I wrote, if anything, provided more features than the commercial version. The instructions occupied about 450 locations, and the data occupied 250 more. Thus, using only 700 locations in total, I was able to produce a working ping pong game.

Another program which I wrote was a simulated baseball game. (The reader can probably tell by now that I like to write computer games.) By feeding the statistics of players on two different teams into the computer, one could get the machine to play out a full baseball game. It was a simplified version of a real game, but it accomplished all of the essentials. This game occupied a total of about 3000 locations, about half of which were simply used for statistical data. So there were only 1500 instructions.

I could go on all day writing about other programs that I have written or encountered, but that is not the point. The point is that the 4096 location machine that we have developed is in fact capable of doing some reasonably complex things. In fact, about the only people who would ever write a program occupying more than 4096 locations are probably professional programmers. But for the average everyday person, 4096 locations do indeed provide a lot of capability.

We should also say something at this point about the speed of a computer. With our current state of technology, which incidentally is constantly changing, it is not unreasonable to

expect a computer to be able to execute well over a million of these instructions in a single second! New technological advances promise to increase that number even further, possibly to a billion or more. Such capability is absolutely mind boggling.

But mini computers are only a part of the computer scene. The recent years have seen the development of computers of far greater size than the one that we have developed thus far, and we shall frequently find ourselves concerned with these types of computers. Let us see if we can take the mini computer that we have developed thus far and expand it further to produce one of these super large computers of which I speak.

Earlier, when we put 4096 locations together into one computer, we decided to refer to them collectively as the computer "memory". From here on in, we shall start thinking of the computer memory as a single unit. This means that we can simply take a bunch of these computer memories and put them together in some way to get a bigger computer. Let us say that we take 4096 of these computer memories and put them into a single computer. To distinguish one memory from the next we could give each of these memories a number from 0 to 4095. Now, our computer looks something like this:

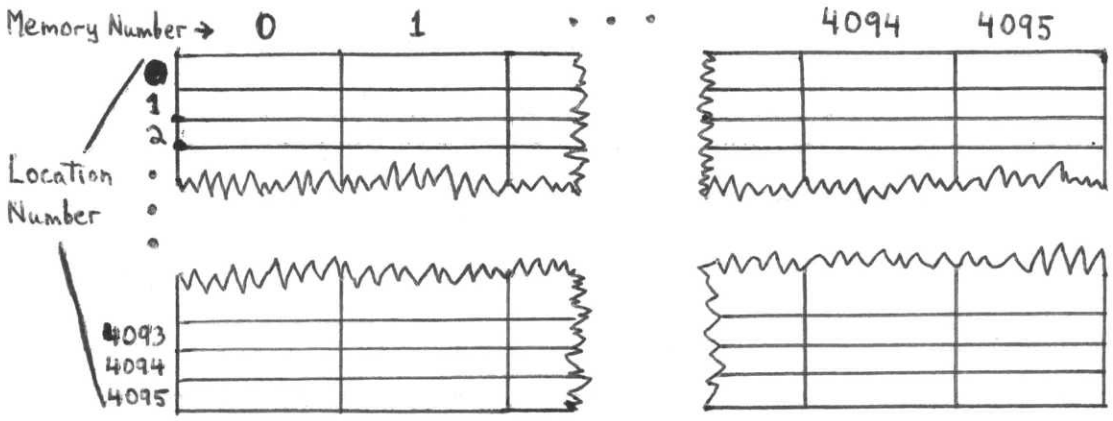

Figure 9

Since we now have 4096 memories of 4096 locations each, we have a total of 4096*4096 or 16,777,216 locations. Now, in order to refer to any of the 16,777,216 locations in our computer all we have to do is specify what [redacted] memory it is in, and what its location number is. Thus, by saying "memory 258, location 1096" we are able to specify exactly which of the 16,777,216 locations we are after. Let's also see what other changes we have to make in our computer to accomodate this increase in size. Let's say that during its operation our computer comes across say, the instruction "lac 3981". What this means is "load the accumulator with whatever is currently in location 3981". But which location 3981 are we talking about? With 4096 memories, we might be talking about any one of 4096 locations, all of which have the number "3981". So it looks like we shall have to find a way to let the computer know which "location 3981" we are talking about.

Let us say that the location 3981 being referred to

happens to be the one in memory 301. We What we can do is add a new instruction. For instance, our new instruction could allow the programmer the ability to say "all instructions refer to locations in memory 301 until further notice".

We shall call this instruction the "memory" instruction, and it will be abbreviated "mem". And when the computer sees the instruction "mem 301", it knows that all instructions until further notice refer to locations in memory 301. If, later on in the program the programmer wanted to have all his instructions refer to, say, memory 1093, he simply says "mem 1093" and the computer will then use memory 1093 until further notice.

Suddenly, with very little effort, we have gone from a fairly small computer to a very large one.

At this point, we shall begin to divert our attention from the technical question of how a computer works to the more social questions of how it is used. It is in this section that many of the social issues of computer use [redacted] will begin to make their appearance. Before going on however, it should be mentioned that the sample machine that we have used for our discussion was derived from a Digital Equipment Corporation PDP-1 computer at M.I.T. This computer happens to be one of the earlier computers that was ever built. Although the computer industry has come a long way since the days when this computer was designed, the basic principles of bits, locations and programs have remained the same. This machine was used here for a teaching model because its design is fairly

simple, and thus easy to understand. In fact, this PDP 1 of which I speak was used some years back to teach an introductory computer course at M.I.T. I had the privilledge being a teaching assistant in this course under the fine direction of the late Samuel J. Mason. Unfortunately, after his death, the course was abolished and this computer is no longer used by anyone.

How a Computer is Used

Timesharing and Multiprocessing

Let us say that a city government somewhere decided to computerize the operations of 100 of its city agencies. There are a number of ways that we might approach such a situation. One extreme would be to buy 100 fairly small computers and give one to each agency. The other extreme would be to buy a single one of these large computers and find some way to split it up among the various agencies. Since one big computer is much less expensive than 100 small computers, the idea of getting the big computer is the one that we shall pursue. Let's say that four of the 100 agencies for which we are buying this computer are the Police Department, the Fire Department, the Motor Vehicle Department and the City Payroll Department. We want the Police Department to have a program which keeps track of all arrests made by its members, along with some pertinent information about those arrests. The Fire Department's program will keep track of all alarms that are sent in, and information regarding them. The Motor Vehicle Department will have a program to keep track of all licenced drivers and registred vehicles. Finally, the Payroll Department program will keep track of the salaries of all city workers, and every Friday at 3:00 it will print out paychecks for all the city employees.

Let us say that the Police Department's program

requires 25 memories. Let us also say that the Fire Department's program requires 15 memories. The Moter Vehicle Department's program happens to require 40 memories, and the Payroll Department's program requires 50 memories. Using our big computer, we could take the Police Department's program and put it in memories 0-24. We can put the Fire Department's program in memories 25-39. We can use memories 40-79 for the Motor Vehicle Department's program and memories 80-130 for the Payroll Department's program. And we still have memories 131 through 4095 left over for use by the other 96 agencies.

If our computer is capable of executing 1 million instructions per second, which is not unreasonable, what would happen if our 100 departmental programs simply take turns being processed? For instance, we could run the Police Department's program for 1/100 second and then kick it off the computer and run the Fire Department's program for 1/100 second. Then, we could go to the Motor Vehicle Department's program and do the same thing. We can simply continue in this fashion until all 100 programs have been given 1/100 second each on the computer, and then go around again. Since each of 100 programs are given 1/100 second on the computer, it takes 1 second to complete a full cycle. And since the computer can execute a million instructions per second, each of the 1/100 second slices of time will still be enough to execute 10,000 instructions. So at worst, each of the 100 programs will get enough time to execute 10,000 instructions per second. This scheme of having programs taking turns on a

computer is generally referred to as time-sharing. Often, a company with one of these large computers will simply sell some of these time slices to various organizations or people who are interested in using a computer, but who don't want to buy a whole one.

The first reaction that one might have to such a timesharing scheme is that it will drastically slow down the speed of the computer for each agency. Let us examine how these agencies might use their programs and see how much of a problem, if any, this slowdown is.

If we were to casually walk up to the person using the Police Department's program and look over his shoulder, we might see him type in a request to the computer to print out the names of all people who were arrested for a felony during February. The computer then looks through its arrest data for a short period of time and finally prints out the desired names. Then, the operator rips off the sheet of paper with the names on it and gives it to his sergeant. Afterwards, he goes back and asks the computer to print out the names of all people arrested by officer Smith the week before. The computer does a quick search through its information and once again prints out the answer. Then, for the next few seconds, the person simply looks at the names that the computer has just given him before he makes another request for information from the computer. Of course, each of these requests can only be made if it is written into the program to allow such requests. Finally, he makes another

request of the computer after which he stops again, this time to go to the bathroom. The point is this: the computer is not being asked to work continuously. Instead it is being asked to do a quick bit of work that takes only a small fraction of a second. Then, there is an idle period of a few seconds where no requests are made. Then, a new request causes the computer to work for another small fraction of a second, and then there is another break. As we can see, it really requires only a small fraction of the computer's time to satisfy the requests of the person using the program. The rest of the time, the computer is idle. Certainly, this seems to be an innefficient use of the computer, especially when one considers how expensive a computer is. If the pattern of use is pretty much the same for the other 99 agencies, then we can probably timeshare among them.

If we look at it from the computer's point of view, we find that when the computer sweeps through the 100 programs, there might only be 3 or 4 of them that want to use the computer immediately. The people using the other 96 or 97 programs are currently engaged in that interval of time where they do not want to use the computer immediately because they are taking a sip of their coffee, reading what the computer has just given them, or whatever. The basic reason for all of this is that the computer is often able able to process requests from a person much faster than that person is able to make them. To show just how much this is so, I cite the fact that when I first typed this thesis on the Multics computer system at M.I.T., a full 8 hour day of typing

often consumed less than 20 seconds of the computer's processing capability. Although others may use more computer time during an 8 hour day, I think that this example makes the point of just how [redacted] faster a computer can work [redacted].

Of course, computers may not always follow this pattern of use. For instance, when we send a manned rocket to the moon, the computer is doing fairly continuous calculations. It is doing everything from calculating which rockets to fire to monitering the astronauts' vital signs and food supply. Because we are constantly using the computer in this situation, timesharing would create a slowdown that might not be desirable.

It seems then, that the number of people that can use the computer at one time depends upon many things including the speed of the computer and the types of programs that we wish to run on this computer. If a particaular computer, without timesharing, can answer a person's request in 1/50 second and the some computer with timesharing causes him to wait 1/10 second (five times as long), that person probably will not care about the difference, because 1/10 second is still very small. But if the timesharing had caused him to wait 5 seconds for his answer, then there is a noticeable difference. In figuring out how to timeshare a computer in the best way possible, we must find some happy medium which uses the computer wisely, yet doesn't cause any really noticeable decline in the quality or speed of service. The decision as to what is "noticeable" is of course, a subjective decision that would most likely be made by the person

designing
buying or using the computer.

The question now arises: how can we get this computer to timeshare among all these programs? The answer: write a special program to do it! All this program would have to do is run around in a circle checking each of the 100 agency programs to see which ones want to use the computer. When we come to a particular program, if it doesn't want to use the computer at the moment, then we simply go on to the next program. If it does want to use the computer, then we give it its 1/100 second time slice before going on to the next program. If 1/100 second is not enough time to finish doing what the program wants, then we simply make a note of where we left off, (which instruction we were about to execute when we were cut off, what the accumulator had in it at the time etc.) and we come back to finish up after we have finished checking the other 99 programs. Of course, a program may not require the full 1/100 second, and in this case we just give it however long it does need, and ther go on to the next program. What we have just laid out, more or less, is a description of how we want our timesharing program to work. Once we actually write up this scheme in the form of a program, we can simply put this program into some of the memories in our computer that are as yet unused. Then by running this program, we'll be able to timeshare our computer. In a sense, this program is in charge of the computer since it rations out the computer's time to each of the other programs.

This timesharing scheme happens to be a very simple

one. There are many much more complex schemes that have been developed. Often, they will reflect some set of relative priorities among programs, with some programs getting larger time slices than others, or with certain programs having their requests responded to faster than others.

As we can see, timesharing doesn't really allow two programs to run at the same time. But since a computer can often process requests so much faster than a person can make them, it appears as if more than one program is running at a time.

So far, we have assumed that our computer has only one processor. In other words, a single one of the hardware units that is used to execute instructions and manipulate data is having its time divided among all 100 programs. If we thought that this was too slow, we could always add another one of these processors. This would give us the ability to literally execute two programs at once, rather than to simply make it look like we are. This is, in a sense, like adding another cashier at the bank when the lines get too long. By putting a number of these hardware processing units into one computer, we can take care of a number of programs all at the same time. This is called multiprocessing.

However, multiprocessing has a problem that the bank doesn't have, which is illustrated in the following situation. Let us say that we are using a computer that has an extra one of these processors installed in it, so that two people's programs can run at the same time. The two people who are running their programs at the same time are named "John" and "Bill". The

overall purpose of each program is unimportant, but let's say that at some point in their programs John and Bill have the following instructions. Let us also say that if we could suddenly stop both programs dead in their tracks and see which instructions were about to be executed in each program, we would find each program running in the place shown on the diagram. Finally, let us say that John's accumulator has a "5" in it, Bill's accumulator has a "7" in it and location 10 of memory 10 has a "3" in it. The reader will also note that each program has recently passed a "mem 10" instruction, which means that all instructions until further notice refer to locations in memory 10.

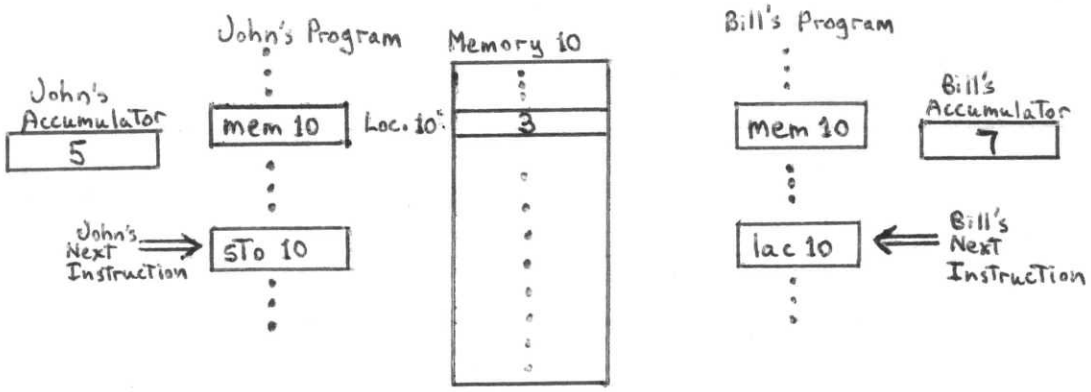

Figure 10

Now, let us look at two different things that might happen from here. We could find that the next instruction that is

executed is either John's "sto 10" or Bill's "lac 10". Since the two processors are running independently of each other, we have no way of knowing for sure which instruction will be followed first. What happens if John's "sto 10" is executed first? This instruction will cause the "5" which is in John's accumulator to be stored into location 10 of memory 10, replacing the "3" that was there to begin with. Then, at some later point in time, Bill's "lac 10" will be executed. This will cause the computer to load whatever is in location 10 of memory 10 into Bill's accumulator. And since location 10 of memory 10 has a "5" in it, Bill will end up with a "5" in his accumulator. What would have happened if Bill's "lac 10" had been executed before John's "sto 10"? The execution of Bill's "lac 10" would have caused the "3" that was originally in location 10 of memory 10 would have been loaded into Bill's accumulator. As we can see, in one case, Bill ends up with a "5" in his accumulator and in the other case he ends up with a "3" in his accumulator. And what he ends up with is not determined by the correctness or incorrectness of his program. It is determined simply by which processor happens to execute a particular instruction first. Certainly, Bill doesn't want to leave what he finds in location 10 of memory 10 up to chance. Yet this type of problem is very prominent in multiprocessing. There is a very big need in multiprocessing to make sure that all processors are coordinated properly. If the programs being run by different processors try to acess the same piece of data, we run into problems. Methods have been devised to

actually coordinate processors properly to get around this type of problem, but it is really very difficult to apply these methods properly, and no sound, systematic way of multiprocessing has been developed. Recent attempts to include multiprocessing on computers, as a result, have run into problems. Note that this type of problem will allow a program that is perfectly correct to make errors by virtue of something totally beyond its control. We shall leave the topic of multiprocessing by asking the reader to remember that it simply adds to the complexity involved in turning out a truly operative program.

The timesharing program discussed earlier is our first example of a more general class of programs known as "systems programs". A systems program is a program that is generally not used directly by any one person. Rather, it is used to control the way in which the computer is used by other programs. There are many types of systems programs that are used to do many different things, and we shall begin to examine some of the more common ones. This will consequently give us some insight into the ways that people are using computers today and some of the ways that they might use them tomorrow.

Program Libraries

Let us say that we want to write a program to play five card poker. At some point in the program, we would have to write a routine that simulates the dealing of five cards from a card

deck. Because there are 52 cards in a deck, we might take each card and give it a number from 1 to 52. Thus, the number 1 might represent the two of clubs, the number 2 might represent the two of diamonds, the number 52 might represent the ace of spades and so on. If we assume that there is some way to get the computer to come up with random numbers from 1 to 52, then the process of dealing out five cards would be fairly simple. All we have to do is pick five different random numbers from 1 to 52. Then, we can simply figure out which cards are represented by the numbers that were picked. But there is a problem here: it is very difficult to write a good program to produce random numbers. Because of this, the person trying to get the poker program to deal out 5 cards would end up spending virtually all of his time writing the program to pick random numbers rather than writing the program to deal cards. And before he even begins to write his random number program, he would probably have to spend his time reading through a few books or papers to learn all about generating random numbers. But the person writing the poker program doesn't want to spend most of his time researching and writing random number programs; he just wants to write his poker program. He wants to use the random numbers once they are generated, but he could care less how they are generated. Certainly, this is a reasonable desire. An auto repairman doesn't want to worry about building a muffler before he puts it into a car; he simply wants to have a complete muffler on hand for use when it is needed. He lets someone else, namely the people at the muffler factory, worry

about actually building the muffler. Of course, there is always the chance that the mufflers sent from the factory are faulty. In this case, even though the auto repairman might install the muffler properly, the car will not perform the way it should. If the random number program that we use for our poker program is faulty, (for instance, it may pick the number 52 more than the other numbers, causing the ace of spades to be dealt out more than the other cards) then the whole poker program will be faulty, even if we use the random number routine properly. This type of situation occurs frequently in computing, and we shall go into some of its implications later on.

Mufflers, engines, batteries and many other parts to a car come pre-made simply because there are lots of auto repairmen who want to use them and not build them. And so it is in programming. There are many programmers who may find the need for random numbers at one time or another. Certainly, they will be needed by anyone writing a game of chance, although they have many other uses too. If each person who needed random numbers wrote his own number program, a lot of people would spend a lot of time writing random number programs, and very little time doing any substantive programming. This duplication of effort can be completely eliminated however if we simply write one random number program that can be used by anybody who wants to use it. And to do this, we can employ a person who really wants to write a random number program, rather than someone who is forced to. This program, when written, could simply allow a person to say

something like "ran 52" to get a number from 1 to 52 for a poker game. A person writing a dice game can say "ran 6" to get a number from 1 to 6 everytime he wants to roll a die. Certainly, making a simple statement like that is easier than spending days researching and writing a random number program. There are in fact many situations where a person has to write a subsection of his program that he does not want to or has no idea how to write. If we simply take a bunch of subroutines that many people are likely to want to use at one time or another and put them into a "library" of sorts, we can make programming much easier for those who take advantage of this library.

Almost all large scale computers and many smaller ones have some sort of program library. Multics, which is the computer system used by a large part of the M.I.T. community has an enormous program library. Some of the programs in the library are designed by the same people who designed the Multics system, and others are written by everyday people who simply felt that a program of theirs might be of interest or use to other people on the Multics system and thus decided to put it in the library. All somebody has to do to use the programs in the library is tell the computer the names of the library programs that he wishes to use. The systems program which is in charge of the libraries will then look for the requested programs and hook them up to the person's program in the appropriate way.

Libraries however, can do more than simply providing

pre made pieces to use in a program. Consider for instance the program we wrote earlier to perform the calculation (3*4)+7 and its bit representation in the computer:

Location Number	"Human" Representation	"Computer" Representation
0	lac 5	000000 000000000101
1	mul 6	000010 000000000110
2	add 7	000001 000000000111
3	sto 8	000011 000000001000
4	hlt	000100 000000000000
5	3	000000000000000011
6	4	000000000000000100
7	7	000000000000000111
8	0	000000000000000000

Figure 11

Here we see two different ways of representing the (3*4)+7 program. The column labelled "computer" shows the actual pattern of bits that the computer would have to contain in order to perform this calculation. The column labelled "Human" shows the shorthand that we developed to make it a little easier for us to understand the program when we look at it. Certainly, it is easier to think of the instruction in location 2 as "add 7" rather than as "000001000000000111", yet it is the "000001000000000111" that must actually be put into our computer if we are to accomplish anything. If we consider the case of a person who is writing a program of, say, 1000 instructions, we can see how it would be much easier for him to write his program using the format shown in the "human" column rather than the one shown in the "computer" column. What we might want to do then is write a program that will enable him to do such a thing. What

this program would have to do is take instructions written in the "human" form and translate them into the "computer" form. If we were to put this translator program into the computer library, then everybody would be able to write their programs in our "human" format, leaving it up to the translator program to translate into "computer" form. Such a translator program is usually called an "assembler". The "human" format for writing a program is generally referred to as "assembly language". This assembler program itself would have to be written initially in "computer" format. But once this assembler has been written and put into the computer, there will never again be a need to write a single program in "computer" format because the assembler will be there to do the necessary translation As a result, we have made it easier to write programs by allowing them to be written in a format that is more understandable to humans than a bunch of 1's and 0's. But certainly it seems, there must be an even easier or "more human" way to write our programs. Programming in everyday English would probably be the easiest thing, and there is in fact a great deal of research going on which is directed toward precisely that goal. If we wish, we can design a scale that measures "ease of programming" of various techniques, and it might look something like this:

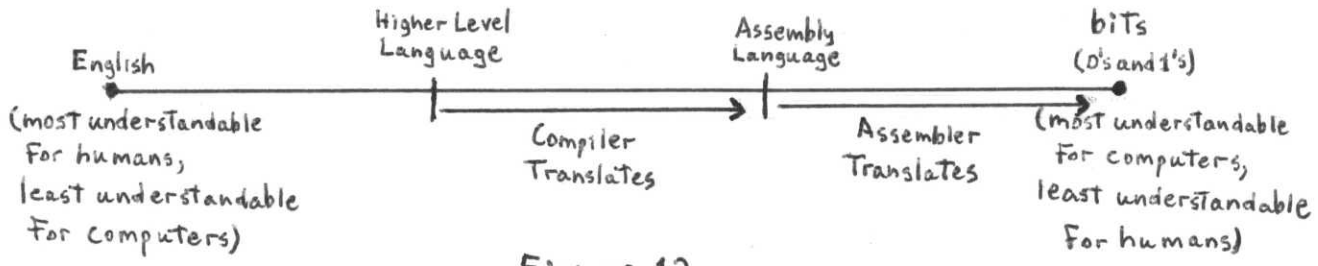

Figure 12

What we are saying here is simply that English is easiest for humans to understand and bit language is easiest for a computer to understand. And there are many ways of programming that are in between. Assembly language is a little easier for people to understand than bit language, and if we want to use assembly language, all we have to do, as we saw before, is write an assembler program to translate from assembly language to bit language. What might be even nicer than using assembly language would be to simply say to the computer something of the sort "calculate (3*4)+7" and have the computer translate that statement an assembly language program capable of calculating (3*4)+7. We can call the computer language that permits us to say such things as "calculate (3*4)+7" a "higher level language" since it permits us to say things that are even closer to English than does assembly language. There are lots of features that we can put in a higher level language to make it useful. Often in programming, we have a section of a program that we want the computer to execute more than once. Thus, a statement such as "do 5 times" might be used in our higher level language to get the computer to execute a group of instructions 5 times. High level

languages also allow us to use symbolic names. For instance, if some location is being used to store a number representing the number of children that somebody has, we might wish to refer to that location as the "children" location rather than as, say, "location 3096, memory 981". A high level language often allows us to assign names to a location that have meaning to us. Thus, whenever we refer to the "children" location, the computer can figure out that we are really talking about location 3096 of memory 981. This is all much easier than using assembly language instructions to get something done five times. All we have to do to have such a high level language is write a new translator program which can translate from our higher level language into assembly language. Then, we can use our assembler to translate the rest of the way into bit language. We can repeat this process again to develop an even higher level language by simply writing a program that translates from our new higher level language to our old higher level language. These translator programs that are used for high level languages are generally called "compilers".

Much of the early research in the computer field centered around the design of computer languages because few people had the desire to program using 1's and 0's all the time. Today, a computer library will typically have a number of compilers and assemblers which people can use to facilitate programming.

Before we leave the subject of higher level languages,

there is something that should be pointed out. When we developed our higher level language, we saw how it allowed us. make statements such as "calculate (3*4)+7". The compiler then translated that statement into assembly language in some fashion. But we do not know precisely how the computer translated the "calculate (3*4)+7" into assembly language. When we wrote this program in assembly language, we first loaded a "3" into the accumulator, then we multiplied the "3" in the accumulator by "4", and finally we added "7". If we simply say "calculate (3*4)+7" to the computer and allow the compiler to translate into assembly language for us, the program produced may put the "4" into the accumulator first and then multiply it by "3". This may seem like nitpicking, because in this case it doesn't really matter whether the "3" or the "4" goes into the accumulator first. But the point is that the programmer who uses this higher level language loses his control over whether the "3" or the "4" goes into the accumulator first, and it is this loss of control that is our concern. If a programmer has no conrol over how the compiler translates his high level language statements into assembly language, but he at least knows how the translation is done and finds the method acceptable, then there isn't much need for concern. But high level languages often have so many different features that it is impossible for a person to always know exactly how his instructions will be translated. He must often trust the compiler to translate his statements into an assembly language program that will do what he really wants it to

do. Of course, one of the reasons we designed a high level language in the first place was to allow us to make general statements about what we want to do while leaving the details of how to do it up to the computer. But if we are writing for instance, a program that will be used to make important decisions about people's lives, it may not always be wise to leave so many details up to the computer. This is similar to the problem mentioned earlier where a person might use a defective random number generator without knowing it, and have his program get screwed up as a result. In a more general sense, we are dealing with the problem that people often allow computers to do things for them without really knowing how it is doing them or if it is doing them correctly.

Sharing of Information

Another feature found on many of the newer computer systems is the ability to share programs and data among a number of users. The library that we discussed before is generally a group of programs to which all users are granted acess. But often someone will have something inside the computer that he wants to share with only a limited number of people. For instance, a group of people working on a program might want to be able to share copies of that program among themselves so that they can all work on it. But they wouldn't want other people on the computer system to have acess to their program until it is

finished and working. And even then, they may not want others to see it. There is the possibility that someone has sensitive information on the computer, and thus only wants a few people to have acess to that information. So our computer must have not only the ability to share information, but also the ability to restrict it. For instance, with the program that we discussed earlier, we want the person using the City Payroll Department's program to have acess to the salaries of all city employees. Yet, we certainly don't want the person running the Motor Vehicle program to have acess to the salaries. If he did, he might try to change the location with his salary in it from reading "$250 per week" to reading "$10,000 per week" every Friday at 2:55. Then at 3:00, the Payroll Department program prints out paychecks for all city workers, including a $10,000 check for the guy who changed his salary. Then, at 3:05, this person can change his salary back to $250, so that nobody will ever know the difference. Although such a ripoff technique seems too simple to be true, many people have in fact committed computer aided crimes using very comparable tecniques. As a result, many computer systems are being designed today which allow a person or organization to specify exactly who has acess to a file, and how much access he has. (Can he only read the contents of a file or can he also change them. Can he make his own copy etc.) But very few computers on the market today are really secure. There are in fact many organizations, who, for precisely this type of concern, have decided to buy a computer which only they can use so nobody

else can get at restricted information and cause problems.

As we can see, the sharing of information in a computer brings out a lot of social problems that must be dealt with. What we must remember is that, in a way, the computer has taken the place of a file cabinet. Information that would have been put in a locked file cabinet or desk 10 years ago is today put on a computer. Certainly, no employer would have let his employees into his file cabinet ten years ago to change their salaries. A person writing a book would not want someone to break into his desk and steal copies of the book. A doctor wouldn't want (hopefully) an outsider to get at sensitive medical information in his files. But nowadays, businesses keep salaries in computers, writers write books using computers, and doctors keep records on computers. It seems reasonable for someone to expect at least the same amount of security for his property when it is stored on a computer as he would have expected if he still kept that property in a file cabinet. The problem is this: In the initial rush for companies to produce working computers and get them on the market, the technological push was directed almost entirely toward simply getting the computers to do useful things. The boom in the popularity of computers saw more and more people and organizations computerizing their operations, thus putting more and more information of various sorts on computers. Gradually, these people began to realize something. Although many technological advances were made in the area of "getting the computer to do something", very few were made to adequately

protect the information which people began to put on computers. This lack of protection encouraged many people to begin stealing or illegally changing information stored by others on the same computer. Some of the computerized ripoffs that have occurred in recent years boggle the imagination in their scope and simplicity. Computer technologists have begun to respond to the problem, and there is a growing field in information protection. But the fact is, that at the present time, very few of the computers on the market adequately protect the information they contain. And crime is not the only concern here. Because computers often contain sensitive material on individuals, this lack of security creates a number of problems in the area of personal privacy.

Throughout this chapter, we have spent a lot of time talking about putting things into and getting things out of computers. But we have not really adressed the question of precisely how this is done. Certainly, it does us little good if the computer performs some task for us but doesn't give us a way of getting at the results of that task. So we come to a discussion of the "extras" that come with a computer, the devices that allow the information inside a computer to somehow be transmitted to the outside world. These devices are most commonly referred to as peripheral devices.

Peripheral Devices

The peripheral device with which most people are most familiar is probably the computer teletype. A teletype is simply a typewriter, except it is connected to a computer. The way in which a teletype usually communicates with a computer is relatively straigtforward. When a person strikes one of the typewriter keys on the teletype, the electronics of the teletype will figure out what the bit representation is for the character on that key, and this bit representation will be left somewhere inside the computer where something can be done with it. For instance, we can have an instruction which simply causes the computer hardware to place the bit string representation for some typewriter key into the accumulator as soon as that key is struck. Once a a key has been strick en and its bit string representation read into the accumulator, the next few instructions of the program will probably be devoted to processing that bit string in an approoriate fashion. For instance, the program might start putting the new bit string together with other bit strings which have arrived recently to form larger words. Then, it will start trying to figure out what those words mean. Once our program has figured out the meaning of the words, it will do whatever it is supposed to do when it gets those words. Sending data out from the computer onto the

teletype will follow the same procedure, but in reverse. We will put a bit string in the accumulator which appropriately represents the character that we wish to type out, and we then execute an instruction to type out that character on the teletype. Since the people timesharing a single computer are often distributed throughout a certain geographical area, their teletypes will be connected to the computer by everyday telephone lines. In fact, the process of using a teletype often entails making a telephone call to the computer. The sudden upsurge in the use of telephone wires to transmit computerized data has certainly proven to be a source of joy to the telephone company in the form of added income.

Although the teletype is the best known peripheral device, it is far from the only one. In fact, just about anything that is attached to a computer is in some sense a peripheral device. For instance, a missle that is being guided by a computer is a peripheral device. The computer might do a bunch of calculations as to whether or not the missle is on course and discover that to get the missle back on course a particular jet on the missle should be fired for 4 seconds. The program might then put the number "4" into the accumulator and transmit it to the missle. Once the missle recieves the "4", it will fire that jet for 4 seconds.

Peripheral devices in fact, go even beyond this. Any medium of cheap information storage is a peripheral device. Just what do we mean by "cheap information storage?" Well, it seems

reasonable that a person who writes a program will want to save that program for future use once he goes home at night. Certainly, he doesn't want it to be erased from the computer between one day and the next. But it happens to be very expensive to keep a program stored in the type of computer memory that we have been talking about. If the person isn't going to use his program for the next six monthes, it would be best to find a cheaper way of storing that program. There are many methods of cheap storage on the market today. Magnetic tape, much like that found in a tape recorder, is probably the cheapest method of storage. Devices called "disks" and "drums" are also used for cheap storage, although it is not really important for the reader to understand how they work. The point is that these methods of storage all allow information to be stored for a cheaper price than the computer memory. One might ask why we don't then use these storage mediums to replace the computer memory. The reason that we don't use them for such a purpose is that they are much slower than the average computer memory. That is, where it might take one-millionth of a second to acess a piece of data in a computer memory, it will take at least a few seconds to find that same piece of data on a magnetic tape, because the tape has to first wind to the proper position. Anyone who has ever operated a tape recorder should understand this fact. Disks and drums are faster than tapes, and are consequently more expensive. However, they too do not approach the speed of a computer memory. So what usually happens is that these devices are used to store data

which is not currently being used by anybody. That is, when a person is in the process of using a computer, the information with which he is working will be stored in the fast, but expensive computer memory, where the computer can get at it almost immediately. Once the person is done with whatever he is doing and is ready to go home, the information he was working with will be copied onto one of the cheaper storage mediums (lets say magnetic tape) and then erased from the expensive memory. When the person next returns to the computer, he simply asks the computer to retrieve what he wants from the tape, and after the tape takes a few seconds to wind to the proper position, the systems program in charge of the tape storage will copy what the person wants from the tape back into the computer memory. And at this point, the person can resume working wherever he left off last time. So as we can see, the computer memory will at any one time contain only the the programs and data that people are currently working with. Everything else will be stored on the cheaper storage mediums.

It is the presence of these very cheap storage devices that has enabled people to compile literally billions of pieces of information for use by a single computer. If the computer memory were used to store so much information, the costs would be astronomical, and nobody would do it. But with cheaper storage, that is not nearly as much of a problem.

Earlier, we showed how the use of pre-packaged subroutines can help to make programming easier for those who use

them correctly. But the notion of a subroutine has much impact beyond this. Consider for instance, the task of writing a baseball game on a computer. For simplicity, all games will last nine innings regardless of score. Before starting however, we can make a few observations about the structure of a baseball game. For instance, if we can write a subroutine which we shall call "playinning", that is capable of playing a single inning of baseball, we can use it nine times in a row to produce a full baseball game. So instead of concentrating on the whole game, let us concentrate on what it takes to write a program to play one inning.

If we can write a subroutine which we shall call "Teambat" which allows a single team to bat until it has three outs, then we simply have to use the "Teambat" program twice, once for the visiting team and once for the home team, in order to play a full inning. So instead of concentrating on a full inning, let's take it a half inning at a time. Well, what is involved in having a single team bat for a half inning? All we really have to do is write a program called "personbat" which allows a single person to bat. Then, by using this routine over and over until a team has 3 outs, we can produce a half inning. Notice what we have done. We have taken the task of writing a full baseball game and reduced it to the task of writing a one person batting routine.

How might we write this program that allows a single person to bat? One approach might be as follows: Let's say a

player has a total of 500 at bats with 40 home runs, 5 triples, 30 doubles and 75 singles. We could use a pre-packaged random number routine to pick a rumber from 1 to 500. Then, if the number picked is from 1 to 40, we call for a subroutine called "Homerun". If the number is from 41 to 45, we call for a subroutine called "Triple". 46 to 75 will cause us to call for a routine called "double", and 76 to 150 will invoke a routine called "single". Any other numbers picked will send us to a routine called "Out". Each of these routines will in turn call on even smaller routines. For instance, when the "double" routine is called, that routine might itself call a routine called "baserunning" which will figure out what happens to all the baserunners, and which might call another routine called "extra base" which allows somebody to try to get an extra base on the play. This technique of breaking a program down into smaller, and thus more manageable parts is known as "structured programming" and is perhaps the most powerful programming tool developed to date. Its obvious advantage is that it limits the scope of what we have to think about at any one time to a very small section of the overall program. Then, once each of these smaller routines is written, all we have to do is piece them together into a full program. In a sense, we are breaking down our program into "ready made parts" which are comparable to the library subroutines discussed earlier. Our program is then a fitting together of these ready made pieces. This approach results in programming that is much faster and much easier than

it would otherwise be if the person writing the program had to worry about every detail of the whole program at once. And if the program isn't doing the right thing when a person gets a triple, the author of the program merely has to look at the triple routine to find his error, rather than search through the whole program. And finally, if more than one person is working on the program, this technique provides a natural way to divide the work. One person can write the single and double routines, one can write the triple and home run routines, and so on. These routines can then be put together to form the whole program, even though each person does not have to have any idea of how the sections written by the other people actually work. It is this particular feature of structured programming that led us to a discussion of the topic in the first place.

The Construction of a Large Computer System

Most of the large programs that we have been and will be talking about cannot possibly be designed by one or even a few people because of the sheer size of these programs. Beacuse we have so many people working on a large program, we are really forced to divide up the work in such a way that each person can work on his own individual subsection without having to ever worry about the rest of the program. As we have seen, structured programming accomplishes these tasks rather well, and is thus an invaluable technique for large scale program design.

This is not to say that structured programming has been used in the past in the design of large programs; in fact, it generally hasn't, since it is a fairly new concept. But then again, it has been very difficult to turn out very large programs that really work. The advent of structured programming however, promises to make large scale programming efforts much easier in the future.

But even so, the design of a large scale program or any program for that matter remains an exacting exercise. One can not make a mistake in writing a program and expect the computer to figure out what he is really trying to do. The computer is a merciless judge of program correctness. Getting a large number of people to design such exacting pieces which in turn must fit together in a very exacting way is far from a simple task, and good management and good communication are as essential as good programming.

One phenomena that is often found in large programming ventures is that the structure of of a programming organization often takes on the structure of the program that the organization is writing. If the program they are writing is initially broken down into five smaller parts, then the organization will itself split into five groups, one to work on each part. Each of these groups will probably have its own boss, as will the entire project. Let us say that one of these five main parts of the program is itself broken down into four smaller parts. Then the group working on this part will itself be broken down into four

smaller groups, each with its own boss and so on. This will keep happening until we reach the level of "one person groups". The rule in almost all programming efforts, be they large or small, is that the design of the program frequently changes as soon as people start working on the nitty gritty and discover that there is an easier approach then the one that was originally taken. If it were suddenly discovered that the above program could be better written with four main parts instead of five, we would have to change the top level of our organization from five groups to four groups. This means that the boss of the fifth major group loses his boss status, and the members of that fifth group must either be relocated or laid off. Beacuse such a situation arises so frequently in programming, such an organization must be able to change its structure radically and frequently. If the boss who was moved in the reorganization doesn't like the fact that he is no longer a boss, we can either tell him "tough luck" or we can decide to stay with the original five part breakdown of the program, even though it might not turn out as well with five parts as it would have with four. What we have assumed here is that the change to four main parts will produce a program to do exactly the same thing that the program with five parts would have done, except more efficiently. It often is the case however that the overall goals of the program being designed are changed to meet programming needs. If it is discovered that a certain feature which was originally included in the programming goals turns out to be difficult to program, that feature may simply be

dropped from the overall goals. Thus, there is often a great difference between what a program is originally intended to do, and what it actually does once it is finally produced. This doesn't mean that the final program doesn't work. It simply means that it doesn't do what people had originally hoped it would do, in the way that they hoped it would do it. This in fact, leads into another major problem. When a large program is designed for an organization, the program simply does not do things in the same way that the organization did them before. As a result, an organization is often redesigned to fit a computer, rather than the other way around. If a program is being designed with some social purpose in mind, social goals in the area with which the program deals may be changed not because they are invalid social goals, but because they are simply not compatable with the program. This phenomena of computer use can cause very severe problems, and will be discussed at great length when we get into artificial intelligence. There is a strong desire among many artificial intelligence researchers to eventually use computers for making important decisions. Certainly, we don't want the process of making complex social decisions to be determined by how easily these decision making processes can be programmed.

Getting back to the management of a large scale programming project, it is apparant that an amazing degree of flexibility is required in such an organization. There are few business organizations today that are flexible enough to accomodate the frequency and depth of reorganizatior required for

a programming project.

Another problem that we might notice here is that there is no one person who really knows what these large programs are doing. There are people who know what certain sections of the program are doing, but it is simply impossible for a single person to be aware of the details of the whole program. After all, the whole reason that we split up the program among so many people was that it was simply tco complex for one person to comprehend. Also, the computer industry has few, if any standards for program documentation. Program documentation is simply the proces off writing, in addition to a program, a description of how that program works. The lack of a good description of how a program works makes it difficult if not impossible for anyone to ever figure out how the program works. Add this to the fact that shortly after the completion of a project, the people who worked on it generally are not available or don't remember how their own sections worked, and we see that our large scale program is nothing more than a mysterious black box. People put information in and get answers out, but they have no idea how the information they put in was used to arrive at the answers they got out. Thus, a person using a computer must rely on a blind faith that the program is really doing what he wants it to do. If he wants to find out how it is arriving at its answers, he can't, since nobody really knows. And if he did know how the answers were being produced, he might decide to never use the computer again!

Since it is difficult to examine someone else's program

in detail, it is also fairly easy for someone writing a section of a program to change around a few instructions to purposely subvert the entire program. As we have said before and will say again, computers provide a means for crime that is unparalled anywhere else.

What all of this results in is a reliance upon a program that nobody fully understands, which may or may not solve the problems it is desigened to solve in the best way possible, and which may have accidental or malicious flaws in it. And even if the computer does solve the problem in the best way possible, (which we have no way of knowing for sure) that method of solving the problem may become obsolete within a short time when some better method comes along. And of course, we have no way of detecting this obsolescence, because we don't know for sure what the program is doing.

Hopefully, the reader has by this time developed a sense for what a computer is, how it works, and some of the problems that may arise if it is not used carefully. More than anything else, the purpose of this chapter has been to give the reader a realistic view of the computer can and can't do. The reader should begin to see by now that computers are only human creations, and as such are subject to the flaws and limitations, and even the corruption of the people who create them.

Footnotes

1. Wall Street Journal, March 22, 1971, "Sabotage, Accidents and Fraud Cause Woes for Computer Centers."

2. ibid

3. ibid

4. ibid

5. ibid

6. Aryeh Neir, "Dossier: The Secret Files They Keep On You," pp. 17-18.

7. ibid pg. 48

8. ibid pp. 133-134

9. ibid pg. 134

10. ibid pp. 78-79

11. ibid pg. 79

12. ibid

13. ibid

14. ibid pg. 80

15. ibid pg. 162

16. Arthur Miller, "The Assault On Privacy," pg. 115. See also "Protecting Privacy and the Rights of Federal Employees, Senate Report No. 534," 90th Cong., 1st Sess. 19 (1967); Ridgeway, "The Snoops: Private Lives and Public Service," New Republic, Dec. 19, 1964, at 13; Zola, "Adventures of a Test Taker," National Review, Jan. 12, 1965, at 21.

17. ibid pg 111.

18. Ibid pg. 116

19. ibid pg. 86. See also Sesser, "Big Brother Keeps Tabs on Insurance Buyers," The New Republic, April 27, 1968, at 11; Morris, "What the Credit Bureaus Know About You," Readers Digest, Nov. 1967, at 85; Wall Street Journal, Feb. 1, 1968, at 1.

20. ibid pg. 86

21. Alan F. Westin and Michael A. Baker, "Databanks in a Free Society," pp. 320-321.

22. Ibid pg. 249

23. ibid pg. 291

24. ibid

25. Ibid

26. Miller Op. Cit. pp. 96-97

27. ibid pg. 97

28. ibid pg. 142

29. ibid pg. 143. See also Steinberg, "Government Records: The Census Bureau and the Social Security Administration," in On Record: Files and Dossiers in American Life, 225-242 (Wheeler ed., 1969); Miller, "Considerations in Determining the Content of the 1970 Census," 4 Demography 744 (1967); Eckler, "Profit from the 1970 Census Data," Harvard Business Review, July-Aug. 1970 at 4.

30. Ibid pg. 55

31. ibid pg. 143. See also H. Taylor Buckner, "Computer Privacy," Hearings before the Committee on Administrative

Practice and Procedure of the Committee on the Judiciary, United States Senate (March, 1967), 264.

32. Computerworld, March 29, 1976, pg. 2.

33. Neir Op. Cit. pg. 30

34. Joseph Weizenbaum, "Computer Power and Human Reason," pg. 123 (From an early draft of the book.)

35. Neil Postman and Charles Weingartner, "Teaching as a Subversive Activity," pg. 20.

36. Weizenbaum Op. Cit. pg. 138. See also H.A. Simon and A. Newell, "Heuristic Problem Solving: The Next Advance in Operations Research," Operations Research, Vol. 6, Jan-Feb 1958.

37. Ibid pg. 205. See also Greenberger and Martin (ed.), "Management and the Computer of the Future," pg. 118.

38. Ibid pg. 207

39. The Boston Globe, Feb. 23, 1976, pg.1, "And One Day They May Bug Your Brain."

40. Weizenbaum Op. Cit. pg. 239. See also New York Times, Aug. 10, 1973

41. Ibid pg. 241

42. Joseph Weizenbaum, "On the Impact of the Computer on Society."

Bibliography

ACM Committee on Computers and Public Policy; "A Problem List of Issues Concerning Computers and Public Policy" (1974)- This paper contains short discussions about sixteen different areas in which computers may have important social implications. Some of the topics covered include "Information Services for Home Use," "Computers anc Education," "Computers and Privacy" and "Computers and Employment."

Ad Hoc Committee on the Privacy of Information at M.I.T.; "Final Report" (1971)- A discussion of existing guidlines for the privacy of information at M.I.T. and an examination of proposals for increasing the level of privacy.

Fredrick F. Brooks; "The Mythical Man Month" (1975)- This book contains a collection of essays that deal with the problems that are encountered when one sets out to actually build a large scale computer program. The book makes numerous suggestions about building large computer systems more effectively, and in particular it discusses some new management techniques for employment in such efforts. A substantial part of the discussion is also devoted to ways to minimize the communications problems that frequently appear during the construction of a large computer system.

Department of Health, Education and Welfare; "Records, Computers and the Rights of Citizens" (1973)- This report deals with some of the harmful effects of the use of computerized data systems and it discusses safeguards to protect the individual against such negative effects. It also contains an in depth discussion of the use of Social Security numbers as universal identifiers.

Robert M. Fano; "Conceptual Approach to Privacy Legislation" (1974)- This paper maintains that the primary focus of privacy legislation should to prevent the gathering of information before that information ever has the chance to become harmful, rather than on records that have already been established.

Robert M. Fano; "The Effects of Time Sharing on the Jet Vac Corporation" (1970)- This paper is a case study of some of the problems that confronted a particular organization when it set about to convert many of its manuel operations into computerized ones.

Robert M. Fano; "On the Social Role of Computer Communications" (1972)- A paper on some of the ways in which computer communications systems can "influence social trends and, in particular, individual freedom." It discusses how the growing complexity of our computerized society threatens our control over

and our understanding of that society and proposes that we must exert concious control over the direction taken by our computer technology.

Robert C. Goldstein; "The Cost of Privacy" (1975)- This book attempts to develop a model to asess the financial impact of potential privacy legislation upon those who would ultimately have to implement the provisions of that legislation in their computer products and upon those who use these products.

Donella H. Meadows, Dennis L. Meadows, Jorgan Randers and William W. Behrens III; "The Limits to Growth" (1972)- This book presents a controversial computer model which attempts to predict man's future on this planet. Taking into account population, pollution, resources, capital and a number of other factors, this model predicted the occurrence of a major global catastrophe within the next hundred years or so if major changes are not made in our adherence to the ideal of continued growth. The book suggests a society based on stability rather than on uncontrolled growth.

Arthur R. Miller; "The Assault on Privacy" (1971)- In many ways, it is this book which initially brought many of the social consequences of computer use before the general public. It is a thoroughly documented book which should serve as a fine starting point for anybody doing serious research on computer

related social issues. This book contains numerous cases of abuses of computerized information, and it discusses many of the ominous implications that future computer use may hold for us.

Aryeh Neir; "Dossier: The Secret Files They Keep on You" (1974)- This book is an almost endless compendium of case studies in which dossiers about individual American citizens have served to damage their lives and reputations, even when the information in these dossiers was false. Included in the discussion are "School Records," "Mental Hospital Records," "Discarge Records," arrest and conviction records, "Credit Bureaus" and "Political Dossiers."

Neil Postman and Charles Weingartner; "Teaching as a Subversive Activity" (1974)- A thorough assault on the conventional methods of teaching and testing youngsters which frequently result in boredom and alienation for the student. Included too, is a compelling discussion of alternative techniques for making the classroom a more relevent place for the average youngster.

Project Search; "Security and Privacy Considerations in Criminal History Information Systems" (1970)- A study of the need of law enforcement officials [redacted] for criminal records vs. the right of privacy of those who these records pertain to.

Alvin Toffler; "Future Shock" (1970)- A book about what happens to people when they live in a continually changing environment such as the technological environment in which we now live. It discusses many technologies with ominous implications that have been proposed for the future including such things as genetic engineering. Finally, it proposes ways in which we can better deal with rampant technological change.

Rein Turn; "Computers in the 1980's" (1974)- This is a forecast about the state of computer technology during the next fifteen years. It takes a look at many of the technological advances in computer science that are just around the corner and it also attempts to build a general model for making technological forecasts.

Joseph Weizenbaum; "Computer Power and Human Reason" (1976)- This book attempts to dispel the notion that computers are simply more intelligent versions of man and it proposes instead that computers are nothing more than an "alien intelligence." The book's fundamental claim is that there are certain tasks which man now performs which computers should never be made to perform, regardless of whether or not they are able to perform them. A good deal of the book focuses on artificial intelligence, in particular.

Joseph Weizenbaum; "On the Impact of the Computer on

Society" (1972)- This paper sets forth the proposition that the indirect effects of a new technology are often much more important and far reaching than its direct effects. It goes on to discuss some of the ways in which computer technology may well manifest this pattern.

Milton R. Wessel; "Freedom's Edge: The Computer Threat to Society" (1974)- A discussion of some of the important social consequences of computer use which includes data banks, "Fair Acess to the Checkless/Cashless Society" and the loss of the "human factor" through computerization.

Alan F. Westin and Michael A Baker; "Databanks in a Free Society" (1972)- An empirical study of computerized data practices in a number of public and private agencies along with recommendations for the future of computer technology based upon the findings made about the agencies studied.