My Adventures with Dwarfs:

A Personal History in Mainframe Computers

by

Russell C. McGee

To Nelma
For her never-ending
Love, loyalty and support

Table of Contents

Preface

"The dazzling capabilities of modern personal computers are based upon the work and experiences gained from many years of use of mainframe computers." This statement is the theme of this book. The book is a slice through the history of those mainframe machines as experienced by the author. It gives one view of what was happening "in the trenches" as programming languages, information codes, subroutine libraries, debugging tools, operating systems and input/output techniques were being invented. It describes where programmers came from before colleges and universities had Computer Science Departments or gave courses that involved the use of computers. It tells how programs and data got into computers and results were delivered to users before an Internet and video terminals were widely available.

"The universe of mainframe computers is inhabited by IBM and the Seven Dwarfs". I don't know its origin, but this statement was often made during the early days of the computer industry and was recorded for posterity by Homer Oldfield in the title of his book *GE: King of the Seven Dwarfs*. I always worked for one or another of the seven dwarfs. One might say that this is a dwarf's-eye view of the mainframe computer industry; hence, the title.

That is not to derogate the efforts of the dwarfs. I believe the average dwarf employee was more involved in activities important to his or her employer than the average IBM employee.

Mainframe computers were giants that occupied fractional acres of floor space. They were housed in special rooms with customized electrical service to supply them with thousands of watts of power and large refrigeration plants to carry off the heat generated as a byproduct of their operation.

These machines had large staffs to operate them and other personnel to prepare the programs to control what they did. Mainframe computers have now been replaced for many purposes by personal computers that sit on one's desk and do one's bidding with a few keystrokes and/or a few mouse clicks. Most personal computers are faster, more reliable and equipped with more effective software support than any of the mainframe computers with which I was involved. Yet it comforts me to

think that all of the experiences we had and the lessons we learned during the early days of the mainframe machines set the stage and established the fundamentals upon which all of this modern achievement has been based.

If computers have not been well understood until recently, then the light is yet to dawn in many peoples' minds about what computer software is, why it exists and how it is created. A historical discourse on computers may provide a way to clarify how software came to be, when it started to be an integral part of the computer-related panoply of capabilities and some examples of modern software with which most people are probably acquainted.

I have recorded some of the technical facts I have learned both as a historical record and to connect my activities with the state of the computer industry at the time. Most of the text is derived from my own recollections of what I did for many years. The historical background of the computer industry prior to my personal involvement is contained in the Introduction. I have interspersed references to additional remarks about technological advancements within the body of the text and in the epilog. Each reference is given an alphabetical designation to distinguish it from other endnotes. Each reference is placed in the text at about the time I would have become aware of the particular advance(s) cited. These occur at intervals of one to several years and may be distractions from the flow of the story just as they were distractions to us as we conducted our jobs. It should be understood that we were always "shooting at a moving target". In spite of our best efforts to stay ahead of the competition, we were often kept off balance by technical advances affecting the entire industry.

Various people have different levels of acquaintance with computer techniques and technology. I have not wanted to make this a computer primer or a teaching tool; at the same time I am not able to present my story without using some technical terms and engaging in some technical reasoning. As a compromise, I have presented the main body of the text using terms and terminology I believe most computer literate persons of the twenty-first century will understand. For those who are insecure about their level of familiarity, I have provided appendices I believe will equip them with what they will need, to understand the main body of the text. In particular, Appendices A through C are intended to prepare the uninitiated reader with the knowledge needed to better enjoy reading the rest of the text. I recommend that readers who have questions about their computer background read these appendices before they read much beyond about page 26. As an additional aid to the uninitiated reader, I have included a Glossary of terms at the end of the book.

Also, some appendices could be skipped if they are not of interest. These are Appendices F and G that give some details about the RAYDAC computer. Appendices H through K are provided so the reader may, if he or she chooses, understand in greater detail some of the information included in the text. Hence, although I think they might be found to be interesting, reading them is optional. Finally, Appendix L gives a summary of my life before I became involved with computers.

Introduction

Before I became involved with computers in a really serious way and for several centuries before, various faltering efforts to develop them had occurred. These efforts can be grouped into the three separate types of devices that evolved. I refer to these three as calculators, automatic calculators and computers. The primary subject of this book is computers; however, the other devices deserve recognition because calculators are familiar to most readers and because calculators and automatic calculators played an important role in the early evolution of computers.

By a calculator, I mean a device a human can use to perform single arithmetic operations—usually, add, subtract, multiply and divide—upon numbers manually entered into the machine at the time the operations are to be performed. The most familiar modern manifestation of this object is the hand-held device many of us own singly or in multiples and can be purchased for a few dollars and powered either by batteries or by an attached solar cell.

The architecture of a calculator is extremely simple and is illustrated

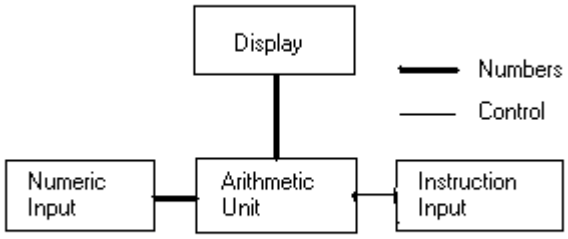

Figure 1 Calculator Architecture

in Figure 1. The display in a modern calculator is usually a Liquid Crystal Panel. It has been various other devices in the past including rotating wheels with numerals painted on them. The numeric input is now usually a small ten-key keyboard. In the past, large arrays of keys were often used for numeric input. If the precision of the machine were 10 decimal digits, then a 10 by 10 keyboard would be provided: ten digits vertically to permit entry of each of the possible decimal digits and ten columns horizontally to permit the entry of the ten different digits of the number. If the machine had only 8 digits of precision then a 10 by 8 keyboard would be provided: again ten rows to represent the digits zero through 9 and eight columns to represent the eight possible digits of each number. However, some machines used other input devices such as, but not limited to, rotary switches in electric or electronic calculators or rotary selectors in mechanical ones. Instruction input is usually an array of

buttons, one for each executable operation. In the simplest case one button each for addition, subtraction, multiplication and division. Also, a "Clear" button usually exists to reset all of the machine's contents to zeros and an "Equals" button to direct the machine to complete a previously designated instruction. The Arithmetic Unit performs the instructions directed by the Instruction Input. A single result is stored in the Arithmetic Unit and displayed on the display device. In some modern calculators an additional storage space is provided within the Arithmetic Unit, called a memory, in which a single additional number may be stored.

The ability of a calculator to perform arithmetic is limited by the speed with which the human operator can enter data and record results. Human operations are generally slow and error-prone. Hence, various automatic calculators were developed that permitted streams of calculations to be preformed at relatively high speed and, in some cases, after verification of inputs. Figure 2 shows the architecture of an automatic calculator.

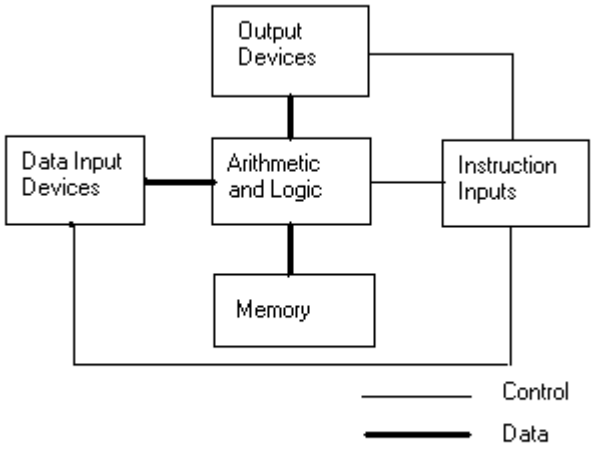

Figure 2. Automatic Calculator Architecture

The similarity to a straight calculator is obvious, but the memory now has been made visible because it typically is capable of containing more than one data entry. Each entry (or word) in the memory could be distinguished from the others by the use of a number called an address. Just as the house number distinguishes the houses on a street, the memory address distinguishes the cells in a computer memory. These cells are generally called words—each with its own address. The data-type inputs can now in some cases be alphabetic as well as numeric, so the term "Data Input" is used instead of "Numeric Input". Also, more than one input device may be provided—cards and paper tape readers, or

multiple paper tape readers, for example. Instructions are read from input devices (usually) distinct from the data input devices. Although only one instruction input is generally provided, such as a paper tape reader, in some machines multiple, alternate instruction inputs are provided. Because the outputs now come out in a stream, it is necessary to provide an output device such as a printer or cardpunch to accept the output stream. Multiple output streams might be provided, each one of which will accept part of the entire output stream. Finally, the Arithmetic Unit is renamed the Arithmetic and Logic Unit (or ALU) because in some machines, this unit could perform logical decisions that could cause the instruction input to switch from one input device to another.

Finally, we come to the computer, also known as the Stored Program Computer or the Von Neumann Computer. Its architecture is the architecture of all modern computers. It is illustrated in Figure 3.

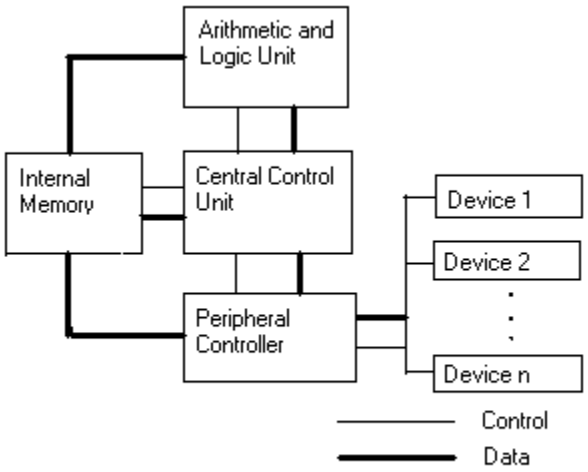

Figure 3. General Computer Architecture

The big change that occurs when moving from the Automatic Calculator to the Computer is that both data and instructions share the memory in the Computer whereas they do not in the Automatic Calculator. This change has a profound affect upon the capabilities of the resulting machines. A Central Control Unit is now at the heart of the system. This unit controls and directs the operation of all other units in the system as specified by instructions stored in the Internal Memory. (These control functions were buried in the Instruction Input Unit in Automatic Calculators.) More importantly, the Arithmetic and Logic Unit is able to perform operations on the instructions contained in the memory, and the Central Control Unit is capable of retrieving instructions from any location (address) in the Internal Memory. These flexibilities give this architecture a vast superiority over its predecessors.

As before, the Arithmetic and Logic Unit performs all actual computational tasks, but, in addition, it can perform logical operations that permit programs to be made intricate and complex to a degree limited only by the imagination of the programmer and the need to be able to maintain and adjust the programs in the future. A Peripheral Controller is now introduced that instructs the peripheral devices and controls the flow of data between the devices and the Internal Memory. In some systems the Peripheral Controller functions are included within the Arithmetic and Logic Unit or the Central Control Unit. The advantages/disadvantages of these arrangements will be discussed in following chapters.

All three of these architectures were employed at various times during the evolution of computers. Table 1 shows the names of a collection of these progenitors and the dates upon which they appeared.

Table 1. Computer Evolution to 1952[1]

Originator	**Device Name**	**Device Type**	**Year(s)**
Wilhelm Schickard	Calculating Clock	Calc	1623
Blaise Pascal	Pascaline	Calc	1644
Sir Samuel Morland	Monetary Adding Machine	Calc	1668
Gottfried Wilhelm von Leibnitz	Stepped Reckoner	Calc	1674
Charles, 3rd Earl of Stanhope	Multiplying Calculator	Calc	1775
Mathieus Hahn	Hahn Multiplying Calculator	Calc	1770-1776
J. H. Mueller	Mueller Difference Engine	Calc	1786
Charles Xavier Thomas de Colmar	Arithmometer	Calc	1820
Charles Babbage	Babbage Difference Engine	Auto	1822-1832
George Scheutz	Scheutz Difference Engine	Auto	1834
Charles Babbage	Analytical Engine Design	Auto	1834
George and Edward Scheutz	Printing Difference Engine	Auto	1843
George and Edward Scheutz	Tabulating Machine	Auto	1853
Charles Babbage	Prototype Analytical Engine	Auto	1871
Ramon Verea	Multiplying Calculator	Calc	1878
Anonymous	Mass Produced Calculator	Calc	1885
Dorr E. Felt	Comptometer	Calc	1886
Dorr E. Felt	Printing Desk Calculator	Calc	1889
Herman Hollerith	Punch Card Tabulator	Auto	1890
William Burroughs	Office Calculator	Calc	1892
IBM	IBM-601	Auto	1935
Konrad Zuse	Z1	Auto	1939
Bell Telephone Labs	Complex Number Calculator	Auto	1940

John V. Atanasoff and Clifford Berry	ABC	Auto	1941
Howard H. Aiken	Harvard Mark I	Auto	1943
Max Newman and Wynn Williams	Heath Robinson	Auto	1943
Wynn Williams and George Stibitz	Relay Interpolator	Auto	1943
Tommy Flowers	Collosus	Auto	1943
John W. Mauchly and J. Presper Eckert	ENIAC	Auto	1945
Howard H. Aiken	Harvard Mark II	Auto	1947
Wallace Eckert	IBM-SSEC	Auto	1948
Max Newman and Freddie C. Williams	Manchester Mark I	Comp	1948
IBM	IBM-604	Auto	1948
Maurice Wilkes	EDSAC	Comp	1949
Presper Eckert and John W. Mauchly	BINAC	Comp	1949
Howard H. Aiken	Harvard Mark III	Auto	1949
National Physical Laboratory, Teddington, England	Pilot ACE	Comp	1950
Jay W. Forrester, MIT	Whirlwind	Comp	1950
U. S. National Bureau of Standards	SEAC	Comp	1950
U. S. National Bureau of Standards	SWAC	Comp	1951
University of Pennsylvania	EDVAC	Comp	1951
Engineering Research Associates	ERA 1101	Comp	1951
Ferranti, Ltd.	Manchester Mark II	Comp	1951
Lyons Company	LEO I	Comp	1951
Remington Rand	UNIVAC I	Comp	1951

Of the 45 entries in Table 1, 13 are calculators (device type Calc), 20 are automatic calculators (device type Auto) and 12 are computers (device type Comp). The calculators range from the crude Calculating Clock in 1623 to the Burroughs Office Calculator in 1892. Of the 13 calculators listed, three were successfully marketed commercially—ten to fifteen Pascalines dating from 1644 were sold by Pascal and additional ones by others (no patent laws existed then), Arithmometers dating from 1820 were sold for 90 years, Comptometers dating from 1886 were sold well into the 20th century and William Burroughs started the office calculator industry in 1892.

Of the 20 automatic calculators, two Scheutz Tabulating Machines dating from 1853 were sold; the Hollerith Punched Card Tabulator was used to tabulate the 1890 census. (Hollerith formed the Tabulating Machine Company in 1896, which merged with the Computer Scale of America Company and the International Time Recording Company in 1911 to form the Computing-Tabulating-Recording Company. It was renamed International Business Machines (IBM) in 1924.) IBM-601s through 604s were widely used starting in 1935 and continuing into the 1950's and beyond. Several of the automatic calculators were special purpose in nature: the Complex Number Calculator, the Heath Robinson and the Collosus, for example. The latter two were built at Blechley Park in England to be used in breaking enemy codes during World War II. Ten Collossi were built during this period and their use was instrumental in breaking the Nazi Enigma code that the Germans thought unbreakable.

The only machines in this early evolution that could be considered commercially successful on a scale comparable to modern computers would be the automatic calculators by IBM—the 601 through 604. However, by the time those IBM machines were available, the true computers were beginning to appear in the commercial market. The last four entries in Table 1 were commercial ventures and all of them were true stored-program computers. Eight Manchester Mark II's were sold. The Lyons Company had originally supported the development of the EDSAC with the intension of using it within the company; however, they soon found themselves in the computer business in addition to their earlier enterprises. (For anyone who is interested, an EDSAC Simulator is available on the Web at http:www.dcs.warwick.ac.uk-~edsac-Software-EdsacTG.pdf. You must use Adobe Acrobat to read the document and receive further instructions. The simulator has an option that permits you to slow it down to the speed at which the EDSAC would have run.) The UNIVAC I was the work of the Eckert and Mauchly Corporation, which by 1951 had been bought out by Remington Rand. It was the beginning of what would become the Sperry Rand Corporation and would survive for many years to come. Engineering Research Associates also later merged with Sperry Rand and survived for several years.

Significant progress in computer technology had been achieved and a common vision of modern computer architecture had occurred during the early evolution of the computer. Starting with gears and chains to perform calculations, then relays, the evolution had ended with the exclusive use of electronic components. Table 2 shows some of the major hardware technological milestones reached in this period.

Table 2. Evolution of Computer Hardware Technology to 1952

Originator	Item	Year(s)
Herman Hollerith	Punched Card	1890
W. H. Eccles and F. W. Jordan	Flip-flop circuit design	1919
E. Wynn-Williams	Binary digital counter using thyratrons	1931-2
George Stibitz	1-bit binary adder using relays	1937
Alan M. Turing	The Turing Machine	1937
Claude E. Shannon	Implementation of symbolic logic using relays	1938
John V. Atanasoff and Clifford Berry	16-bit adder using vacuum tubes	1939
Schreyer	10-bit adder using vacuum tubes and memory using neon lamps.	1939-40
Atanasoff and Berry	Regenerative memory using capacitors mounted to revolving drums	1941
Max Newman and E. Wynn-Williams	High-speed Paper Tape Reader—2000 characters per second	1943
John von Neumann	Stored program computer architecture.	1945
Frederick Viehe	Magnetic core memory	1947
Various Independently	Magnetic drum memories.	1947
Freddie C. Williams	Cathode Ray Tube Memory	1948
An Wang	Magnetic core memory as a serial delay-line memory.	1949
William Shockley and Presper Eckert	Delay line memory.	1949
Remington Rand	Magnetic Tape	1951
Jay Forrester	Magnetic core memory as it is to become commonly used (as random access store).	1951

Herman Hollerith introduced the punched card into the computer industry, but it was not a new and unique invention. Punched cards had been used previously for control of the Jacquard Loom. Hollerith's cards had 80 columns. Remington-Rand also used cards, but theirs were 90-columns wide. Flip-flop circuits are notable because they were the first electronic bi-stable devices; switches and relays preceded them.

Alan Turing's machine introduced in 1937 is not a real computing machine, but a theoretical, simplified computer used as a mathematical device in problem solving. It was introduced in a paper [2] on

"computable numbers" and has since been a cornerstone of computer science.

Some of the contributions to technology turned out to be dead ends, such as the neon lamp memory and the capacitor memory. However, regenerative memories have been and are still in frequent use. The paper tape reading speeds achieved with the Blechley Park machines were indeed impressive. They were reading at 5000 characters per second by 1945, but magnetic media have superseded paper tape in modern computers.

The von Neumann computer architecture might be the most significant item contained in Table 2; however, von Neumann's claim to credit for that breakthrough was challenged. Various people felt von Neumann had given insufficient credit to others in introducing the stored-program idea. In addition to the objections of Eckert and Mauchly, some [3] thought the original thinking was due to Turing in the form of the Universal Turing Machine described in his landmark paper of 1936.

Without question, magnetic cores won the battle in the 1950s for best internal memory. The three contenders are listed as the last three entries in Table 2. Having used all three at one time or another, I can assure the reader, my colleagues and I preferred magnetic cores without reservation. Williams Tubes were said to be inexpensive, but Whirlwind converted from Williams Tubes to Magnet Core Memory in the 1952-3 time frame because, among other things, they were spending $32,000 per month in replacement CRT tubes. Delay lines, being serial memories, were just too slow to be competitive.

In the area of peripheral storage, the introduction of magnetic tapes on the UNIVAC I was a major breakthrough. That along with magnetic drums would be the paramount bulk storage media for several decades to come. However, the tapes used by Eckert and Mauchly were not the plastic tapes with which most of us are familiar. Their tapes were nickel-coated bronze with a recording density of 128 characters per inch.

Because the von Neumann Computer is so important to us, its first conceptualization and the immediate consequences it had upon the evolution of computer technology is worth reviewing. It all started with the ENIAC [4], developed under contract to the U. S. Army's Ballistic Research Laboratory at Aberdeen Maryland and was the first all electronic computer. Ballistic Research Laboratory, also known as BRL, and previously known as APG, the Aberdeen Proving Ground, was

responsible for providing the Army with "firing tables" used in artillery aiming.

During the early days of World War II, these tables were calculated with the assistance of mathematicians at the Moore School of the University of Pennsylvania. Women who operated hand calculators had originally performed the necessary calculations and a single 60-second projectile trajectory took about 20 hours to complete. The university had a digital differential analyzer that was put into use, but it too took about 15 minutes to complete a 60-second trajectory. The demand, in 1943, for the computation of new cases was greater than the computational capabilities could accommodate.

In response to this need, BRL issued a contract in June 1943 to build a computer called ENIAC, for Electronic Numerical Integrator And Computer. The ENIAC, which was actually an Automatic Calculator, was completed in 1945, too late to make any impact during the war. (However, it could calculate a 60-second trajectory in 15 seconds—a shorter time than a shell would be in the air.) It was then moved to Aberdeen and again put into operation and continued in use until 1955, when it was permanently shut down. In the intervening period, it was modified several times including the addition of a 100-core magnetic core memory provided by the Burroughs Corporation. (It was also later modified to permit instructions to be stored and retrieved from its memory. Based upon this fact, some people claim that ENIAC was the first stored-program digital computer. However, the machine that resulted was rather different from modern computers and I, for one, prefer not to include it in the class with other von Neumann machines.)

However, the greatest reward resulting from the ENIAC development wasn't the machine itself, but rather the gathering together of the best minds of the free world on the subject of automatic computation. The cross fertilization of thoughts and ideas that resulted from this gathering resulted in a previously unprecedented explosion of innovation in the computer field. J. Presper Eckert was the principal engineer on the project and John W. Mauchly was the prime consultant on the ENIAC. The staff was of the highest quality; most of them were well known for personal accomplishments in related fields. John von Neumann, then of the Institute for Advanced Study at Princeton became affiliated with the project in 1944. He was a member of the BRL Scientific Advisory Board. On August 29, 1944, that board recommended funding of EDVAC, Electronic Discreet Variable Automatic Computer.

In June 1945, von Neumann produced a paper entitled "*First Draft of a Report on the EDVAC*". The stored-program concept was first documented in this paper.

In July and August of 1946, a series of 48 lectures were given at the Moore School entitled "*Theory and Techniques for Design of Electronic Digital Computers*". Although only 28 persons attended all of the lectures, several others attended one or more of them. Most importantly, the biggest names and most influential persons in what would become the computer industry were in attendance.

Construction of the EDVAC at BRL began immediately after it was recommended. Shortly after the conference, the EDSAC was started at Cambridge University in England as was the Manchester Mark I at the University of Manchester. The SEAC, Standards Eastern Automatic Computer, was started in the U. S. Also, Eckert and Mauchly had formed their own company by the time of the Moore School lectures and had begun construction of the BINAC for the U.S. Air Force. All of these machines were based upon the EDVAC Report, were influenced by the lectures at the Moore School and represented the true beginning of the "Computer Industry".

Whereas electronic technology and the other devices needed to complete physical computer systems advanced rapidly in the period preceding 1952, the same cannot be said for the art of programming. Some people say Ada Byron Lovelace [5] was the first computer programmer. That may or may not be true, but even if it is, it is probably irrelevant because the Babbage Analytical Engine upon which she was said to have done the programming was in no significant way similar to modern computers and the machine was never built. Hence, Ada's experiences are not in any way applicable to the work performed by modern programmers. She did recognize the importance of subroutines and program loops, but these are concepts that can be taught in a few minutes to neophyte programmers so they are not major intellectual resources.

In 1945, a remarkable German named Zuse, who had by then invented several automatic calculators, invented a programming language named Plankalkul [6]. Commentary on the language on the World Wide Web is very complimentary; however, it was never implemented as far as I have been able to determine. Furthermore, the accomplishments of Zuse were unknown (or unrecognized) until long after World War II, so even if the language had been implemented, it could not have influenced programming in the U. S. and the U. K.

The Web site referenced in Note [1] says under the heading "Jun 1948", "Turing joins the team [at the University of Manchester] and devises a primitive form of assembly language, one of several developed at about the same time in different places." The only other reference in this time period to any programming aid appeared in a book entitled *The Preparation of Programs for an Electronic Digital Computer,* by Wilkes, Wheeler and Gill [7] of the EDSAC computational laboratory at Cambridge University in England. This is a complete description of the EDSAC Assembly Program along with a description of its use of a subroutine library. It was, no doubt, one of the assemblers referred to in the Note [1] reference. (See Appendix E for a brief description of various computer languages including assembly language.)

In 1951, Grace Hopper of Remington Rand (which had bought out Eckert and Mauchly), invented the modern concept of the compiler and the A0 programming language. These were, no doubt, major contributions, but they were baby steps in the long road to the achievement of useful and well-accepted high-level languages. In the same year Betty Holberton, recipient of the 1997 Ada Lovelace Award [8], introduced a sort-merge generator for the UNIVAC I. This was the germ of what would be some very important future developments.

Other accomplishments surely occurred in the programming field in this time period, but references to them are unavailable. It is clear that these machines needed to have loaders to load their programs, subroutine libraries for calculation of mathematical functions and debugging aids such as memory dumps and program trace routines. Of all these needs, only the book by the EDSAC group provided any description. The use of flow diagrams also arose in this period and was of assistance in the design and planning of programs. Their original use is attributed to John von Neumann [9].

Getting Started

I was born and raised in Stockton, California and received my early education there in the public schools. As with many others, my education was interrupted by World War II. I spent a short time, while in the Army at the University of Idaho, in Moscow, Idaho, but shortly thereafter was trained to be a Cryptographic Technician and was sent to Greenland and the Azores where I spent the remainder of my Army career. [A more complete personal history may be found in Appendix L.]

After discharge from the Army, I received a baccalaureate in Physics from the University of California at Berkeley. I was married during my sophomore year of college and our first daughter was born during my junior year. During my senior year, I received an offer from the U. S Naval Air Missile Test Center at Point Mugu, California to go to work after graduation. The salary was to be $2,475 per year and I accepted it.

When our little family arrived at Oxnard in June of 1950 to report for work at Point Mugu, our first challenge was to find housing. It was still close enough to World War II that housing was at a premium and it seemed our only choices were to buy a new house, which we were financially unable to do, or rent. The problem with renting was that the only things available that we could afford were units in government-built housing developments left over from the war. Since these rentals were our only choice, we moved in.

All of the units were located in long, narrow quadraplexes. The area was referred to as "Kerosene City" because all the appliances—stoves, space heaters, water heaters and refrigerators—were fueled with kerosene. No electric or gas appliances were available during the war, so these were the substitutes. However, our unit suited us very well and though it was quite humble, the rent fit our salary.

When I reported to work, I discovered I had been assigned to the Computational Branch of the Mathematical Services Division of the Range Instrumentation Department of NAMTC (Naval Air Missile Test Center). Of course, all of these organizational names meant nothing to me, but on the first day, I was asked if I would be interested in working with an electronic digital computer. Several other recent graduates and I were very interested. Hence, we all accepted our assignments to the Computational Branch.

I was now beginning to meet the people, locations and organizations with whom and within which I would be working. Some of the people who started when I did were Norman Potter, Jim Tupac, Chuck Aronson and Bob Causey. Our immediate supervisor was Don Dufford and his boss was Margaret Swanson. She reported to a man named Ed Ring (I think). Others already on board when I arrived were Shirley Colton and Phyllis Mastracola. All of these people had degrees in mathematics except Norm Potter, who had a degree in engineering of some sort. The idea then was that computers had something to do with mathematics, which was only somewhat true.

Later, but while I was still reporting to Don Dufford, Max Eaton joined our staff. He was one of the greatest characters I ever met. He was a statistician who had been trained at the University of Kansas and all of his examples involved Pigs and bushels of wheat, etc. He had also been a Major in the Army and was about fifty years old, which to me at the time seemed ancient. He had opinions about everything that he shared spontaneously. He invested in the stock market, played poker in Gardena and loved to play chess. We used to play chess at lunchtime and he was extremely good at it. I don't think I ever beat him. He was always eager to learn and always enrolled in the UCLA courses many of us took. Knowing Max was one of the great positive experiences of my life.

When I first arrived at Point Mugu, all of the buildings were leftover, one-story World War II barracks-type structures. Ours was located right across the street from a large diesel-electric generator that was started at about 8:15 every morning and ran all day until 4:45, fifteen minutes to quitting time. At first it was annoying, but soon we didn't hear it at all during the day. The entire Mathematics Branch, including Don Dufford had desks in one room of this building. There was no privacy; nowhere to go to think without possible interruption.

Since I had first heard of computers, I had learned that most of them were named after either a wind and/or had a name ending in "ac" for "automatic computer". Our machine was to be called the RAYDAC or the Hurricane Computer. In 1949, the EDSAC and BINAC; in 1950 the Pilot ACE, Whirlwind and SEAC; and in 1951 the SWAC, EDVAC, ERA1101, Manchester Mark-II, LEO-I and UNIVAC-I had first been brought into operation. In 1952, [10] the MANIAC-I, ORDVAC, ILLIAC, RAYDAC and ELECOM 100 joined this list. There were a variety of less well-known computers introduced so that RAYDAC was actually the 33rd [11] computer ever built. It is also one of those that came into being during the highly productive period following the building of the ENIAC.

Most of these early machines were unique in their designs and configurations and were built to satisfy rather specific application objectives. It is not surprising that few companies wanted to get into the "general purpose" computer market. Such sages as Howard Aiken of Harvard had predicted only six computers would be needed to fulfill the computational needs of the United States for the foreseeable future, so who would buy these machines?

The problem was that these "sages" were thinking of computers in terms of their own backgrounds in which automatic calculators were used simply to print page after page of tables. These would then be published for others to use manually. Examples of these tables were the "firing tables" the ENIAC was created to produce and the mathematical tables, such as tables of logarithms produced by the Bureau of Standards. However, a few people, such as Eckert and Mauchly and the people who built the ELECOM 100, were able to see beyond this limited horizon and went into the general-purpose computer business.

In any case, our computer was to be the RAYDAC, built by the Raytheon Manufacturing Company under contract with the Special Devices Center of the Office of Naval Research. (I discovered while writing this book that the head of the RAYDAC development at Raytheon was Richard M. Bloch, a person that would enter this story again much later.) RAYDAC was a little different from some other computers in that it was one piece of a larger real-time data reduction system. The system was known as the "Hurricane System" and hence the computer was called the Hurricane Computer. The system was said to have a price tag of four million dollars, so it is hard to say how much the computer cost. Hence, we were the employees who were to work on a non-existent computer for which the manuals had not yet been written. In the meantime, until the computer became a reality, we were given other jobs to do that would relate (it was supposed) roughly to our work when the computer was ready.

The work we did usually involved using mechanical calculators such as those made by the Friden or Marchant Corporations. These usually occupied about a sixteen by sixteen inch square on a desk and stood about 10 inches high at the back and slanted downward toward the front. They had a keyboard of 100 keys in a 10 by 10 array for data entry and additional keys for controlling the machine. A carriage at the top of the machine displayed operands and results. Their capabilities were somewhat less than the pocket calculators that have been commonly available since the advent of the transistor. They could add, subtract, multiply and divide; but could not compute the square root of a number.

(I believe a "Square-root Friden" came out during the 1950's, but it cost about $4000 compared to about $400 for a standard Friden).

The assignments we executed using these machines varied. Often someone would have created a form describing a particular sequence of calculations. Our task consisted of filling out page after page of such forms based upon the calculations they specified. After a while, though, most of us were assigned to help someone else in performing his or her job. In my case, I was assigned to work with Harold Gumble, who was a member of the Aerodynamics Branch, working on what was called "The Sparrow Flutter Problem".

Flutter is a phenomenon that occurs when the physical bending of an airframe becomes coupled with the lift of the control surfaces. If a control surface causes the airframe (in this case a Sparrow Missile) to climb, this can cause the airframe to be bent in such a way that an additional displacement of the control surface occurs causing additional climbing that causes additional bending of the airframe, etc. This can result in loss of control, or, in extreme cases, destruction of the airframe. Gumble's problem was to examine the degree to which flutter could occur with the Sparrow Missile and the conditions that would cause varying degrees of flutter to occur.

Gumble was pursuing the problem using a Digital Differential Analyzer at California Institute of Technology in Pasadena and also by manual calculations. I got involved in the manual calculations and also made several trips to Cal Tech with him. Of course, the Differential Analyzer could produce results much more rapidly than I could. My job was to compare the Differential Analyzer results with my manual calculations and try to explain any differences.

It should be understood that a great debate [12] was going on in the early 1950's about what kind of computers were to become dominant: digital or analog? Of course, most of the "ac's" and "winds" were digital computers. They performed ordinary arithmetic and logic at very high speed and under control of a program prepared in advance. While digital computers were beginning to be used and recognized as valuable, analog computers were also being used and were producing useful results. These were computers that used the values of voltages and/or currents in electrical circuits or mechanical parameters in mechanical systems to represent the values of the parameters in a problem as the dynamics of the problem system were carried out.

People who preferred analog computers argued that analog computers were easier to program, and much less expensive than digital computers. Digital computer proponents argued that analog computers provide insufficient precision and flexibility for many uses and digital computers could do any computation whatsoever. They said that with the development of better programming techniques and methods digital computers would win the day.

Hybrids such as Digital Differential Analyzers existed that performed the same mathematical functions as the components of analog computers, but did these things digitally. This provided the ease of use of an analog computer with the precision of a digital computer. It is clear today that digital was the way to go; analog computers and Differential Analyzers are now hardly ever even mentioned.

In addition to providing a job with a small income, working at Point Mugu provided some other benefits. For example, we had the chance to watch the launch of some of the navy's most advanced missiles. It was a short walk from where our offices were located to the launch pads from which the tests, which were the bread and butter of Point Mugu, originated. We often took this short walk to the beach to "watch the show". I am sorry to report that the "show" was usually a flop. Very frequently, the missiles just plopped into the surf. The most interesting ones to watch were the Loons that were U. S. manufactured copies of the German V-1s. They usually made it off their launching rails and actually flew. Only once while I was there did a missile (it was a Sparrow) leave the pad, fly about five miles out to sea and demolish a B-17 drone. From where we stood, it looked more like the drone ran into the missile than the other way around, but we were happy to see it nevertheless.

A substantial benefit of being at Point Mugu was proximity to UCLA. The U. S. Bureau of Standards, Institute for Numerical Analysis (referred to as INA) was located there. During the same time we were awaiting the RAYDAC, the people at INA were awaiting their computer, the SWAC (Standards Western Automatic Computer) or Zephyr. They had weekly symposia to which we were always invited. Because of its location on the university campus and its connection with the Bureau of Standard, the speakers at the symposia were often people of note in the field at the time. Hence, we had an opportunity to listen to and rub elbows with some outstanding authorities at a time when we were totally uninitiated neophytes.

To understand the function of the Institute for Numerical Analysis, it is necessary to understand that prior to the advent of modern computers and their associated software, Scientists of all sorts—Astronomers,

Physicists, Statisticians, for example—had to look up the values of mathematical functions in books of tables. (Slide rules provided some functions, but only at low precision; high precision values had to be found in tables.) Where did they get these tables of functions? Universities and the Bureau of Standards supplied them. Hence, the art and science of Numerical Analysis was an important area of expertise for the Bureau.

It turned out some of the employees of INA were computresses who had worked for the bureau for many years. They were all experts in numerical analysis and were exceptionally skilled at the use of desk calculators. Each of them had contributed to the creation of many volumes of mathematical tables. They were generally kindly, elderly, spinsters that knew more about how to calculate mathematical functions to any degree of precision desired than almost anyone. They were another valuable resource to which we had access.

The other advantage of being close to UCLA was they offered courses at Point Mugu in which we could enroll. During the time I was there, I took 21 semester hours of graduate mathematics via this program. The courses were fascinating and I think I got A's in all of them. I could have gotten a Masters Degree in Mathematics if I had completed an additional 3 units, fulfilled a residence requirement and written some sort of thesis. Because of the residence requirement and the thesis, I didn't complete the degree, a fact I have often regretted.

During the fall of 1950, the U. S. Government let a contract to the Raytheon Corporation to provide two courses of study on the RAYDAC Computer to certain Civil Service Employees. The courses were to: first, prepare a group of engineers to maintain the computer after delivery and second, to train a group of programmers to program the new machine. The courses were to be conducted at the Raytheon Factory in Waltham, Massachusetts starting in January and continuing until the spring of 1951. The hardware types (engineers) chosen to go were Harold Baugh, Sigmund Yelen, Matt Gibson, Bob Waller, one Point Mugu employee whose name I forget and one man from NSA (National Security Agency); the programmers were Norman Potter, Jim Tupac, Don Dufford, Bob Causey, Chuck Aronson, a person named Pederson from NSA and myself. The Navy paid our transportation to and from the factory and gave each of us $900 to cover our other expenses including rent. To us, this seemed like a lavish stipend, but, in truth, it was just about right.

The decision to participate in the class created the very serious problem of separating me from my little family. By the time we were to leave, we had moved to a little rented house in Oxnard. We were very happy in

Oxnard and the prospect of being separated for three months was not at all a joyous one. However, the career enhancement aspect of taking part in the training could not be ignored, so, in her usual supportive and good-natured manner, my wife went along with the decision to have me participate.

My Introduction to Computers

Norman Potter, Jim Tupac and I were together most of the time during our educational journey to Waltham, Massachusetts. The three of us left the Oxnard Airport aboard a DC-3 on a very rainy night in January 1951. Because of the storm, the airline put us up overnight in Los Angeles. We completed our flight the next day on a TWA Constellation—a four-engine airplane that was the height of luxury in its day. When we arrived in the Boston area, we arranged to find a place to stay in a private residence.

The residence was the home of an elderly lady and her daughter who was probably in her thirties. It was a pleasant, two-story house with three bedrooms and a bathroom upstairs. The daughter occupied one of the bedrooms and the three of us shared the other two. We took turns occupying the single room, so we each had an opportunity for some complete privacy from time-to-time. The house was within walking distance of the Raytheon Plant and a diner where we usually ate. It was also within walking distance of Watertown Square where we could get a trolley that connected with the subway to Boston.

Image 1. The RAYDAC Class, Waltham Mass., 1951

Of course, we had had a bit of an introduction to computers from our visits to INA in Los Angeles and from informative meetings we had at Point Mugu. However, the assumption of our course at Raytheon was that we knew nothing about computers and so our training started us out from scratch. That was a reasonable assumption and, for practical purposes, it was true. The first part of the course was called logical design and was taken with the hardware and the programming personnel all in attendance. In the second part of the course, the two groups had separate classes according to the needs of their separate technical specialties. Each of the classes was taught in the morning and the afternoons were free to do homework and to study the material.

The attendees and instructors at the RAYDAC course are shown in Image 1. The instructors are all standing; the students are seated. There are only two of the teachers that I remember: Emma Cummerford, the only woman, and Robert Brooks, standing at the right rear of the room. Seated from front to back are: front row, Unknown #1, Norman Potter, Jim Tupac; second row, Don Dufford, Bob Waller, myself; third row, Unknown #2, Sigmund Yelen, Jim Harvey, Jim Pedersen; fourth row, Hal Baugh, Matt Gibson and Bob Causey. Unknown #1 was an engineer from Point Mugu, but I don't remember his name. Unknown #2 was also an engineer, but he was from the National Security Agency (NSA). I don't think I ever knew his name. All of the students were from Point Mugu, except for Unknown #2 and Jim Pedersen. Both of them were from NSA. According to Jim Tupac, there was another student from Point Mugu, not in the picture. His name was Chuck Aronson.

Robert Brooks of Raytheon taught the Logical Design course. Some of the general and introductory material he covered is described in Appendix A—Some Computer Fundamentals. [This would be a good place for readers unfamiliar with computers to read appendices A through C.] During one of the first days of the course, he handed each of us a copy of huge blueprints. These were the logical design of the RAYDAC. They did not consist of electrical circuit diagrams, but logical circuit diagrams. The elements of the logical diagrams were flip-flops, logical **and** gates, logical **or** gates, delay lines, one-shot multi-vibrators and probably other things I have forgotten. Corresponding to each of these logical elements, electrical or electronic elements existed. So if one had the logical design and the electrical design of the logical elements, one could, in theory, build the computer.

We were to spend many hours, both at the Raytheon Plant and at our rented quarters, pouring over the logical design blueprints of the RAYDAC.

In the end, we really knew what the RAYDAC was, how it worked and why it worked the way it did. I have long since forgotten much of the detail we learned, but the general knowledge was to serve me well for many years. I believe the Logical Design Course lasted for 5 or 6 weeks. We had a test at the end of that period and I got the highest grade in the class; Jim Tupac got the next highest.

After the Logical Design Course was completed, the Engineers and the Programmers were divided into separate classes. Our class was, of course, RAYDAC Programming. The class was taught by the only woman from Raytheon we had any contact with during our stay. She was Emma Cummerford—a young woman, age 30 or so, who was very bright. Some of the material she taught us, as I can recall it, is given in Appendix F—RAYDAC Programming. Emma was an excellent teacher and was well liked by the class.

It should be understood that learning to program, or teaching programming, in those days was a different task than learning to use a modern computer. We had nothing to work with except the bare computer. The manufacturer provided no programs except a loader—a program to move other programs from an external medium into the computer's memory. No compilers or assemblers or subroutine libraries were supplied. Many, if not most of these things had yet to be invented. No books or journals or manuals on RAYDAC or any other computer or the general principals being used, were available. So Emma Cummerford had the task of teaching us about programming the RAYDAC, but also to suggest to us the mountain of other programs and capabilities we would need to make this machine a useful and useable tool.

One important area of knowledge we needed to give life to RAYDAC was numerical analysis. This brought us back to our seminars at INA and to some extent also to the courses we had taken at Point Mugu. Many of the ideas our west coast experiences had introduced were now becoming tools of our trade and not just abstract ideas floated by us in lectures. Ms. Cummerford had a good feel for these skills and passed along much of her knowledge to those of us in her class.

At the end of the class, we had another test. Again, Jim Tupac and I got the top two scores in the same relation to each other.

Whereas most of us had flown to the Boston area, Don Dufford had driven his car. On several occasions, he took Norm and Jim and I on excursions around the area. Once we went to the Harvard Computation Laboratory in Cambridge. Here we toured the facility of the Mark IV, the fourth in the

series that started with the Mark I. Howard Aiken was the director of the laboratory and was a well-known name in the computer industry at the time. The facility was very attractive and glossy, but I didn't get much out of the tour. The Mark's were not stored-program computers—their programs were stored on long strips of punched paper card stock and were separate from their data. As such they represented a less advanced version of computer technology than the RAYDAC. They also had things called "magnetic buckets" that were used for data storage. I never did understand what they were or how they worked.

On another occasion, we visited the Whirlwind Computer Laboratory at Massachusetts Institute of Technology, MIT. This was more to my liking and understanding. The Whirlwind was a 16-bit stored-program machine that had been built at MIT and bore many characteristics that were similar to RAYDAC. It was built on vertical racks exposed to view. As I recall, they were arranged in two parallel rows. It, unlike RAYDAC, was air cooled, so the room air was re-circulated and refrigerated, hence it was noisy but temperate in the computer room. It had Williams Tube Memory (later replaced by magnetic cores). It had magnetic tape handlers and Teletype input and output devices. It also had a very advanced library of program subroutines.

When we got back to Point Mugu, RAYDAC was still in Waltham and was not in operation. That was okay, though, because no building existed to put it in, but one was under construction—Building 50, the Range Instrumentation Building. We watched as it was being built and we were very anxious to occupy it in lieu of our old wooden hovels.

Building 50 was a concrete structure with many windows. In general, the front of the building contained offices and the rear contained laboratories and workspaces with a hallway down the middle most of the length of the building. Entry to the building was by way of a lobby connected with this hall. The lobby was bare except for a doorway on its left (east) wall that led to a nice conference room convenient for meetings of up to 40 people. As you entered the hall from the lobby the opposite wall was bare except for two doors. The first one, to your left, was large and heavy. This was the door to the RAYDAC computer room.

At the end of hall past the computer room door to the right was a snack bar where you could buy beverages and small food items. To the left was a double door that entered into a large bay, where our offices would be, including the office of our manager and his small staff.

The other door in the wall opposite the lobby entrance was to the right as you entered. It was a standard door that entered a room that contained the RAYDAC peripheral equipment—the Problem Preparation Unit (PPU) and Output Printer. This was called the PPU Room. The PPU Room had a back door that led to a laboratory area where our engineers worked. There was another door to the computer room just inside of this lab on the east wall in the northeast corner.

Down the hall to the west, past the PPU Room, offices were located on the right and labs on the left. One of the first labs you would encounter was the one containing IBM punched card equipment. That is all I recall about the layout of Building 50.

The evolution from the EDVAC to the RAYDAC is not clear, except RAYDAC was part of the explosion of machines developed after the publication of the EDVAC report mentioned in the Introduction. However, RAYDAC had several features that were unique or, at least, new in relation to some of the others in the group. First, it was self-checking. Unlike the BINAC that performed checking by having two complete computers, which checked on one another, the RAYDAC used mathematical algorithms similar to the method of "casting out nines" in decimal arithmetic to check each arithmetic operation. (Actually, RAYDAC performed its checking in base 32 and cast out 31's.) It also checked all internal information transfers with parity checking that would catch most errors of movement of data within the machine. It also had four magnetic tape handlers that used a unique method of searching for information, which involved the optical sensing of block numbers printed on the back (non-magnetic) side of the tape.

Image 2. RAYDAC without its skins in the Raytheon factory.

Contrary to most other computers of its day, the RAYDAC was Freon cooled. Most others were air cooled and had their cooled air delivered to them through a false floor upon which the computer equipment sat. The RAYDAC electronics were mounted on racks that were two feet wide and eight feet tall. These racks had plumbing along each side that carried liquefied Freon gas, to carry off the heat from each rack to be exchanged with the outside atmosphere. Each rack had studs screwed into it on each side that accepted aluminum blocks drilled with holes to accommodate heat-generating electronic components, such as vacuum tubes. These blocks carried the heat from the heat-generating components to the Freon in the racks to which they were attached. These details can be seen in Image 2. The upshot of all this was the RAYDAC computer room didn't have a false floor nor was the room plagued with the constant noise of rushing air as were most computer rooms of the time.

The external appearance of Building 50 was unique because it had a tower. This was right above the RAYDAC computer room. It contained the air conditioning equipment that supplied the RAYDAC with liquid Freon and contained the heat exchangers, which disposed of the heat removed from the computer.

The RAYDAC computer room was a 40x40x12-foot box enclosed in a double wall of quarter-inch copper. The two copper walls were about six inches apart and the floor was covered with interlocking rubber tiles. The reason for the copper enclosure was to shield the computer from bursts of radar energy that were frequent, powerful and abundant around the missile range in which we were located. The computer room contained two large doors: one on the north, near the northeast corner of the room; the other on the west, near the southwest corner. The north door was the one visible from the front hall; the west door opened onto a laboratory and workshop area described above.

The RAYDAC did finally arrive and we did finally move into Building 50. Our offices were in the large bay at the east end of the building. The bay was filled with perhaps 16 desks. Two enclosed offices opened off of the south wall of the bay. The one that shared the windows on the east end of the building with the bay in which we sat was for our manager and the other was unoccupied. A secretary's desk and some file cabinets sat outside the manager's office. Our manager turned out to be Lt. Commander J. C. Aller, a Naval Officer who knew very little about computers. Hence, the RAYDAC and its associated needs had had a building built for it but the people who were to provide it with its life's blood—programs—were still housed in what was referred to as a "bull pen". However, we knew of no other arrangement, so we proceeded ahead happily.

I remember when we found out who our boss was going to be; the big topic of conversation was how to address him. Finally, Jim Tupac asked him how we should handle this important issue and he said, "Just call me Jim". He turned out to be a really nice person and we all liked him. By now, I must explain that "all" meant those of us who had been in the Waltham programming class less Don Dufford, who went on to other endeavors, Chuck Aronson, who left to attend graduate school and the NSA person. Also, a few new people who had not gone to Waltham joined our group: George Kendrick, who along with his wife Barbara had just graduated from the University of Kentucky and Walt Soden and another man named Lloyd Hines. Walt Soden and Lloyd Hines either were not selected to go to Waltham or elected not to go. So the whole staff

consisted of George Kendrick, Norm Potter, Jim Tupac, Bob Causey, Walt Soden, Lloyd Hines, myself and our boss, Jim Aller. Shortly after we moved into Building 50 Jim Aller hired a secretary whose name I don't remember.

After the RAYDAC arrived and was reassembled in the waiting computer room and PPU room, another major hurdle needed to be crossed before it would be ours to use as we saw fit—the machine had to pass its acceptance tests. This involved execution of extensive programs (prepared and debugged at the Waltham plant) in the presence of a host of critical dignitaries. While we waited for this momentous occasion, we busied ourselves programming utilities (see Appendix K—Utility Programs) we knew we would need when the machine was turned over to us.

It would have been 1952 before we seriously got down to RAYDAC programming. First, we all worked on writing a better program loader than the one supplied by Raytheon. It was like a contest—each person trying to make a faster and/or smaller version. We finally arrived at several loaders both faster and smaller than the starting version. The other set of programs we knew we would need were those for converting between binary and BCD and binary and six-bit numeric code. We used the same approach on these with the same results.

I don't remember any other programs of significance we worked on prior to acceptance tests, but one memorable event did occur. Our old friend Don Dufford had made contact with a man from England who worked in the EDSAC Computation Laboratory. The EDSAC was one of the premier early computers and the group at the laboratory had performed some excellent programming including the writing of (as far as I know) the world's first assembly program. (More about this below). We had all seen and read the book they had written about their assembler and were greatly impressed.

This English visitor was in the U. S. (I think at INA) and paid us a call. We were all expecting to talk shop with him, but instead he regaled us with the story of the Lady Lovelace who was the daughter of Lord Byron and a protégé of Charles Babbage the creator of the world's first computer (so some people say). Lady Lovelace, whose given name was Ada, was said to be the world's first programmer. His presentation was entirely fascinating and we all enjoyed it and had reason to remember it in the years to come when Ada's name was given to a very famous programming language touted for use as an international standard. [A][13]

In due course, the RAYDAC acceptance tests were successfully run and the machine was turned over to the Navy for use. The only item of note in that connection is one person on the acceptance committee. He seemed determined to find cause for not accepting the machine. He created no end of troubles during the tests, being an obstructionist all the way. I don't remember his name, but he was to turn up later as a Vice President of Honeywell.

At long last, we had the opportunity we had been waiting for—to run our own programs on the real RAYDAC hardware. What a shock! Every programmer has, no doubt, experienced it: the realization that a program must be PERFECT in order to run correctly and nearly perfect in order to run at all. We learned to debug—to strive to find every single bit in our programs that was not correct. This is a difficult process that no one really enjoys. However, we survived it and lived with it forever more.

The step-by-step process we went through to get a program in operation was as follows:

1. The programmer wrote the code for the program to be created on two forms: a programming form that described the instructions and a data form that described the data. The code on these two forms constituted the program. These were both written in octal and the octal address into which each word was to be loaded for execution was assigned as the programming proceeded.
2. After making the hand-written code as perfect as possible, it was taken to the PPU Operator, who copied the hand-written code to punched paper tape with a paper hard copy using a teletype machine with tape punch. The hard copy was then returned to the programmer for inspection.
3. If the paper tape represented what the programmer wanted, he or she would go to step 4, else he or she would return it to the PPU Operator for corrections.
4. The PPU Operator would process the paper tape through the Problem Preparation Unit (PPU), which would read the paper tape and transfer its contents to RAYDAC magnetic tape. The tape on its reel would be returned to the programmer.
5. The programmer would schedule a block of machine time to test the program.
6. When the machine time came up, the programmer would go to the computer room with the magnetic tape and hard copy. When the programmer gained access to the machine, the tape reel would be mounted on the tape handler designated as number 2; a tape containing the program loader would be mounted on the handler designated as number 1 and any other tapes needed

would be mounted on the other two handlers. At this point the test could have begun.

7. The programmer would set the start selector to 1 (for tape handler 1) and press the Start Button on the console.
8. This would cause the first block of 32 words, which contained the loader, to be read into the memory buffer associated with tape handler 1 and would instruct the computer to begin executing instructions beginning with the first word in the buffer. Having done this, the instructions of the loader would load the instructions of the program that (by convention) were on the tape on handler 2 into main memory and would command the computer to begin executing the instructions just loaded.
9. When this point was reached, the programmer had in mind a certain sequence of events that would transpire as the program executed with perfection. What usually happened was something entirely different, which often involved the display of some sort of error condition. In the unlikely event that the program ran as expected, the process jumps to step 12 otherwise it continues at step 10.
10. When an error was detected, it was the programmer's responsibility to determine if the error was caused by the program or by a malfunction of the computer. If a malfunction had occurred, an engineer would be called and the programmer would restart the process at step 8 after corrective action had been taken. Otherwise, he would use the CRT display on the console to look at various words in memory in binary to try to determine why his program caused the error to occur. Having determined this, the programmer would relinquish the machine to the next user.
11. The programmer would now return to his or her desk and find a way to repair the error in the program. Having done so, he or she would correct the hand-written version of the program and return it to the PPU Operator for correction on magnetic tape. This would return the process to step 2.
12. At this point there was some hope that the program was working correctly, however this hypothesis would need to be checked further. If it ran successfully, it would have produced some output, most likely on one or more magnetic tape handlers, so the programmer proceeded to step 13.
13. The programmer delivered the magnetic tape(s) produced to the PPU Operator who mounted it(them) on the output-printing unit and converted the results on magnetic tape to hard copy on a Teletype printer.
14. The programmer then took all of the results from the "successful" run to his or her desk and analyzed the results in detail. There was still a

great likelihood that some error would be found, in which case the programmer would return to step 11. Finally, everything would be correct including the final output answers and the job would be complete.

Of course, at this time, we had no programming aids of any kind. We had no assembler and no assembly language and no debugging aids whatsoever. We had to write our programs in octal and perform octal hand calculations to help determine if the programs were working correctly.

We soon became dissatisfied with writing programs in octal and sought better ways of getting our jobs done. The assembly program produced by the EDSAC group inspired us, but it seemed like a huge challenge for us to undertake, so the group considered various suggestions. After much discussion and debate the choices came down to two: a program to create complete programs from a collection of subprograms written with relative addresses and a modification of the EDSAC assembler. I had introduced the first choice and Bob Causey the second. The selection of which one was to be implemented was left to our boss, Jim Aller.

He chose in favor of my simplified version. I suspect he did so because he had very limited discretionary funds available. I will admit I was very positively impressed by Bob Causey's proposal and would not have been greatly disappointed if the choice had gone to it. But as it turned out, this was one of the first, if not the first real programming assignment I received. A brief description of the result is contained in Appendix F—The RAYDAC Assembly Program. (I would discover some time later that what I had created would more appropriately be referred to as a relocatable loader, not an assembly program.)

The other major contribution to our programming productivity came to us as if "out of the blue". Any mathematical function (such as a logarithm or a sine or a cosine) used in a calculation had to be derived by use of a numerical approximation. The determination of the exact algorithm to be used to approximate each function to a desired degree of precision is a difficult problem in itself. Those of us at Point Mugu had no particular background or experience to guide us in dealing with such problems.

Then we became aware of the algorithms published by Cecil Hastings of the Rand Corporation. These were published in loose-leaf form, one sheet per function and were distributed for the asking by Rand. This was like a programmer's cookbook. If you needed to program the calculation of a function to a given degree of precision, you could look up the Hastings

sheet and it would tell you exactly how to do the necessary approximation. We were greatly indebted to him as I suspect were most programmers throughout the country. This was one of the really valuable contributions to programming made in the early days of the profession.

At this time, Northrup Aeronautical Institute personnel were being hired under contract to perform manual data reduction tasks at Point Mugu. A freeze on civil service employment was making this advantageous. This suggested that a similar contract could be let to obtain personnel for the operation, programming and maintenance of the RAYDAC. In due course, that is what happened. The contract was with Computer Control Company of Framingham, Massachusetts—a brand new company composed of former Raytheon employees, mainly the engineers that had built the RAYDAC.

Computer Control Company

Before the new contractor took charge of RAYDAC, some of the group that had been trained to program for it decided they had had enough. Norm Potter was the first to leave. He went to Douglas Aircraft in the Los Angeles Area. They were paying high salaries and found his experience very attractive. Then Jim Tupac accepted an offer from The Rand Corporation. He, too, was pleased with the opportunity the move provided. I'm sure they were both excellent employees in their new environments. They had always been valuable contributors at Point Mugu.

After the decision had been made to contract the operation and maintenance of RAYDAC to Computer Control Company, all of us still present who had previously worked on the machine were offered employment with the contractor. We all accepted except Lloyd Hines, who stayed with civil service and who seemed only marginally interested anyway. The transfer to the new employer was effective in December 1953. I was given number 16 on the company pay roll. The first few numbers on the payroll went to the President of the Company, Louis Fein, and his engineers that transferred out from the east.

I don't remember the names of any of the engineers Lou brought with him, but I got to know him fairly well. He had a PhD in electronic engineering and was looked up to by everyone in the company. He had been the principal designer of the magnetic tape handlers of the RAYDAC. Lou had a pleasant personality and was generally well liked.

While Jim Aller had been our leader, he had delegated to me the keeping of records in regard to what jobs we had, who the customers were and who was working on which job. Hence, it was natural for Lou to ask me to fill him in on this information when he arrived. This was not a big deal because not many jobs or customers existed then. Lou also suggested a better way to keep track of such things than the informal notes I had kept. He introduced me to Gantt Charts. These permitted us to keep the same information in a much more concise form and also provided for visibility of the expected future progress of each job. Other than this change, I did not notice much change in my day-to-day activities, nor, I think, did any of the others of us that made the change of employer.

However, our personnel picture changed in a big way. Lou, of course, replaced Jim Aller in the manager's office and Lou hired an accountant-type person named Bill Murphy (or something like that) who shared the office with him. Jim Aller's secretary went with him and was replaced by a young, blond woman named Doris Hermanson. He also hired a dear middle-aged (she looked old to me at the time) lady to run the equipment in the PPU Room. Her name was Christine and everyone liked her. She did her job well and seemed to want to be everyone's mother.

Our programming staff also began to grow. Lou hired Joseph Weizenbaum who had just finished his PhD at Wayne State University. In addition to having a more elegant degree than the rest of us, Joe was a very bright person and a good programmer.

Lou also hired Kay and Frank Stockmal—a wife and husband team. They both had previous programming experience, I believe on a UNIVAC I.

Bruce Blasdell joined us at some point. He was about Christine's age and at some point had gained some management experience. Bruce seemed to feel he had a special role to play in advising me, as the acting supervisor of the programming group. That was okay with me. I needed all the help I could get.

Jack Anchagno (I'm sure of the name; not of the spelling.) was a local boy that joined us. His family had been the recipients of a Spanish land grant out east of Ventura towards Ojai, California during earlier generations. Jack had lived in the area all his life. He was a really nice person and everyone liked him. He had contracted polio as a child and had the use of only one arm. John Hanson, who was a very sharp person with a strong math background, was one of the last persons I remember joining the programming group.

I don't remember the names of all the engineers we came in contact with. However two deserve mention: Roger Sisson and Dick Canning. Roger was a young electronics engineer with a brand new Master's Degree from MIT and Dick was an electronics engineer that lived in the Point Mugu area in the hills above Camarillo. They went on to form Canning Sisson and Associates, a well-known consulting firm, and several other ventures that were influential in the industry for many years to come.

One Navy Seaman had been assigned to work on the RAYDAC in addition to these engineers. He was John Haanstra and he had a Master's Degree from MIT in electronics engineering and had been

drafted into the Navy during the Korean Conflict. John was exceptionally bright, and everyone liked him. His name will reoccur within this story.

Finally, Computer Control hired another person to help Christine in the PPU Room. She was Sandy Dennin, the very pretty wife of a navy pilot on the base. I'm not sure how much help she was to Christine, but I do know she spent lots of time in the computer room and the adjoining laboratory frequented by our engineers, and they did not complain about her presence.

I have trouble remembering much about the jobs we had on the RAYDAC. I do remember a few, mostly because they were more spectacular than others or because I had a more personal involvement with them. We called one the *Missile Matricide Study*. This was a program that simulated the firing of air-to-air missiles from an aircraft and computed the probability the missile would miss its target and lock onto the aircraft that had fired it. I don't remember the details of the problem, but I always liked the name.

Walt Soden worked on a similar program related to range safety. This program determined the safety of the ships and boats traveling through the ocean within the test range. The problem was to determine the probability that one of these vessels would be struck accidentally by one of our missiles. The result was that the probability was negligibly small. Hence, clearing the range before a launch was not required.

At some point, Lou Fein asked me to write a program to keep track of the company's checking account. I did this with no difficulty and described my resulting program to one of Lou's friends during a visit to INA. His friend criticized my program for not using a sort program to order the information about checks by check number. This was, no doubt, a valid criticism, but we didn't have a sort program and it was not my assignment to create one. In any case, the program did what Lou wanted, so I was satisfied. He later asked me to write a payroll program, which I did, but it was kind of a bust because we had no way to print checks. Hence, all it did was create some accounting inputs based upon the payroll expenses.

At the time we were creating these programs, there was an IBM-CPC (an automatic calculator, not a computer) in use down the hall. CPC stands for Card Programmed Calculator and that is what it was. The program that it executed was stored on cards and some of the data it worked on were stored in units called "Iceboxes" because they looked like little short, gray refrigerators. Each icebox stored about thirty words, and I think the CPC at Point Mugu had three of four of them. A CPC consisted of a set of

separate units cabled together. The one in our building had a card reader, a calculator called a 604, a parallel printer and some number of iceboxes. Typically a set of auxiliary equipment went with the CPC such as a card sorter, a reproducing cardpunch, one or more keypunch machines and perhaps a card collator. All of this equipment had to be leased from IBM—it could not be purchased.

A man named Chuck Wimberley ran the IBM facility. I don't remember whether he was a Northrup contract employee or a civil servant. In any case, he was very loyal to his job and what and how it was being done. He constantly argued that the RAYDAC was a big waste and that he could do anything we were doing faster and cheaper. In retrospect, I think he was probably right because he had access to programming aids, tools and techniques developed by and shared among all of the CPC customers in the Greater Los Angeles Ares to use to leverage his small programming tasks. We had to develop everything we used from scratch, so we were always at a disadvantage.

Reliability was our other problem. Ours was very poor and his was very good. The RAYDAC was a self-checking machine, which meant it fully checked everything it did and if it found any error, it would stop. It was nice that it didn't create errors at electronic speeds, but it was not so nice to be stopping for errors all the time. In Wimberley's case, the machine continued with the (correct or incorrect) assumption the results were correct. This made his life much easier than ours though his customers were not necessarily being better served.

Because the running of the CPC was pretty straight forward, it was possible for Wimberley to utilize semi-skilled operators to run his programs. The need to analyze each error and go through a careful and often complex procedure to restart the machine, made it necessary for a programmer (usually the one who created the program) to act as the RAYDAC machine operator. This was inconvenient and a serious drain on our already skimpy resources. So, all in all, I think Wimberley had a pretty good argument.

We did everything imaginable to improve RAYDAC reliability. At one point, closing the door to the computer room was found to cause

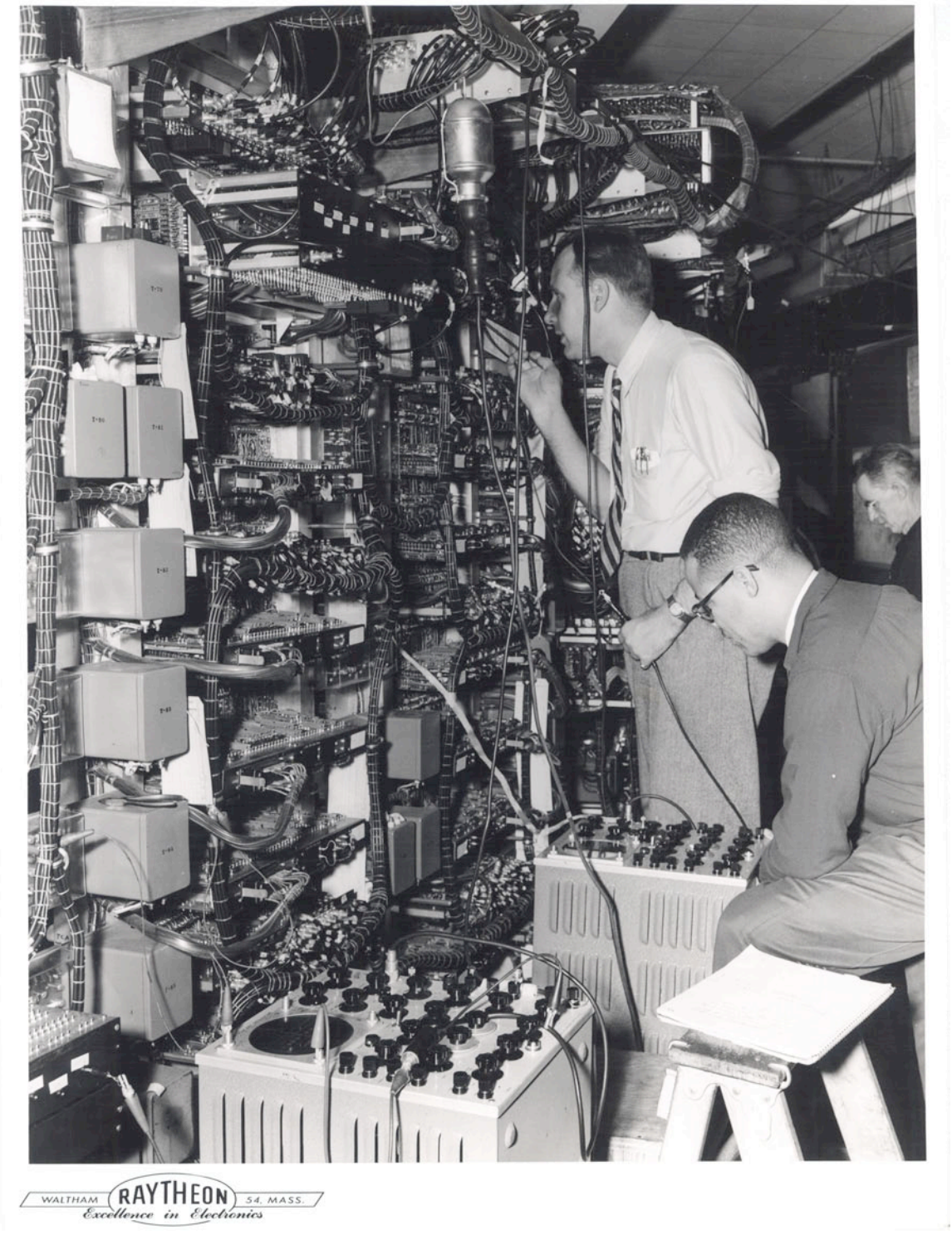

Image 3. The Engineers' View of RAYDAC

acoustical shock waves to be propagated into the mercury of the delay-line memory. These shock waves were causing memory errors. The memory was in a separate rack located about 10 feet inside the back door of the computer room. Because the doors were quite heavy, it was customary to close them with a bang; hence the problem. So everyone was cautioned to close the door easily or use the other door if possible. This helped some, but errors continued to occur too frequently.

At one point, the thousands of solder joints that held the circuits of the machine together were suspected of being faulty. Hence, all of us including the programming staff were brought in after hours and given dental mirrors to observe each and every solder joint. We were all given instructions on identifying faulty or cold solder joints. The scene we confronted in performing this job is shown in Image 3. Some bad solder joints were found, and improvement in reliability may have occurred, but the bottom line was the RAYDAC was NOT a reliable machine.

In August of 1954, a man named Don Cambellic reported for work as the supervisor of the RAYDAC programming staff. This was kind of a shock to me because I had performed that function since the group was formed and, though I was never given the formal title, I knew of no reason for me to be replaced. I don't remember how Lou Fein handled the thing, but however it was, it didn't set well with me. Furthermore, Cambellic was not a person I particularly liked or admired. He was given a large private office and I did what was expected of me—tried to teach him what we were doing as an organization and as a group of individuals.

The arrival of Cambellic had the advantage of permitting me to do some things I would not otherwise have had the chance to do. Chief among these was the opportunity to work with Joe Weizenbaum on the solution of a set of 65 simultaneous differential equations we had gotten from the Chance-Vaught Company. They described the flying characteristics of one of their missiles. Differential equations are mathematical expressions that describe how each variable in a system changes as a result of a small change in some other variable. It is not, in general, possible to obtain a closed-form solution to an arbitrary set of simultaneous differential equations, but it is possible using the differential equations to examine how a system will respond to a change in certain of its variables when changes are applied to other variables of the system. This is done using numerical integration.

Joe had been working on various methods of numerical integration and he and I discussed his results and challenges as he proceeded. This permitted me the opportunity to understand what he was doing and to

act as a sounding board for him as he achieved results. The Chance-Vaught problem came along when Joe had implemented what we concluded was the best method of numerical integration for our purposes. Hence, he implemented the solution of that problem and started running cases.

However, the cases took a VERY long time to run. We usually scheduled the daytime hours for program debugging and tried to do whatever production runs we had at night. So Joe was stuck with working nights, but progress was slower than anyone wanted. So I volunteered to work a second night shift. We took turns working swing and graveyard shifts and we gradually worked through all the required cases.

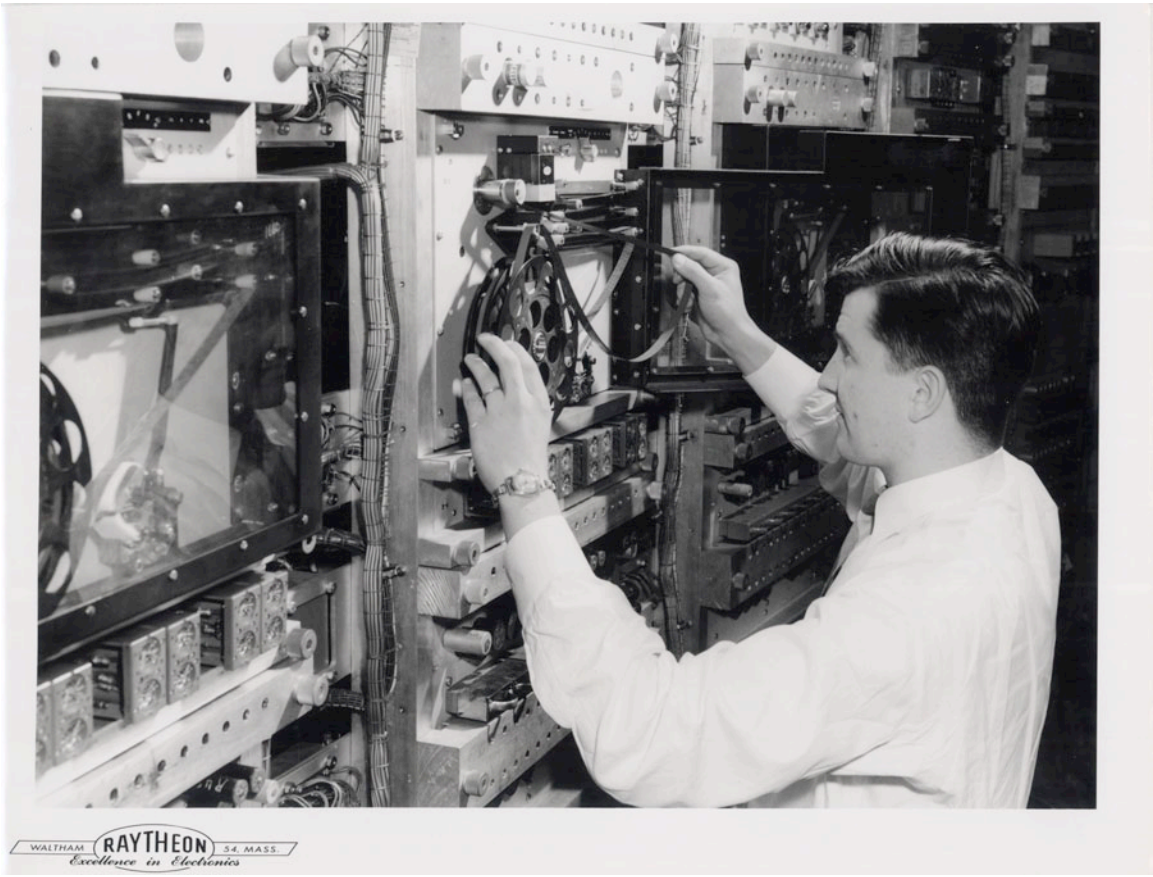

Image 4. RAYDAC Magnetic Tape Handler

Even with two shifts per day, the progress was much slower than anyone wanted. As a result, Joe came up with an improvement that was a credit to him and his ingenuity. To explain what it was, I should explain that Joe had implemented the Runge-Kutta integrator by placing the program on magnetic tape in 32-word blocks each of which would issue a command to read in its successor after it had been exhausted. In this way, most of

the main memory of the computer was available for storage of the problem data and this speeded up the calculations. Each block of code was read into the 32-word delay line associated with the magnetic tape handler upon which the tape containing the program instructions was mounted. The instructions were executed from where they were stored in the delay line without any need to be moved into the main memory. However, after the computation sequence was completed, the tape had to be repositioned to the beginning. This repositioning took about the same amount of time as the whole calculation, so if it could be eliminated, we would have doubled the computation speed.

The idea dawned on Joe that if we had a tape loop instead of a tape reel, we could adjust the length of the loop so that the next block after the last block of the loop would be the first block of the next rotation of the loop. (The general configuration of a RAYDAC Magnetic Tape Handler is shown in Image 4.) To implement this idea, he first got some scrap magnetic tape and made a loop of the right length. Then he got an idler from one of the engineers. (An idler is a little nylon pulley over which the tape normally passes while it is being handled by the tape drive.) He used a paper clip as an axle for the idler, connected the paper clip to a rubber band, then connected the other end of the rubber band with a paper clip to an overhead fluorescent light fixture. The gadget worked by running the tape handler with its door open to permit Joe's contraption to reach the light fixture. One end of the loop passed over the read/write head of the tape handler. The idler assembly supported the other end. The fluorescent light fixture, in turn, supported the idler assembly. After the blocks of code had been written to the tape loop, he tried it out and it worked.

The expected doubling of the calculation speed took place and we were off and running, well, at least walking a little faster. The only problem was the tension provided by the rubber bands varied from time to time, presumably due to changes in temperature and humidity. So the first challenge of getting the program to work was to select the rubber band, which would work for that night. Whenever we changed shifts, we would have a little powwow about which rubber bands had worked and which ones had failed in case we had to make adjustments during the night.

In spite of good experiences like this one it was difficult to remain positive about the RAYDAC. I decided that I might like to make a change myself, but I didn't want to be part of the Aircraft Industry and I didn't want to live in the LA area. So I tuned up my resume and sent it out to various potential employers. The ones that responded and offered me interviews were Convair Aircraft in San Diego and General Electric at Richland,

Washington. The Convair offer was very attractive even though it was in the industry I had vowed not to join. However, they were working on the Atlas ICBM and they had a computer that looked very attractive to me (I think it was an Electronic Research Associates 1103). But after the trip to the Hanford Atomic Products Operation at Richland, I was hooked. Hanford had an IBM-702. IBM had introduced the 702 in 1954, but when I arrived in Richland for an interview, the Hanford organization already had theirs installed. The general environment and the interview with Harry Tellier, the manager of the computer center, made the decision for me. I wanted to go to work for General Electric.

I cannot leave the discussion of my experiences at Point Mugu without mentioning some of the background we were experiencing. It was at the height of the McCarthy era. Joe McCarthy was the senator that was getting everybody stirred up about our government and society being infiltrated with communists. All of us at Point Mugu had security clearances and often worked on classified projects and dealt with classified materials, so the normal paranoia associated with security was whipped into a fever pitch by the senator's ramblings.

This tension came too close for comfort in two instances: one was the accusation that a female employee who many of us knew was a communist; the other was the lock out from the base of an employee Lou Fein had hired because of a security problem. The female employee was one of those people who had clear and distinct political ideas and liked to express them. I remember she used to hold forth about the evils of Chang Kai Chek and the Kuomintang Party. This seemed strange to me because, as far as I was concerned, Chang was a hero of WWII to be honored and respected. (That error was clarified for me by reading I did later in life.) My wife and I had attended a cocktail party at her home. In any case, she was suddenly on administrative leave because of a security problem. Then some FBI agents showed up and interviewed various people, including me. She was gone for a couple of months and then she returned.

In the other instance, Lou Fein had hired a man from the east and moved him out to California. He was ready to report, but base security would not allow him to enter. He claimed he was not accused of any specific activity or association but was simply denied entry. He showed up every few days and someone would go out and talk to him. I was chosen to meet with him on one or two occasions. He seemed like a regular guy, but he finally gave up and left. This was the unhealthy climate in which we lived for longer than we should have.

Hanford—Getting Started

Our family had grown while we were at point Mugu. By the time we moved to the Richland, Washington area, we had two daughters and one son. We arrived in time for me to report to work on September 1, 1955.

The Hanford Atomic Products Operation had been started during World War II for the production of Plutonium for use in atomic bombs. The Columbia River makes a big loop to the east just north of Kennewick and five nuclear reactors, a chemical separation plant and a new city, Richland, were built where before nothing but desert and a tiny village called Hanford had existed. Richland contained the offices for the whole operation and the living quarters for many of the employees. For security reasons, the office areas were off-limits for non-employees, but Richland was an almost regular civilian community. All of the rest of the reservation was off-limits except for employees with appropriate security clearances.

Actually, some aspects of Richland were not at all regular. For one thing, most of the houses in the community belonged to the government. The assortment of house designs was limited; each one was given a letter. So when you met someone, you might ask, "Do you live in an A-house or a C-house?" Having gotten the answer you would immediately know the floor plan of the house because all of the houses of a given letter were identical. So a certain military-like uniformity existed in the town, but the residents generally exhibited pride of ownership that led to a good appearance.

The other unusual thing was that many of the employees worked in an environment where exposure to radiation was a danger. For this reason, many employees wore film badges when they were on the job and exercised other precautions with which I am not particularly familiar. The most unusual thing was that every employee, including myself, was given a urinalysis on a periodic basis to check for any signs of excessive radiation exposure. To facilitate this, a truck would drive around the area distributing metal cases containing two large flasks. When the flasks arrived at your house, you were expected to fill them within a time period, I think it was a week, at which time the truck would return and pick up the case containing the now filled flasks. All of the residents knew when each of their neighbors was going to have a urinalysis because the metal cases were very unique and recognizable on front door steps.

DuPont built the Hanford reservation during World War II. They ran it for a number of years and then General Electric became the prime contractor to the Atomic Energy Commission. The big attraction of the area was its remoteness and the large supply of cold river water that was supplied to cool the reactors. A pear orchard had been planted near the village of Hanford at one time, but had been left to die away. Other than that and the production facilities, the area was a barren desert. It got quite hot in the summer and cold in the winter and we often had wind throughout the year. The old-timers spoke of "termination winds" that were the normal winds, which blew regularly. However, before the town built up and became vegetated, the winds were always accompanied by dust and sand storms. This prompted many employees to want to terminate and apparently many did just that during these events.

The main office complex was in the middle of Richland, in the business district such as it was. The office complex consisted of a thirty or forty-acre fenced-in area with two entrances: a north gate for pedestrians and vehicles and a southeast entrance through two different pedestrian doors of the main administrative building, the 703 building. The administrative building was quite large and was really a series of interconnected wooden barracks-type buildings typical of World War II construction. In addition to the administrative building, the compound contained many large warehouses and shops. The computer center was located in one of these warehouse-type buildings, the 713 building.

Harry Tellier, the boss of the computer center, often said, "The great advantage of our building is no one else wants it". It was, indeed, humble, but I must say I rarely noticed its limitations while carrying out my duties. The front third of the building consisted of offices in the form of cubicles containing two to three people each. A total of about twenty of these were arrayed on each side of a U-shaped hall. This was a pleasant change from the bullpen arrangement at Point Mugu.

The two-to-three-persons-per-cubicle rule was violated in some instances: one supervisor and his secretary occupied one of them and Anna Mae Nielson occupied another. Anna Mae was a long-time veteran of Hanford and kept us supplied with the supplies and services we needed. She had an office of her own chock full of things we all needed and we were all glad for her to have the space because she always knew how to get things and get things done.

The open end of the U of the U-shaped hallway opened onto another hallway that led into the next third of the building. The outside of this connecting hallway contained one-man offices for supervisors and

consultants, a conference room and Harry Tellier's office including the space for his secretary. The far end of the connecting hallway led to the second third of the building. Most of this middle part of the building contained punched card equipment gradually becoming obsolete and being replaced by the IBM-702 that occupied the third part of the building. The remainder of the middle part of the building contained the Control organization whose function shall be described below.

The walls of the whole building were painted a light green. Some one had done a study that showed light green was the best choice for worker morale. Except for the false floor in the computer room that accommodated the air conditioning needs of the 702, all of the floors were concrete. Evaporative coolers on the roof handled the space cooling and overhead space heaters provided heat when it was needed. Once again the 702-area was the exception. It had 55 tons of refrigeration at the far end of the building. It was very nice to have an excuse to go to the computer room on a hot, humid day.

When I arrived, the organization reporting to Harry consisted of four units: Computer Operations, Application Control, Business Programming and Scientific Programming. In addition, he had a consultant reporting to him. His direct reports were Clarence Poland, Computer Operations; Pop Vought, Control; Chuck Thompson, consultant; Bill McGee, Scientific Programming; and. Kendall Wright, Business Programming. I'm unsure of the academic backgrounds of any of these people except I know that Bill McGee has a master's degree in Physics from Columbia.

Clarence had a very sharp mind. He was a wiz of a computer operator and could imagine sequences of details better than anyone I have known.

Pop Vought was the eldest of the staff, probably approaching sixty. I don't know what his given name was. Everyone just called him Pop. He had a reputation for being a hardnosed supervisor, but was, in fact, a very kindly gentleman. Today, we would have called his function Quality Control, but I never heard anyone refer to it as anything but Control. It was his responsibility to schedule all of the financial work, to make sure it was run and to make general checks of the results to ensure that they made sense. He was the real world, practical replacement for the self-checking of the RAYDAC. I had never before been exposed to a function such as his, but I soon became a real fan of the idea.

Chuck Thompson was the son of a Seattle-area auto dealership owner. He always made a good appearance and was very pleasant in his social

mannerisms. He was very kind to me while we were both at Hanford. We were about the same age.

Bill McGee was also about my age. He was a "clean-desk" man—always cleaning up and filing one task before starting another. His father was a judge in a county east of Stockton, and he was raised in the California Mother Lode country. He was usually reserved and dignified.

Kendall Wright, like Clarence was probably in his thirties. He was a Mormon and at some future time he accepted a job with the Mormon Church placing their genealogical files on computers. Kendall, Clarence and Harry all had extensive punched card backgrounds, which had a profound influence on our work.

Finally, Harry was perhaps in his early forties. He was the person that made everything work harmoniously within the computer center and controlled our relationships with outside organizations to our benefit. He had worked for the State of Washington at Olympia, for the Kaiser Steel Works at Fontana and others. I don't think he had a degree, but he had very distinct ideas about how computers should and should not be used. He was dead set against the decentralized use of computers—believing this led only to duplication of effort and disagreements over results from different sources. He also felt quality control suffered in the absence of centralization. Harry's view of the ideal system was a single powerful computer upon which all data for the organization (company, country, factory, etc.) was stored and available for reporting and processing. He was a great manager of people and knew how to apply his ideas without any heavy-handed tactics.

When I first arrived, I was put in an office with Fred Gruenberger. We were in the office right next to Anna Mae Nielsen who was in the office right next to the connecting hallway. Fred had a Master's Degree in Mathematics from the University of Wisconsin at Madison and was very interested in many things including games, movies and Number Theory. He also loved to write [14]. He and his wife published one of the very early news magazines about computers. It was called *Computing News* and it came out monthly. It contained articles by anyone he could get to write one and other news clips about events around the industry he collected from a vast network of friends. Fred and his wife made the magazine look bigger and more imposing than it actually was by each having many names and many functions. Their masthead looked as if the company had twenty or thirty employees when, in fact, it had only two. They had a circulation of three or four hundred in the U. S. and up to twelve foreign countries.

At one point, Fred cajoled me into writing an article for Computing News. It was on flow diagrams or flow charts as we came to call them. Fred loved it and built an entire issue around it. To help fill it out, he included a short humorous piece by Jackson Granholm. Jackson was a friend of Fred's who lived in the Seattle area—I think he worked for Boeing--and he had this act in which he was a German Professor named Dr. Stoerben von Hunger. In this persona, he would carry on in pigeon German. As a sample, "Wenn man, jemand Nincombpoopen beim cottenpicken Hanz, die ganze library Tape erased hat, es gibt no verdammte use!"

Dan McCracken had worked at Hanford until shortly before my arrival. He wrote a book [15] on programming in the late fifties and included my Computing News article (including all of its errors) as Chapter 6.

Like many of our employees, Fred had extensive experience with punched card equipment. He loved to do his own keypunching, sort his own card decks and be his own computer operator. This was great for me because I was short on knowledge about card technology. When it came time for me to wire my first control panel, Fred was there, ready and eager to be my teacher. On the other hand, he was eager to learn from me about my experience with the RAYDAC. We soon became fast friends—a friendship I always treasured.

After just a short time, Sarah Saco joined us in our cubicle. (Coincidentally, Sarah and her husband and little son had moved into the house we temporarily rented from the company in Richland.) We were a very compatible trio. Sarah had a degree (I don't remember from what school) and was a pretty, blond girl in her twenties. She was very bright and a good worker; her greatest interest seemed to be her husband and her little boy Gus. She was a good office mate—always good-natured and mostly down to business. To Fred's delight, she was always willing to join in at lunchtime playing the weird games he liked to play. I don't know where he got them, but he knew a whole galaxy of variants on well-known games from three-dimensional chess to bizarre card games.

The IBM-702 had been installed less than a year when I arrived. It had 10,000 characters of Williams Tube Memory, a 60,000 character magnetic drum and about 12 Magnetic Tape Handlers. The computer performed its arithmetic in decimal and the memory addresses were also in decimal. The card reader read about 200 cards per minute and the console output device was an IBM Typewriter instead of the Teletype machine I was accustomed to. I don't believe it was equipped with an on-line printer,

but if it had one, it would have printed 150 lines per minute. Most input and output was done by way of magnetic tape.

Separate peripheral subsystems were required to copy information from cards to magnetic tape (200 cards per minute), lines of print from magnetic tape to printed page (150 lines per minute) or card images from magnetic tape to punched cards (100 cards per minute). These subsystems were a major step forward from the puny Teletype output and PPU input we had on RAYDAC. (Chuck Wimberley would have cheered!)

The reliability of the machine was not spectacular, but it was worlds better than RAYDAC. The Williams Tubes failed fairly often (a few times a shift) and magnetic tape errors were very frequent. Much time was spent recovering from memory failures and adjusting gains on tape handlers while a tape block was reread time after time trying to glean its information in an error-free form.

Sometimes, the engineers would do what they called "develop the tape" to try to see on which track the unit was failing or to see if the tape was dirty. This process consisted of removing the tape from the handler and swabbing its magnetic surface in the failing region with a fluid containing iron powder. The iron would collect around the magnetic spots on the tape rendering them visible. The engineer could then read the binary encoded information and determine which character or track, if any, was failing the parity check. Dirt was often the reason for a failure even though it was standard procedure for everyone to wear white lab coats in the computer room and to maintain very strict cleanliness.

The greatest failing of the 702 was lack of adequate software. IBM supplied a simple assembly program and a sort routine. That was it! The assembly program was comparable to that of the EDSAC I had been introduced to at Point Mugu several years previously. While it was certainly much better than the assembler I created for RAYDAC, it was not then state-of-the-art and had one extreme shortcoming for use at Hanford—it supplied no capability to perform floating-point arithmetic. Although the majority of the work on the 702 was of the business variety, many scientists and engineers at Hanford wanted to use a computer in connection with their work. Hence, floating point arithmetic and a full library of mathematical subroutines was needed.

The Hanford organization satisfied this need in the form of its own homegrown assembler. It was called SCRIPT and Chuck Thompson and an IBM employee named John Jackson developed it. They had it written and working before the arrival of the 702. It encompassed all of the

features of a conventional assembler of the time plus the needed floating-point arithmetic and a subroutine library.

Getting to Work at Hanford

I was assigned to Bill McGee's Scientific Programming Unit when I first arrived at Hanford. The unit consisted of perhaps 8 of us. I don't remember many of my assignments, but I do remember one. It was the calculation of neutron flux densities in various cross sections of a reactor. I wrote it in SCRIPT and it was pretty straightforward, but I remember trouble getting it debugged thoroughly. The work was done for a physicist at one of the reactors who was a friend of Bill's. He would come around just when I thought everything was working fine and look at the results for a few minutes and then point to a number and say to me, "That doesn't look right." I would then go back to the code and pore over it and look for a mistake. I would always find one or more and then we would go through the process again.

I had enough other assignments that in a short time I became very familiar with 702 programming and how to get around in the Hanford Computer Center. I was soon to find that the programmer at Hanford had a great deal more support provided in getting his job done that had been the norm at Point Mugu. Because the primary medium for supplying raw input to the 702 was cards, we didn't need to deal with paper tapes and special equipment for converting it to magnetic tape. We could supply cards that were read by the computer itself and we even had keypunch operators that, if we wished, would punch our program decks for us including verification for correctness of the punching.

We employed what was called "closed-shop debugging". That meant that a program to be tested would be submitted to the computer room with operating instructions and a computer operator would run the test. For training purposes we all had the opportunity to run our own programs and, of course, when the circumstances warranted one could make arrangements to do their own debugging. In case of any failure during a normal test, the complete state of the computer registers and a memory dump of all the cells of memory would be recorded and printed out for the perusal of the programmer. This information along with any output that was created would be delivered to the programmer after the test.

The procedure a programmer would normally use at Hanford to create a SCRIPT program was as follows:

1. The programmer wrote the program on forms created for the purpose. Each line of the form consisted of a location, an

operation, an operand address and a comment—that is, it was in one-address format with a comment. This was like the RAYDAC instruction form except, of course, the RAYDAC form was in four-address format. It also differed in that the location, the operation, and the operand address were given in more or less symbolic form. That is, the operation was written as some mneumonic triad, such as **ADD** for add, or **SUB** for subtract. The location and operand address (both of which are addresses) were written as eight-digit sequences of the form **xx.yy.zz,** where **xx** is the number of a block of code, **yy** is the number of a line within the block **xx** and **zz** is a sub-line within the line **xx.yy** . This was not entirely symbolic but it was a vast improvement over the use of octal for all instructions and addresses as in the RAYDAC. The sub-line numbers would initially be all zeros.

2. The programmer would send the hand-written program to the keypunch room to be transferred to punched cards and verified. He or she would receive the hand-written program and a deck of cards in return.
3. The programmer would prepare a run request form describing how the program was to be run and submit the program deck and the run request to Computer Operations for execution. He or she would receive the program and any printed or punched results the run created and, if the run was unsuccessful, notations on the run request form by the computer operator who ran the program and a memory dump. If a memory dump was created, it would show the entire contents of memory at the time the program terminated.
4. If the run were successful, the programmer would exit this procedure and enter the celebration procedure. Otherwise, he or she would analyze the operator comments and memory dump to find the reason for the failure.
5. The programmer would construct a modified version of the code excluding the error. At this point, sub-line numbers could be used to insert new lines between previous ones, or previously used line numbers could be abandoned. Either or both of these methods of creating modified code could be used without starting over from scratch.
6. The programmer would create new cards corresponding to lines added to the erroneous code. Because the number of new cards was frequently small, the programmer often punched the cards corresponding to new lines instead of having them punched to shorten the time needed to get a new run submitted. Once the new cards were available he or she would modify the deck containing the program to conform to the corrected code. Printing the contents of the cards on a tabulating machine could produce

a listing of the modified code, if desired. The process would continue at step 3.

Although I was never a great advocate for the use of punched cards, they were a great improvement for use by programmers as compared to the paper-tape machinations we had to employ on the RAYDAC. In addition to being convenient to use as described above, it was possible to sort a deck if it was dropped and shuffled, or make copies of it using a card reproducer or carry out a whole array of other operations that were the specialties of a wide variety of punched-card machines.

Within three months of my arrival, the memory of the 702 was upgraded to 20,000 characters of magnetic core. This not only doubled our memory size, but also increased our reliability. Magnetic cores were a big step forward in memory technology, as anyone who witnessed this transition will testify.

As a result of the new memory on the Hanford 702 we experienced an increase in capacity. However, in spite of this improvement, we were short of machine resources. We were running a four-shift operation, three shifts every day around the clock plus weekends. A few idle spots occurred, but not very many. Debugging was scheduled in the daytime if possible, but the needs of the business often didn't permit this and if the programmer's presence was required, as was sometimes the case, we ended up coming in at odd hours to get some machine time. Only rarely did idle time occur.

In fact, idle time never occurred because my good friend and office mate, Fred Gruenberger, had written a program to calculate prime numbers and was in the process of calculating all of the prime numbers in the 40,000,000-range. He had made a deal with the computer operators that whenever idle time was about to occur, they would start up his program. (Even though this time was usefully utilized, it was nevertheless charged to idle time. This was done with the knowledge and support of IBM and simply permitted the use of time that would otherwise have gone unused.) Fred's program was written so that it would start up from where it had last stopped and continually added the new results to those previously calculated. Why was he doing this, you ask? It was never quite clear to me except Fred had a burning desire to be published and this was an easy way to get his name in print. Apparently, having this large supply of prime numbers was of some possible value to number theorists.

In any case, the scientific users of the computer were being deprived of computer time just like the rest of us. Harry knew they would get a

machine of their own if he didn't provide them with what they needed, and he didn't want that to happen. Hence, to provide for the needs of this group he arranged for an IBM-650 to be ordered. [B]

The IBM-650 at Hanford was to be for the exclusive use of the scientists and engineers and it appeared it would be well able to take care of the need. The only problem was that, like most machines of its day, no easy-to-use programming language existed. Hence, on a day in mid-1956, I was called into Chuck Thompson's office and he showed me the manuals for an assortment of different programming languages he had studied. He wanted me to look them over and recommend one I thought would be appropriate for the use of our scientific customers. The idea was if IBM didn't provide us with the language our customers needed to get their work done, we would provide one for them.

After doing the required study, I found that none of the samples provided what seemed like it would suit our needs. The most promising candidate would have been something similar to Fortran that was a brand new language then and was being implemented for the IBM-704. However, as attractive as that may have been, it didn't exist for the IBM-650. Hence, I suggested a language of our own that would be in part algebraic and in part English-language-like. After some discussion, I was given the task of completing the language and implementing a processor for the 702 that would produce code to run on the 650—in modern parlance, a cross-compiler. By taking this approach, we could utilize the convenient variable word-length capabilities of the 702 to get the compiler implemented quickly and provide a powerful tool for our scientific users to use to their hearts content on the 650.

That was a turning point in my career. I was never again a programmer who wrote application programs, but was always involved in the computer system or what we would come to call (much later) system software. I invented the language, and implemented the cross-compiler. It was not an easy assignment and I spent many late nights in the computer center getting the last persistent bugs out of the code. After the implementation was complete, I wrote the manual and designed a class to teach the scientists how to use it. When the compiler was finished, Chuck insisted it needed a name. His wife came up with the name OMNICODE. I had help from a summer intern during the summer of 1956 and from Ralph Eichenberry of IBM, but I think I did most of this work myself.

As I was about to finish OMNICODE, I was again summoned to Chuck Thompson's office. This time he had an announcement for a conference

to be held at the Franklin Institute in Philadelphia on January 24-5, 1957. He suggested I should prepare a paper for presentation to the conference and see if it would be accepted. I told him I thought it was very nice of him to think of me in that context but I had this problem with a tremor, which got very extreme and out-of-control when I tried to appear before a large gathering. I didn't think I should embarrass myself and/or General Electric Hanford by making a spectacle of myself. He said he was aware of my problem and he thought I should do it anyway—it would help me learn to control the problem and it would be very good for me professionally. Well, he was not a car salesman's son for nothing. I accepted the challenge and did the presentation at the Franklin Institute.

As I look back on that experience, I realize Chuck was very correct and I shall always be grateful to him for having insisted. He also suggested professional help was available at Hanford to get my materials ready in an artful and upscale fashion. I followed up on this suggestion and got the help of an artist to prepare some cartoon-like slides for me that were interesting and eye-catching. All in all it went very well. I got the shakes, as expected, but after a few minutes at the podium, I began to get into the subject and forgot myself and settled down very nicely.

The conference was entitled *Symposium on Automatic Coding*. About 260 people were in attendance. I was very flattered at the banquet held the first night. The people running the conference made a point of having me seated next to Grace Hopper. She was a well-known personality in the industry and was one of the panel moderators. She also had the title Manager, Automatic Coding at Remington Rand Corporation. She was the author of several programming languages starting with the A-0 language, then the A-1 and A-2 and was now working on the B-0 language. These were all English-like languages for UNIVAC machines and were the progenitors of COBOL—a language that became somewhat of an industry standard for business programmers. Grace Hopper later became the first female admiral in the Navy and was very influential in guiding the future of data processing and computing within the military services.

Other people on the program were Richard M. Petersen of General Electric, Louisville, where a UNIVAC was in operation, and who would later transfer to the General Electric Computer Department; Charles Katz who worked for Grace Hopper and would be my first boss at the General Electric Computer Department in Phoenix; Robert W. Bemer whose name I had known from the LA Area and who would work for Honeywell when I did and Alan J. Perlis who was the Director of the Computation Center at Carnegie Institute of Technology and who would become well-known in

artificial intelligence and throughout the industry. Panel moderators in addition to Grace Hopper included John W. Backus of IBM, the developer of FORTRAN; and John W. Carr of the University of Michigan.

I must have finished work on OMNICODE by the end of April 1957 because in May 1957, I became the supervisor of the computer room. I believe Chuck Thompson and Clarence Poland swapped positions at about the time I started the work on OMNICODE—that is, Chuck became the Operations Manager and Clarence became a consultant reporting to Harry. I know I took over the machine room from Chuck and Clarence was still around as a consultant. When I became machine room supervisor, Chuck took Kendall Wright's place and Kendall took a consultant position reporting to Harry.

Supervising the computer room was a whole new experience for me. It involved directing the work of a large crew of non-exempt employees, scheduling their activities, the shifts they would work and appraising their performance. It also involved scheduling the work to be done on the computers and the associated peripheral equipment. Finally, it involved negotiating with IBM to keep the equipment up and running and to provide the IBM engineers adequate machine time to perform their preventive maintenance.

Before I arrived on the scene, Chuck Thompson had installed a set of procedures for the scheduling of work and the retention of files on magnetic tape that were efficient and effective. All I had to do was use them. I found working with the computer operators to be pleasant and learned to have a deep respect for their ingenuity and persistence in dealing with problem situations. By far my greatest challenge was dealing with hardware failures.

The IBM management tried to avoid calling their engineers out in the middle of the night to effect repairs. They wanted to make sure all efforts had been exhausted before the engineers were called. To make certain of this, they insisted all calls to them come from me. As a consequence, I was personally involved with every computer outage that occurred no matter the time of day or night. I was expected to quiz the operators if they had done this or that corrective measure or workaround and make sure all that could be done had been done before I would call the IBM engineer.

I played the game as expected, but it was often very exhausting not only for me but also for my wife. On the other hand, it was inevitable that I would become involved, because when an outage occurred, it meant

work would be behind schedule and many customers would need to be informed and much work would need to be rescheduled.

The problems weren't limited to the computers either. The air conditioning often gave us problems. Failure of an air conditioning unit could shut us down just as completely as a memory failure. And also, the computer failures were not limited to the mainframe—the peripherals could shut us down also. If a card reader died, input from cards could not get to the machine—it would die of starvation. If a printer was down, we could have all of the desired answers on magnetic tape with no way to get them to hard copy. We had a printer that could print at 500 lines per minute, but its reliability was terrible. Fortunately, we had two 150 line per minute printers we could use instead, but all of these problems had to be dealt with.

On the plus side, I had, for the first time, the opportunity to be a member of Harry Tellier's staff. This was the first time I had been a member of a manager's staff and I enjoyed it very much. Not that Harry's staff was like many others. I would learn to my great regret I had been in a rare and unusual environment then and often wished other managers could have half the skill and talent of Harry in carrying out their jobs. But at the time I found the experience stimulating and enjoyable and I, for one, think we did a great job.

I held the machine room supervisory position for only four months. It was a valuable experience for me because I had learned the solutions to a whole new set of problems. I had gained a new respect for the importance of reliability. I had learned how to deal with employees and customers in tense situations. I was involved with checking the billing from IBM, which taught me where we were paying too much for too little service. I gained a sharpened awareness of how programs should display their status to operators so the operators could respond as intended.

While I was gaining all of this valuable experience, various other important developments were taking place that must be explained. SCRIPT was used for most of the programming done by Harry's organization in the early days of the 702. This permitted the installation to get started in its use of the new machine and replace some of the great mass of punched card equipment it had formerly consisted of, but an assembler is not an efficient programming tool and Harry and his staff knew it.

What they wanted was a way to create magnetic tape files to take the place of the huge card files they had previously used. They would call these magnetic tape files "source files". In addition, they wanted the

ability to quickly and efficiently extract, format and report information from these source files and perform processing on their contents when needed. Instead of control panels to define the processing and reporting their applications required, (as they would have done using punched card equipment) they visualized a programmed counterpart that could be created and maintained without the inflexibility and inscrutability from which control panels suffered.

Parameterized sort programs had been in use for several years by this time. These permitted files stored on magnetic tape to be sorted into any desired sequence under control of a set of parameter cards fed to the sort at the beginning of its execution. In this way a user could specify the result desired of the sort program and the program would achieve this result without the user being involved in the programming details. If only they could perform the remainder of their tasks in a similar manner, their lives would have been very much simplified and their customers served in a much better fashion. They would call the programs that would achieve these goals "Generalized Routines" and this is what they developed[16].

The first of these was the 702 Report Generator. The best way to describe it is to use the words of its co-inventor, (along with Chuck Thompson) Ed Roddy. He said [17],

> "During the summer of 1957, I was transferred from 702 Computer Operations to work for Charles E. Thompson as a programmer. About that time, he recognized the need for management and staff to be able to obtain special reports to supplement the reports already being produced for Payroll, Work Order Cost, and other key projects. To accomplish this I would obtain a definition of the required reports and then write the code [in SCRIPT] and produce their reports. This process was repeated again and again over a period of several weeks. This led to a discussion with Chuck to determine if the process could be made more efficient. Several approaches were studied including the use of canned routines to prepare the headings, detail, and total lines but the variable formats would still require special code to be written for each report line. [After] Another look at the information that was being provided for each report, that is a definition of each print line and of the fields to be inserted in the lines, it became apparent that if the format data were input to a program that the code could be generated to produce the report. I was allowed to develop this concept and within 10 weeks (early November 1957) we had a fully operational Report Program Generator."

The Report Program Generator mentioned by Ed Roddy was referred to internally as the RG. It was the first of our generalized routines. It was used to extract reports from source files. Two key inputs to the RG were required for each report to be generated: a description of the records included in the source file—referred to as the source file dictionary--and a description of how the report was to appear, provided by the person requesting the report. The source file dictionary was the first file on the first reel of every source file and was unique for each version of the source file. That is, the format of the source file could be changed from one update to the next and the dictionary at the front of each version of the source file would define the format of the data contained in that version of the source file. Hence, the source file format and content were not frozen in time but could vary with the needs of the business and the wants of the customers.

The appearance of each report was recorded on a form consisting of two parts: a pictorial representation of the appearance of the desired report and a set of expressions specifying the origin of each report entry. The pictorial representation consisted of an example of each line that was to appear in the desired report shown as a sequence of characters. For example, a fixed-point numeric entry might be shown as xx.xxxx where each of the x's (representing a digit) and the "." would be shown on the form in the columns they were to appear in the final report. This example would indicate a six-digit numeric entry with four decimal places. Different codes, in place of the "x" in the example, were used for each possible different type of entry. Each report line was given a label that identified the type of line it was: heading, detail or total. The entries on the second part of the form specified where the data for each report entry was to be obtained. This would often be from some field of the source file, but could also be a literal value or a calculated value involving various fields from the source file or the sum of various values from a lower level line in the report.

The content of each of these forms was punched into cards. The card deck corresponding to each report was referred to as a packet. No programming in the usual sense was involved in creating a packet and only a packet and its corresponding source file was needed to create a report.

When it was time to process a particular source file, all of the report packets available, referring to that source file, were placed in the card reader and the RG would read the packets and the dictionary from the source file. The RG would then generate a program to create the specified reports. During execution of the generated program, all of the

reports would be stacked, one after another, on an output tape with a single pass over the source file. This tape would then be printed to create the final outputs. The format of the source file and the particular selection of reports to be created were variable from one RG execution to the next.

The source files tended to become quite voluminous. For example, I think the Personnel Source File contained 2000 characters of information about each employee. These large source files would be passed over once or twice a day so many customers could have their routine or spontaneous needs satisfied with a reasonably short turnaround time.

The second generalized routine was a modification of the sort program supplied by IBM to use parameters specified in terms of the names of fields defined in the source file dictionary instead of the physical locations of fields in the source file. The reason for this was to make the sort parameters largely invariant to changes in source file format and content. As long as the fields upon which the sort was performed were in the source file dictionary, the sort routine could always locate them even if they changed location from one sort execution to the next. I have been unable to determine who performed the changes that came up with this version of the sort.

The third generalized routine was the file maintenance generator. With this tool we were able to feed data arising from the normal running of the business into the file maintenance program so it could update the appropriate source file to contain the latest values of these data. So, for example, time cards, changes in grade or level, changes in marital status and a plethora of other information that needed to be reflected in the Personnel Source File each week were placed there by the generalized file maintenance program. These changes were needed in order to properly run the Payroll and other Personnel Accounting applications. Having performed these source file updates, most Personnel Accounting reports could be obtained by an appropriate set of RG packets being inserted in the card reader on the next passage of the Personnel Source File following the execution of the file maintenance run that provided the needed data.

The file maintenance generator was also the tool that was used to make changes to the format of a source file and to perform corresponding changes to the source file dictionary. It was possible on a single file maintenance run to change the format of the source file and incorporate any data changes that were then available. Joan Cannon created the original file maintenance generator, but I have been unable contact Joan to gain any further information about its development.

This approach permitted us to provide a wide variety of services to our customers in an efficient and rapid-response manner. However to take maximum advantage of what we had, it was necessary to inform our customers how they might use these new tools. As soon as the first generalized routines were operational, Chuck Thompson, Ed Roddy and George Gillette put together what was referred to as a "dog and pony show" to inform our customers around the Hanford reservation of the new capability and to encourage them to use it.

Chuck had some large displays made on pressed board panels showing the approach being used by the generalized routines. These were hung in the hallways of our building and used in the customer presentations. The customers were introduced to the generalized approach and then asked to create a report of their choosing from a sample source file that had been created. The customers could ask for anything they wanted and the RG forms to express their desire would be created by them during the presentation. These would then be quickly keypunched and taken to the computer room for execution. The resulting report would be presented to them while they were still in attendance.

For a time when it took months to get a modification to an existing report or to cause a new one to be prepared, this was a very spectacular demonstration. Our customers responded as expected and we were off and running. I had no part in any of this development, but it would be critical to some of my later activities.

Ed Roddy and George Gillette developed a second version of the Report Generator [18] in 1958 that included:

- The ability to create reports ordered in different sequences than that of the source file.
- The ability to include calculations utilizing source file data in a report definition.
- The ability to refer in Report Generator calculations to source file data in terms of their source file field designations.

These were capabilities that were omitted from the original version but significantly enhanced the overall capabilities of the programs—often resulting in improved performance by permitting repeated file maintenance runs to be eliminated.

The 709 and SHARE

In September of 1957, I was made a consultant to Harry, and Jim Marshall became the machine room supervisor. I think this occurred when or near to the time Clarence Poland left to accept a job with IBM. Kendall Wright continued as a consultant reporting to Harry and I became his second consultant. Both Kendall and I had the title Representative, Electronic Data Processing. In general, we were expected to amplify Harry's capabilities by taking on specific tasks he would otherwise have needed to perform. Kendall spent most of his time working on new applications with the business customers. My area was the evaluation of equipment and programming systems for potential use at Hanford. Clarence had started the work I inherited and I expanded the job from there.

I believe I got my first private office and secretary with this promotion. The office I got was, in my opinion, the best in the building. It was in the northeast corner and had its own outside door. It was also very spacious compared to my previous quarters. I also inherited a set of partially filled file cabinets that had been Clarence's. They had a good start on data about computer-related equipment that seemed interesting for use at Hanford.

We were constantly running low on machine time because of the high productivity of our programming staff and the pressure from our customers to get more and more work on our equipment. IBM had often applied pressure to convert to a 705, but our studies showed the increased cost of the hardware would not produce a sufficient boost in productivity to justify the upheaval of a conversion. It was at about the time of my change in duties to consultant that IBM announced the 709. Evaluating it was my responsibility. It was a fixed word-length binary machine like the 704, but it had some features supposed to make it useful for business applications. It had the added advantage that it would be a perfect tool for use by our scientists and engineers who were in the process of overloading the 650. I went through all of the calculations and arguments and came up with a recommendation to replace the 702 with a 709.

Other possibilities had to be looked at though they were never serious contenders because they didn't have the programming support we needed and no conceivable means of achieving it. Among these was the ERA 1103A, an upgrade of the machine Convair had when I interviewed with them. Its most attractive feature was it had an interrupt

[19] capability that could be (and in the future would be) used to improve system efficiency.

Some other alternatives were the supercomputers--probably too expensive for us in addition to their deficit in program support. The first was the IBM-STRETCH introduced in 1955. It was to have been ten times faster then any machine in use in 1955. (It never made the target.) The other two were introduced in 1956. They were the UNIVAC-LARC (for Livermore Automatic Research Computer--to be used at the Lawrence Livermore National Research Laboratory in California) and the Ferranti-ATLAS. Although these were out of our reach, they had some interesting features. Among these was virtual memory--a new concept for the management of memory. This and other capabilities were introduced on the Ferranti-ATLAS and would be of great importance in the future.

Other alternatives that came out in 1957 were the Datamatic-1000 from Honeywell and the Philco-2000 from Philco-Ford. They both seemed like reasonable machines, but had the same lack of programming support the others suffered. (The AN/FSQ 7s and 8s also came out in 1957. I was interested in them because they were also known as Whirlwind IIs. Whirlwind had acted as the prototype computer in the Sage system and these were its replacements. Also the Lincoln-TX0 was interesting because it was the first fully transistorized computer.)

Also in 1957, IBM introduced the 305 RAMAC, which stood for Random Access Memory Accounting and Control. (I found out while writing this book that the RAMAC had been the work of John Haanstra—the young man who had been a Navy seaman when we both worked at Point Mugu.) This was the first magnetic disk memory system and used the first commercially available magnetic disk storage units. The disk memory consisted of 50 magnetic disk surfaces, each 24 inches in diameter. Their total storage capacity was about 25 mega-characters. Also in 1957, the first FORTRAN compiler was delivered after three years in development. The RAMAC was not anything we could have used in its original form; however, it alerted us to be on the lookout for the future use of this technology. However, the completion of FORTRAN was a distinct advantage for us because it was something our scientists and engineers could use immediately, if and when we got a 709.

Only the generalized routines made recommendation of the 709 feasible. The plan was to re-implement the generalized routines for the 709 and the same source files and packets could be used on the new machine as on the old. However, execution times would be reduced because of the increased speed of the processor and increased simultaneity between

the processor and the Input/output equipment. Simultaneity improvements occurred because in many instances two tape handlers and the computer could be in concurrent operation on the 709 while each of these took place at separate times on the 702.

That was the plan and the recommendation and it was provisionally accepted by our management and by the AEC, which was paying the bill. But where were the generalized routines for the 709 going to come from? That was the big question. Based upon the provisional acceptance of our plan, a letter of intent was issued to IBM. The management of IBM was contacted and petitioned to pick up the generalized routines for general use, but they were not interested. This was a radical new idea and besides, it was not their idea so they were not interested. The other avenue we could pursue other than implementing the programs for ourselves at Hanford was the SHARE Organization. [C] On the strength of our letter of intent, we were eligible to become members of SHARE and we might be able to get help from SHARE in undertaking the development.

SHARE was an organization of users who had banded together in their own self-interests to try to make up for the serious deficiencies in programming and other support from IBM. The organization held meetings twice per year. The SHARE members were from some of the biggest and most influential companies, universities and governmental institutions in the country and these organizations had sent some of their best people to attend SHARE. SHARE used various means to deal with or in some cases to circumvent IBM. They exerted pressure on IBM by adopting resolutions representing the consensus of SHARE, they provided opinions and feedback to IBM about the acceptability or lack thereof of the proposed designs of upcoming programs and in worst case, they would muster SHARE personnel to perform development on behalf of the membership.

I don't remember more than two or three of us from Hanford attending SHARE meetings, but many organizations would send up to a dozen people. As a result, the general sessions of SHARE were usually held in the Grand Ballroom of some large Hotel in a large city. Fortunately, general sessions were infrequent. Much smaller committee and subcommittee meetings in decent sized rooms with twenty or thirty people in attendance carried out most of the business and work of the organization.

I was to discover the creation and dissolution of these committees and subcommittees were accomplished by simple word-of-mouth agreements between the chairpersons and the SHARE Officers. The Officers were the

President, the Vice President and the Secretary. The President ran the general sessions, approved or disapproved the committee/subcommittee structure and represented SHARE in communications with other organizations. I don't remember the Vice President doing anything except being there in case something happened to the President. The secretary was the one that did the most work, and the institution to which he belonged accepted a financial burden when it permitted him to take the position. He collected and published the minutes of all the general sessions and distributed them to the entire membership. These distributions would typically be about three inches thick because they included the entire transcript of the general sessions along with a written report from each committee and subcommittee chairperson.

SHARE was one of several vendor-specific organizations that existed at the time. I am only familiar with two: SHARE and GUIDE. They were the two users' organizations for the IBM-701/704/709 computer family and the IBM-702/705 computer family respectively. The 701/4/9 family was the set of 36-bit, binary IBM machines and the 702/5 family was the set of variable word-length decimal IBM machines. We at Hanford would have been eligible to belong to both organizations, but past experience had indicated the GUIDE organization was not particularly productive, so we never participated in it.

Harry was the first to attend a SHARE meeting. He attended SHARE IX in San Diego at the end of September in 1957[20]. This must have been before our order for a 709 was approved. However, on that occasion, he met Charlie Bachman, the Chairman of the Data Processing Committee, from Dow Chemical and we were in touch with Charlie from that time on. I attended SHARE X in Washington, D. C. in February of 1958 and gave a presentation on the Generalized Routines. I presume Harry had arranged with Charlie for me to be on the agenda but I remember I had never met Charlie face-to-face as of the time of the presentation. In any case, I recall presenting to an audience of about fifty people in a very large, mostly empty meeting hall or grand ballroom. The presentation went very well and the audience was extremely attentive. Many questions were asked and, at the end, people stayed around to talk to me. I remember one of the very interested people was Fernando Corbató of MIT. I believe he was the President of SHARE at the time because I recognized him. Another interested person I finally got to meet in person was Charlie Bachman of Dow Chemical Company in Midland Michigan.

The Data Processing Committee of SHARE of which Charlie was Chairman, was mostly a group of 709 users who, like we at Hanford, had significant amounts of business data processing (as opposed to scientific

calculations) to do. They had no agenda other than to commiserate with one another about what they might do to solve their (not so easy) problems. Of course, Charlie and most of his committee members were very interested in what I had to say. I hardly had to say it might be a good idea for us to band together to produce generalized routines for the 709—it occurred to all of them simultaneously. Needless to say, that was the beginning of a long and, I think, fruitful friendship with Charlie and several other SHARE members.

Either at that SHARE meeting or at the very next one, the idea of a combined effort had spread like wildfire. The product of the effort had been named 9PAC and a responsible subcommittee had been created, the 9PAC Subcommittee, with me as chairman, to work on its creation. Of course, this all required the approval of my management in Richland, but they were overjoyed to have this opportunity to solve the big hitch in their plan to upgrade to a 709. Many of the committee members expressed interest in working on the project, but they too needed to receive approval from their managements.

This eventually led to me making frequent trips to various companies to tell my story to the brass who would, we hoped, be supplying free manpower to achieve the programming support we needed. I must have succeeded because we had plenty of manpower. In addition to manpower, Union Carbide Corporation contributed office space for us to use for the 9PAC Report Generator effort in Long Island City, a short subway ride from Manhattan.

Actually, the SHARE effort was limited to creating the Report Generator of 9PAC. The 9Sort, as the 9PAC sorting program was called, was a modification of the one being supplied by IBM and was done completely at Hanford. The new 9PAC File Maintenance Program was developed at Hanford with Hanford manpower.

After my first SHARE meeting, Harry and I both attended regularly. We both made many good friends among the membership and spread Harry's data processing gospel at every opportunity. It was helpful to me to have his support. When I first started conducting subcommittee meetings, I brushed up on Robert's Rules of Order and tried to proceed in a very formal way, but some of my good friends took me aside and said, "This is SHARE, we do things in an informal way." From then on, we just had nice chatty, though organized discussions.

In addition to the business I was involved with at SHARE meetings, the General Electric attendees usually succeeded in getting together a

General Electric caucus to take advantage of being physically together. The meetings would simply be a dinner together where we could meet one another and chat about what we were doing at our different departments. It was at one of these that I met Fred Banan, Harry Cantrell and Jane King all of whom I worked with later.

Also, during this period, I had the opportunity to visit a variety of General Electric and IBM sites. The General Electric sites that I remember were Appliance Park in Louisville, Kentucky where I had a chance to get reacquainted with Dick Petersen, Power Transformer Department in Pittsfield, Massachusetts where I met Stan Williams, Large Steam Turbine in Schenectady where I had a chance to visit with Jane King and Harry Cantrell and General Electric Corporate Headquarters in New York where I had a chance to use the executive washroom.

I also visited the IBM Homestead in Endicott, New York and the factory where the 700-series machines were manufactured in Poughkeepsie. I don't remember whom I saw or why I went to these IBM sites, but they treated me very well. I also visited IBM World Headquarters (Fred Gruenberger always called it Galaxy Headquarters) at 570 Madison Avenue in Manhattan. I was trying to get them to divvy up support for 9PAC. As in an earlier meeting that Harry and I had made there for the same purpose, they didn't treat me very well.

I met many people during my SHARE days. One worthy of note was Fletcher Jones. He worked for North American Aviation and his office was in Columbus, Ohio. Paul Tani worked for him and Paul was a great friend of Harry Tellier's. On one occasion, I was invited to visit them in Columbus. I had been in New York so I took a sleeper on the train to get to Columbus. I arrived there in the morning and Paul met me at the train station.

My recollection is that Fletcher headed up the Surge project in SHARE. Surge was a compiler for the 704 and its language was supposed to do wonderful things for data processing users. Many SHARE members viewed us as rivals and I thought Fletcher wanted to meet to discuss this rivalry. I was wrong. He wanted to talk about nothing except his plan to form a company along with Roy Nutt (another SHARE acquaintance) and he was looking for people who he could hire as managers. I guess he was fishing to get Harry or me or both to volunteer, but neither of us was interested at the time.

Anyway, he took Paul and me to lunch. We had to go in separate cars because he had just bought a new Carmen Ghia sports car and it was a

two-seater. Well, we finished lunch and I left and he formed his company. Its name was and is Computer Science Corporation and it has had many years of success and has been listed on the New York Stock Exchange for many decades. [D]

The development of 9PAC proceeded apace and the product was put into use in May of 1959. I can't remember the names of all the participants in the 9PAC Report Generator development. Some of the companies were Dow Chemical, Union Carbide, Chrysler, Lockheed Missile and Space Division, North American Aviation, Northern States Power, Phillips Petroleum, Thompson Ramo Woodridge and Space Technology Laboratories . The Thompson Ramo Wooldridge and Space Technology Laboratories contributions were in the area of user documentation. Although IBM had one or another observers present during the development, I don't think that either of them contributed a single idea or line of code. However, by the summer of 1960, IBM Applied Programming had taken over maintenance of 9PAC. University of California at Los Angeles was also on the list of contributors. [21] I think that was in the form of computer time at the Western Data Processing Center, which was a joint venture between IBM and UCLA.

The person who did most of the programming work was Ed Roddy. Ed not only worked very hard, he provided the leadership I could not provide because I was back at Hanford or running around the country keeping various contributing companies happy with their participation. Harry used to recall Ed when he first came to work at Hanford. He was not a complete stranger because his father had been a long-time Hanford employee. Ed was hired as a computer operator when Hanford had nothing but punched card equipment. Whatever his job, Ed was an outstanding employee. He had quickly become one of the outstanding 702 operators and was given a programming assignment that rapidly led to the development of the 702 report generator as described above. He was chosen to work with us on 9PAC because of his report generator background and his high level of ability.

While the 9PAC Report Generator development proceeded, I had George Gillette, Joan Cannon, Ron Pulfer and Glen Otterbein working for me at Hanford on the 9PAC File Maintenance Generator. This and the 9PAC sort rounded out the 9PAC system. (I don't remember who did the sort. Perhaps Ed Roddy whipped it out after dinner one night.) In any case, one big difference existed between the 702 generalized routines and those for the 709—the latter permitted a more complex file structure. This was a concession to those involved in the project that felt the need to

economize on storage space by tailoring their file structures as a way of reducing file size (and consequently processing time).

The source files of the 702 generalized routines, were required to consist of records of a single length and format. In the 9PAC-generalized routines, the source files could contain multiple record types in a hierarchical organization. In this way, redundant information could be eliminated.

For example, suppose your company had three divisions and the data to be processed in each of the divisions was unique in some respects but common in others. In the 702 source files, it would have been necessary to have each record contain data common to all three divisions plus data unique to each of the three divisions. Then two out of three division-unique areas would have been left null in each record.

In 9PAC source files you could define one record type to contain the information common to all three divisions and a separate record type for each division to contain its unique data. The three non-common data types would then each be defined as subordinate to the common record type and the system would adjust itself to operate on all four of these as appropriate. With this capability, two records would exist in the file for each item: one containing common data and one containing data unique to the division to which the item pertained.

This change in design came at some costs: it made preparation of 9PAC packets more complex than preparation of their 702 counterparts, it destroyed the upward compatibility of 702 packets with the 9PAC generalized routines, and it complicated the development of the 9PAC generalized routines compared to their 702 counterparts. By providing this added capability, the programmer productivity was negatively impacted and the labor-free conversion from the 702 at Hanford was no longer going to be possible. Nevertheless, the trade-off was judged acceptable, so as to gain the "free" manpower that would accrue from SHARE participation.

One of the functions SHARE performed for the users was to keep track of the performance of the various member installations. During each meeting, the SHARE secretary gathered the data and then consolidated it for incorporation into the minutes of the meeting. The key data were reported on the mainframe and each of the peripheral controllers. The data were production time, preventive maintenance time, unscheduled maintenance time (down time), red tape time and idle time. Red tape time occurred when the unit was unable to proceed while it waited for

some manual operation to be completed such as mounting or dismounting a magnetic tape or reloading the card reader.

I believe the secretary published only the summary information so no installation got paranoid about where it stood in relation to the whole. However, knowing its own information, each installation could compare itself to the whole. I found the data to be very valuable in the planning of our own installation. Where should we concentrate our effort to get more productivity for the large sums of money we were spending? We, of course, wanted to maximize the production time and minimize the red tape time and down time. It also made good and fruitful conversation for Share meetings where we could have private conversations with people from other installations. We could see how they were or were not performing and what techniques they were finding useful in their quest for higher productivity and lower down time.

As I had discovered when I was the machine room supervisor, we were paying many dollars for off-line peripheral equipment. Each of these pieces of equipment was a device combination consisting of a controller, an input/output device and a magnetic tape handler. Each triplet would perform one of the three conversions: card-to-tape, tape-to-card or tape-to-printer. We needed two or three tape-to-printer converters at times and hardly ever needed the tape-to-punch one. The use of the card-to-tape combination was moderately busy.

I found a peripheral device controller being built in Southern California capable of converting between several pairs of devices of the types: magnetic tape handler, card reader, cardpunch and printer. (I don't remember the name of the company). It could do little else, but that was not my concern. I reasoned that if we could get one of these controllers and if it could perform the various conversions we needed done simultaneously, we could save the cost of several IBM-supplied controllers and a couple of tape handlers. I contacted the company and we decided their device would do our job. Hence, I justified the purchase and got it into the paper mill at Hanford for approval.

I was convinced this would have been an improvement in our productivity and a reduction in our costs and was able to show this. However, before the approval was complete, IBM came out with the IBM-1401. This was a small variable word-length computer. It would do all of the same things and more than the gadget I had proposed to purchase for even less money. The 1401 was able to accomplish this feat because it utilized multiprogramming. (See Appendix H for a description of multi-

programming.) Reluctantly, I withdrew the proposal and we obtained a 1401 as a replacement for our multiple off-line controllers. [E]

We at Hanford and others kept a sharp eye on red tape time reported by SHARE. This was time that could be reduced substantially by a good staff of operators. We did pretty well in keeping it down, but the variation from one crew of operators to the next was substantial. We hoped to be able to do something about it.

This was a problem not only with Hanford but also with all the SHARE members. As a reaction to this problem, several installations developed job monitors. These were programs that remained in memory at all times and at the termination of one user program started the next. In conjunction with this job-to-job control, a file was provided, usually called SYSOUT, upon which the running programs could record their printer outputs. The outputs would be stacked on the output file and converted to hard copy on peripheral equipment after the monitor run.

This was a good solution for installations that predominantly ran purely computational programs that were well checked out and involved the use of few, if any, magnetic tapes. In this case, many programs could be run one after another with no red-tape time, with no operator intervention and with no lost time between jobs. This was ideal, but the circumstances that permitted it to happen were rather rare, especially at Hanford.

IBM attempted to provide a more general solution to this problem with IBSYS. This was a monitor that tried to deal with the problems of mounting/dismounting magnetic tapes. Instructions to the operator were given during the current run to mount tapes for jobs that were not yet in execution—a procedure that good computer operators followed anyway. In this way the tapes would be ready to use when their job(s) was(were) ready to run. Similarly, IBSYS would issue instructions to operators to dismount tapes no longer needed on-line after their jobs had finished execution. With this approach, the operator involvement at the outset of execution of a job was (hopefully) limited to adjusting the identification of tape handlers between jobs, which was accomplished by simply turning a knob.

I had no experience using any of these. They didn't seem to meet our needs and our statistics were pretty good without the use of monitors (probably because of the high quality of our operators). Furthermore, if any program in a sequence of programs being controlled by a monitor had a bug in it, it would probably crash in a way that would destroy the monitor in memory. In this case, the operator would need to restart the

monitor and execute a complex recovery procedure that could easily eat up the hoped for timesaving. However, this experience pointed to the need for further improvements in the management of large systems. I would have the opportunity to deal with that need in the future.

Some other efficiency problems that had nothing to do with SHARE statistics needed attention. Many magnetic tape files utilized only about half of the tape—the rest was entirely blank. Also, even though the hardware of a computer was capable of executing programs with the central processor and simultaneously performing input and output, it didn't mean this would always happen. Both of these problems are solved by use of Blocking and Buffering. These techniques are explained in Appendix H. Both of them and various other developments with which I would be involved in the future were incorporated into 9PAC to the extent possible.

At some time in 1959, I received a letter from Bernie Galler of the University of Michigan (a coworker of my Franklin Institute panel moderator) inviting me to teach a one-week class on 9PAC. This would occur at the summer program on computers held at Michigan every year. He apologized that he would be unable to offer me a salary for this job, but offered an honorarium of $800. I, of course, wanted to do it and, in particular, I wanted to combine it with some vacation time and to take the family along. I think I offered to take the time off without pay so no one could accuse me of "double dipping". I'm not sure how that turned out, but I did get permission, and I did do the class.

By this time, SHARE XI had taken place, and Charlie Bachman and I both brought our wives to the meeting in San Francisco. The wives had gotten along famously and had a great time in the big city. So when we went on the trip to Michigan, Charlie invited us to stop by at their house in Midland, and to drop by their cabin in northern Michigan on the way for a swim. We did both of these things and much more. I remember Charlie and me sitting in his back yard in the evening and talking with pride about our 9PAC accomplishments.

Not long before that conversation, IBM had announced the 305 RAMAC. The idea of RAMAC was you didn't need to do your processing using rows of magnetic tape handlers but could instead put your data on magnetic disks and update the data in place as the need presented itself. Having done so, you could access any particular piece of data directly on the magnetic disk, or you could start a program to create a report containing sequences of data just as you had done with magnetic tape. Charlie and I felt this new mode of processing would have a large impact on the

future of data processing. We, as the keepers of that domain within SHARE, decided we should try to do something to help our companies and our fellow users be prepared to use this advanced technology. What we didn't know at the time was that the capacity and reliability of disk storage were not going to be as great as one would have liked for yet a couple of decades. Also, the networks and remote workstations permitting users to have easy access to these data stores were even further away. Nevertheless, we both resolved to try to prepare ourselves for this new era.

Soon after this Charlie accepted a job with the General Electric Production Control Service in New York. I was made chairman of the SHARE Data Processing Committee to replace him and Charlie became a friend less frequently heard from for Harry and I. However, Charlie did not let any grass grow under his feet. He developed Integrated Data Store (called IDS), which was (and perhaps still is) a generalized program for the storage and retrieval of data on magnetic disk files. The original IDS development took place in 1962, but underwent an evolutionary sequence of improvements after that.

To describe my contribution to this issue, I need to say a bit more about the organization at Hanford, because some of these elements were involved in what I did. Harry Tellier reported to the Manager of Finance at Hanford, as did Lou Hereford, who was Manager, Business Systems and Procedures. The Manager of Finance was named Ken Robertson. I knew Ken, and he was always very good and friendly to me both at Hanford and after we had both left there. However, he and Harry didn't get along particularly well, and their disagreements seemed often to involve Lou Hereford and/or his organization.

Lou was an ex-marine and it showed. He was very aggressive and terse in his communications. He was very bright and sharp and had opinions on most things. Though they often locked horns at work, Lou and Harry were fairly friendly in a social setting, especially if a chance existed to have a few drinks. (I found out years later Lou had been a young marine on board the Battleship Nevada during the attack on Pearl Harbor. It was the Battleship saved as a result of being beached after it had been severely damaged thereby avoiding blockage of the harbor entrance by the sunken hulk.)

At a time after the main pressure of 9PAC had settled down, I had an opportunity to meet one of Lou's employees, Jim Fichten, and we hit it off very well. Our favorite topic of conversation was what kind of business systems we would like to see at Hanford, and what hardware system we

would like to see to support them. Since we were just "blowing smoke", we didn't limit ourselves to presently available hardware, but imagined things we would like to have that were perhaps some distance into the future. We enjoyed these sessions, and apparently Jim discussed them with Lou, because after a short time, Lou came up with the idea that we should prepare a five-year plan for Systems Support at Hanford. Of course, Lou didn't limit the scope to business systems but also included data collection and processing at the reactors and chemical plants and data and voice transmission between all of the components on the reservation. (semper fi !)

Lou assigned Jim Fichten and Chuck Buchanan and perhaps some others to the project, and I supplied the data processing projections. In a couple of months, we came out with a dandy piece of science fiction. In my own words written at the time, the plan was for, "… an integrated information system fed by automatic and semi-automatic data acquisition equipment, a microwave communication network with satellite computers and a central 'super data processor'". It was all probably achievable, but who knew when? We didn't propose a schedule or submit any justification to proceed with implementation (it was impossible). It was an interesting exercise, but sterile. It led to nothing--at least not immediately, but in the future . . .?

By this time, Kendall Wright had left to accept a job with IBM and Chuck Thompson had gone to the General Electric Computer Department in Phoenix. Also, IBM had announced the successor to the 709, the 7090. It was my responsibility to evaluate the 7090 and recommend its acceptance or rejection. It really was a no-brainer. The 7090 was almost totally program compatible with the 709 and its cost, though higher than the 709, was much smaller per unit of data processing than the unit cost on the 709. Of course, our workload had increased and we needed the increase in productivity. I recommended and justified the conversion. The IBM-7090 was to be installed in October of 1960. [F]

During 1960, Harry and I started to talk about my next assignment. He opined that I might be able to get a job at the General Electric Computer Department in Phoenix. This was fairly appealing to me. I was very loyal to General Electric, and the company was not doing particularly well in the computer business. I thought I might be able to make a real contribution to the success of the department. On the other hand, I realized I would be moving from a situation where I was usually working on the biggest, latest and fastest computers available to some much smaller and limited equipment.

The prospect was not strong in either direction, but Harry arranged for me to be interviewed and I made the trip to Phoenix. I talked to many people while I was there, and many of them did not impress me, but one prospect was dangled before me that really sparked my imagination. General Electric was designing a new line of computers including a small machine called the X and a large one called the Y. I was offered the opportunity to work on the programming for the Y.

The General Electric Computer Department had a definite appeal for me, but I also appreciated (and grew to appreciate very much more) that the working environment at Hanford was exceptional. (I came to view it as the Camelot of electronic data processing.) It became more and more apparent that throughout the universe of computer vendors and computer users, programmers were second-class citizens. They were looked upon as some sort of clerks who did something only marginally necessary and as a result caused cost and schedule overruns and generally mucked up the works. They also had an aura of being impractical, pointy-headed intellectuals in some cases. By contrast, hardware engineers were awarded great respect and dignity. This dichotomy showed up in public attitudes and the salaries the two groups received—the programmers were always at the low end. The idea was even put forward that programmers should be classified as semi-skilled workers whereas hardware engineers were always accepted as high-level professionals.

None of this dichotomy showed itself at Hanford. We were sheltered from it by Harry Tellier. Part of his personal philosophy was that it was the programmers that made the computer liked or disliked by the customers; the hardware didn't matter so long as it had enough capacity and availability. I hesitated to turn my back on this, yet this issue was probably not in as sharp focus to me then as it is today.

After much debate, I decided to accept the offer from the General Electric Computer Department. I made the decision to join one of the real product producing departments of my company. My father had always been a big fan of General Electric, and I think my opinion was influenced by the idea it would please him if I stayed with the company. Furthermore, during my employment at Hanford, IBM had earned in my mind the characterization of the enemy. By accepting this offer I had, for the first time, the opportunity to face this enemy head on. In any case, the decision was made. My wife and I went to Phoenix over the 1960 Christmas Holiday and bought a house, and I was to report to work in Phoenix on March 1, 1961.

I attended one more SHARE meeting. While there Harry Husky approached me. (He was the President of the Association for Computing Machinery (the ACM), and had an incredible history in the industry. He had worked on the development of the ENIAC and the Pilot Ace, and had been in charge of the development of the SWAC, Bendix G15 and Maniac I.) He said he had been trying to get a data processing activity going in the ACM and in looking for someone to head it up, many people had suggested my name. I was, of course, flattered.

The ACM was and is the umbrella organization of computer organizations in the U. S. It is not affiliated with any particular vendor and attempts to address problems and solutions that pertain to the entire industry. I was forced to tell him I didn't think it would be appropriate for me to take on that position while I was employed by a computer vendor and told him of my forthcoming move. He reluctantly agreed.

Getting Started at General Electric

One could say General Electric got into the computer business by accident [22]. In the early 1950s IBM was General Electric's second largest customer after the U. S. Government. The General Electric President and CEO, Ralph Cordiner, had made it clear that no one was to do anything to perturb this excellent market for vacuum tubes, motors, transformers, switches, etc. However, several in the company thought the company should enter the field. Dr. W. R. G. (Doc) Baker, Vice President and General Manager of the Electronics Division in Syracuse, was one of these, but he was limited to producing special-purpose computers. In no case was any General Electric organization to compete in the general-purpose computer market.

Homer R. (Barney) Oldfield ran the General Electric Microwave Laboratory on the campus of Stanford University in Palo Alto, California. A representative of Doc Baker approached him in 1955 to study a solicitation to bid on a contract from the Bank of America. The contract was to produce a special-purpose computer to automate the paper processing throughout the many locations of the Bank of America in California. It was the largest bank in the state and, at the time, no computer vendor provided the systems it needed. However, Stanford Research Institute (SRI) located nearby had studied the problem and had created a moderately successful prototype. The system was called ERMA for Electronic Recording Method—Accounting.

Doc Baker thought his Division would be able to respond to the bid because of its special-purpose nature. Resistance from the executive office of General Electric was diminished because all of Cordiner's direct reports insisted IBM would win the award, so no need to oppose it internally was indicated. As a result, Barney Oldfield's laboratory, responded to the bid request.

As it turned out, three responses to the solicitation were received that the bank considered seriously: those from Texas Instruments, RCA and General Electric. General Electric ended up winning the contract. They were to deliver thirty-one ERMA systems to the bank beginning in 1958 at a cost to the bank of $31 million. Oldfield proceeded to assemble a staff in Palo Alto to get the job done (including Joe Weizenbaum and George Kendrick, my friends and colleagues from Point Mugu).

As the project proceeded, General Electric Headquarters informed Oldfield that for a manufacturing job as big as his, it would be necessary for a production site to be selected meeting certain company standards. As a result of this, six sites were selected as candidates: Berkeley, CA; Austin, TX; Phoenix, AZ; Urbana, IL; Nashville, TN and Richmond, VA. Phoenix was selected.

Phoenix office space was initially rented in the KTAR radio building. Then space was acquired on Peoria Avenue in Phoenix and on the Black Canyon Highway north of town. In addition, a service center was opened on the campus of Arizona State University (at the time called Arizona State College). These facilities along with the Palo Alto laboratory became the domain of Barney Oldfield as General Manager of the General Electric Computer Department.

While ERMA was in development and the facilities in Phoenix were being acquired, Barney Oldfield arranged to obtain a contract to manufacture the NCR-304 computer for National Cash Register. For a company not in the computer business, General Electric was beginning to look very much like a company in the computer business. Unfortunately, shortly after the move to Phoenix, Barney had to leave the company because of the illness of his wife. He was replaced as General Manager by a man named Clair Lasher.

In 1956, Arnold Spielberg left RCA to join General Electric. He had worked on the development of the BIZMAC while at RCA. He was not assigned to work on the ERMA project, but instead worked on process control computers. This was considered as non-competitive with the business machines market so was seen by the General Electric executive office as an acceptable undertaking for a General Electric Department. In fact, process control was something the company considered to be its natural area of endeavor.

In a short time, Arnold had organized a small group, and they had developed a small computer called the GE-312. It was sold initially to Jones and Laughlin Steel and McClouth Steel. So now General Electric was really in the computer business, but only the process control part of it. This development was to have important repercussions in a few years.

In the meantime, the ERMA development proceeded apace and was accepted by the bank at the end of 1958. The acceptance was marked by a large celebration attended by many dignitaries (including Ronald Reagan who was then working for General Electric) and high-level management from both Bank of America and General Electric, but

excluding Ralph Cordiner. Also, shortly thereafter, orders from other banks began to pour in. A message arrived at the Computer Department from the executive office signed by Ralph Cordiner stating that no banking systems were to be sold beyond those contracted for as part of the ERMA project.

This was a severe blow to the Computer Department and to those who had worked so hard to bring the system to reality. However, an interesting thing happened. Within a short time, President Cordiner began receiving congratulatory messages from all over the country regarding the ERMA success. Apparently the volume of these was so great that he saw fit to retract his earlier edict and merely stated the Computer Department was not to go "head-to-head" with IBM in competing for computer business. That restriction remained for the remainder of his tenure as CEO and beyond. It is a good thing this restriction was not known to the working level people of the department who were trying to get a job done. The plant and offices would have been vacated quickly if they all knew their greatest competition was located in the executive levels of their own company.

By 1959, General Electric seemed to be in the computer business but had no general-purpose product to offer. At this point Arnold Spielberg came up with the idea of providing a general-purpose product by adding general-purpose peripheral devices (card readers and card punches, printers and magnetic tape handlers) to the GE-312 and renaming it. This is what was done and the production prototype was completed in five months. The new product was named the GE-225 and it was the main product for several years to come.

The development of the GE-225 was an important stopgap. It now added to the profits coming in from the ERMA, the GE-210 (ERMA systems sold to banks different from Bank of America), NCR-304s and the GE-312s to make the financial impact on the company more positive than it would otherwise have been. However, what was really needed for success was a product line salable to customers requiring capacity ranging from small computers to the largest then in existence and beyond. It should, in addition, have been possible for customers to change from the smaller members of this family to the larger ones without needing to reprogram their applications. The event that permitted progress to be made toward this objective occurred in mid 1960 when Ralph Cordiner, for the first time in history, approved the business plan of the General Electric Computer Department.

The product line that was to provide all of these necessary capabilities was visualized and was called the w, x, y, z product line. (It is said that it was also called the mosaic, but I do not ever remember hearing that name while I had any contact with the product line.) At some point, the w was renamed the s (for small) because of the similarity of w to the Westinghouse W. The S systems were to sell for about $30,000 and the large Z systems would cost about $2.5 million.

All of these events and ideas occurred before my arrival in Phoenix. We drove in at the end of February 1961. It had been a cold trip through Idaho and northern Arizona, and we were pleased as we approached Phoenix because the atmosphere warmed and was sweet with the smell of citrus blossoms. We took the time to drive by our house before renting rooms in a motel. We stopped in front of the house and walked around the yard. The swimming pool was empty, and the bottom of the pool was full of grapefruit from the 13 trees in the yard. We walked through the house to the back and looked out the back window. A red cardinal perched on the grapefruit tree just in front of the window like a beautiful picture. Our welcome to Phoenix couldn't have been better.

When I reported to work, I discovered I was to report to Charlie Katz, my fellow presenter at the Franklin Institute and one of Charlie's direct reports was George Kendrick, my old friend from Point Mugu. I didn't know any others who reported to Charlie Katz, but they seemed a reasonable group to work with. I also discovered that both Charlie and George lived within three blocks of our new house. [G]

The organization of the General Electric Computer Department had several sections in the standard GE format reporting to Clair Lasher, the General Manager. The sections were responsible for Manufacturing, Hardware Engineering, Programming, Marketing, Finance and Personnel. The Manager of Programming to whom Charlie Katz reported was Helmut Sassenfeld. Helmut was a German (surprise!) who had worked at Pienamunde during World War II and had been hired in the U. S. after the war as one of the group that immigrated with Werner von Braun to Huntsville, Alabama. He spoke English with a German-Alabama accent. I was to later become very close to Helmut.

As a company, GE had two well-known philosophies that hadn't affected us much at Hanford because of the close supervision of the AEC: decentralization and professional management. The idea of decentralization was that each General Electric department was a separate, autonomous profit center and it was to be run by its General Manager with complete authority. The General Manager could do

whatever he needed to do to achieve the success of the department's products including competing with other General Electric departments. If a department was not successful, then its General Manager would be replaced, or the department would be sold or shut down. This always seemed practical to me, especially when viewed from the standpoint of managing an enterprise as large as the General Electric Corporation.

The other idea was not so obviously a good one. It was that management was a profession in itself and regardless of the product of the department; any manager could manage it whether or not he knew anything about the product. In my opinion, this philosophy caused us great grief at the General Electric Computer Department. We were in a difficult business with many startup woes and the company was expecting it to turn a profit in a short time. Many of these woes were caused by poor business decisions at the top. The wrong products were often produced at the wrong times, and the markets served by them were not as lucrative as those that could have been served including General Electric itself. Also, not knowing the product and not having worked in the industry, the General Managers didn't know whom to hire to lead the workers, nor did they have a good understanding of the competition, it strengths and weaknesses. Of course, the General Manager can't do everything and is not expected to, but experience in a field gives a manager intuition and insight he would not otherwise have.

I have no idea how many General Managers we had while I worked under the management of the plant in Phoenix, but I don't think I have enough fingers to count them. At one time one of our executives had just come from the Outdoor Lighting Department. Each time we got a new General Manager, one or more of several things would happen for sure: we would reorganize and/or we would rearrange and redecorate the offices and/or rearrange the factory. These were mostly annoyances, but they had very little to do with achieving success and diverted our attention from our major problems. I think if our General Managers had known more about the business we were in, they would have spent more time and other resources on things that mattered much more. I don't intend to dwell on this, but the reader might look for signs of these weaknesses while reading the text.

It turns out the strategy being employed to develop the S, X, Y, Z product line was to start with the X and the Y and later build down to the S and up to the Z. Arnold Speilberg was heading up the X development and a man named Art Critchlow who had recently been hired from IBM had responsibility for the Y development. I naturally gravitated toward the Y, because most of my background involved large computers. At some

point, shortly after my arrival in Phoenix, I was assigned to Y programming development. What this meant was I was assigned to help create a proposal describing the programming support that would be developed for delivery to the customers of Y computers. This was very similar to the work I had been doing at Hanford, but it was for a new computer that was still a figment of our collective imaginations.

During my days at Hanford, an interest grew in what was called Operating Systems. Operating Systems are a set of programs generally supplied by the vendor, always available to users and to other programs and performing services for them.

By using an Operating System, one hopes to achieve an uninterrupted flow of jobs through a computer system and to simplify users' applications by permitting them to use the Operating System's versions of frequently needed programs instead of supplying their own. Each job in those days was represented by a deck of cards (similar to a 9PAC packet but usually larger) that would supply the program to be run, its identification and each of the resources to be used in running it. The Operating System would use the information in this deck to facilitate the performance of services, to provide the resources the program would or might need and would place the program in execution when all of its needs could be satisfied.

For example, one would expect an operating system to perform some if not all of the following: accept a queue of programs (jobs) to be executed, schedule these jobs for execution, allocate resources (memory, magnetic tapes and disk space) to the jobs, begin execution of jobs as resources become available, issue messages to operators to perform support functions as needed (i. e. mount/dismount magnetic tapes, replace printer paper as needed, remove cards from full cardpunch hoppers, etc.), monitor execution of input/output so that one job does not interfere with another in use of I/O devices, recover from hardware errors if possible, queue printer outputs on a system output device (SYSOUT), cause SYSOUT to be printed when appropriate, provide blocking and buffering support for applications and keep careful measurements of the resource utilization of each job for billing and analysis purposes. No Operating System had yet been built including all of these capabilities—the hardware to support them had not been available—but it seemed feasible to create a system including them all plus others.

My job on the Y computer was to define the Operating System and the input/output system contained therein. The Operating System everyone imagined supported multiprogramming. That meant the hardware

needed to be able to isolate system programs—those like the operating system that were omnipresent in the machine—from user programs—those that were executed under control of the Operating System. It also needed to isolate user programs from one another so as to prevent one user program from damaging any other or the Operating System.

The hardware also had to have an interrupt system to permit the Operating System to gain control before the start of any input/output operation to ensure its execution would not be harmful to the Operating System or any other user program. Interrupts also had to give control to the Operating System at the end of every I/O operation to inform the system and the user program the operation was complete. If an I/O operation completed successfully the system would start a new I/O operation using the resource released by the completed operation. If the I/O operation completed unsuccessfully, the Operating System would initiate a recovery procedure that would either cause the operation to be successfully executed on retry or would turn the problem over to an operator. Other capabilities of a more detailed nature existed that would have been nice to have, but which were negotiable. The Y computer was to have had all of these (since it was a "paper tiger"—it existed only on paper, not in reality).

My work on the Y computer kept me busy until January of 1962. This involved many meetings with Product Planning, Systems Engineering and Hardware Engineering. These meetings gave me an opportunity to find out how things worked in Phoenix and to meet many people with whom I would work into the future. It also involved writing down some of our ideas and negotiating with various people to try to achieve what I thought would lead to the best product. This was all interesting and useful for my education, but it was all for naught because the Y computer was cancelled. However, the X computer survived and became the GE 400-line. [H]

After the demise of the Y computer, I was given a unit called Operational Programming to manage. This unit was responsible for creating and maintaining I/O and utility programs (Blocking and buffering, card-to-tape, etc.) for the GE 200-line. (By now a new member of the line existed—the GE-235.) The unit had had a previous manager from whom I took over. I don't remember what happened to my predecessor, but much of the work was ongoing and I just took over the day-to-day management of the work and the personnel. However, we had just begun offering magnetic disk storage on the 200-line and we had to implement the input/output for that device type. This was a substantial challenge and we were able to handle it on time and within budget.

At about this time a new employee appeared upon the scene. He was Dave Latimore and he had come from the General Electric site (in Idaho I think) where they were trying to implement a nuclear propulsion system for aircraft. That endeavor had folded after a several-year effort to come up with a feasible design and Dave had become available to us. Dave was an ex-marine and it showed in his demeanor. As soon as he joined us, he became productive and I was more than glad to have him on our team.

He soon became aware of the problems and opportunities of our department. He had had some experience using job monitors in his previous work and was unhappy we didn't have that capability, in a form our customers would use, on the 225/35. One day he came to my office with a plan. It was a plan to implement an Operating System on the 200-line. He had done his homework and presented a very tempting case to proceed. It was not my decision to make, but I supported him in getting his plan to the right people. The Operating System, Bridge II, was created under his leadership just as he had planned. (A Bridge I had existed, but was under-used by our customers.)

As far as I am concerned, Bridge II and the disk I/O were the most useful things done in my unit while I was its manager. Nothing else that was innovative or had a significant impact on the business of the department took place during my tenure.

GE-625/35

In early 1963, we began to hear about a computer—the M236--being built on contract for the Defense Department by the Military Systems Department in Syracuse, New York. It was a machine very much like the 709/7090 machines—had a 36-bit word length and a similar interior décor. It was being developed for some secret project and the computer inputs and outputs were apparently all real-time in nature. Hence, the machine had no conventional input/output for devices such as card-readers, cardpunches, magnetic tape handlers, printers or magnetic disk handlers. Quite a few people in the company were saying, "We pay IBM many dollars each month to lease their computers…the computers our own Computer Department builds don't do most of the jobs the company needs to get done, so why don't we manufacture this IBM look-alike and keep the profits in the company?"

This argument was difficult to ignore and, in the end, it won the day. The decision to build the General Electric 600-line, which was our version of the machine originally built in Syracuse, came in May of 1963. I was transferred immediately to work on the new product. [I]

I very much wanted to become the manager of software for the new line; however, that was not to be. [By this time, the term "software" was coming into common use, so I will also use it here. However, one General Electric manager found it objectionable, saying when people said it he thought they were talking about baby diapers.] Our general management had hired a Ph D physicist, John Weil, to take charge of the 600-line. John Weil had run the Philco-2000 computer installation in San Jose, where the company had a Department called Nuclear Products or some such name. In any case, Ed Vance had managed programming for John Weil in San Jose, and John made sure Vance got the software management job. I thought this was an odd choice, since Philco-2000 software bore very little relationship to that which the General Electric users had been employing. Our users were accustomed to IBM 700-series software and that was my paramount area of expertise. However, Weil was the boss and he made the decision in favor of Ed Vance. I doubt if I was even considered.

However, I did get a good job on the team: Manager, 600-line Operating Systems. This was the area of my most intense interest at the time, and creating the Operating System for this series of machines was going to be a real challenge.

Ed Vance was younger than I and had gotten some kind of degree in education from Redlands College in California. Whenever I was with him I had the feeling I had been sent to the principal's office. He was very demanding and offered praise in small doses. However, he was very smart and knew computers in general and software in particular quite well. He had been active and influential in the Philco-2000 users' group as I had been in SHARE.

Early in the life of the 600-line, perhaps before its birth, Ed and I and a few others went to Syracuse to meet the engineers that had built the M236. As far as I was concerned, the main topic of discussion was how we were going to make up for the absence of an I/O system to handle conventional peripheral devises. On this trip we met John Couleur who had been the main designer of the machine. He was a likeable person, and shared our desire to come up with a useful computer for our company. We had a general conversation about what we would need in terms of a general purpose I/O Controller and he seemed to think what we were asking for was achievable. On the same trip, Ed Vance met with John Couleur's boss, a man named Walker Dix, with whom Ed was very impressed.

As we recruited for our 600-line workforce, I was able to hire Fred Banan, my old friend from SHARE meetings. He had been working for the Jet Engine Department in Cincinnati and was glad to move to Phoenix. Fred was a little older than I and had been a cryptanalyst during the war. Cryptanalysts are the "top drawer" of cryptography, the ones who break enemy codes. Furthermore, Fred had worked at the "top drawer" cryptographic facility of World War II: Blechley Park in England, where the Nazi Enigma code was broken. While in England, he had married his wife Betty who was just as pleasant to be with as was Fred.

Fred and I worked with John Couleur over the telephone to flesh out the 600-line I/O Controller (called the IOC). This involved designing the interrupt logic of the 600 computers as well as the interior décor of the IOC itself.

The initial 600-line products were the GE-625 and the GE-635. The only difference between them was the GE-635 had a faster memory than the GE-625.

The 600-line machines evolved as being memory oriented. That is, the memory was viewed as a passive unit at the center of the machine, and other active devices would cluster around it. The active devices were

CPU's and IOC's (Central Processing Units and Input Output Controllers)—they caused data to be processed or moved; passive devices merely stored or retrieved data when instructed to do so. In this way, it was possible to build systems with multiple CPU's and multiple IOC's; hence, it was possible to expand a system by the simple expedient of adding CPU's and IOC's. The memory size, though, for a single processor or IOC was limited to 256k (=262,144) words (equivalent in modern terms to 1 megabyte, which at the time seemed enormous).

In any case, we finished the design of the IOC, and Fred and I later shared a patent for its creation. Getting patents was very much encouraged within General Electric. The company paid for all of the legal work involved and owned the resulting patents. They paid each of us a hundred dollars (or some such amount) when the patent was awarded.

After finalizing the design of the IOC, we had what looked like a good commercial product. It would be a multiprogramming, multiprocessor system with a state-of-the-art Operating System and language processors for several languages: COBOL, FORTRAN and GAP, where GAP stood for General Assembly Program. It would also provide a complete library of mathematical functions in a subroutine library. Other utility functions were provided including input/output blocking and buffering. The operating system would schedule jobs, allocate resources to jobs before their execution, manage execution of input and output commands and the resulting interrupts during execution and provide for recovery procedures to be automatically executed in the event of I/O errors.

The hardware provided fences in the address space so no user program could encroach upon an Operating System Program or another user program. The Operating System would control the location and setting of these fences. A SYSOUT file would be provided by the Operating System for collection and stacking of printer and punch output. The SYSOUT files would be printed and punched by the Operating System at appropriate times. The Operating System would also direct the computer operators in the mounting and dismounting of tape reels and dismountable disk packs, changing of printer paper, supplying additional paper to empty printers and loading and unloading of card stackers and hoppers.

With this computer hardware and Operating System it was unnecessary to have separate peripheral hardware to execute card-to-tape, tape-to-card and tape-to-printer transpositions. These were just additional jobs executed concurrently with other jobs on the main system.

Having gotten the core of a development staff together, it was time to specify the 600-line software in detail. To do this, it was thought useful to get away from the office and the telephones and the regular interruptions of every-day business. The idea was to go to a place where we could concentrate our efforts exclusively on the software definition. To this end, a design group went to the town of Skokie, Illinois, on the outskirts of Chicago, and lived in a motel with many nice meeting rooms for a week or two and worked on the design. (Why we couldn't have done this in Phoenix with the chance to see our families every evening I'll never know. I guess some people didn't want to see their families.) Some of the people I remember there were: Ed Vance, manager; Leroy Ellison, SORT; Ed Somers, COBOL; Bill Heffner, FORTRAN; Jim Porter, GAP; Morgan Goldberg, Subroutine Library, others whose names I cannot recall and myself, Operating System.

My main recollection of the meeting is sitting in my room all alone trying to imagine the perfect Operating System. I wrote things down, I made charts and I presented my ideas to the group. We would all get together in a meeting room to present our ideas and criticize those of others. After a group meeting it would be back to solitude to modify what had been done and to make more charts for more presentations. At the end of the trip we had a close approximation to what the software package would be to a reasonable level of detail.

Some of the things decided at the meeting were the names of things. The Operating System was to be called GECOS for General Comprehensive Operating System. Most people thought this stood for General Electric Comprehensive Operating System, but some legal reason prevented us from calling it that.

So now we stayed home and worked on the implementation of the design. For me this meant hiring more people and getting them trained and put to work.

It should be understood that from the beginning of the industry, there were no colleges of universities training candidates for employment in the computer industry. The people who were hired to do programming generally had educations in other fields—engineering, mathematics, psychology, theology, all sorts of things. Sometimes especially promising secretaries or clerks or computer operators were offered programming positions and some of them were very successful, but we had to search far and wide for qualified employees. For example, Ed Vance and I took a trip to IT&T in Paramus, New Jersey where they were having layoffs and hired Bob Hobbs and Dick Foster. At some point Jane King and Dave

O'Connor joined the group, both from within General Electric. Lois Kinneberg joined us thanks to being recommended by George Kendrick, who was by then working for General Electric in Virginia.

The product was announced in 1964 and the salesmen began talking to customers and potential customers. GECOS was one of the most unique and visible aspects of the system and the salesmen were poorly equipped to talk about it, so they often asked me to tell the hot prospects, either in Phoenix or at the customer's site, about our new Operating System. I had a one-hour presentation that had evolved from our debates during the design period and I adjusted this for use with customers. It informed the customers, and the salesmen loved it. Hence, I frequently visited customer sites and potential customer sites all around the country and talked to customers who visited the plant in Phoenix.

At some time, I was asked to estimate how much memory the GECOS operating system would occupy. The users viewed this as space they were paying for but would be deprived of using, which was true. They took no account of the capabilities this use would give them. However, I was happy to respond. I reasoned that the GE-635 was about like a 7090 as far as interior décor was concerned and many 7090 installations of the day got along with 8k (= 8 * 1024) words of memory in total, so it was reasonable that the Operating System, just a bunch of little programs, would be able to fit in 8k of storage.

I was to live to regret that estimate. As the development proceeded, a continuous stream of small changes was added to GECOS. These were little features of "great importance" to various people. In the end, the estimate was quite a bit under the real figure. (I don't know what the real number turned out to be.)

In any case, the "8k" number had been told to customers so the company felt they had to (in some way) stand by it. Before this issue had arisen, the smallest memory size had been 32k (=32,768) words. This was for the so-called "low ball" system. Now these "low ball" customers were screaming they were getting less than 24k of memory, though they were paying for 32k, because GECOS was using over 8k. To respond to this problem, the company decided to offer a 40k memory to their "low ball" customers at the cost of a 32k memory.

This was a real blooper! Who ever heard of a 40k memory? No one! This became an industry joke and who was the butt of the joke? You know who—the author.

Yet the development of GECOS proceeded. However, not fast enough. A program had been prepared so the bare hardware of the prototype could be used without any Operating System in uniprogramming mode. This meant the machine was entirely under the control of the human operator and had none of the "bells and whistles" of GECOS available for running multiple jobs. This permitted testing to proceed on the various pieces of the software. It also meant, though, GECOS developers faced great competition for the small amount of available test time on the prototype machine. As a result the testing proceeded at a slow pace.

At some point, Dave O'Connor came to me with a proposal to scrap the present implementation and replace it with a new one, which could more easily be achieved. I don't remember what the details were, but he convinced me it would be a good thing to do. So I went to Ed Vance and conveyed O'connor's proposal and offered to step down in favor of O'connor so the new version could move ahead with maximum haste. He accepted and I was out of a job except that the salesmen still liked to have me help on customer presentations from time to time.

GE-645

After I gave up responsibility for GECOS development, I was not out of a job after all. For some time, John Couleur (who had by now moved to Phoenix or was about to do so) had been working on an upgrade to the GE-635. It was in response to a need of MIT for a computer system to support what they called an Information Utility. MIT had created a timesharing system in the 1960s using IBM 700-series hardware. (There is dispute whether the MIT system or the GE-235 system at Dartmouth College was the first timesharing system. They both occurred at about the same time.) This system was called CTSS for Compatible Time Sharing System and had been quite successful. On the basis of this success, they had formed Project MAC (Machine Aided Cognition). They had access to funds from ARPA to carry out the development of a prototype Information Utility. They were looking for a computer vendor able to support them in the execution of the project.

An Information Utility was to be a central information store to which users could gain access from remote terminals and/or remote computers—in other words, a very advanced timesharing system. It was similar to what we call an intranet today. At some point, Bell Telephone Laboratories joined into the project. [J]

I was assigned to work on the project as soon as my GECOS responsibilities ended. Our initial efforts concentrated on a meeting scheduled for the near future at which we would attempt to convince the combined team from MIT and Bell Labs we were the vendor that had the competency to provide them with what they needed. Some of our participants likened the event to a Ph D candidate taking oral exams.

The meeting took place as planned. Two of the participants were Fernando Corbató and Doug Eastwood—people I had known during my SHARE participation. The visiting evaluation team consisted of a total of about ten people. I presented the GECOS design as an illustration of our competency in the Operating System area, John Couleur presented some of his ideas about CPU's that would suit the needs of the project and a Manufacturing representative dazzled them with our ability to produce what the engineers had designed. It went very well. The audience was very polite and responsive and it was really a very pleasant experience after we got into it.

We won the competition at some time in the fall of 1964. The product was to be called the GE-645 and the Operating System was to be called MULTICS [23][24] standing for MULTiplexed Information and Computing Service. While I had been working on GECOS, Helmut Sassenfeld had been reorganized out of his former job and was now also assigned to work on the GE-645. The GE-645 and MULTICS were the heart of the system and everyone at MIT and Bell Labs having anything to do with the project was focusing on these two items.

The big question was, what is MULTICS? An idea of what it might be like existed because of the continued use of CTSS on a daily basis in Cambridge. CTSS was the point of departure for MULTICS. Everyone who was to be involved in the MULTICS design either had already had some experience using CTSS or, like myself, was invited to use it to gain some familiarity with its capabilities and feel.

To answer the question, "What is MULTICS?" a design team was formed consisting of members from the three organizations: MIT, Bell Labs and General Electric. The membership of the team consisted of Fernando Corbató, Bob Graham, and Ted Glazer (and Butch), from MIT; Vic Vissotsky plus others from Bell Labs and Ed Vance, Helmut Sassenfelt and myself from General Electric. In addition to the design team, a group called the "triumvirate" existed consisting of the managers of Corbató, Vissotsky and Vance: their names were Robert Fanno of MIT, Ed David of Ball Labs and John Weil of General Electric. All members of both groups were Ph Ds except Vance and myself.

The CTSS system had been documented in the form of two books, each two to three inches thick. The first described the general purposes, structure and use of the system and the second gave a complete description of each command provided to users of the system. The second was in loose-leaf form so that it could be easily updated and re-alphabetized. The objective of the design team was to get a good start on developing a similar pair of books for MULTICS. One caveat became clear: nothing went into the books unless Bob Graham had written it. That was a bit of a bone of contention at times, but we lived with it. It was necessary for a certain critical mass of the content of these two books to be completed before MULTICS could be put into use.

The design team met about once a month for a week to hammer out concepts and problems. As with the GECOS design team meetings, these all took place at locations remote from our regular work sites. Corby, as Fernando Corbató was called, was a good leader and kept things on track. Bob Graham's contribution I have already mentioned.

Ted Glazer was the most astonishing member of the team. He was totally blind and was never separated from his guide dog, Butch. He had an incredible range of knowledge in both hardware and software and was only slightly slowed down at times by his impairment. On occasions, say when someone was writing on the blackboard, he would say something like, "I am visualizing, from what you have said, a rectangle in the center of the blackboard, and within the rectangle are two squares representing page tables. From each of these an arrow points to the words in memory that are to be addressed. Is that correct?" In other words, he was able to construct in his mind a picture of what was happening in the meeting and would ask for verification that his picture was correct. Sometimes he would describe what he was visualizing so that it could be written and/or drawn on the blackboard by one of the sighted members of the team. He always made a positive contribution and we all admired him.

Vic Vissotsky was one of many remarkable people at Bell Labs. It was one of the outstanding experiences of my life to have worked with them. They were uniformly bright and stimulating. It was said that even the janitors at Bell Labs had Ph D degrees. Of course, it was an exaggeration, but not by far. As I recall, though, the only member of the design team from Bell Labs that always attended was Vic. I remember Peter Neumann having been there and Doug Eastwood and I think Joe Ossana, but I don't think they attended every meeting.

The geography of the project gave me a problem. The center of gravity of the participants was definitely on the east coast. The headquarters of the activity was clearly at MIT. Corby and his staff were located in a high-rise building at 545 Technology Square in Cambridge, just across the street from the MIT campus. Helmut Sassenfeld leased offices in the basement of the same building. He had a small staff. The people on his staff whose names I can remember were Braxton Ratcliff and Mauro Pacelli at MIT and Dave Levinson and myself at Bell Labs. The difficulty with this arrangement was I lived in Phoenix. Helmut relocated, but I had no intention of doing so. It ended up that I stayed most of the time in a motel in Union City, New Jersey, a few miles from Bell Labs in Murray Hill, and went home every week or two for a week.

However, as it turned out, I wasn't continuously at Murray Hill. The meetings of the MULTICS design team took place at various places in the East and often reasons arose to go up to Cambridge to meet with Helmut and his staff and/or the people at MIT. So, one way or another, I spent a great deal of time living out of a suitcase. [K]

I had become well versed in the GE-645 hardware early in the history of the project. I had made presentations on the subject to various interested people within the Computer Department and to our new employees. It turned out that the interior décor of the GE-645 was only slightly understood at Murray Hill, so one of my first duties was to give my presentation there. It was very successful and appreciated by the people in attendance. As with every other customer site I had visited, a large group at Murray Hill would have preferred if their company had been working with IBM, so those in this Bell Labs group had been making many waves internally about "riding the wrong horse". One can hardly blame them. No one had told them what the General Electric horse had to offer.

This was very good for our image with that important customer and partner, but it also set me up as the patsy to talk to other prospective customer sites after the GE-645 had been announced. These customer presentations added even more to my "away' time. However, I was glad to do it because, especially after the MULTICS plans began to develop, we had an increasingly attractive story to tell.

The GE-645 was different from any processor I had ever encountered at the time and much like those that followed it. The other 600-line machines were all limited to memories of 256k (=262,144) words. This was because the addresses in the instructions were only 18 bits in length and eighteen bits can describe only 2^{18} = 256k unique values (k = 1024). The GE-645 got around this limitation and was able to directly address many times 256k unique locations. It accomplished this by using "virtual memory".

The idea of virtual memory had been introduced in 1962 when the Ferranti Atlas [26] Computer introduced the use of paging. This permitted an address space larger than the available high-speed memory to be directly addressed. While only part of the storage corresponding to the address space would be resident in the available memory—the remainder would be resident on a backing store, usually a magnetic drum. The address space was divided into "pages" which were blocks of storage that might or might not be resident in the high-speed memory. If a word not in high-speed memory was addressed, it would cause the processor to stop running the program it had been running. This automatically removal of a program from execution was called a "page fault". When a page fault occurred, it would cause the processor to automatically run an operating system program that did the following: [L]

- Find a low-use page in memory that could be replaced.

- Start a **read** operation from backing store to replace the found page with the page containing the data at the sought address.
- Give control of the processor to another program for which all needed memory is available while the **read** operation is completed.

The GE-645 and MULTICS created virtual memory by a combination of segmentation and paging [26][27]. The hardware provided the ability for each user to have up to 256k segments each containing up to 256k words. Each segment was a separate address space of 256k words. Only pages of segments currently in use were kept in the high-speed memory of the system the remaining pages were stored on magnetic disk or drum. This swapping process between high-speed memory and backing store was carried out by MULTICS and the GE-645 hardware without the user being aware of its occurrence. (Special hardware support to facilitate this is described in Appendix J--GE 645 Addressing.) [M]

The design team made an early decision to implement MULTICS in a high-level language. Most operating systems were written in assembly language. That is, one or more lines of code existed for every machine instruction of which the operating system consisted. Use of a high-level language could reduce this amount of writing substantially and had other benefits for easy maintenance and debugging. Some vendors had experimented with this approach—primarily Burroughs. As I recall they attempted to use Algol and mixed stories circulated about how successful they had been. However, PL-1 was the latest language with everyone's attention and it was at some preliminary stage of availability. In any case, it was not available on any General Electric machine, so MIT proposed to develop an MIT version, for our equipment. It was called EPL, for Early PL-1, and was used successfully throughout the development until a complete PL-1 could be created.

The user interface of MULTICS was largely specified by CTSS. The modern reader should imagine an Intranet constrained to the limitations of the 1960s. The most obvious limitation was the types of terminals available. They were, with few exceptions, typewriter-like devices—Teletypes or the like. However, the full upper and lower case capability was utilized in MULTICS. I don't remember what character set we used. That was never a big issue. Of course, no color display or variation in fonts was possible. The font was what was molded into the print mechanism of the printer with underlining as the only embellishment—no bold face or italics.

A user would log on and log off as in a modern system with asterisks printing as the user entered his password. Once logged on, the user could

send or receive email, chat with other users, access bulletin boards, write programs in EPL or various other languages, run his own programs or look at his own files or, with proper permissions the files of others. A file system was provided that allowed the user to create a directory/file structure just as in today's systems where we have folder/file structures. A general objective of MULTICS was to make the entire machine available to each user and it came close to doing just that within resource and security constraints.

A word processor was provided but not like those of today. Because video monitors were not the norm, the word processors of the day used little short commands to instruct the system how to assemble the text the user typed. To look at the new version of the text, one had to ask to print the section in question on the paper in his terminal. This worked and one could develop a fair amount of skill in using the tool, but it lacked the get-what-you-see feature of modern word processors. A capability called "runoff" was also provided allowing the user to format documents utilizing the maximum abilities available at the users' terminals or the system's printers. To use runoff, one had to embed formatting commands in the text stream to direct the software in achieving the desired appearance.

The MULTICS design that evolved was very elegant and beautiful, and, as it turned out, very durable. As with any operating system, a part called the "Kernel" was always present in the core memory. When a user logged onto the system, the Kernel would be present to administer or reject his or her access. If the log on were successful, the kernel would provide the user with a command processor that would become part of what was called the user's "process". The process was the entity on the system representing the user during his MULTICS session. A process differed from a GECOS job in that it was a variable entity and it included only the MULTICS instructions and data the user activated. As the user executed more and different commands, the process would expand and contract as needed.

One advantage of this approach was that ways were provided to permit standard MULTICS parts to be replaced by alternatives of the user's choice. In this way, new versions of system modules (or commands) could be introduced at any time without shutting down the entire system and introducing a new version of the whole operating system. This was a system programmer's dream.

Also, the MULTICS code was all constructed as "pure procedure". That meant none of the instructions in the code were ever changed in any way. As a result many user processes could concurrently utilize a single copy of the code if each had a private copy of the data to which the

code referred. This, of course, lead to an economy of storage in a system where many users were executing the same code, for example, the central loops of a word processor. Use of pure procedure also permitted code to be reentrant. In this case the code could call upon itself to accomplish a task if, as was sometimes the case, this act would lead to an economical achievement of a result.

Each time a MULTICS user typed a command, the command processor would parse it. If the command contents were legitimate, the command processor would append the code for that command to the user's process (if it was not already there) and place it in execution. In doing these things, the command processor would utilize the File System containing the names and other attributes of all virtual memory segments available on the system. When the code for a command began execution, the virtual memory pages of the command would be brought into core memory, as they were needed. If a particular page were not in core when it was referenced, a "page fault" would occur and the desired page would be retrieved as described above.

Of course, the virtual memory pages of a process that were in real memory didn't need to be located in contiguous real memory addresses. The page tables, maintained current by the File System, indicated where each page of a segment was located. The processor translated the effective address of each instruction to refer to the actual location of the word in question regardless of where it was in high-speed memory. This made the process of memory management much easier than it would otherwise have been.

One of the big problems of the File System was to develop algorithms for the removal and replacement of pages (page turning) in real memory so as to generate the minimum possible delay. These algorithms interacted with the level of load on the machine at a given moment. Failure to manage these variables effectively could have lead to a situation in which the computer was spending most of its time moving pages into and out of memory instead of servicing the needs of its users. This is a phenomenon called "thrashing" that substantially reduces the throughput of a system by creating unwanted overhead.

MULTICS was also sensitive to data and program integrity. A comprehensive and automatic backup system ran as part of the File System. Whenever a page was turned out of memory, a copy of it was written to magnetic tape along with its identification and time recorded. However, this recording would occur only if contents of the page had changed from the time it had been created or turned into memory. The

processor automatically kept a record in the page table of whether or not the page to be removed had been changed since it was read into real memory. (See Appendix I). If a system crash occurred, the system could be restored to its state at the time of the crash by restoring the File System from the magnetic tape records to the values it contained at the time of the crash.

I am not aware of any MULTICS commands permitting a general user to utilize magnetic tapes, cardpunches, printers or similar peripherals. Commands were provided that would cause information to be transferred to/from cards and/or magnetic tape, but the actual commanding of the devices was done by MULTICS.

The boundary was never really clear to me between which of these features had been present in CTSS and which were new in MULTICS. However, they were all eventually duly recorded in the MULTICS documentation.

Quite a large portion of the designing of MULTICS occurred during the summer of 1965. Before our children were on vacation from school, I went to Helmut and asked permission to rent a house and a car in the area for the summer. The whole deal saved the company money, so he guided me through the procedure to get permission from Finance. This all went off without a hitch and we rented for the summer a schoolteacher's house in Berkeley Heights, New Jersey, conveniently located in relation to Bell Labs. This was a great opportunity for the family because on most occasions when I had to travel, we arranged for them to go along. Then while I was busy doing my job, they would go about being tourists. In this way, the family got a chance to see the Boston Area, the New Jersey area and in one case we went down to DC. Of course, we also took advantage of our proximity to New York City to visit there on a few occasions.

By the time the MULTICS design was winding down the GE-645 prototype was well on its way and Helmut was hiring people to join with the MIT staff to begin the implementation. The GE-625/35 and the 400-line were muddling along. Neither of them was spectacularly successful and I don't think the department had yet turned a profit. I believe the 200-line had been pretty much phased out; at least as far as new sales were concerned. Art Critchlow had long since left and Arnold Spielberg had accepted a job with Scientific Data Systems in the LA area (much to his son Steven Spielberg's delight, I am sure).

Throughout this time period, I continued to give presentations to potential customers. IBM had announced a product competitive to the GE-645—the IBM-360 model 67—and at every presentation, we were asked to compare ourselves to the IBM offering. Their design was said to also be capable of supporting virtual memory and large numbers of concurrent timesharing users, but our engineers and our partners in MULTICS said our approach was more sound and practical. They must have been correct because IBM eventually withdrew their offering. I don't remember the details now, but the debate went on and on. In most cases I did the best I could and then went home. However, one case occurred in which I took the consequences more seriously.

I knew a man in Phoenix with whom I had worked frequently during the early days of the 600-line and perhaps also during our work on the Y-computer. I don't remember his name, but I remember he was a little older than I, had a Ph D degree and was always pleasant and gentlemanly. At some point, he left the company and accepted a tenure-track position at Washington State University, in Pullman, Washington. At some point, he requested that I come to Pullman and present my story on the GE-645 and MULTICS. The visit was approved and I went.

I remember it was cold in Pullman and snow was on the ground. I met my old friend and he welcomed me with great warmth. He introduced me to several of his graduate students and we had a very pleasant reunion. When it came time for the presentation, I gave the one I had given many times and it went well until about halfway through the question and answer session. One young lady in the audience kept insisting thrashing would be such a difficult problem that virtual memory wouldn't work. I gave her the best argument I had been taught to give but she persisted. I finally just lost my patience and responded to her in a very ugly fashion. That cast a negative aura on the session and it ended shortly thereafter. I never again heard from my former friend from Washington State University. (I don't think they were serious customers anyway, but I was sorry to see the termination of what I had considered a valuable friendship.)

At some time after our summer in the east but still, I think in 1965, Bell Labs withdrew from the project. I am not sure of the reasons. Although squabbles had occurred between a few members of the MULTICS Design Team and individuals within the other organizations, I never witnessed anything that would have led to a rupture between the partners. In any event, the Triumvirate became a Biumvirate and we continued on our way.

However, this left me without a major focus. At about this time, people began to think about what we were going to run on the prototype GE-645 hardware whenever it was able to get up and running. Of course, test and diagnostic programs would be written to exercise the machine as it was being assembled. But what would be run after that? MULTICS was not scheduled to be completed or in test until quite some time after the prototype was up and running. Anyway, testing a machine and its operating system concurrently was not a pleasant prospect, as many of us knew. Therefore, it was decided we would create a version of GECOS II, 645 GECOS, which would run on the GE-645 hardware. The responsibility for doing this was given to me in Phoenix.

I really don't remember where this job was done, in Cambridge or Phoenix and I don't remember who worked on it. I think I continued to report to Helmut Sassenfeld, but I'm not sure. It was really a small job, though to people not familiar with the GE-645 hardware it seemed quite difficult. What we did was use the same registers, descriptors and page tables intended to create virtual memory in MULTICS to simulate the address fences and relocation capabilities of the GE-625/35. This required only a small amount of programming and the hardware did the rest. For those who were not familiar with the GE-645 hardware in detail, it looked like magic.

This job was completed by April of 1966. That was the end of my connection with the GE-645 and MULTICS and, I am sorry to say, almost the end of my connection with Helmut Sassenfeld. I believe our relationship had been mutually beneficial and satisfying. The fact that his name occurs rarely in this history is emblematic of our relationship. Our contacts were always friendly and positive. His management style was always supportive and we discussed all subjects openly and without inhibition. He ranked with Harry Tellier in terms of management skills if not in philosophical leadership.

After GE-645 GECOS was completed, I again reported to Ed Vance. I was given a unit to manage called 600-line System Support. We were receiving and tracking customer problem reports, keeping the customers posted as their problems were addressed, distributing software corrections, as they became available and creating system documentation. This was a challenge different from any I had previously had. It was kind of a mess when I took it over.

The easy part was documentation. Ed Vance had hired a man named John Maynard who was a great writer and had in depth knowledge

about computers. He was a real pro and a very nice person and all I needed to do was give him responsibility and stand back. He did a beautiful job.

I don't remember whether or not I had responsibility for the maintenance of the library of completed software versions. This was a huge job and was undertaken very ably by Bob Jordan, a former employee of the Jet Engine department in Cincinnati. Bob's organization maintained the very extensive library of the piece parts from which each software release was made. The people in his organization were the first users to assemble these parts and test them out as a whole before distribution. They had to try to make sure each version of the software, as released, would work on all of the different hardware configurations in the field. They then had to deliver the releases to all the installations. They were also responsible for sending out what were called "patches" between releases. If I was not responsible for this group, I worked very closely with them.

I do remember I was responsible for processing all of the Ck-97s that came in for the 600-line. A Ck-97 was a form upon which a customer would record a system error or malfunction—in other words, a complaint. When I took over the unit, the procedure for handling the Ck-97s was kind of like a leaky bucket—some complaints would go unanswered, their receipt would often not be acknowledged and much customer dissatisfaction resulted. The solution to the problem was to create a database of pending Ck-97s and a system that would prioritize their acknowledgements and responses and would report periodically on the progress in solving each one.

The submitters of the complaints had to be kept informed throughout this process. We were committed to provide a solution or a work-around in each case. Unfortunately, it was necessary, in some cases, to tell the customers they would need to wait until the next release of the offending software to obtain a solution to their problem. These were the cases requiring the greatest diplomacy in their communication.

This job required a great deal of human relations skill because the customers were usually not in a good mood and our management was disappointed that our products were not perfect. In addition, our developers didn't want to spend their time and energy on that old stuff they thought they had completed and would never see again. Yet every complaint had to be dealt with and answered with as little ruffling of feathers as possible. The key was keeping up the level and quality of communication, but that required a great deal of discipline.

During all of the time since Charlie Bachman had joined the company, I had seen him once in a while, but we were both very busy and had little time to socialize. His IDS, Integrated Data Store, had become an integral part of our product offerings and was in regular use in many customer sites. One of these was at the Weyerhaeuser Company in Tacoma, Washington. They had had 200-line equipment for a number of years and were big IDS users. They were about to convert to the 600-line in a big way and the GE Computer Department was looking for someone to manage the development of some special purpose software for them. Charlie threw my name into the hat as a candidate.

I had had the System Support job for about ten months and had made some progress, but I wasn't really happy in that line of work. I put out feelers to Harry Tellier who was by then with IBM, Helmut Sassenfeld with RCA and Ken Robertson who by then was working for Douglas United Nuclear. I interviewed and got an offer from Douglas Aeronautics and Space Division. I even contacted Vic Vissotsky, thinking I might get a janitorial job at Bell Labs, but that lead to nothing. However, I was really interested in what was being done at Weyerhaeuser, so I agreed to take a look at it. But that is another story I shall tell in the next chapter.

WEYCOS

When I was a small boy, a big story appeared in the news shortly after the Lindberg kidnapping. Another kidnapping had occurred: the scion of the Weyerhaeuser family, George Weyerhaeuser, had been kidnapped. The Weyerhaeusers ran and controlled the company that bore their name. It was one of the largest forest products companies in the country and the family was thought of as being very rich. As it turned out, a great deal of publicity and public anxiety was generated, but little George was safely recovered.

Years later, George became the President of the company bearing his family's name. It happened that he and Lou Hereford sat next to one another on an airplane. At the time, Lou worked for the GE Internal Automation Operation (IAO) in Schenectady. Lou gave George the story of the dream system we had proposed for Hanford years before. George was very interested and after more discussion and meetings, the Weyerhaeuser Company ordered a feasibility study to be performed by IAO. Its mission was to investigate means whereby the order processing and inventory control in the Weyerhaeuser Wood Products Division might be streamlined. Both the company headquarters and the headquarters of the Wood Products Division were then located in Tacoma, Washington.

Lou and Bill Helgeson, also from IAO, carried out the feasibility study. Bill Helgeson knew Charlie Bachman and was intimately knowledgeable about the General Electric data management product, IDS (Integrated Data Store), and its implementation. I like to believe Lou and Bill dreamed an updated version of the old Hanford dream; this time it was not just a fantasy but also the beginnings of a plan to be implemented for the Weyerhaeuser Company.

The Weyerhaeuser Company accepted the plan presented in the feasibility study and when that happened, Lou changed employers and took charge of Weyerhaeuser information and communication systems. A large GE-235 installation with a DATANET-30 communications computer (a combination known as the GE-265) was planned for installation in Tacoma. This system would communicate with terminals throughout the Weyerhaeuser Company. Any order taken anywhere in the company would come in to Tacoma via a torn-tape Teletype system and be recorded in an IDS database. The filling of the order would be recorded and its delivery and billing would similarly be followed in the database. In

modern parlance, they planned to create an "intranet" for the running of the business of the Wood Products Division.

By this time, IDS was a stable product but was designed to obtain its data in the form of punched cards. It also lacked a recovery capability that would protect the data contained in the database from corruption or destruction. Hence, modifications were required to support the on-line acquisition of data by way of the DATANET-30s. Grayce Booth of Weyerhaeuser and Paul Kosct of GE (who also came from IAO) implemented these modifications in 1964 with some help from Bill Helgeson. In addition, Grayce and Paul maintained the data integrity of the system after it was put into operation. They were on 24-hour call during the life of the system to restore the database and restart the computers in the event of any mishaps.

The initial on-line application was limited to order entry, order acknowledgement, shipping and invoicing. It was highly dependent on IDS. Even the queuing of orders and their prioritizing and selection for processing was handled by a "Problem Controller" developed by Charlie Bachman for manufacturing control applications within GE, then re-written for the on-line system by Grayce Booth of Weyerhaeuser.

When put into operation in 1965, the system was very successful, but it had one problem—no one from computer operations could tell what it was doing. As originally installed, its only operator interface was the set of switches on the computer console. No output was available to the operators. The system just sat in the corner, 24 hours a day, seven days a week and blinked its lights at the operators. The operators were very nervous because they could not tell whether it was operating correctly or not. So, to relieve the tension in the computer room, an electric typewriter was installed and set up for the Problem Controller to type out the length of the problem queue once an hour so that the operators could tell something was going on. Additionally, a method was provided of suspending on-line processing in the evening hours for Batch Processing Applications requiring the IDS database.

At one point in time, according to Allen Smith, Vice President of Operations, Weyerhaeuser [28] encountered a big influx of orders. Orders flowed in faster than they could be processed. They were stored in the IDS database. All week long, the order-processing backlog would build up. However, over the weekend, the order inflow stopped and the system kept running, 24 hours a day, and the order backlog was worked off before Monday morning. He said what amazed him, was the higher-

priority problem classes, such as assigning customer numbers to new customers, were returning new customer numbers in less than a minute when one was requested by the sales offices. This was excellent performance when one considers that telecommunication was being handled with Teletype machines in the sales and shipping offices.

The original Weyerhaeuser system was later referred to as WEYCOS I. It was quite successful but stories such as the preceding made it clear the system lacked the capacity to continue to perform the job if any workload increase occurred. However, the company was pleased with the technological path it was pursuing and was anxious to look for a way to expand the system's capacity to give better service to more users within the company. This led to the plan to develop WEYCOS II, a large GE 600-line installation with modified software that would continue and expand the capabilities of WEYCOS I.

As the plan evolved, the computer system in Tacoma would continue to serve the Wood Products Division as with WEYCOS I, but with additional capabilities. There was also an expectation that enough capacity would be available to serve some of the other divisions of the company in a similar manner.

By the time anyone had talked to me about this project, the idea was already implanted in the minds of the Weyerhaeuser personnel and the affected GE personnel. My old friend from Hanford, Jim Fichten, was now working for Lou Hereford in Tacoma and Chuck Buchanan was working for Weyerhaeuser in their Chicago office. A preliminary plan for the GE-635 system involved the sale of many (I've forgotten how many) millions of dollars worth of equipment. However, the standard software was not up to doing the job the customer wanted to do, so a special version of GECOS was to be developed that would fill the bill. This modified version of GECOS would be WEYCOS II to support just the Weyerhaeuser requirements. Agreement between the two companies was achieved during July of 1966 when Stan Williams, representing GE Marketing, and Joe Handros, GE Computer Department Legal Counsel, met with their Weyerhaeuser counterparts in Tacoma. The agreement to proceed with a joint development between the two companies was reached and signed into being during this visit.

Some recruitment of staff took place in 1966 both within GE and within Weyerhaeuser. Of course, Grayce Booth and Paul Kosct continued on the project but now on different hardware and a different final system target. Bill Conlen , Jim Everett and Don Faul were team members on the Weyerhaeuser side. On the GE side, my old friend Ed Roddy had been

transferred from Phoenix, as was Bob Hobbs from early GECOS days. Ron Pulfer was also there and Dick Carlyle had been transferred from Phoenix. I had met Dick in Phoenix, but we had never before worked together.

These people all had their offices on the second floor of the building right across the street from the Weyerhaeuser Building in Tacoma. Lou Hereford had his office there and the entire second floor was available except for a little tailor shop with which several of us did business. Charlie Bachman, "Mr. IDS, himself" and Ron Pulfer came to Tacoma during the week each week starting with the signing of the joint agreement and Charlie and Ron Pulfer flew home to Phoenix on the weekends. [N]

At some time early in 1967, I went to Tacoma on an interview—I should say a seduction. They had my office already furnished with my name on the door. Everyone gave me a hero's welcome. I was introduced to everyone and was told everything known at the time about the project. It was very flattering and the project was extremely challenging. I couldn't turn it down.

After I accepted the job, I learned it would result in a nice promotion for me. I reported to work around the first of March in 1967. It was no longer necessary for Charlie Bachman to commute to Tacoma every week.

Ron Pulfer and I stayed during the week in the two-bedroom apartment in Tacoma he and Charlie had formerly occupied. We batched there until my family came up in June after the end of the school year in Phoenix. In the meantime, we sold our house in Phoenix and bought a new one in the suburb of Fircrest a few miles from downtown Tacoma. We didn't move into our house until after I brought my family up from Phoenix. In the meantime, I spent the weekdays in Tacoma and the weekends in Phoenix. Everyone told me a beautiful view of Mount Rainier could be enjoyed from Tacoma, but the weather was such that I had never seen it until the day my family and I rolled into the area. Starting at Fort Lewis, which is just south of Tacoma, the mountain was dead ahead of us on the horizon—a gorgeous view we never tired of seeing.

As soon as I arrived in Tacoma some key decisions had to be made. But before I describe them, readers need to understand some things going on in Phoenix. As computer technology proceeded, the demand for timesharing capability steadily increased. More and more people wanted to do their work from a terminal even if that terminal was just a typewriter-like device. The only facility GECOS II had to support that capability was a time-sharing supervisor executed just like a user job. This really didn't give the same level of service some of our competitors'

systems were able to achieve in the timesharing mode. The development of GECOS III had been proposed to take care of this problem and several others. A big debate was occurring in Phoenix as to whether it would be better to undertake a large revision to GECOS II or to start over with GECOS III. George Gillette, a Hanford transplant, led the fight for revision of GECOS II, but, sadly, he and his followers lost.

This issue was of great moment to us at Weyerhaeuser because we had no wish to remake an operating system from scratch. It was our hope we could achieve our goals by modifications to GECOS II. Indeed, no other practical alternative existed because GECOS III didn't yet exist and waiting for it was not acceptable to the customer. So one of our first tasks was to work through this issue and convince ourselves GECOS II was the base from which we would start.

We also needed to delineate responsibilities between GE and Weyerhaeuser. Since I had seen him last, Lou Hereford had become very involved in computer communications and had a staff of competent people working on that side of the problem. Some discussion of me also assuming responsibility for computer communications took place, but I was not anxious to do so. I had plenty to challenge me with WEYCOS. Hence, it was decided WEYCOS was my responsibility and the staff Lou already had on board would handle data communications and they and we would report to Lou.

The next step, at least as far as I was concerned, was to make sure just what WEYCOS was going to be. So we went about painting the picture of the system. The primary issue to be addressed in WEYCOS II was system capacity. To understand some of the issues relating to this problem it is necessary to know something about what is involved in performing a database access. One can separate the time spent in performing a database access into four parts:

1. Time spent by the CPU executing the logic of the program performing the access.
2. Time spent waiting for the read/write heads of the disk to be positioned over the proper disk and track.
3. Time spent waiting for the disk to rotate to the sector containing the sought-after data.
4. Time spent transferring data between memory and the device where it is stored.

Having made this separation, it is easy to show for most computers and disk handlers; the sum of parts 1 and 4 is much smaller than the sum of parts 2 and 3. That is, the system spends most of its time waiting for mechanical devices to move and very little time deciding which device is to access what data and disposing of the sought-after data.

In changing from a GE-235 system to a GE-635 system, the increase in CPU speed was greater than the increase in the speeds of the disk drives. Therefore, how were we going to gain a large increment in performance?

The answer was the GE-235 system was uniprogrammed. That is, with few exceptions, each of the tasks to be performed to achieve a database access was done in sequence. By contrast, the GE-635 system was multi-programmed. Hence, while the CPU was waiting for a mechanical device to move, it could be dispatched to work on another program, when that program had to wait for a mechanical device to move, it could be dispatched to another program, etc. In this way, the CPU could be more fully utilized, but, more importantly, several disk handlers could be set in motion in parallel looking for sought-after sectors. This multiplicity of disk activity was the major factor in achieving the necessary improvement in throughput in WEYCOS.

This meant the primary objective was within reach, but at the cost of some programming complication. Take an airline reservation system as an example of why some of this complication was necessary. Suppose you have a database with multi-programmed access and two customers looking for a seat on a plane with only one seat available. The first customer asks to be reserved for the available seat and before the transaction is closed, the other customer asks for the same seat. Because the transaction is not closed, the seat is still available, and, hence, it is sold to two customers. Access to the "number of seats available" record must be locked whenever it is accessed with the possibility of being updated. This can be done (and was done in WEYCOS) but it leads to another problem. It is possible that two programs each accessing the same database can lock records such that neither of them can proceed—they become deadlocked. It was necessary to put "deadlock protection" into WEYCOS, to prevent this from happening.

In addition to the increased throughput resulting from greater parallelism, the disk handlers used on the GE-635 system were larger and faster than those on the GE-235 system. They were our newest and best disk handlers—the DSU-270s. I don't remember their specs, but they were considered quite large and fast for their time. We at Weyerhaeuser had many of them.

However, we needed to do more than just access a large database rapidly. We also needed to be able to restore it in the event of a failure of any type. The solution to the recovery problem involved both hardware and software. In the hardware sphere, we created what was called a "multi-wing configuration". This took advantage of the memory orientation of the 600-line machines and permitted up to four GE-635 machines in a configuration, all of which had access to a common module of 64k (=65,536) words of memory. The in-memory buffer storage for the IDS database would be kept in this common area. Hence, the processors and IOCs (input/output controllers) in the wings could be added and removed from the configuration and only rarely require disabling of the entire system. Figure 4 shows the major components of the Multi-Wing WEYCOS hardware system. Not shown are the input/output devices and their controllers.

"P1" relates to the first processor. "I1" relates to the first IOC. The first processor and the first IOC have exclusive access to "Memory 1", which represents three memory modules. The "Common Memory" represents the single memory module addressable by all four processors and all four IOCs.

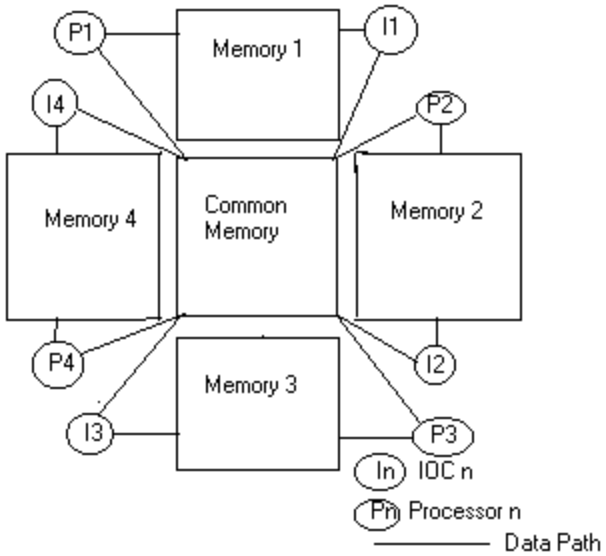

Figure 4. WEYCOS Multi-Wing Hardware Configuration

This configuration could be supported with off-the-shelf hardware except for one small hardware feature. A mechanism was needed to prevent interference between active devices accessing the Common Memory.

This mechanism was designed but never installed on any of the equipment.

The software also had to be able to withstand outages with a minimum of disruption and in a more automatic manner than in WEYCOS I. Hence, an extensive Recovery and Restart (R & R) system was developed in the WEYCOS software.

Unlike the MULTICS backup system that had hardware assistance, the WEYCOS R & R system was required to backup database content from copies of logical records stored in the IDS database. Copies of records, both before and after modification were stored on magnetic tape and processed to permit a restart from any specified clean point. A clean point was a point at which the database was known to be intact and without error. A great many details of this process escape my memory, but I do remember four forms of recovery were provided, depending upon the severity of the failure. Each of these was given a name: "normal recovery" that was essentially automatic; "emergency recovery" that required some human intervention; "penultimate recovery" that required substantial rollback and reprocessing; and "ultimate recovery" that was essentially whatever could be worked out if penultimate recovery didn't work. Each of these successive layers of R&R was engaged if the preceding layer had been unsuccessful; however, it was expected that the frequency of occurrence of the four layers would diminish substantially from normal recovery to ultimate recovery, so human intervention would be minimized.

The project made progress as soon as we had our staff assembled. However, the progress was too slow. It was too slow to suit the customer, too slow to suit the employees and too slow for me. We tried everything we could think of to get it on a faster track, but nothing worked. When adequate debug time was unavailable, another system was ordered to provide the time. When insufficient visibility of project tasks and progress within tasks was suspected, we started using the critical path method for tracking progress and created an elaborate display for everyone to see in our hallway showing where every task stood at any time. Weyerhaeuser hired psychologists and put all of their managers, including me, through multiple sessions of sensitivity training in the hope we would learn to more effectively deal with each other and/or our employees. (Perhaps the fact that anyone associated with the project was referred to as one of "Hereford's Raiders" suggested this approach.) None of these things seemed to help.

On the other hand, it should have been a surprise to no one that trying to fit an innovative development into a fixed time block is a fool's errand. Plenty of history existed within the computer industry and other technical endeavors to make the folly of this practice apparent. I blame myself for having been coerced into giving time targets in advance. Of course, we were operating within business organizations operating on the basis of budgets set in advance and this was nothing new to me so I caved in to the pressure. I can say, though, I have never seen such an effort made by any management team to keep on schedule. [O]

After the project had been going for about a year, we had a visit from our GE Department General Manager—George Woodward. Our project had always been unpopular in Phoenix because the working-level people and the low-level management viewed us as a "brain drain" and a drag on our department's limited resources. When George Woodward arrived, he seemed to be reflecting these same attitudes. However, by lunchtime, we had had time to tell him our story and explain what we were doing. By the time we had returned from our midday repast, it was clear we had a convert. He began to smile frequently and actually became quite enthused about what we were doing. This helped to bolster our position in Phoenix; however, of course, George didn't last very long.

As it turned out, his replacement was probably the best General Manager we ever had—John Haanstra. Yes, the man who was the young seaman at Point Mugu was now our General Manager. He had spent many of the intervening years with IBM and was an excellent choice for his new job. He too came to visit us at one point and although impressed, was not as much so as George had been. Nevertheless, for the first time since I had worked for GE, I felt we had a man at the helm that really knew what was going on and what was important for our business and technical success. Unfortunately, shortly after he visited us he crashed his private airplane with his wife and child aboard and they were all killed. I believe this happened in the summer of 1969.

However, the worst blow to the project was to occur in late 1968 or 1969. The person who really kept everything going and was the thought leader of the entire endeavor, Lou Hereford, died. I had found out during one of our sensitivity sessions that Lou had Hodgkin's disease. He had apparently had it since our days at Hanford. He would often disappear for a few days or a week, but convenient cover stories were always provided. We found out later, he had been in Seattle getting chemotherapy.

Even while Lou was alive, some people were lobbying to shrink the project or to discontinue it. People in Phoenix argued against the project as

described above. People at other Weyerhaeuser sites would have preferred to see a more decentralized way of running the business. Throughout the Weyerhaeuser Company, as with any customer, some people would have liked to see GE thrown out and replaced by IBM. Lou always led the defense against these assaults and now we were getting pressure from within the sponsoring organizations to bring the project to a conclusion.

The decision was made to complete single-wing WEYCOS and then discontinue the project. That seemed to me to be a good practical decision given all of the negative pressure. However, the decision didn't change anything we were doing. We had been viewing the achievement of single-wing WEYCOS, to be a first-stage target in leading to the more ambitious goal of multi-wing WEYCOS. So we had simply redefined the definition of completion. We continued in our efforts and WEYCOS gradually sputtered into life in 1969. Of course, maintenance and bug fixing efforts would continue for some time to come, but it was opined the Weyerhaeuser employees that had reported to me during the development would be able to handle those needs. [P]

As WEYCOS II was coming to life, talk of GE selling the Computer Department to Honeywell persisted. That is what eventually happened and the transition was scheduled on October 1, 1970. So most of us GE employees were expecting to have a new job in the fall and we were expecting it to be working for a new employer.

As for the McGee family, we were not anxious to leave Tacoma. We all liked our house, the two children still at home had a very good situation in their school and we all enjoyed living in the area. Unfortunately, Boeing was going through layoffs at the time and it was a bad time to be looking for a job in the Seattle area. I tried to find a job at Weyerhaeuser, but no one was interested. So we put our very dearly loved house on the market and prepared to move back to Phoenix.

At some time during 1969, I was asked to come to Hollywood, Florida to participate in a meeting on the proposed "new product line" for GE. None other than Richard M Block headed the new development—the man who led the development of RAYDAC in the 1950's. My first impression of the meeting was that this was undoubtedly the biggest boondoggle I had ever witnessed. The approach being taken was more lavish than anything I had ever seen, yet I was shown very little evidence of progress toward a realistic design. My main recollection of the meeting, which was called Shangri-La, is the hundreds of land crabs roaming around the grounds at night. However, my opinion was not

universally held. Here is the description of one of those who took part in the meeting [29]:

> "The Shangri-La meeting was scheduled to last three months and had participants from GE Bull (France), GE Italia (Italy) and the GE Computer Department in Phoenix. The meetings were held at the Diplomat Hotel, in Hollywood, Florida. Participants came for the summer with their families while schools were not in session. An extra fourth month was tacked on for the employees without their families to finish up the report.
>
> "Was that meeting a success? Yes and No! When the effort to build the new computer line was estimated and top GE management notified, they decided to bail out of the computer business itself. The consequence was that GE sold its computer business the next year to Honeywell who went forward to implement the Shangri-La developed plans."

Early in 1968[30], Walker Dix and his entire organization had transferred to Phoenix. Walker became the Manager of Engineering of our Department. I had reported to him since his arrival. One day he gave me a call and said a job was available at the AMEX, American Stock Exchange, and would I be interested. The job consisted of developing a new electronic trading system for the AMEX. It would be something like the Weyerhaeuser project, and I was being looked at as its manager. I was definitely interested. So we moved to Phoenix with the prospect my future might be in another project, this time in New York.

VMM

As it turned out, neither General Electric nor Honeywell was awarded the AMEX contract. But we did arrive in Phoenix in the summer of 1970 and moved into yet another new domicile. At the time, I reported directly to Walker Dix, the Manager of Engineering. I had the title Senior Consulting Engineer. Two of us had this title and reporting relationship—the other was Jim Wilde, a man from England with a beautiful beard. We had adjoining offices close to Walker's and shared a secretary.

My duties were rather vague. When I arrived, Walker said something to me like, "I want you to do something about this software mess". As usual, the software types had a bad odor. Everything was behind schedule and nothing was working the way people wanted. I knew of nothing I could do to change what was going on in the software organizations, but agreed to try.

To me the bigger problems were with the combined product line we had to try to manage and profit from. We had the General Electric 400-line that still existed and was not particularly successful, the General Electric 600-line consisting of the GE-615/625/635 (the 615 had been added) running partially under GECOS II and partially under GECOS III and the GE-645 running under MULTICS. We now also had inherited the Honeywell 800 and the Honeywell New Product Line, NPL, specified at the GE Shangri-La meetings. The Honeywell 800 seemed to be mature and not growing, but the NPL was a whole new universe.

The GE-Bull participants from France were the main proponents of the NPL as it was conceived at Shangri-La but people from GE-Italia and our own people from the GE Computer Department in Phoenix contributed to its design. We in Phoenix had always had an information exchange agreement with Toshiba in Japan and Honeywell had a similar arrangement with Nippon Electric. I am not sure whether or not Toshiba had any interest in the NPL after the merger, but I do remember they had representatives in Phoenix then. NEC also participated in NPL activities after the merger. So now the parties to the NPL were people from France, Italy, Japan, Billerica (the principal Honeywell site in Massachusetts) and Phoenix.

When I looked at the NPL, I found a brand new computer architecture and interior décor including many of the latest and most advanced concepts in the industry. However, it existed mostly in the brains of the various participants and their individual conceptions were not identical in many instances. Also, the software definition was very skimpy and ambitious. The magnitude of the undertaking was frightening.

The "party line" was that we would all learn about the NPL and we would pull together as a single company to create a unified product. Ugo Gagliardi, Walker Dix's counterpart in Billerica had directed the NPL development since the merger. (In addition, he either was or had been a professor at Harvard.) Phoenix software developers were expected to contribute to the NPL software, but just how the efforts in Phoenix, Billerica, France, Italy and Japan would be integrated was not clear.

Walker suggested to me that I should spend some time in Billerica and get to know some of their people and participate in some of their NPL meetings (that included overseas participants). He also arranged for Ugo and me to have lunch together during my visits. I did as suggested and read everything I could get my hands on about the NPL so I could talk to these people. I found their meetings interesting and productive. They were generating what I considered to be useful documentation as a byproduct of their meetings. I also found my meetings with Ugo to be pleasant, but it was not at all clear to me where all of this was leading and how Phoenix was going to get "plugged into" this process.

At some point while I was making these trips and trying to get reestablished with the Phoenix organization, I had a visit from Richard Carlyle who had been with us in Tacoma. We were just chatting about this entire morass of products and he said something like, "I don't see why we couldn't build a machine and a Virtual Machine Monitor that could run more than one of these systems. Then we could run GECOS and MULTICS, or GECOS and NPL, or any combination we would like—each in its own virtual machine. We could permit existing users who wanted to grow from GECOS to NPL, say, to continue using GECOS while they conducted a leisurely conversion to NPL."

On the face of it, this seems like a new complication, but I kept thinking about Harry Tellier emphasizing the terrible difficulty of performing a conversion. In changing systems, high cost, dissatisfied customers and much confusion and frustration were inevitably involved. Of course, the biggest hazard of introducing a new product line is "pulling the rug from beneath" the existing customer base. If the Carlyle suggestion could be

implemented, we could protect our existing customers and use the superior characteristics of the NPL to attract new ones.

After my conversation with Dick Carlyle, I began to think about the experience we had implementing 645 GECOS. It was a simple job. We intercepted the execution of programs at a few critical points and, through the use of some carefully crafted programs, made the intercepted programs think they were operating within the interior décor for which they were written. In fact, the programs were operating within a different interior décor and what they were experiencing was merely a simulation. The important part was the programs worked; they worked correctly and efficiently. In effect, we had created a virtual GE-635 machine on a real GE-645 machine. We could almost as easily have created multiple virtual GE-635 machines on a single real GE-645 machine with a virtual GE-645 thrown in for good measure.

I thought this idea might sell well within the Honeywell environment because I remembered that at one point in their history; Honeywell had introduced a new machine with the interior décor of the IBM-1401. This permitted them to attack the market of the 1401 users who had no growth path offered them by IBM. Honeywell offered them growth without the cost of a conversion they would have had to make if they had moved to higher-powered IBM equipment.

I put together a proposal describing a new machine including the interior decors of both the GE-615/625/635 and the NPL. This proposal wouldn't have been possible a decade before, because the cost of the dual personalities would have been prohibitive, but with the arrival of semiconductor technology, such limitations were rapidly falling away. In any case, I submitted my proposal to Walker and he bought it. It was known as the VMM (Virtual Machine Monitor) proposal because of the software that would exist at the heart of the system to control all of the hardware and the software it contained.

After Walker had accepted my proposal, he did what people who knew him expected he would do. He got in touch with his friends in Finance and evaluated the financial impact implementation of this proposal might have on the Corporation over the next several years. All of this financial thought and analysis was distilled into a few spreadsheets Walker used as his focal point in selling the idea to everyone of importance. The spreadsheets, of course, showed that our profits would be increased and our liabilities reduced. If this had not been the case, the proposal would have been dropped.

This is what Walker did; what he didn't do was anything at all to get the project moving. Talk of the project spread far and wide both in Phoenix and also in Billerica, Paris and (I assume) Tokyo, but not a stone was turned to get the project started. Of course, strong resistance to the idea existed in many circles. The Marketing people didn't like it because it made them rethink all of their current plans. Some of the Billerica and Paris people didn't like it because it seemed to take some of the luster from the NPL. Some people didn't like it because they didn't understand it or because their brains were full of other thoughts and didn't want be bothered with another one. [Q]

It should be understood that at this time, the 600-line was the most successful computer product in the new company and protection of the 600-line customer base was of great importance. That issue had motivated my proposal from the start. At the same time, we had a new product line coming along to which we would have liked to see many of these customers convert. The big question was: How would the customers carry on their business while their programs were being rewritten for the NPL? The VMM provided an answer to this question! They needn't convert if they didn't want to; they could continue on the new equipment in 600 mode. They could then write all of their new programs in the language of NPL and take advantage of its improved capabilities.

This argument had one flaw. If a user had a database on the old equipment he wanted to access from the NPL programs how was he to do this? This is a very annoying problem, but it is not insurmountable. I wrote a paper on methods by which this problem could be addressed and dealt with, and asked for comments. The silence was deafening. I doubt if many people read it and certainly no one replied to it.

On a particular day while the acceptance of the VMM was still, to some extent, in doubt, a meeting was held at which Walker gave his presentation to a group of brass from Honeywell in the east. Present at the meeting was a group of our Marketing foot-draggers, among whom I was sitting. During Walker's presentation, the Marketing people kept talking to me and making distracting remarks. I tried to be off-hand about the whole thing and to keep a good humor. At some point, Walker asked me a question that I missed because of the confusion caused by the Marketing group. Instead of casting the blame upon them, on the spot, I responded in a flip manner by saying, "What was the question?" Walker repeated the question and I answered it, but he was clearly annoyed with me. I think our relationship ended right there--vacuous as it had been before.

During these early days of the VMM project, I continued to go to Billerica and have lunch with Ugo and attend the NPL meetings. Ugo was friendly towards the VMM proposal and as soon as he learned of it he arranged for me to meet two of his employees. They had gotten their Ph D degrees under his tutelage. They were Harold Schwenk and Robert Goldberg. The subject of Bob's Ph D dissertation was Virtual Machines. We immediately became fast friends and allies. It turned out Bob Goldberg was working on putting together a Symposium on Virtual Machines at just about the time I met him. I, of course, attended, but didn't have a paper to deliver.

A new version of the 600-line was announced not long after the merger with Honeywell. It was called the Honeywell 6000-line and was the old General Electric 600-line re-implemented in semiconductor technology. In its new manifestation the 615/625/635 machines were known as the H6030, H6050 and H6070. The old GE-645 became the H6180. The 6000-line machines sold like hotcakes and we began seeing success like none we had seen in the past.

The Engineering Student Intern Program was started within General Electric and continued within Honeywell. In this program, promising engineering students, in our case juniors and seniors in engineering at Arizona State University, were hired and worked part time until their graduation. Most of them were then hired as full-time employees. While they were in the program they underwent some special in-house training that acquainted them with our company and methods and also worked on three-month assignments with various organizations within our plant. These interns called themselves "Turkeys" for reasons I never understood. In any case, in the absence of Dix doing anything to get the VMM going, I hired some Turkeys (they were not charged to the hirer's cost center) and started working on implementation of the 6180 VMM.

At some point, I was given one regular employee to join in this effort. So we had one actual VMM project in operation.

During the summer in which this effort was going forward, we had some kind of conference in the Phoenix area attended by an important guest: Dr. Painter, from the Defense Department. I was asked to come to the meeting and give a private presentation to Dr. Painter on the VMM. I did so and he liked it very much. The idea was the Department of Defense might be interested in the VMM for reasons not clear to me. However, before I left the company of the good doctor, he emphasized he would want to see the VMM kept simple. I don't remember much else he said, but the "keep it simple" aspect stayed with me. [R]

At some time in early 1972, Walker Dix was promoted to Vice President of Engineering. I don't remember the organization chart but I guess that meant he was the highest-ranking computer engineer in the company. Lee Wilkinson replaced him as Director of Engineering in our Phoenix Division and I ended up reporting to John Couleur who reported to Lee and had an organization called Special Projects Engineering. This was a demotion for me because I now appeared two levels below the Director of Engineering in the organization chart whereas I had previously been a direct report. The worst part of it was that, a short time thereafter, I lost my privilege of parking in front of the building and had to park in the mega parking lot surrounding the west and north of the factory. However, these things did not give me great angst because I got a good performance appraisal in July of 1973 and my project was moving ahead well.

Shortly after my 1973 performance appraisal, I was transferred to the Multics Development Project under Bob Montee, a person I hardly knew. Being part of the MULTICS world again certainly was no problem for me, and the move didn't change my working situation in any important way. I might have taken note, however, that Montee had come from FSO, the Federal Systems Operation whose sole function was to sell Honeywell Products to the Federal Government.

At some time shortly after we got the VMM running on the H6180 hardware in the factory, we received an order through our Federal Systems Operation in Washington, D. C. for a VMM for use by Grifiss Air Force Base in Rome, New York. This order was duly filled and delivered at a cost to the customer of $100,000. The VMM also became a required deliverable in a MULTICS System sold to the University of Southwest Louisiana in Lafayette.

As I recall it, we received a contract for $25,000 shortly after the delivery to Griffis Air Force Base to perform a VMM Study. The thrust of the study was for us to evaluate the VMM against a set of requirements. The implication was if we were to come up with the correct set of answers, a prospect existed of selling several systems to the Department of Defense including the VMM. In executing the study, we were to work with a consultant from Los Angeles.

The consultant from Los Angeles was a man I had known in SHARE named Ascher Opler. It was pleasant working with him and he was very sharp. I was determined to go down the path of the VMM we had already installed and had in operation. He wanted to add some hardware changes responsive to some security requirement. I dragged my feet

because I kept thinking of Dr. Painter's mandate to "keep it simple". The consultant and I agreed to disagree on this subject, but he

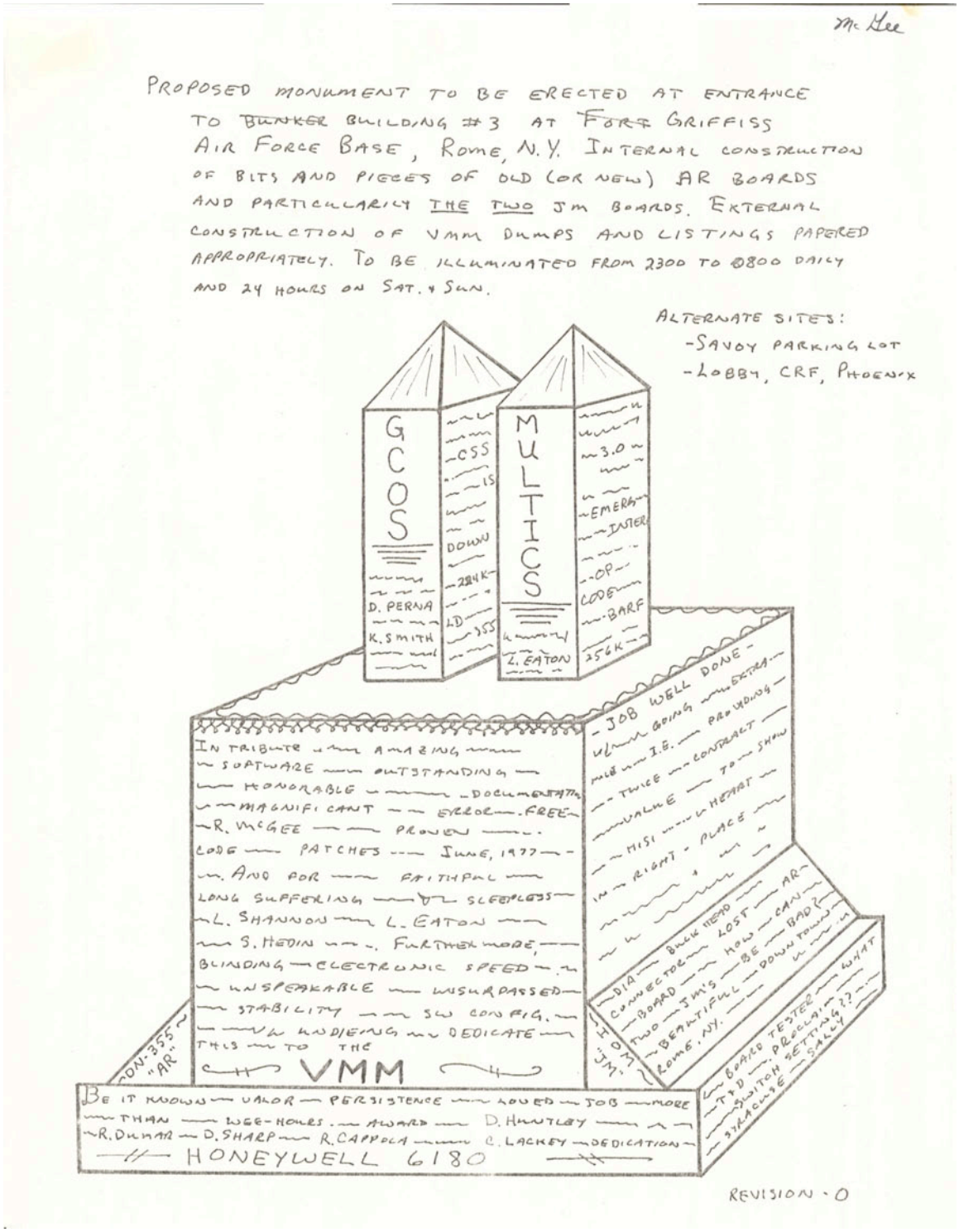

Image 5. Monument proposed in jest for Griffis Air Force Base

continued to stress this point until the time the final report had to be submitted. It was my contract and I submitted the report in the way I wanted.

If I had agreed with the consultant, it would have involved making hardware changes I was sure the engineers would resist. At the time, we were under very tight budget constraints and I knew they were already strapped for manpower. I thought we would lose the battle even if we had recommended the changes. However, if the sale of enough systems had been involved . . .? In retrospect, it is clear I made a mistake. I should have gone along with the consultant, but I was sure I was responding the way that Dr. Painter wanted.

Not long thereafter, I was called to a meeting in Washington. I made the trip and discovered that some people from Federal Systems Operation and I were going to visit Dr. Painter at the Pentagon. We arrived at his office and were admitted. A fifteen or twenty minute lecture followed in which he let me have it up one side and down the other about what a worthless, low-down, cantankerous, back-biting, no-good person I was for not paying attention to the consultant. That was the end of the meeting and the end of the trip.

At some time before 1976, the "other shoe dropped" on the Dix promotion. John Couleur had been working on a new hardware design based on the 6000-line hardware. It was called NSA for New System Architecture. I was generally aware that John was working on this, but thought it was just a hobby. I was wrong. The new design was announced as the wave of the future and the VMM and NPL were dropped. This was a severe blow to many including myself. It seemed to me like burning the future. The 600-line was an ancient design when we adopted it over a decade before. If it had not been for the peculiar situation within General Electric, it would not have made sense then. Now we were going to prolong its life and jettison its successor? It just didn't make sense.

The one good thing that could be said about the NSA was it was (at least partially) upward compatible with the existing 600/6000-hardware--meaning one could upgrade to it without substantial reprogramming cost. However, it was not long before a clamor arose for GECOS IV—a brand new GECOS that would take full advantage of the NSA features. Of course, GECOS IV would introduce a whole host of new problems including incompatibilities resulting from running existing user programs on the new operating system.

Real effort in Phoenix on the development of an operating system for NSA was slow in getting started. Toshiba didn't show a similar tardiness in their efforts. They developed ACOS-3 in a short time after the NSA design was made known. When Phoenix people were finally assigned to the task, they were hungry for information. What the Japanese had done in ACOS was of great interest. A library of information in Japanese was being translated to English, but it was rather rough and available in limited copies. I was aware of a government-subsidized program in which the company participated to help worthy but disadvantaged students by giving them jobs. So I hired two of them to help me with the NPL documentation and get it into clean form with adequate copies.

At the time this was going on, our building was in the throws of an enlargement. To make room for the renovators we had been squeezed into a small space adjacent to what had been the stage of the auditorium at the west end of the building. When my two new hires arrived, we had no place to put them. We searched and searched and the only thing we could come up with was a small closet a few feet from my tiny office. We had to put them there. Those two employees were Stephanie Knox and Edward Landrum and they would be with me for quite a long time. They were each given a MULTICS terminal and were put to work entering the translated documents into a database from which we gleaned hard copies as needed by the developers. This beginning grew into a staff of about eight young people who performed MULTICS input for this and other projects. They were all female except for Edward Landrum. [S]

Of course, our offices eventually returned to their normal configurations and I had gotten a proper terminal room for my MULTICS terminal operators. They were a grand group and I enjoyed working with them. I received some complaints about their work—it was neither polished nor perfect—but it was better than should have been expected for personnel with their levels of experience and pay. They had all started out with no skill other than typing and had learned rapidly and worked consistently without any major discipline or absentee problems.

Some of them said that they had trouble with arithmetic. So I asked them if they would like to have a math class at lunchtime. Four of them accepted. They all bought books and we all brought lunches and we would eat our lunches and have math class every noon. This went on for quite a while and they were apt students and made good progress. At one point, the girls plus some of their friends and sisters took me to lunch at an upscale country club not far from the plant as an expression of appreciation. I felt very honored and fussed over.

I carried away one other good thing with me from the VMM experience. After we got the first one running in the factory, the Turkeys all went together and had a copy of the specification I had written bound. They presented it to me in a little ceremony they got together in the factory. It is one of the treasures I have carried along with me ever since.

While all of this was going on, we began to hear stories out of the Pentagon about an approach to data processing and computing being fostered by my old acquaintance Grace Hopper. Large computers were going to be replaced by networks of small ones capable of communicating with one another and with common databases that might be on computers of any size. Any user at any node could communicate with any other user at any other node provided he or she had the proper clearance and permission to do so. Also, any user at any node could access data on any database in the network with the same provisos.

At the same time, personal computers were beginning to appear in their original primitive and relatively incapable versions. Nevertheless, a new vision of how computing and data processing were going to be done seemed to be gathering momentum. This would have been very much to Harry Tellier's distaste, but it seemed to be on its way nevertheless.

WELLMADE

Almost immediately after I returned from Tacoma to Phoenix, I met Tony Pizzarello. Tony was a new immigrant from Italy and was very friendly and interested in the work we had done at Weyerhaeuser. He was also a very bright person and was well versed in the more advanced thinking in the field of computer science. He was a Doctor of Engineering from a school in Milan, Italy and was clearly well educated though when I first met him, his English was still a bit rough.

Many people were talking then about what was called Structured Programming. The idea was to derive a set of techniques and approaches that would at the same time make software development more manageable, efficient and economical and would lead to error-free, readable, maintainable and flexible software products. These were all easy-to-sell goals and everyone was buying them. Two general approaches to the problem were being considered: one was an informal set of techniques, procedures and attitudes; the other was a formal, mathematics-like method potentially involving proofs of correctness of programs. The first of these approaches was being promoted and sold by a man named Yourdan and several others who went around the country giving seminars. The other approach was by Professor Edsgar Djikstraa, from Denmark, who wrote many learned papers on the subject and also frequently read papers at computer conventions and symposia.

Tony and I discussed the subject frequently and he studied the subject in depth and kept me apprised of his progress. Both he and I were strongly attracted to the Djikstraa approach. It suited our backgrounds in which one demonstrated that things were correct using mathematics. We liked the conciseness and rigor of the method. We also thought it was possible it could be made into a methodology that would result in all-inclusive designs. Hence, if one followed the methodology in designing a program, errors would be impossible that in a less formal system would be caught only by careful scrutiny and close attention or, worse yet, disasters at a customer installation. Yet the Djikstraa work was not yet in the form of a methodology but only in the form of a meta-language that he claimed could be used to design any program.

The Djikstra meta-language looked much like a high-level programming language, yet that is not what it was. Instead it was a set of language constructs each of which carried a set of input conditions into a set of output conditions. These input and output conditions were described

rigorously in logical statements as were their transformations by the various language constructs.

Hence, I visualized a scheme similar to designing and constructing a computer where the design is described in the form of logical expressions. The implementation then consists of merely creating a physical component capable of performing the function of each of the logical elements and connecting many of these components together as specified.

In the Djikstraa design, the logical design elements corresponding to the logical **and** and **or** gates and **flip-flops** and so forth of the computer design were the meta-language constructs. The program design was a set of input states and output states written as logical statements and a set of meta-language constructs transforming the input states into the output states. Implementation then consisted of converting the meta-language statements into an implementation language and the program was complete.

At some point, I went to his management and asked that Tony Pizzarello and his work be transferred to my organization. This was done without difficulty—I don't think they knew what to do with him. He proceeded to create a complete methodology based upon the Djikstraa constructs and ideas. I wrote a manual for it and we named the methodology WELLMADE, which stood for "HoneyWELL Method for Algorithmic DEvelopment".

Tony started making presentations to internal programming organizations and the idea got mixed reviews. In the best case, the programmers said they were willing to try it; but the more normal response was this was just more unnecessary paperwork being piled on them that they didn't need or want. It certainly was more work. However, it was our belief the additional work done at design time would more than be repaid by improved visibility of the product's operation and huge reductions in debug and maintenance time that were the elements currently eating us alive.

Word of WELLMADE spread rapidly throughout the company. It was pretty much ignored or scoffed at in Billerica, but it was paid serious attention at the Corporate Computer Science Center (CCSC) in Minneapolis. It was not long before Tony was invited to CCSC to give one of his presentations. It went over very well and established a relationship with one of the young Ph Ds there named Don Boyd. Don was very positive about WELLMADE and arranged for Tony to spend an extended period in Minneapolis to

work with them on its further development and use. Tony did so and that resulted in a much closer relationship with CCSC though I'm not sure it contributed much to WELLMADE or its future or to the fortunes of CCSC.

In the meantime, back in Phoenix, I was having my Turkeys use WELLMADE in their work as much as I could. The results were not very encouraging. Its use was, indeed time-consuming. Furthermore, the application of the Djikstraa constructs to the input states to obtain the output states was not a simple, straightforward process. Many ways often existed to achieve the same transformation and some ways were much better than others. This could have been viewed as an advantage—find the "best" way to accomplish the job at design time—but the effort involved was really outlandish.

I tried it myself and could not claim any better results than my employees. Still I tended to blame myself and continued to believe in the methodology. One reason for this was whenever Tony was confronted with a problem, he would plow right through it and come up with a clean result. However, he tended to be confronted with small sample problems and not with entire applications or systems to design, so I'm not sure his experience was particularly significant. Worst of all, we would often get a design just the way we wanted it and then it would fail in implementation due to silly mistakes such as typographical errors on input or misspelling of the names of variables. So we definitely didn't have a panacea.

Yet I continued to support WELLMADE with all of the political and persuasive capital at my disposal and I did so for a long time. Finally, one day Tony came to me and said in essence WELLMADE was an idea that had no place in the world of practical program development and he recommended that we abandon it. He had totally lost faith in the idea and recommended I do the same. It took me a long time, but I came to realize much later that he was correct.

At some point during VMM development, we had another change in leadership. Ed Vance returned, after a multi-year absence, as Director, Software Engineering—his was a new organization reporting directly to Walker Dix. This meant we software types no longer reported to Lee Wilkinson. Ed was later promoted to Vice President of Software Engineering. I suppose that I should have been happy about the change because it put us one notch higher in the reporting chain. However, I don't recall any tangible benefits.

At about the same time, Honeywell acquired Xerox Data Systems. I don't remember the details, but we ended up with a software group in Los

Angeles, not far from LAX, under the direction of George Gillette. This group had some very talented people and had developed what they called a Software Factory for their development work. This was a set of on-line programs and utilities the developers could utilize from timesharing terminals. It was what is typically included with a modern compiler one purchases for his or her PC (like my Visual Basic package) only it was broader in the sense it served several programming languages employed in implementation.

I was given the chance to visit the LA installation and observe what they were doing. I was unimpressed. I had the idea this was an expensive toy and it would be unnecessary if only people would use WELLMADE. However, Ed Vance thought it was grand and gave me the responsibility of implementing it on our equipment.

I was really uninterested in the Software Factory but I was familiar with one problem I thought fell into that realm: the processing of Ck-97s. I had obtained two employees, Bob Grimes and Bill Dibble, who were anxious to implement something in C-language. I gave them the task of writing programs that would convert the existing Ck-97 processing programs to on-line versions running under GECOS timesharing and implemented in C. They tore into the task and got it done with little fanfare and very good results. They were very impressed with C-language although they expressed some criticisms I don't now remember. I should have kept them going on other Software Factory tools, but I didn't.

After Tony had finished his tour at Corporate Computer Science Center and WELLMADE had settled into nothingness, an opening developed for the leadership of the CCSC. I was asked to interview for it. I made a trip to Minneapolis and had a meeting with the staff of CCSC and then with a Honeywell Vice President. It turned out the Vice President was the great obstructionist on the RAYDAC acceptance tests. The meeting with the staff went very well, I thought, though I was not particularly impressed with them, and so did the meeting with the VP until he noticed I had worked on RAYDAC. He asked me if I had worked on RAYDAC and I replied affirmatively. At that point, he glazed over; and, for practical purposes, the interview was over.

I didn't get the job, but I didn't really expect to. I wasn't qualified to lead a staff of all Ph Ds most of whom had years of academic background in the computer science field. On the other hand, they had little practical experience. Also, it was not clear where they would get someone who was better qualified. In any case, it was an interesting exercise and to this day I don't know who got the job.

One other high note comes to mind from this period and, again, it involved my students. A girl named Sharon Marion who was attending Tuskegee University augmented our student staff temporarily for the summer. She and Edward Landrum and I think Jackie Williams expressed dissatisfaction that they didn't know how to swim, or how to swim well. I asked them if they would like to come to my house after work some evenings and take swimming lessons. They loved the idea and we duly started after hours swimming lessons. Sharon and Edward attended every time and Jackie frequently, but she already knew how to swim and only needed some practice. Well, of course, Edward and Sharon both learned and were doing pretty well by the end of the summer.

To commemorate their success, we decided to have a party. It was a swimming party, of course, and the high point was Sharon and Edward swimming the length of the pool. All of the students attended and some of them brought friends. In addition, Tony Pizzarello and his wife Sandra came because the students were all fond of Tony. We had a grand time and at about 10:30 P. M. the students all pitched in and volunteered to clean up the debris of the party. They were all gone before midnight and the house was spotless. I sure liked those kids. [T]

During July of 1977, I received a performance appraisal from Ed Vance. When I read it, I thought it must have been for someone else, because so many of the comments were so off target. Also, it had been three months since it had been written before he had gotten around to reviewing it with me. After I had a chance to read it and think about it I wrote him a note asking for clarification of the various points that mystified me. We eventually had a private meeting to discuss it, but I learned very little as a result.

That was bad enough, but things were to get even worse. We had a new General Manager and he was from the Federal Systems Operation, FSO, in Washington. FSO was something of a darling within the company because they had landed some very lucrative government business. I'm sure that is why the particular General Manager was chosen. Shortly after his appearance on the scene, he began bringing in his protégés to fill software management positions. For the most part, these were brash young kids who knew very little except a few buzzwords. I ended up reporting to a whole series of them and that was not enjoyable.

One experience I endured during this period was typical. Dave O'connor and I were in a meeting with one of the pipsqueak "leaders" one day and this one lectured to us about how GECOS was constructed. Here we

were, the inventors of the whole thing, and he was lecturing to us on that subject? That was too much to take.

I wanted to leave, but could not figure out how to get it done. I had one interview in Los Angeles set up by the man who had been the Regional Sales Manager for General Electric when I was at Weyerhaeuser and had gone into the "headhunting" business. The interview didn't go well at all. I think I had begun to loose my confidence and it probably showed. I got in touch with Harry Tellier, but nothing was available for me at IBM. I had lost track of Helmut Sassenfeld. I assume he had retired. I had also lost track of Ken Robertson and anyway, I had turned down the offer his company had previously made to me.

So when I had an opportunity to take early retirement in 1982, I jumped at it. It was not a golden parachute as some people imagined, but it did continue me on full pay for six months during which it was imagined I would seek other employment. I did a little seeking, but mostly I got into the idea of spending full time with my wife and didn't wish that to change. So then I really retired and I have never regretted it for more than about five minutes at a time or in the bizarre dreams I sometimes have.

Summing Up

As I look back on my career I have an overwhelmingly positive feeling. Although my colleagues of the time and I did not have the opportunity to contribute to the more modern aspects of the computer industry, I think the work we did, set the stage and made possible the giant leaps that have taken place.

It is true that I encountered some disappointments during the period after I returned to Phoenix from Tacoma. However, in the bigger picture, none of them mattered at all. Because Honeywell and its followers never did get into the personal computer business, they probably wouldn't have had a large impact in the industry as a whole. As it evolved, they probably did about as well as they could have for the Honeywell stockholders.

I also had some experiences few can equal. The good feeling of delivering a useful tool for our scientists and engineers to use at Hanford was one of the early ones. Carrying the message of Hanford Data Processing to SHARE and to various presentations around the country was heady and gratifying. Helping to mold 9PAC into a tool that could be used by the contributors was exciting and rewarding. The creation of the Five-Year Plan for Computing and Communications at Hanford was fun and satisfying.

The creation of the original GECOS design during those lonely days outside of Chicago was an accomplishment of which I have always been proud. It was entirely my own response to a great number of strong demands and it endured for the life of the 600 and 6000 lines. New implementations and additions were made to GECOS, but the basic functionality and guiding concepts were the same in 1982 when I retired as they had been in 1962 when I specified them.

My contributions to the MULTICS design were small, but I always enjoyed working on and with MULTICS. I especially enjoyed the opportunity to work at Bell Labs and the opportunity to interact with that exceptional group of people. I had fun creating 645 GECOS.

Being one of "Hereford's Raiders" was certainly an experience in a lifetime. We took on a tough job, put everything into it and ended up with a good result, though not as ambitious as our starting goal.

Although the outcome of the VMM effort did not lead to the consolidated product line that I had envisioned, defining it and specifying it and imagining the positive impact it would have were their own rewards. It was good work and I knew it and that could not be taken away because someone decided not to use it.

The WELLMADE problems were of my own making. I became emotionally involved with the method and refused to see the proper balance between its beauty and its practicality. This was a serious mistake that I came to regret.

One occurrence of particular importance to me was the introduction to psychology resulting from the teaching efforts undertaken at the Weyerhaeuser Company. I had never had a course in psychology and didn't have any notion what it was about or what it could do for or to people. In my view we were born with a certain personality and that was the way it was going to be. Each human would get along with some people and not with others and nothing could be done to change interpersonal relationships. Of course this was the exactly opposite message from the one the psychologists had to deliver.

I was astonished and intrigued with what they had to say and the techniques and methods they had to teach us. I started reading what my offspring referred to as "Pop Psychology"—*I'm Okay, You're Okay, The Games People Play,* and so forth—and found it absolutely spellbinding. I started studying Transactional Analysis and also reading some of the seminal works in the field like those of Abraham Maslow.

The most rewarding part about the lessons these people had to teach was that they were useful not only at work but at home and in our social lives. Thereafter, I took every opportunity to attend seminars or lectures or classes touching on interpersonal relations. In one instance, I had a chance to attend a one-week seminar at the University of Wisconsin at Madison. At a later time, I was able to convince our management to have the leader of The University of Wisconsin seminar to come to Phoenix and give a short course for our managers. Of course, the usual thing happened: those who needed it the most derogated the course; those who needed it less, embraced it.

Finally, I think I had an unparalleled opportunity to work with bright, young students—both Turkeys and my other young people. They were all loyal and mostly supportive of what I was trying to do and I shall always be grateful to them for their efforts and attitudes.

Epilog

The various products I worked on and the installations and organizations with which I was associated came to diverse ends. The RAYDAC was dismantled in the early 1960's.[31] Given its poor performance, it is remarkable that it lasted that long. Apparently, the parts from which it was constructed were used to fabricate other electronic gear at Point Mugu. Point Mugu never reached the heights of notoriety that have been attained by Vandenberg Air Force Base to the north near Lompoc, California or the Naval Ordinance Test Center at China Lake. Its greatest notoriety came when President Reagan used it as a stop over on trips to his ranch near Refugio Beach to the north.

Computer Control Company continued in business headquartered in Framingham, Massachusetts. They developed a line of minicomputers called the DDP-series. The company was sold to Honeywell in 1966 and played an important role in the early development of computers in that company.[32]

Of course, Omnicode had a short, but I think successful life at Hanford. FORTRAN and the IBM-709 replaced it and the IBM-650 when the 709 was installed.

The history of 9PAC is long and colorful. Of course, it was put into service at Hanford in 1959 and continued in use until a Univac machine replaced the 7090 in 1965. [33] I know 9PAC was also used elsewhere, but I am not sure where. Upgrades called 90PAC and 94PAC were created by SHARE and IBM continued to maintain and support them including the preparation and publication of manuals. [34]

A potential 9PAC replacement was developed called COGENT for Cobol GENerator and Translator.[35][36] The idea was to generate COBOL source programs from 9PAC packets and feed these into a COBOL compiler to produce executable programs. For various detailed reasons, this was never particularly successful.

However, other 9PAC descendents and look-alikes lived long and prosperously. According to Fry and Sibley [37] (in 1976), "9PAC is the principal ancestor of most commercial report generators developed since 1960. Foremost among these is the Report Program Generator (RPG) developed for the 1401 in 1961; this has evolved into the RPG for the IBM

System/360 and an enhanced RPG II for the IBM-System/3, System/360, and several other computers."

It is clear that IBM finally "got the message". Other comments [38] from Phillip L. Frana, Ph. D., of The Charles Babbage Institute added the following commentary based upon information in the CBI archives:

> 'RPG II was released in September 1970 by the Rochester, Minnesota, division of IBM for use in System/3. While popularly considered a programming language, RPG II varies from the common definition of a language in that it involves solving problems by filling out preformatted forms rather than using problem-oriented notation. Regardless, RPG II made the quick and flexible production of a wide variety of reports possible. IBM described the product as the "highest-level language in data processing today," and as "the easiest to learn, easiest to use, easiest to maintain, fastest to code, and most machine-independent, problem-oriented language available." Because of these properties, RPG is arguably the second-most popular "programming language" of all time, behind only COBOL.
>
> 'IBM's RPG II software was usually bundled with a compiler deck and library of subroutines, and made available for monthly lease. RPG II also worked on the IBM-1130, the IBM-1800, the RCA Spectra-70, as well as on the UNIVAC 9200/9300-Series. Its successor, RPG III, was developed for use on IBM's popular AS/400 minicomputer system. The development manager for RPG II was Tom Hoag, who also defined the original extensions for RPG III.'

In the early 1960's John A. Postley began work on what Postley called File Management Systems.[39] This lead to development of 9PAC-like systems called Mark I and Mark II to run on the IBM-1401 and Mark III to run on the IBM-360. By 1965, he was working for Informatics and obtained sponsorship from outside companies of $500,000 to develop an improved version called Mark IV. This was accomplished and the system was put into service in 1967. After some lean startup years, the product became very successful.

Informatics sold Mark IV for $30,000 per copy and at some point garnered additional profits for maintenance and updates. Postley reported that in 1979, when he retired, there were more than 4000 Mark IV installations in 45 countries. Mark IV was the first software product to have cumulative sales of $10 million and then $100 million.[40] It was acquired by Sterling Software in 1985 and Sterling was acquired by Computer Associates in

March of 2000[41]. Computer Associates renamed the product VISION: Builder. It may still be in use.

Postley did not acknowledge any relationship to previously existing software in any writing that I have seen. I remember seeing a paper about Mark IV in the early 1970's and thinking at the time that someone had reinvented 9PAC. Whether or not it was done with knowledge of 9PAC and its capabilities is impossible to tell. Certainly, the founders of Informatics—Walt Bauer, Werner Frank, Richard Hill and Frank Wagner [42]—were well aware of 9PAC. I met all of them while I was active in SHARE and I am sure that they knew about 9PAC, but whether or not they knew of it in enough detail to have influenced Mark IV is unknown to me. I prefer to believe that these were parallel inventions based on a single set of good ideas.

According to W. C. McGee, there were some 20 similar programs developed in the period following 9PAC development. [43] I think the importance of the Hanford Generalized Routines as a "spark" to start the development of all of these has been established.

During the 1960's, the Hanford Plant was reorganized in a major way. Instead of being run by a single prime contractor, as it had been under Dupont and then General Electric, the operation of the plant was divided among a large group of contractors each of which was responsible for a single function. The computer center became the responsibility of Computer Sciences Corporation and later of Boeing Computer Services. This change would have been a disaster in the eyes of Harry Tellier, if he were still there; however, I believe he accepted a job with IBM before the big change took place.

Activities at Hanford continued along two general lines that involved a large group of contractors [44]: hazardous waste cleanup and general research and development. The need for hazardous waste work is well known and results from decades of operation of the facility with a primary emphasis on production and little knowledge or concern for protection of the environment. The general research and development activities may not be as well known.

In 1965, the Battelle Memorial Institute entered into a contract with the Atomic Energy Commission to establish a laboratory adjoining the northern boundary of the City of Richland, Washington. In this laboratory, they were to perform research and development on behalf of the AEC, but were also given the opportunity to sell their technical expertise to the general public. They called the laboratory the Pacific Northwest

Laboratory [45][46]. In the years since then, the laboratory has performed research in a wide variety of technical areas including Information Technology, Biotechnology and nanotechnology. In 1995, they became one of nine national laboratories funded by the Department of Energy [47]. At that time their name was changed to Pacific Northwest National Laboratory.

In 1996, the laboratory ordered a Cray Supercomputer and recently that machine has been replaced by a supercomputer said to be the 5th fastest in the world [48]. It is a Hewlett-Packard machine capable of performing 11 gigaflops per second. (That is, 11 billion floating point operations per second.) So the Hanford tradition of fostering advanced computational knowledge and techniques lives on.

The plant that I went to each day in Phoenix is still there and, the last I heard; it is still a computer factory. However, Honeywell no longer has any connection with it. In 1986, Honeywell Bull was formed—a global venture with Compagnie des Machines Bull of France and NEC Corporation of Japan. The ownership was originally 42.5% Bull, 42.5% Honeywell and 15% NEC, but the Honeywell portion diminished annually for five years until, in 1991, Honeywell was no longer in the computer business. [49][50]

Bull was nationalized during the 1980s, but then became involved in a dizzying set of negotiations for partnerships with other companies throughout the world, some of which were carried out. Some of the companies involved were IBM, Packard-Bell, Motorola, Zenith Data Systems, and Processor Systems of India. In 1987, Compagnie des Machines Bull changed its name to Bull. It is still in operation.

It is difficult to determine the extent to which the GECOS design that I developed in 1963 survived and influenced the various operating systems that followed it. Certainly, its capabilities have been extended and expanded enormously. Just how much of the original thinking is still present is probably impossible to determine.

There has always been some confusion about the name. When we originally named it, we intended the name to stand for General Electric Comprehensive Operating System, but the lawyers told us that we could not use the company's name in that manner. So we just said that it stood for GEneral Comprehensive Operating System. That worked well until the merger with Honeywell, but they decided they wanted to break the (supposed) connection with the former product owner and changed the name to GCOS for General Comprehensive Operating System. That name has endured into the twenty-first century.[51]

Actually, the name GCOS has been used in conjunction with a vast array of products offered by Bull over the years. Toshiba and then NEC developed an ACOS-6 operating system that was based on GCOS-3. It utilized the features of the Honeywell New System Architecture (NSA). In this sense, the GCOS influence was transmitted to Japan as well as Europe. GCOS-8 (originally named GCOS-4) was developed to utilize NSA and used code acquired from Toshiba as a starting point. [52]

There have been GCOS's for small computers and GCOS's for medium computers in addition to GCOS-8 for the large computers. As the Bull products evolved, the Internet tells us, "GCOS-8 enterprise servers … process millions of transactions and hundreds of batch jobs and … bring together the advantages of mainframes and open systems."[53] It also tells us, "DPS9000, which runs under the GCOS-8 operating system … can handle more than 1000 on-line transactions per second."[54]

My conclusion, from what I have been able to read, is that the DPS9000 computers are the direct descendents of the old Honeywell, 36-bit, machines. DPS7000 machines, which run under GCOS-7, are also offered that use 32-bit words. In addition, there are smaller computers, many of which run under something named GCOS, and a full array of peripheral devices sold under the Bull logo.

MULTICS Systems also had a long and colorful life. [55] A serious effort to increase MULTICS System sales took place in the mid 1980's. At one point there were 100 MULTICS Systems in the field. There were various efforts started to develop a more modern, better performing hardware base for the system, but Honeywell management never saw fit to provide the financial support needed to bring these efforts to fruition. There were also several attempts to form MULTICS companies that might have supported a hardware upgrade, but none of these succeeded.

Eventually, no maintenance was provided to keep the machines in operation and various companies rose to the occasion and provided the support that was needed. These arrangements helped to keep the system alive, but throughout the 1990's systems began to shut down—one after another. Finally, on October 30, 2000, the last MULTICS System was shut down at the Canadian Department of National Defense in Halifax, Nova Scotia. After thirty-six years, the dream of the Information Utility, as manifested in MULTICS, came to an end.

WEYCOS had a shorter life than MULTICS; it ceased operation at Weyerhaeuser in the mid 1980's. [56] It was replaced by IBM equipment. I

do not know what software was used to replace the transaction processing done by WEYCOS. Knowing what we know today, it is regrettable that WEYCOS was not re-implemented on a GCOS-3 base with the full Multi-wing capability. I could possibly have drummed up some interest in such a project, but this seemed an unlikely possibility with the WEYCOS project termination occurring as it did just as we were being taken over by Honeywell. However, in later years, transaction processing became an important capability in the sale of Bull systems. Nevertheless, a productive life of a decade and a half is not a bad "track record" for a system that was built for a single customer on an obsolete base (GCOS-2).

Because the VMM and WELLMADE never went beyond infancy, there is nothing to report about their life cycles. They were both more or less dead on arrival.

I would like to be able to give a complete rundown of what happened to each of the good people mentioned in the body of this book, but I am not able to. Suffice it to say that it was an interesting and exhilarating experience to have worked with them. Several of them are still actively employed in Information Technology either as owners of their own companies or as employees.

Acknowledgements

I am deeply indebted to many people for helping me to recall the many activities and events that I have reported. In particular, George Kendrick and Jim Tupac helped me to recall the people and events at Point Mugu both before and after Computer Control took over running the RAYDAC.

Bill McGee and Ed Roddy were of great help in getting the story straight on the development of the Hanford Generalized Routines. The Generalized Routines were an important "springboard" in my career, but I was not involved in their development, so getting the story straight was a challenge. George Gillette and his wife Dorothy helped me to get the Hanford Atomic Products Operation organization straight in my mind and to remember the designations of some of the buildings within the "downtown" complex in Richland. Glenn Otterbein and Fred Ouren were of enormous help in providing me with and directing me to information on the events at Hanford after my departure.

I was able to benefit from some of the documents Charlie Bachman shared with me in recollecting the early days of 9PAC. In addition, of course, Ed Roddy was of great assistance and I was able to remake contact with Art McCabe who was also a 9PAC contributor and provided important criticism of my early draft.

Again, George Kendrick helped me to reminisce about our early days in Phoenix. Lois Kinneberg and Jane King helped me to recall their roles in the development of GECOS I.

In describing my involvement with the GE-645 and MULTICS, John Couleur helped me to reconstruct some of my prior knowledge about the hardware. Also John Gintel who was a major contributor to MULTICS at MIT helped me in this regard.

Grayce Booth and Paul Kosct were of invaluable assistance in helping me reconstruct a brief summary of the original WEYCOS system on the GE-265. Stan Williams provided information on the initial arrangements that led to the development of WEYCOS II. Paul Kosct also provided me with historical information about the life of WEYCOS after the completion of the project. Also, Paul, Grayce and Ed Roddy reviewed my manuscript of the WEYCOS Chapter and Grayce did a spectacular job of editing it, for which I am very grateful.

Finally, profuse thanks are due the members of my immediate family and Dr. Bruce Mason for reading and criticizing the manuscript. They found things that were invisible to me in spite of many rereadings.

Appendix A—Some Computer Fundamentals

The major components of a stored-program computer are discussed in this appendix. Each of these is described briefly in sections A1 through A4. How they work with one another is then described in section A5—the General Organization of a Computer.

A1. Internal Memory

When one uses a hand-held calculator to perform complex calculations, it is necessary to use scratch paper to record intermediate results as the calculation proceeds. (Some modern calculators have one or a few cells of memory that can be used to store intermediate results, but because there is no standard way in which these operate they will be ignored in this discussion.) Computers have the same need to store intermediate results and other values as a calculation or sequence of calculations progresses. To satisfy this need, computers have a major component referred to as Internal Memory (or just memory, if not otherwise qualified). The Internal Memory consists of an array of storage for fixed-length data strings called words each of which has a unique identifier called an address.

When thinking of a computer memory, one should visualize a neighborhood filled with houses each having a mailbox. Each mailbox has an address and any item of the type "mail", be it letters from Mom or utility bills or advertisements or other junk, may be placed in the mailbox by the mail carrier and removed by the resident or vice versa. In a similar manner, a computer memory has a set of cells (mailboxes) capable of holding anything of the type "word", be it a number or a string of alphabetic characters and/or punctuation marks or a computer instruction. Each of these cells has an address that the computer uses to specify which word is to be affected when it stores or retrieves words to/from memory. The things stored would usually be the results of operations it has performed; the things retrieved would usually be operands upon which operations are to be performed or instructions that are to be executed.

Again referring to hand-held calculators, you normally enter one number at a time, indicate an operation to be performed and then enter another number. Each of the numbers you enter is a word. It is the unit of information upon which the calculator operates. Each calculator has a word length that is fixed. For example, most popular hand-held calculators have a word length of eight digits. That is, you may not enter a whole number larger than 99,999,999, or if you are working with dollars and cents, you may not enter an amount greater than $999,999.99. Of course, some calculators have larger word lengths, such as 10 digits or 12 digits, which permit entry of larger values. But whatever the word length, that is the number of digits that the calculator uses in every one of its operations.

It is generally possible to enter amounts shorter than the word length of the calculator, but when doing so, the missing digits are assumed to be zero in value. For example, you may perform the operation 1 + 2 = 3. In a calculator with an eight-digit word length, you would enter 1 and the calculator would assume this to be equal to 00000001. You would then press the "+" key and then enter 2, which the calculator would assume equaled 00000002, and then you would press the "=" key which would cause the calculation to be performed and would cause the result 3 to be displayed. The leading zeros in the result 00000003 would be replaced by blanks so that the result would appear the customary way.

Most computers deal with numbers in the same way as calculators except they store the numbers upon which they operate in internal memory. So, let us assume that we have a number whose value is 1 stored in internal memory address 1000 and we have another number whose value is 2 stored in internal memory address 1001. We want the computer (which we will assume has an eight-digit word length) to add the two numbers and store the result in internal memory address 1002.

So, in our example, we are assuming that the value 00000001 has been stored in the mailbox at memory address 1000 and the value 00000002 has been stored in the mailbox at memory address 1001. We instruct the computer to add the contents of address 1000 to the contents of address 1001 and store the result in memory address 1002. (Notice: this sequence could be used to add any two numbers by simply placing the values to be added into addresses 1000 and 1001 before asking the computer to perform the addition.)

However, computers are asked to perform more and different operations than the simple set (add, subtract, multiply and divide) found on most hand-held calculators. For example, they perform operations on

alphabetical and special characters like those that must be handled by word processors. In that event, the words of the computer use some scheme for storing these non-numeric characters in the words of the internal memory. For example, let us imagine that we have a computer that can store nine digit decimal numbers in each word or six non-numeric characters. In this machine, the name "GEORGE WASHINGTON" would occupy three words. It could be stored in memory, starting at address 1100, as follows:

Address	Contents
1100	GEORGE
1101	sWASHI
1102	NGTONs

The small letter "s" in the above represents the space character. Note that the space is used to separate the first and last names and also to fill out the unused last character of the word in address 1102.

Each processing step a computer executes involves either retrieving or storing (or both) a value in internal memory. In modern computers internal memory is used to store the instructions the computer executes, the data upon which the instructions operate and the results created. Hence, it is important for the memory to be rapid in its responses and for it to be large so it can accommodate voluminous programs along with all of their data.

A2. Central Control Unit

The Central Control Unit (CCU) does what its name suggests—it controls the other units of the system. It does so according to the instructions it finds in the program it is executing. For each instruction executed, the CCU does the following:

- Retrieves operands from and/or stores results in Internal Memory at addresses specified in the instruction.
- Sends the code for the operation to be performed to the unit that will perform it.
- Retrieves the next instruction to be executed from Internal Memory.

A3. Arithmetic and Logical Unit

The Arithmetic and Logical Unit (ALU) performs all processing operations. That is, it carries out the following steps:

- Accepts operands from the CCU.

- Performs an operation upon these operands according to the operation code provided by the CCU.
- Delivers results to the CCU for storage in Internal Memory.
- Maintains internal triggers and values that carry over from one instruction to the next.

A4. Peripheral Devices

Peripheral Devices perform various other functions. Many of these functions help to make computers usable by humans. Some examples of these are given below:

- Operator's Console. This gives a human operator an opportunity to start or stop programs and control devices manually instead of via the CCU.
- Keyboard. This gives a human the opportunity to provide key-driven inputs.
- Console Display. A way for a human operator to view messages originating in programs in execution or to display specific words of Internal Memory.
- Card Reader. A device capable of reading punched cards to be used as program input.
- Card Punch. A device capable of punching cards as outputs from a program.
- Printer. A device capable of printing output from programs
- Secondary Memory. A memory larger, but slower than the Internal Memory. Such memories are used to expand the apparent size of the Internal Memory by permitting large blocks of storage to be read into the Internal Memory when needed or large blocks to be written when completed. In PC's these are usually referred to as Hard Drives.
- Magnetic Tape Handlers.
- Magnetic Disc Handlers.

The list of such devices is virtually endless. Some peripheral devices have their own controllers that match their internal requirements with those of the CCU and others share a single controller among a group of devices. In many instances the controller has the ability to address memory independent of the CCU. The CCU interprets instructions tailored to control each of these devices in the same way it interprets instructions for execution by the ALU.

A5. General Organization of a Computer

The general organization of a stored-program computer is shown is Figure A1. Each of the components described above is shown within a rectangular box. A fine line shows the control flow of each component; a bold line shows data flow. The CCU is at the center of the picture controlling everything and is getting its input from the internal memory. One or more I/O Controllers may be present depending upon the requirements of the devices being controlled. It is also possible to build systems with more than one CCU/ALU pair, but that complication may be ignored here.

In some computers, it is not possible for more than one component to access Internal Memory at once. In these instances, it is necessary for the CCU to act as a traffic cop and inhibit the initiation of any new operation while a previous one is in progress. This leads to slow processing because many components are often waiting to gain access to memory.

In most machines, it is possible for the I/O controllers and the ALU to access memory simultaneously. This simultaneity is depicted in Figure A1 by the data flow lines being connected directly from each of these components to the Internal Memory. In this arrangement, the CCU can start an operation on a peripheral device and then proceed executing ALU operations. This is a great advantage because peripheral devices usually run much more slowly than the ALU, so that the ALU can perform many operations while a peripheral executes just one.

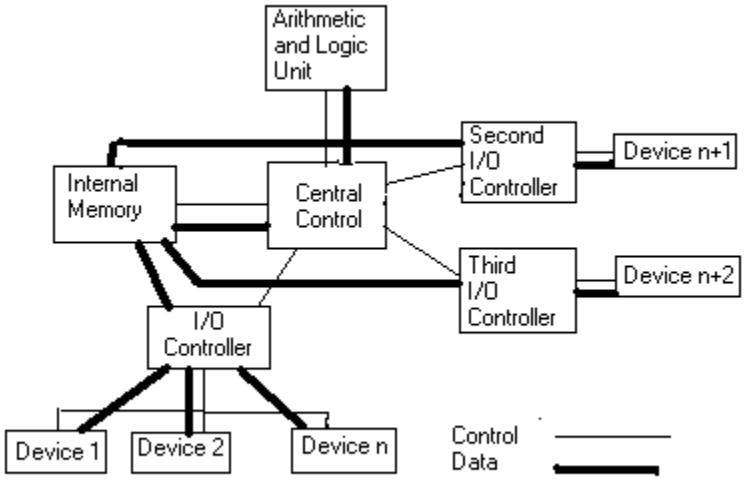

Figure A1. Organization of Computer Components

Appendix B—Representation of Computer Information

Information within a computer is, in most cases, represented as a string of binary digits or bits. Depending upon the type of information to be represented, different combinations of bits may be used to represent different quantities or symbols.

Each machine has its own number size referred to as its word length. The word length is the quantity of information operated upon when the computer performs its fundamental operations. For example, when two numbers are added together, it will add one word to another word. If one or the other of the numbers being added is shorter than the word length, then it will be filled out with leading zeros.

To illustrate this with a simple example, suppose we consider an ordinary hand-held calculator. Most of these have a word length of 8 digits. The display is eight digits wide and when you enter information or when results are obtained they must fit within this window. When you enter data, they fill the display from the right (low-order) end and leave the unused leading digit positions blank (zero). If some result is larger than the width of the display, an error is indicated, or some alternate procedure is initiated.

In discussing how numbers are represented in computers, we shall deal mainly with whole numbers. Similar methods exist for fractions and mixed numbers and may be studied and understood by the reader separate from this overview. It is possible to describe number representation in two different ways: algebraically—described in **B2** below—and non-algebraically—described in **B1** below. The algebraic way leads to more complete and general results, but either way will accomplish what is needed for this book. Readers who find the algebraic approach difficult should feel free to use the non-algebraic one; they are both described below.

B1. Number Representation—non-algebraic

The decimal number system uses ten characters called digits to represent any possible number. The digits are the familiar 0, 1, 2, … 8 and 9. When we count, we run through this sequence and when we get to 9, we have run out of unique characters, so we add 1 to the column to the left of the previous digits (which was previously blank and interpreted to mean zero) making it 1 and repeat the sequence from 0 through 9. When we wish to continue beyond 19,

we add one to the digit in the preceding column making it 2 and then repeat the sequence from 0 through 9. We are all familiar with the process. When we get to 99, we add one to the column to the left of the first 9, which makes it 1 and resets both of the columns previously in use to 0. When we get to 999, we add one to the column to the left of the first 9, which makes it 1 and reset all three of the columns previously in use to 0, and so on.

This is called the Place-Value System and the Arabs invented it many centuries ago. What may be a surprise to some people is that it works equally well for any whole-number base. That is, if you have only eight characters to represent unique digits, a number system every bit as useful as the decimal number system can be devised along the very same lines. This base-8 number system is called octal. If you have twenty unique symbols, you can create a number system to the base twenty, as the ancient Mayans did. If you have sixteen unique symbols, you can create a system called hexadecimal. If you have only two unique symbols, you can create a binary number system. We will look at some examples of octal, hexadecimal and binary numbers because they are all used in connection with computers.

In octal, the characters 0, 1, 2, … 6, and 7 are available to represent individual digits. Hence, we count 0, 1, 2, 3, 4, 5, 6, 7 and then we run out of digits just as we do in decimal when we get to 9. To continue counting we again do just as we do in decimal; we add one to the digit in the column to the left of the column that has run out of digits and then continue in the original column with the value 0. So we count 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17 and then we again run out of unique digits. At that point we again add one to the column to the left, reset the exhausted column to 0 and continue, 20, 21, 22, etc. When we got to 77 we continue 77, 100, 101, 102, 103, 104, 105, 106, 107, 110, 111, etc. In this manner we obtain an octal number to represent any possible whole quantity.

In hexadecimal, we have sixteen unique digits which are usually represented by 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. (Although A through F are conventionally used to represent the digits with decimal values 10, 11, 12, 13, 14 and 15, any other six unique characters could be used instead.) Counting is very simple. It progresses 0, 1, 2, 3, … 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, … 18, 19, 1A, 1B, … 1E, 1F, 20, 21, 22, … 2E, 2F, 30, 31, 32, … 99, 9A, 9B, 9C, 9D, 9E, 9F, A0, A1, A2, … AF, B0, B1, B2, … FA, FB, FC, FD, FE, FF, 100, 101, 102, … FFA, FFB, FFC, FFD, FFE, FFF, 1000, 1001, ETC.

In binary, we have only two unique digits: 0 and 1, but the scheme for representing numbers is just the same as with any other base. We count 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, 10001, 10010, 10011, etc. Once again, we can count as high as we like and represent any possible quantity using binary numbers, and this is what computers do.

A relationship exists that is utilized in the text of this book and needs to be understood. It is often useful to know in decimal how many unique values can be expressed by a given number of digits in another number system. So for, example, we might want to know how many different whole dollar amounts could be represented by a binary number containing ten bits. To determine this, we multiply two by itself ten times. Hence, the answer would be 2x2x2x2x2x2x2x2x2x2 = 1024. If we wanted to know the same thing for a four digit octal number, we would multiply eight by itself four times. Hence, 8x8x8x8 = 4096. If we wanted to know the same thing about a three digit hexadecimal number, we would multiply sixteen by itself three times. Hence, 16x16x16 = 4096. In general, a number to a certain base with a certain number of digits can represent a number of unique values equal to the base multiplied by itself a number of times equal to the number of digits.

B2. Number Representation—Algebraic

I shall refer to a number using the symbol N. Each number will contain n digits. So, for example, the decimal number 56924 would have n = 5 and N = 56924. I shall refer to the digits of the number as d_k, where k is a whole number greater than or equal to zero. The digits are counted from the right to the left starting with zero, so that if N =56924, then $d_0 = 4$, $d_1 = 2$, $d_2 = 9$, $d_3 = 6$, $d_4 = 5$. Notice that the largest subscript, k, is equal to n – 1. In our example, n = 5 and the largest value of k = n-1 = 4.

In addition, I shall use the notation B^k to indicate the number B to the k^{th} power. (The number B is called the Base or, in this context, the radix; k is called the exponent.) That is, if B = 5 and k = 3, then B^k = 5x5x5. In other words B times itself k times. So that, 3^7 = 3x3x3x3x3x3x3. The only case that is a bit troubling is that for which k = 0. In this case the value is always equal to one. Hence, for any value of B, $B^0 = 1$. The other case that may be a bit confusing is when k = 1. Here $B^1 = B$; so, for example, $9^1 = 9$.

Although numbers with bases different from 10 (decimal) seem daunting when first encountered, they are really quite easy to deal with after a little experience. One must understand the meaning of any number in any base to obtain an idea of how the scheme works. That is, any number used in modern calculations and processing, is represented in place-value notation invented by the Arabs many centuries ago. In this notation, when we write the decimal number 2693 we mean

$2 \times 10^3 + 6 \times 10^2 + 9 \times 10^1 + 3 \times 10^0 = 2 \times 1000 + 6 \times 100 + 9 \times 10 + 3 \times 1 = 2693$.

This illustrates the formation of decimal numbers. In the general case of an n-digit number, N, in the base B, the notation is

$N = d_{n-1} \times B^{n-1} + d_{n-2} \times B^{n-2} + \ldots + d_1 \times B^1 + d_0 \times B^0$ (explanation follows),

where d_k is the k^{th} digit of the number, and (for whole numbers) k = 0 for the column that would contain the number with value one. So in our example above (where N = 2693 and B = 10), 3 is in the $zero^{th}$ (k = 0) column and $d_0 = 3$. The values of k are then increased as one proceeds to the left; so that for k = 1, $d_1 = 9$; for k = 2, $d_2 = 6$; and for k = 3, $d_3 = 2$.

One may construct the decimal place-value sum that defines the value of the number as follows. First determine the value of n by simply counting the digits. In our case, n = 4; therefore, n-1 = 3. Hence, the first term will be $d_3 \times B^3 = 2 \times 10^3$; the second term will be $d_2 \times B^2 = 6 \times 10^2$; the third term will be $d_1 \times B^1 = 9 \times 10^1$ and the final term will be $d_0 \times B^0$. These are the terms contained in the description above and are correct.

Actually, after one goes through the description of the preceding paragraph as a means of understanding the notation, there is an easier way to write the same representation of a number. Just start out by counting the number of digits, n. This will tell you that the exponent of the first digit on the left will be n-1. In our example, n-1 = 3. You can now immediately write the first term ($d_{n-1} \times B^{n-1}$) in our example, 2×10^3. The second term ($d_{n-2} \times B^{n-2}$) will be 6×10^2. Continue in this manner until the exponent (superscript) of B equals zero. That will give you the rightmost term ($d_0 \times B^0$) equals, in our example, 3 x 1 = 3.

Some other examples that could be considered are the following:

In binary, $10011101 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^o$

In base 5, $3424 = 3 \times 5^3 + 4 \times 5^2 + 2 \times 5^1 + 4 \times 5^0$

In base 8 (octal) $74635 = 7 \times 8^4 + 4 \times 8^3 + 6 \times 8^2 + 3 \times 8^1 + 5 \times 8^0$

A distinction will be made between signed and unsigned numbers. Most numbers such as temperatures, can take on both positive and negative values, notated by a "+" or a "-"sign. These numbers are called signed numbers and are meaningless without their sign appended (unless a plus is implied by absence of sign as in temperature). On the other hand, some numbers such as serial numbers or a person's height have no sign and need none. These are referred to as unsigned numbers.

The word length of the RAYDAC was 30 bits (binary digits). This meant that any RAYDAC word could represent any unsigned decimal number from zero to 1,073,741,824 (2 times itself 30 times) or a signed number from -536,870,912 to +536,870,912 (plus or minus 2 times itself 29 times). The reason the maximum magnitude of the signed numbers is smaller is that one of the thirty bits must be used to store the sign leaving only 29 bits to store the magnitude of the number.

This binary mode of representing information has become rather well known, but at the time we were learning about RAYDAC, binary representation was not well understood by the public (even the well-educated public) at large. Of course, binary representation is important because each bit of a binary number (or word) can be represented by the on/off state of a switch or an electrical voltage changing from 0 to 1 and back or from + to -. Although binary mode is good for computers, it is not particularly convenient for humans. For example, to represent the unsigned number equivalent to 1,073,741,824 in binary, one would write "111111111111111111111111111111". This is clearly not the kind of number one would hope to use in balancing a checkbook.

However, such numbers can easily be represented in either octal or hexadecimal. The conversions between these systems are very easy and that is why we like to use octal and hexadecimal when we are dealing with a binary computer. To illustrate this, consider the following comparative values:

Binary	Octal	Hexadecimal	Decimal
100101001010	4512	94A	2378
111111111111	7777	FFF	4095
001111000111	1707	3C7	967
001010011100	1234	29C	668
111110101001	7654	FA9	4009

The first three digits of the first binary number are 100. They have the same octal value as the octal digit 4—the value of the first octal digit in the same row. The second three digits of the first binary number are 101. They have the same octal value as the octal digit 5—the value of the second octal digit in the same row. The third three digits of the first binary number are 001. They have the same octal value as the octal digit 1—the value of the third octal digit in the same row. Finally, the fourth three digits of the first binary number are 010. They have the same octal value as the octal digit 2—the value of the fourth octal digit in the same row. Hence, we could convert the binary number to octal by simply converting groups of three bits at a time to their octal equivalents and writing them down. We could perform the reverse procedure—conversion from octal to binary—by simply converting each octal digit to its binary equivalent in turn and writing it down. The reader should perform the conversions both ways on the numbers in the remaining rows as an exercise.

Now looking at the first four bits of the first binary number, we see 1001. This binary number has the same hexadecimal value as the hexadecimal digit 9—the first hexadecimal digit in the same row. The second four digits of the first binary number are 0100. This binary number has the same hexadecimal value as the hexadecimal number 4—the second hexadecimal digit in the same row. The third four digits of the first binary number are 1010. This binary number has the same hexadecimal value as the hexadecimal digit "A"—the third hexadecimal digit in the same row. Hence, we could convert the binary number to hexadecimal by simply converting groups of four bits to their hexadecimal equivalents and writing them down. We could perform the reverse procedure—conversion from hexadecimal to binary—by simply converting each hexadecimal digit to its binary equivalent in turn and writing it down. The reader should perform the conversions both ways on the numbers in the remaining rows as an exercise.

Note that in the examples above, there is not a simple way to convert the binary, octal or hexadecimal numbers to their decimal equivalents. Indeed, these conversions involve a sizeable amount of multiplication, division and addition and are best done by computer programs. Hence,

programmers—especially in the early days—preferred to work with octal and hexadecimal.

In the RAYDAC world, we always used octal because it was convenient and also, The Marchant Company marketed a calculator that operated in octal. We had Marchant octal calculators and used them fairly often.

Here are some various whole numbers in the decimal, binary, octal and hexadecimal systems:

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
23	10111	27	17
31	11111	37	1F
32	100000	40	20
63	111111	77	3F
64	1000000	100	40
100	1100100	144	64
255	11111111	377	FF
256	100000000	400	100

Note that in binary, in the absence of many symbols, you must use many places. In octal, you have the symbols with which you are familiar in decimal except for 8 and 9. Hence, you gain some economy in places over binary, but are still not as well off as in decimal. But in Hexadecimal, you need more symbols than in decimal to represent the numbers. You

will note, though, fewer places are used to represent large numbers than in decimal.

So, once the numbers upon which we wish to perform calculations are converted to binary within the computer, we can proceed with blinding speed. However, how do we get the decimal numbers we are given with our problem into the computer and how do we present in decimal the results our customer is going to read? A way to do this must be provided. The answer on the RAYDAC (and on many other machines) was to use BCD, which stands for Binary Coded Decimal. In this scheme, each thirty-bit word of the computer is divided into seven four-bit groups (with two bits unused) and each four-bit group represents a single decimal digit in binary code as follows:

Dec. Digit	BCD Representation	Dec. Digit	BCD Representation
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Expressed algebraically, 0000 in BCD = $0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 0$ decimal; 0001 BCD = $0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1$ decimal; …; 1000 in BCD = $1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8$ decimal; and 1001 in BCD = $1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9$ decimal. Ways are then provided for making these BCD words readable by the computer and for displaying them for external viewing. These ways will be described elsewhere.

The ability to perform fixed-point arithmetic had been possible on computers from the very first ones invented. That is, the computer was capable of performing add, subtract, multiply and divide operations (actually some could not perform divides) on data all of which had the same decimal points (or binary points if they were binary machines). That is, you could perform adds and subtracts on whole numbers (integers), or fractions or numbers of the form x.xxxx, where x represents a decimal digit and the decimal (binary) point could be anywhere in the number. However, early computers had no capability to operate on numbers in which the decimal (binary) points changed from one number to the next.

For example, one could add words containing the values 125.33, 528.64 and 987.12 because they all have the same decimal precision; however, one could not add words containing the values 125.33, 52.864 and 9871.2 without performing auxiliary operations to align the decimal points. (The problem is even more difficult with multiplication and division.) Because

this type of problem occurs frequently, a separate form of representation was invented to facilitate its solution. This form of representation is called floating-point.

A floating-point number is represented as a mantissa and an exponent. It may be floating-binary or floating-octal or floating-decimal, but we will first discuss floating decimal. A mantissa is a number within a specified range, which when multiplied or divided by 10 an appropriate number of times will equal the fixed-point number to be represented. An exponent is the number of times the mantissa must be multiplied or divided by ten to obtain the fixed-point number. The general form is eexxxxx, where ee is a signed exponent and xxxxx is a signed mantissa whose absolute value is in the range 0.1<=|xxxxx|<1. (The range of the exponent depends on the particular computer.) Some examples are as follows:

Fixed-Decimal Number	Floating-Decimal Number	Mantissa	Exponent
125.33	0312533	0.12533	3
52.864	0252864	0.52864	2
9871.2	0498712	0.98712	4

To convert a fixed-decimal number to floating-decimal one first determines the mantissa. In the first example above, one examines the quantity 125.33 and notices that to make it fit into the required range, greater than or equal to 0.1 and less than 1, it will be necessary to move the decimal point to the left three places. This will produce the mantissa, 0.12533. To recreate the original fixed-decimal number it will be necessary to multiply the mantissa by 10 three times, which will move the decimal point back to where it was originally. Hence, the exponent of the floating-decimal number is three. The same process can be used to convert the remaining numbers in the table.

Floating-decimal numbers such as those shown above are needed for accepting input and presenting output where the range of numbers involved is highly variable. Note the similarity to scientific notation, which for the same numbers would be 1.2533×10^2, 5.2864×10 and 9.8712×10^3.

However, for use in computations, one wants to deal with floating binary numbers of the form (in a thirty-bit word) seeeeeeeetmmmmmmmmmmmmmmmmmmmm, where s is the sign of the binary exponent, eeeeeeee is the binary exponent, t is the sign of the binary mantissa and mmmmmmmmmmmmmmmmmmmm is the binary

mantissa. In the case of floating binary, 0 <= eeeeeeee <512 and ½ <= mmmmmmmmmmmmmmmmmmmmm < 1.0 in value. In early computers, programs provided floating-binary (often packaged as reusable subroutines); in modern computers, floating binary arithmetic is built into the hardware.

Some other scheme is required to deal with non-numeric information. For purposes of this description, I shall assume the use of a computer with a 36-bit word length. With such a machine each word could store nine BCD numeric codes with no unused bits or six six-bit codes.

Various machines have used various six-bit codes to represent what is called alphanumeric information, that is, a combination of decimal digits, alphabetic characters (26 of them) and some selection of special characters. Such codes fit nicely into the 36-bit word. Just what their codes are, is unimportant. The proliferation of different codes in use has been a severe problem. One has often found it necessary to convert between various versions of such codes, which has high nuisance value, but is certainly doable.

However, alphanumeric data don't fit the world of words so nicely. For example, if I were to have data for First Name, Middle Initial and Last Name, I couldn't simply assign these to consecutive addresses. The first and last names wouldn't fit into a single word and it seems a shame to take up a whole word to store a single initial. So one could decide to place First Name in addresses 1004, 1005 and 1006, Middle Initial in 1007 and Last Name in 1008, 1009 and 1010 or one could allow for 17-character first names and store First Name and Middle Initial in 1004, 1005 and 1006 and 18-character last names in 1007, 1008 and 1009. The second scheme would save one word of storage, but would increase the processing time because of the need to separate the Middle Initial from the First Name during preparation of the output display. All of these things and many others are possible and have been used, no doubt, but they lack the straightforward simplicity of the word-oriented addressing found in numerical calculations.

The use of six-bit codes for representation of alphanumeric data has two disadvantages then: the proliferation of codes in use and the awkwardness of using word addressing to deal with data that are variable in length by their nature. An additional problem of six-bit codes is their inability to represent rich character sets such as those containing both upper and lower case alphabetic characters and/or those with special characters for foreign languages or for entire foreign language character sets.

The problems of the proliferation and richness of character sets were addressed concurrently by the American National Standards Institute (ANSI) and the International Standards Organization (ISO). This work came to fruition in the United States in the 1970s with the adoption of the ASCII code by the U. S. computer industry. ASCII stands for American Standard Code for Information Interchange. It consists of the eight-bit code in general use now for several decades and is an admirable solution to these early problems. In 36-bit machines, each word contains four eight-bit characters with four bits that can be used for parity or ignored.

The awkwardness of word-oriented addressing has been solved by the use of addressing at the character level—so-called variable word-length processing. This was first done with the introduction of such machines as the IBM-702, 705 and 1401. Although these machines had character addressing, they used six-bit characters and, hence, were only a partial solution. Most, if not all, modern machines use character addressing and use eight-bit characters that solve both the proliferation and richness problems. These eight-bit characters are referred to as bytes.

Historically, variable word length machines were at a disadvantage for performance of calculations. For example, if the average length of numbers in a calculation is 18 bits, then on average a variable-word-length machine with six-bit characters would need to access memory three times each time a number was retrieved or stored whereas a fixed-word-length machine would require only one memory access for the same retrieval or storage.

Most, if not all, modern machines can handle both situations efficiently by utilizing byte addressing, while accessing memory with fixed words. With inexpensive models, it is common to access memory eight bits at a time; while higher performance models do so sixteen or thirty-two bits at a time. Hence, in higher performance models, all modes of processing benefit from a reduction in memory accesses but more complex hardware is required to get the same job done as is performed in lower performance models.

In a few machines, the hardware was given the ability to operate in decimal; hence, data were represented in codes other than binary. I believe Univac used BCD and I know the IBM-650 used bi-quinary. In bi-quinary, five bits are used to represent the digits from 0 through 4 and a sixth bit is used to determine whether or not to add 5 to this amount. Hence, if the sixth bit were equal to 0 then the digit would be equal to the number of the first five bits that was on. So if the second bit were on, the value would be two, if the fourth bit were on, the value would be four,

etc. If the sixth bit were equal to one, then five would be added to the number of the first five bits that was on. So if the second bit were on, then the value would be seven, if the fourth bit were on, then the value would be nine.

How were data and instructions recorded on magnetic tape? In the early days, it depended upon the machine. For example, on the RAYDAC, data were always recorded in blocks of 32 30-bit words. I don't remember just how the magnetic spots corresponding to the bits were arranged on the tape. But what is more important, a standard for recording was arrived at fairly early in the industry permitting interchangeability of seven-channel and nine-channel tapes to occur between machines from various vendors.

Seven-channel tapes were used in the days of character machines. They recorded information in seven information channels arrayed across the width of the half-inch tape. Each "frame" recorded across the width contained six information bits and a seventh "parity" bit. The parity bit would be a one if the number of information bits was even and it would be a zero if the number of information bits were odd. That is, the frames were said to have "odd parity"; they could never contain all zero bits.

Nine-channel tapes were used in machines that used bytes to store alphanumeric information. Their data frames contained nine bits recorded across the width of the half-inch tape. Each frame contained eight information bits and one parity bit. They also used odd parity.

Most punched cards used the Hollerith Card Code. Each eighty-column card contains twelve rows and 80-columns. Each column contains a single character of information. If the information is numeric, it will be punched into the row number equaling its value. That is, a zero will be punched in row zero, a one in row one and so forth. If the information is alphabetic or a special character, it will be punched in a numeric row and one or both of the so-called X and Y rows located on the card atop the numeric rows.

Another important code is binary card code. In this case, each of the possible holes in the first seventy-two columns of each eighty-column card is used to represent a zero or a one—i. e. a bit. The last eight rows are used to contain Hollerith codes. In this way, a card can store 72 * 12 = 864 bits of binary information and still be sorted or selected by punched card equipment based upon the information contained in its last eight columns. The use of binary cards was used extensively on many mainframe computers for the storage of programs.

Appendix C—Computer Memories

The Internal Memories of computers come in a great many varieties and have evolved greatly over the short history of the technology. The first major division among the many varieties is serial versus parallel. This has to do with the mode in which words are entered into and retrieved from the memories. In a serial memory, each bit of a word is stored or retrieved in sequence; in a parallel memory, all of the bits of a word are stored or retrieved simultaneously. As one would expect, parallel memories are much faster than serial memories. Historically, serial memories were used because they were less expensive than parallel ones, but they became less and less advantageous as memory technology evolved. Now, most if not all memories are parallel.

The class of serial memories has two members: rotating magnetic memories and delay lines. The members of the class of parallel memories with which I am familiar are: Williams Tubes, magnetic cores and semiconductor memories. Each of these is described briefly below:

1.1 Serial Memories

C1. Rotating Magnetic Memories

Magnetic drums, one form of rotating magnetic memories, have been used from very early in the computer industry. They consist of a ferromagnetic cylinder mounted on its central axis. A set of read/write heads is placed in a row parallel to the axis along the periphery of the cylinder so each head can record bits on the surface of the drum in a fixed track and read bits from the same track.

The cylinder (drum) is rotated at a fixed rate while it is in operation so a constant number of bits pass beneath each read/write head in a single rotation. As the drum rotates each head may read a sequence of bits from its track or record a sequence of bits as magnetic spots on its track of the drum. Each drum has a track length of something like 16 or 32 or 64 words. If the machine had a 30-bit word and a 32-word track length, this would mean each track on the drum would have 30 x 32 = 960 magnetic spots. If the same drum had 32 read/write heads, its total storage would be 32 x 32 = 1024 words of storage.

The storage on each track is divided into sectors. So if the track capacity were 32 words, the track would be divided into 32 sectors numbered 0 through 31. So the storage of a word on a 32-track drum with 32 words per track could be addressed using 10-bit addresses; 5 bits for specifying the track and 5 bits for specifying the sector in which each word is to be stored. (How do I know this? Each track and each sector must have a unique binary address. Since there are 32 of tracks and 32 sectors, it requires some number, p, such that $2^p = 32$, to provide the correct number of addresses. The value of p that satisfies this relationship is p = 5. Try it: $2^5 = 2x2x2x2x2 = 32$.)

The words to be stored must be presented to the drum serially, so if the other components of the system deal with words in parallel, it is necessary to provide a parallel to serial conversion between the other components and the drum. Similarly, serial to parallel conversion between the drum and other components is necessary if the other components deal with words in parallel.

Of course, drums, like all serial memories, suffer a serious disadvantage—latency time. That is, the waiting time for the word being sought to arrive under its read/write head. Drums often rotate at 1800 rpm. This means a single rotation takes (60 seconds/1800 revolutions = 60,000 milliseconds/1800 revolutions =) 33 1/3 milliseconds, or the average wait for a word to arrive under its read/write head is half a revolution or 16 2/3 milliseconds. This is referred to as the drums latency time.

Having arrived under the head, it would take the time for the word to pass under the head to read or record the bits of the word. With a 32-word, track length this would be 33 1/3 milliseconds /32 = 1.067 milliseconds. So the average time to access a word would be 16.667 + 1.067 = 17.734 milliseconds.

The other variant of rotating magnetic memories is the magnetic disk. They are the same as magnetic drums in many ways except the rotating medium is a disk instead of a cylinder and instead of having one read/write head per track, one read/write head on an arm that can be repositioned over any desired track serves the entire disk. The time it takes to reposition the head is called seek time and must be added to the latency time and data transmission time. Hence, disks are somewhat slower than drums in general.

However, disks are usually used to transmit blocks of information to/from internal memory whereas drums are often used to move single words at a time. Therefore, disk sectors are generally large and contain a large

amount of information, so the seek time penalty is shared by many words so that the average seek time per word may remain relatively low.

C2. Delay-line Memories

Delay-line memories consist of some delaying medium through which a train of bits can be passed and then retrieved. If you have such a delaying medium, it will perform just like a magnetic drum. Each delay-line performs like a track on a magnetic drum.

The RAYDAC (and Univac and several other early computers) used Acoustic delay-line memory where the acoustic medium was mercury. Each delay line consisted of a path through a pool of mercury in which bits were entered by an electric-to-acoustic transducer and were exited by an acoustic-to-electric transducer. The 36 delay-lines in the RAYDAC Internal Memory were 32 words long. They all shared the same pool of mercury.

In the case of the delay-line memory, one considers the time for the entire 32 words to pass through a delay-line instead of the time for a drum to rotate. This time was referred to as the major cycle and was equal to 9.76 milliseconds in the case of RAYDAC. Hence, the average latency time is 4.88 milliseconds. The time for a single word to enter or leave a delay-line is called a minor cycle and equals 305 microseconds. Hence, the average access time to the RAYDAC delay-line memory was 5.185 milliseconds, or a little less than most drum memories of its time.

1.2 Parallel Memories

C3. Williams Tubes

Williams Tubes were small (under six inches in diameter) cathode ray tubes upon whose surfaces spots representing binary digits could be displayed. They also had the ability to read the spots on their surfaces. Hence, they were used to store and retrieve words of computer memory.

Williams Tubes were quite fast in relation to serial memories, but were not particularly reliable. They had to be run in darkness so the illumination of external light would not be sensed during retrieval as previously stored bits. On the other hand, they were convenient because you could look at the tubes and read the bits stored in memory (though it was difficult to associate a given bit with the word to which it belonged). The maintenance personnel liked this visibility because they could store

certain known patterns in memory and then look at the tubes to see if those patterns were the ones being displayed.

The access time of Williams Tubes was, as I recall, on the order of 20 microseconds per word.

C4. Magnetic Cores

The first really good parallel memories were supplied by magnetic cores. These were little ferromagnetic doughnuts strung in rectangular arrays with wires passing through them. Each core could be set, reset, or read by passing appropriate currents through the wires passing through it. In this way, many words of memory could be provided with very acceptable access times.

When they were first introduced, core memories were about the same speed as Williams tube memories, but far more reliable. The speed of a core memory is inversely proportional to the size of the core. So as it became possible to manufacture smaller and smaller cores, the memory speeds increased. At the same time as smaller cores were faster, they were also more reliable, occupied less physical space and utilized less energy. As this trend proceeded access times became as low as 2 microseconds per word and memory sizes of up to 265,000 36-bit words were common before core memories were superseded by semiconductor memories.

C5. Semiconductor Memories

The exploitation of semiconductors has provided the latest step forward in Internal Memory technology. Although some semiconductor memories are faster than any core memories, their biggest contribution has been the availability of larger memories. The contemporary practice of referring to memory sizes in terms of megabytes makes this increase clear; with core memories, we always talked about memory sizes in terms of kilobytes or kilo-words. For example, the machine upon which this is being typed has a 128-megabyte memory and that is not considered to be unusually large. I'm not sure of the memory speed, but I think it is 1 microsecond access time or perhaps it is even faster. In any case, it is plenty fast and well matched to the speed of the ALU that is, of course, also using semiconductor technology.

Many varieties of semiconductor memory exist—more than I am aware of in detail. A separate book, or a substantial essay could be written on that subject. Most of them are capable of being read and written, but some

are read only. Some are destructive read-out and some are not. In destructive read-out memories, reading a byte also resets it, so if it is to remain in memory, it must be rewritten. However, this all happens automatically and at very high speed. Best of all, semiconductor memories are more reliable than any of their predecessors.

Appendix D—Computer Interior Decor

The capabilities of a computer may be made visible to the programmer in a large number of ways. These are referred to as the "Interior Décor" of the computer. Some of the important versions of these are summarized below.

In addition to Internal Memory described elsewhere, certain forms of storage exist within the ALU. These are generally electronic storage devices and are called registers. Most computer operations involve three registers. The first of these is the IC or instruction counter, which specifies the memory location from which the next instruction will be retrieved. The other two are often referred to as the A and B registers. These are used to store the results of one operation so they will be available for use in the next operation. One or the other or in some cases both of these are involved in the execution of most operations the computer is capable of executing.

How does the computer know what to do? It does what is specified in the instruction most recently retrieved by the CCU from Internal Memory. In general, each type of computer has its own unique instruction format, but these can be broken into five categories each of which is about the same from one computer to the next. These are single-address, two-address, three-address, four-address and variable formats. Each of these is described below:

D1. Single-address Format

In single address format, each instruction contains an operand address and an operation code. These instructions are generally stored one to a word. So, in a machine with a 36-bit word length, an instruction might specify an 18-bit address, an 8-bit operation code and ten bits for other uses (that we will not get into in this brief overview). This would allow addressing 256K of memory (where K is 1024 words) and 256 operation codes.

Typical instructions might be as follows (all codes and addresses are in octal on an imaginary machine):

Instruction Address	Operation Code	Operand Address	Description
001000	001	123456	Load A register from memory location 123456
001001	010	123457	Add the contents of 123457 to the A register
001002	014	123460	Subtract the contents of 123460 from the A register
001003	004	123461	Store the contents of the A register in 123461
001004	001	123462	Load A register from memory location 123462
001005	020	123464	Perform the logical **and** of the contents of location 123464 and the A register
001006	030	123465	Perform the logical **or** of the contents of location 123465 and the A register.
001007	004	123466	Store the contents of the A register in location 123466
001010	070	001200	Obtain the next instruction from location 001200.

This small set of computer code might represent a small snippet of a program. During its execution, addresses 123456 through 123471 and 001200 are addressed. Their contents would need to be listed in the program and would consist of an area of memory (excluding 001200) such as the following:

Memory Address	Octal Value	Comments
123456	000000000144	Octal equivalent of 100 decimal
123457	000000000100	Octal equivalent of 64 decimal
123460	000000000002	Octal equivalent of 2 decimal
123461	000000000242	Octal equivalent of 162 decimal after execution of

		instruction stored in 1003
123462	626563630043	Octal equivalent of “RUSS C”
123463	005543474545	Octal equivalent of “ MCGEE”
123464	000000000077	Mask for use in instruction at location 1005
123465	202020202000	Octal equivalent of five blanks followed by zeros used in instruction location 1006
123466	202020202043	Octal equivalent of five spaces followed by the letter C. This is the result of executing instructions stored in 1004 through 1007—the middle initial, “C”.

The first four instructions, in locations 1000 through 1003, perform a small and trivial calculation that should be easy to understand. The instruction in 1004 loads the alphabetic string “RUSSbC” into the A register, where “b” stands for blank. The instruction in 1005 then sets the first five characters of the A register to zeros and leaves the “C” unchanged. The instruction in 1006 then replaces the leading zeros in the A-register by the octal code for blank—“20”. The instruction in 1007 stores this result in 123466. The last instruction at address 1010 instructs the CCU to retrieve its next instruction from octal address1200.

This example gives the general flavor of a one-address interior décor except for a few internal details. Some of these are important to get a more complete picture. In addition to the registers, the ALU has many triggers used to store the existence of certain conditions. These are generally exception conditions that may be tested for using special instructions. Examples of such exception conditions are overflow and divide by zero. In general, any condition leaving the result of the operation ambiguous will result in an exception that will raise some one or more triggers.

An example of an overflow would be adding two numbers together in the A register whose sum is greater than the capacity of the register. Divide by zero is clear from its name. These triggers are not peculiar to one-address machines, but will be present in all of the following.

D2: Two-Address Format

In this format, two addresses are available to specify operand addresses and additional bits in the instruction specify an operation code. It is

conceivable to define a two-address instruction format in which both addresses are used to refer to operands, but I know of no real example of such. However, the IBM-650 used a two-address format in which the first address was an operand address and the second was the address of the next instruction. This was to facilitate what was known as Minimum Latency Programming. To distinguish between a true two-address format and one like the IBM-650, the latter is referred to as a 1+1-address format.

The description of the one-address format will give the reader what he needs to know about understanding the use of the operand address and its connection to the operation code.

If a magnetic drum machine (or any other serial memory machine) is implemented with one-address instruction format, a serious problem with performance exists because the sequence of instructions to be executed will be one after another around the periphery of the drum. This is a very poor choice because instruction execution times are usually a small fraction of the time of one drum rotation. So after one instruction has been executed the computer must wait for almost an entire drum rotation to find its next instruction. It is the use of the second address, the instruction address, which may ameliorate this problem. This permits the sequence of instructions to be placed in locations so they will be under the read heads at exactly the time they are needed. This placement is specified by the second address of the instruction of the previously executed instruction.

In a machine with a serial memory, this capability can materially speed up the execution time of a program by placing instructions and their operands in memory locations so each will be under the read/write heads at just the time it is required to be there. By this ploy, much of the latency time characteristic of a serial memory is nullified with a very substantial improvement in performance. Programming so as to take advantage of this time saving is referred to as Minimum Latency Programming.

The IBM-650 had ten decimal digits per word. Eight digits were allocated for addresses and two for the operation code. Although this would have provided for memory addressing of up to 10,000 words, I don't think that any drums were offered larger than 2048 words.

The creation of minimum latency programs is a very tedious and labor-intensive task, not to mention the resulting code tends to be hard to debug and very inflexible. Some of these shortcomings were ameliorated by the use of compilers and assemblers capable of creating minimum latency code from sequential program descriptions. However, the much

better solution has been to use parallel memories where the problem being solved doesn't exist because every word in a parallel memory is accessible in the same small amount of time.

D3. Three-address Format

In this instruction format, the first two addresses specify operand addresses and the third specifies where the result is to be stored. Instructions are retrieved from sequential addresses unless the instruction counter is set by an instruction to a non-sequential address. If the word length is 36 bits, this scheme will provide for three 10-bit addresses and a six-bit operation code, for example.

Each instruction in the three-address format occupies more space than an instruction in the single address format. For example, a single three-address instruction could add the contents of two memory locations and put the result in a third. Let us suppose a single address machine is available with a sixteen-bit word length in which ten bits are used for the address and six for the operation code. In this latter machine, three instructions would be required to do the same job as a single three-address instruction. So our hypothetical three-address machine seems to have an advantage in terms of bits of instruction to do the same job versus our hypothetical single-address machine—that is, 36 versus 48. However, if the problem we used for comparison were to add two numbers, subtract a third, then store the result, the outcome would be different. In this case, the three-address format would require two words (seventy-two bits) of storage for instructions; the single-address format would require four words (sixty-four bits). Now the preference seems to have switched the other way. Of course, then you would want to argue about the validity of the cases chosen because the three-address machine is operating on 36-bit words and the single address machine on 16-bit words.

Needless to say, this problem has been extensively studied. It would seem the single-address format has won, because most modern machines tend to have mostly single-address characteristics.

1.3 D4. Four-address Format

It is not clear why anyone would want a four-address format containing four operand and storage addresses. Indeed, I know of none. However, a three-plus-one-address machine is possible and has been built and operated—RAYDAC. In this case, the word-length is 30 bits and the

instructions are stored in pairs of words the first word of which is stored in a word with an even address. The first instruction word contains two operand addresses and some unused bits; the second instruction word contains an address in which the result is to be stored, an operation code and the memory location in which the next instruction pair is to be found. For further explanation of this interior decor, see Appendix E.

The justification for this format is to facilitate the implementation of minimum latency code. (See discussion above, D2 Two-Address Format.)

D5. Variable-length Format

With the advent of character addressing, it was possible to introduce variable instruction length. In this scheme, an operation code occupies a character and the length of the instruction is determined from the operation code. In this way, each instruction uses only as much storage as it needs. And because the machine is character addressable, it is possible for code to completely fill the space in which it is stored. [Actually, performance reasons argue against always filling all of the space, but that is another topic and will not be discussed here.]

I was first exposed to variable word-length processing in the IBM-702. In this machine two modes of processing were provided: numeric and alphanumeric.

If you wanted to do some arithmetic, you would do a *RESET ADD* operation with the address of the first character of the operand. That is, the instruction would consist of the character used to designate the *RESET ADD* instruction followed by four characters of operand address. This would cause the ALU to go to the operand address and begin to load a large register from the specified address. If the character found there was a numeric character, it would be loaded into the register and the next character in memory would be examined. If it were also numeric, the loading process would continue. This would proceed until a non-numeric character was encountered. The numeric characters were all in BCD so their two most significant bits would be zero. The terminal character was also in BCD, but had one of the leading bits set to one indicating two things: this was the last character of the number and the sign of the number--positive or negative.

The register loaded by a *RESET ADD* was quite large—256 characters I think, so lack of precision was never a problem. It performed the role of

the A and B registers in other decors. To proceed with a calculation after the *RESET ADD*, other numeric operations existed in the same format—one character operation code followed by four-character address. A complete set of fixed-point arithmetic operations was provided.

To operate on alphanumeric data, the same register was utilized, but it was loaded in a different manner. In this case, one used an operation called *SET LEFT*. This instruction set the length of the register. Having set the length, one could perform a *LOAD* instruction consisting of the *LOAD* operation code followed by a four-character address. The ALU executed the *LOAD* operation by copying the characters starting with the specified address from memory to the register starting with the character to which the register is *SET LEFT*. Having loaded the register, it is now possible to execute a whole suite of other character operations provided by the ALU.

Appendix E—Computer Programming Languages

In modern computers, a user has many choices from which to choose when writing a program. This has not always been the case. On early computers such as EDSAC and RAYDAC, (or even in the factory where entirely new computers are being made) the only language originally available in which to program is the language of the computer itself. This will generally be binary and it is very awkward to work with. The computer will generally have a certain word length and will store one instruction per word (this is not necessarily the case, but is a useful simplifying assumption for descriptive purposes).

For example, let us assume, for purposes of illustration, we have a one-address computer with an 18-bit word. Let us further assume the machine is capable of performing 64 different operations and has a memory containing 4096 words. In this case, each instruction will consist of a six-bit operation code, followed by a 12-bit address. The operation code will instruct the computer what to do and the address will give the location in memory where the operand of the instruction is to be found. We will assume further, the machine has an 18-bit A-register and an 18-bit B-register.

Now the computer will, no doubt, have some rows of switches permitting an operator to insert instructions, one at a time, into memory locations also specified by setting switches. This will be useful for engineers performing maintenance and hardware debugging, but will not be efficient for entering programs of any significant length. Rather, one will wish to have a way of planning and expressing an entire program in written form readable by others and used by the program's author to refresh his or her memory after the initial creation has been accomplished.

One choice would be to use binary. In this case, we could write a small program in a sequence something like this:

Location	Op. code	Operand address	Comment
000000000000	000001	000001000000	Load contents of 64 into A
000000000001	000100	000001000001	Add contents of 65 to A
000000000010	000101	000001000010	Subtract the contents of 66 from A
000000000011	001000	000001000011	Store contents of A in location 67
.			
.			
000001000000	000010000000		The value 128 decimal in binary
000001000001	000000000100		4 decimal in binary
000001000010	000000000010		2 decimal in binary
000001000011	000010000010		the result (130) in binary

This is not a very lucid description. If it were not for the comments, it would be almost entirely incomprehensible. Even so, it is probably worth going through the example in words, to make sure it is clear. We will assume the computer begins executing at location 0—the word whose location is all zeros, in binary. Apparently the op code 000001 instructs the ALU to load the A-register with the contents of the address field. The address field contains the binary equivalent of 64. Looking to the location with this address in the location field, we see it contains the value 128 decimal expressed in binary. Hence, the result of the first line of code is to place the binary equivalent of 128 decimal in the A-register.

The computer now proceeds to location 000000000001 for its next instruction. Here it finds an op code of 000100, which apparently means add the contents of the address field to the A-register. The address field contains the binary equivalent of 65. Looking to the location with this address in the location field, we see it contains the binary equivalent of 4 decimal. Hence, the result of the second line of code is to place 128 + 4 = 132 in the A-register.

The computer now proceeds to location 000000000010 for its next instruction. Here it finds an op code of 000101, which apparently means subtract the contents of the address field from the A-register. The address field contains the binary equivalent of 66. Looking in location 66, we see it contains the binary equivalent of 2 decimal. Hence, the result of the third line of code is to place 132 - 2 = 130 in the A-register.

The computer now proceeds to location 000000000011 for its next instruction. Here it finds an op code of 001000, which apparently means store the contents of the A-register in the location specified by the address field. The address field contains the binary equivalent of 67. Hence, the result of the last line of code is to place the contents of the A-register (= 130) in address 67.

This is all correct (I think) and one could code in this manner, but it is unnecessarily labor intensive and hard to follow. The very next step in clarity and efficiency is to use either octal or hexadecimal numbers to express the binary quantities. After making this change to octal, the same code looks as follows:

Location	Op. code	Operand address	Comment
000	01	0100	Load contents of 64 (octal 100) into A
001	04	0101	Add contents of 65 (octal 101) to A
010	05	0102	Subtract the contents of 66(octal 102) from A
011	10	0103	Store contents of A in location 67(octal 103)

.
.

100	0200	The value 128 decimal in octal
101	0004	4 decimal in octal
102	0002	2 decimal in octal
103	0202	the result (130) in octal

This is, at least, more efficient of writing effort and paper. It will need a program of some kind to convert the octal digits written down above to binary and place them in the indicated addresses. This program would be called a loader and is needed anyway for other reasons.

However, it still lacks anything that could be characterized as application related. For example, suppose the problem involved here was to adjust the balance in an account by adding a deposit and subtracting a withdrawal. It would be useful to express the code so the significance of the quantities being dealt within the code were clear to the reader. To achieve this objective, symbols are introduced.

The first thing to do is to make the op codes more mnemonic. For example, if we use three characters to represent op codes, we could make the four we have used in our example be LDA for Load the A-register, ADA for Add to the A-register, SBA for Subtract from the A-register and STA for Store the contents of the A-register. Such assignments will be done for each of the op codes of the machine and used in place of any binary or octal coding by everyone who uses the symbolic language.

However, the user will be permitted, within some limitations, to select symbols of his or her choosing for the variables of the problem. The limitations arise because of the need to translate this language into binary machine language. A program called an assembler or an assembly program will perform this translation. In the process of doing so, the assembler will need to build a table, called a symbol table, relating the various symbols to their locations in the resulting machine language program (the so-called object program). To keep the assembler's job manageable, limits are placed upon the length of the symbols programmers can utilize and the characters symbols may contain. In our case, we shall assume symbols may be up to six characters in length and may contain any combination of 26 upper-case letters and ten decimal digits.

Finally, we will need a way to declare to the assembler what storage to allocate in performing the actions of the program and the initial values to assign to them. For this purpose, we shall introduce a single pseudo-operation. (In an actual assembly language, many pseudo-operations

might be provided.) The one we shall use is DEC, which allocates a single word of memory and sets its value to the binary equivalent of the decimal value provided with the pseudo-operation. With this background, the sample problem shown above can be rewritten as follows:

Location	Operation	Operand	Comment
BALANC	DEC	128	Initial value of balance
DPOSIT	DEC	4	Amount of deposit
WTHDRL	DEC	2	Amount of withdrawal
NEWBAL	DEC	0	New balance value (before run)
START	LDA	BALANC	Load A with balance
	ADA	DPOSIT	Add deposit
	SBA	WTHDRL	Subtract withdrawal
	STA	NEWBAL	Store new balance

Although this is not entirely plain English or mathematics, it is a large step forward toward the objective of problem orientation. Assemblers can be made very capable, but they still reflect the characteristics of the machines on which they are used. Notice the language written here must be processed by an assembler before it may be in a form acceptable to a loader, which will place the assembled bits in their proper locations in memory ready for execution.

Machine-independence is a long-sought-after objective in computer languages. Many languages have been invented approaching this goal and have had varying degrees of success in achieving it. It is not the purpose of this appendix to provide a complete compendium of such languages, but we shall take a look at BASIC as an example.

If we were to rewrite the program for our simple problem in the language of BASIC, it might look like the following:

```
MAIN DIM BALANCE, DEPOSIT, WITHDRAWAL, NEWBALANCE;

     PRINT "ENTER BALANCE, DEPOSIT, WITHDRAWAL";
     INPUT BALANCE, DEPOSIT, WITHDRAWAL;

     NEWBALANCE = BALANCE + DEPOSIT – WITHDRAWAL;

     PRINT "NEW BALANCE = "NEWBALANCE;
     END
```

First, we notice the program is given a name: "MAIN". This is, in fact the name of a block of BASIC code. A complete BASIC program might contain many such named blocks. The remainder of the first line of the

program is a "dimension" statement. It is used to declare by name the variables to be used in the program. In our case the variables are BALANCE, DEPOSIT, WITHDRAWAL and NEWBALANCE. Now skipping to the fourth line, we see the entire remainder of what was contained in the earlier versions of the program. The other lines in the BASIC program provide a way for the user to provide input values and to see the result of the calculation. (These details were omitted from the earlier examples because no method of providing input and output had been invented for our imagined machine. The machine-independent language of BASIC provides means for accomplishing these tasks.)

Users at time-sharing terminals were the persons for which BASIC was originally created. One should imagine such a user while reading the BASIC program. The PRINT statement on the second line invites the user to enter values for the BALANCE, DEPOSIT and WITHDRAWAL variables separated by commas. When he or she does so, the values entered will be assigned to the three variables as specified in the PRINT statement in the third line. The NEWBALANCE will be calculated as per the fourth line. The fifth line tells the computer to PRINT the computed value of NEWBALANCE preceded by the text "NEW BALANCE = ".

Programs written in languages such as BASIC, like more machine-oriented languages, must be translated to the language of the machine upon which they are to execute before they may be used. These translators are called compilers and will be referred to in the text of this document.

Appendix F—RAYDAC Programming

Some additional information about its interior décor is necessary before a discussion of RAYDAC Programming is possible. The main internal memory of the machine contained 1024 30-bit words. These were addressed with the octal addresses 2000 through 3777. Its A and B registers were referred to with the addresses 0100 and 0200. In addition, two special registers were provided: one with octal address 0300 that facilitated locating information on magnetic tape and one with the octal address 0400, used to send information to the console Teletype unit. The machine had four magnetic tape handlers, each of which had an associated 32-word memory buffer into which it read input and out of which it wrote output. The four buffers associated with the four handlers were addressed with the octal addresses 1100-1137, 1200-1237, 1300-1337 and 1400-1437 and could be used as extensions to the main internal memory.

With this information, we can repeat the sample code given in Appendix D for single-address machines here in the 3+1-address format of the RAYDAC.

Location	Address 1	Address 2	Operation
	Address 3	Address 4	Comment
2000	3000	3001	01
	0100	2002	Add the contents of 3000 to the contents of 3001 and leave the result in the A-register. Get the next instruction from 2002.
2002	0100	3002	04
	3003	2004	Subtract the contents of 3002 from the contents of the A-register and put the result in 3003. Get the next instruction from 2004.
2004	3005	3007	63
	0100	2006	Shift the contents of location 3005 right by the amount in location 3007 and leave the result in the A-register. Get the next instruction from 2006.
2006	0100	3010	34
	3011	2010	Perform the logical **or** of the contents of the A-register and the contents of 3010 and place the

			result in 3011. Continue to 2010.
2010	0100	3012	52
	2012	2100	This is a conditional branch instruction. If the contents of the A-register equal the contents of 3012, then the next instruction is taken from 2012-13, otherwise it is taken from 2100-1.

The storage for this program snippet would be as follows:

Address	Octal Value	Comments
3000	0000000144	Octal equivalent of decimal 100.
3001	0000000100	Octal equivalent of decimal 64.
3002	0000000002	Octal equivalent of decimal 2.
3003	0000000242	Octal equivalent of decimal 162 after execution of instruction in 2002-3.
3004	6265636320	Octal equivalent of "RUSSb" [b = blank]
3005	4320554347	Octal equivalent of "CbMCG"
3006	45455202020	Octal equivalent of "EEbbb"
3007	0000000030	Octal equivalent of decimal 24—the amount of shift in the instruction at 2004-5.
3010	2020202000	Operand, "bbbb0", for performing logical **or** in locations 2006-7.
3011	2020202043	Octal equivalent of "bbbbC" after execution of instruction at 2010-11 [Middle Initial].
3012	2020202043	Constant for performing test at locations 2012-13.

The first two instructions calculate the result of the expression 100+64-2 and place the result in location 3003. The next two instructions, at 2004 and 2006, load an alphanumeric string into the A-register, perform a right shift to place the six bits originally in the high order positions of the string "CbMCG" into the low-order six bits of the A-register. The spaces vacated on the left are filled with zeros. A logical **or** sets these zeroed bits to blanks and stores the result (the middle initial "C") in location 3011. The final instruction, in 2010, is a conditional branch demonstrating a very important class of operations, which permit programs to adjust their flow to the particular data they are processing.

In many instances in this code an operand has been retrieved or stored in the A-register in lieu of a memory location. This choice was made to

increase the speed of the code. The access time from registers was much faster than from memory.

In truth, my memory does not recall the exact operations the RAYDAC performed or their operation codes. However, the instructions in this code are representative of the ones it performed. In particular, I am sure add, subtract, multiply, divide, various shifts, logical **and**s and **or**s and a variety of conditional branch instructions were provided. You could branch on equal or not equal, or on greater than or less than, or on greater than or equal to, or on less than or equal to. This small sample of RAYDAC code gives a slight flavor of how a three plus one address décor appears.

The magnetic tape handlers used standard half-inch magnetically coated plastic tape (to the extent a standard then existed) of, I believe, 1200-foot length. However, they had optical block numbers on their non-magnetic side—a feature unique to RAYDAC. These were printed on the tapes using a special press, which placed the block number in binary code on the back of each tape before it was put into use. Each block was a 32-word parcel of storage, which would just fill one of the four above mentioned buffer delay lines.

The optical block numbers were recorded as a sequence of bars representing ones on one side of the tape and bars representing zeros on the opposite side. They were kind of a precursor to the bar codes used today to read the identity of your purchases at the supermarket.

To read information into the machine from magnetic tape, one would place the number of the block one wished to read into a register addressed with the octal address 0300, and then one would issue a **hunt** command. The tape handler would find the desired block on the specified handler by reading the optical block numbers on the non-magnetic side of the tape and would leave the block that had been **hunt***ed* for under the read/write head. If the program then issued a read instruction, the contents of the magnetic tape block would be read from the tape into buffer memory and could then be addressed directly using the buffers's set of 32 addresses. If, instead, the program issued a write instruction following a **hunt**, the contents of the buffer would be recorded on the magnetic tape block for the specified handler. This special ability to position the tape where one wanted it gave the RAYDAC magnetic tape handlers some of the abilities later available on magnetic disk handlers, though at much lower speed and reliability.

To complete the system, two additional devices were needed that were peripheral to RAYDAC. These were the Problem Preparation Unit, PPU,

and the Output Printer. The PPU was used to record information on magnetic tape typed on a Teletype machine by a human operator. The Output Printer was used to print information recorded on magnetic tape on a Teletype machine.

Appendix G—RAYDAC Assembly Program

The main problem I was trying to solve with the RAYDAC Assembly Program was the ability to create a program from a set of previously written/tested and used subprograms [=subroutines]. A reduction of duplicate programming and debugging effort results from this approach. It is also a way of permitting specialization among the programming staff. Each programmer can be permitted to create those subprograms that his or her background and skills have best prepared him or her to create. In the ideal case, one would be able to create a new program by writing a single master subroutine that performs its functions by merely calling other pre-written and pre-tested subroutines in the correct sequence and circumstances. It is unusual to find a problem and a set of subprograms complementing one another this well, but it is an ideal to which programmers aspire.

In the case of the RAYDAC and other machines of its time, the programmer was provided with no greater capability than the instruction set built into the hardware. Any other capability had to be provided by programs—hopefully in the form of reusable subroutines. These had to be provided to perform floating-point arithmetic, trigonometric functions, logarithmic and exponential functions, conversions between number systems, program/subprogram loading and many other tasks and capabilities.

The first problem one must solve to achieve this building block approach to program construction is the ability to relocate subprograms. That is, it is customary when writing RAYDAC code to begin at location 2000 octal and place succeeding instructions in successively higher-addressed word pairs. However, it is obvious every subroutine cannot start at address 2000 at execution time—only one may be there and the others must be displaced to other locations.

If this problem is solved, then it is necessary to solve the similar problem with respect to the subprogram's data. Suppose each subprogram starts its data storage in the word at octal 3000 and addresses that increase in sequence from there. It is now necessary to relocate the addresses of the data of each subprogram so each may properly address its own data and not be allowed to address the data of another subprogram.

Finally, a method must be provided to permit interchange of data among subprograms. This is often accomplished by providing a common area every subprogram may address.

Because the ability of the RAYDAC to handle alphanumeric data was limited and awkward, I did not attempt to utilize any alphanumeric characters in the RAYDAC Assembly Program. Instead, I took advantage of the fact that many of the addresses in the total address space of the RAYDAC were not used by the hardware. So, out of a total address space spanning the octal range from 0000 to 7777, only the addresses 0100, 0200, 0300, 0400, 1100 through 1277 and 2000 through 3777 were used by the hardware—all other addresses were illegal and unused.

I defined a standard subroutine coding convention that worked with the assembly program to facilitate subprogram relocation and data reference. When writing code according to this convention, you would start all subroutines at octal address 4000 and add instructions in address pairs in the range 4000 to 4777. You would place all data used internally by the subroutine in addresses starting at 5000 and increasing to as high as 5777. All common storage would be assigned to addresses starting at 6000 and increasing to as high as 6777.

The proper use of common storage was key. Each subprogram had certain common storage addresses containing its inputs and certain common addresses containing its outputs. It was the responsibility of the calling program to place the necessary inputs in the appropriate common storage addresses before calling a subprogram and to remove the results from the appropriate common storage addresses after the execution of a called subprogram. It was also the responsibility of the Assembly Program to relocate those addresses to that the caller and the called addresses matched up as they were required to.

The assembly program would create a program ready for execution by going through each of the program's subroutines and reassigning the fictitious addresses in the 4000 through 5777 range to legal addresses in the 2000 through 3777 range, leaving all addresses starting with 0 unchanged. It also created a common area of sufficient size and reassigned all addresses used in the 6000 to 6777 range to it. This is the general scheme that was utilized, but I have been unable to reconstruct just how the assembler established communication between the various common areas of various subroutines, especially when subroutines were calling subroutines.

Appendix H. Multi-Programming

Multi-programming is the ability of a computer to simultaneously progress in the execution of more than one program at a time. This ability is facilitated by the introduction of an event called an "interrupt". An interrupt causes the processor to cease what it is doing, store the state of all its registers at the time it was interrupted and continue execution at a place, called the INT for interrupt handler, in a memory location that is a function of the particular interrupt. When the processor arrives at the INT it will execute a sequence of instructions tailored to the needs of the particular interrupt.

Interrupts can occur for a wide variety of reasons, but some of the most common occur in conjunction with the execution of input and output. For example, an interrupt will typically occur after the completion of an input/output operation. It will be either a normal interrupt indicating successful completion of an operation or an abnormal one signaling an error or malfunction. A distinct interrupt will occur in each of these cases.

In the case of the normal interrupt, the interrupt handler will return an okay signal to the program for which the command was being executed. If the interrupt signals a malfunction, a recovery procedure will normally be attempted. For example, if this is a read error on a magnetic tape handler, the tape will be backspaced and a new read instruction issued. If the new read is successful, an okay signal will be returned to the program that originally issued the read instruction and it can then continue processing. Otherwise, several attempts will be made to reread the record. If after several attempts the instruction has still not succeeded in executing, a message is issued to the operator and he will be required to take corrective action.

Of course, the use of interrupts can become complex because additional interrupts can occur while an interrupt handler is in execution, so interrupts can be interrupted by interrupts at multiple levels. This is not a problem normally, and interrupts usually obey some priority rules so lower priority interrupts do not interrupt higher ones. Also an ability to mask interrupts is usually provided so the occurrence of interrupts can be inhibited; however, this capability must be used with extreme care because excessive inhibition of interrupts can, in some circumstances, induce errors.

When the interrupt capability is present, the user normally expects interrupt handlers will be supplied automatically when he/she executes a program. That is, the handling of interrupts is part of the system programs loaded before the user's application program so the user program is not concerned with these details.

Given the interrupt capability, it is not difficult to write programs that can progress concurrently—in other words, to achieve multiprogramming. To illustrate the point, consider the media conversion programs utilized in the IBM-1401. In this case, the media were cards, printer paper and magnetic tape; the conversions were card to tape, tape to card and tape to printer. A conversion program was present for each conversion supportable by the peripheral hardware configuration. For example, if the configuration contained one card reader, one cardpunch, two printers and four magnetic tape handlers; one card-to-tape, one tape-to-card and two tape-to-printer programs could be in execution concurrently and keeping all of the peripheral devices except the tape handlers operating at their full rated speeds.

Each of these programs would have been trivially simple. The tape-to-printer program, for instance, would have consisted of a program that said, "Read Magnetic Tape, Write line to printer" and would have repeated this instruction pair without limit. This program would have been supported by a full array of interrupt handlers instructing the operator on his or her console to perform certain tasks. For example, if printer2 ran out of paper, an interrupt would have occurred activating an interrupt handler that displayed, "Load Paper on Printer2" on the operator's console. When the conversion was completed, an End-of-File signal would have occurred on the Magnetic Tape Handler that would have said "End-of-File" to the appropriate interrupt handler. The handler would then have displayed "Tape-to-Printer Conversion on Printer 2 Complete" on the operator's console.

How did the programs get started? When an operator was ready to do a conversion, he would have mounted the magnetic tape to be used and then readied the other peripheral device. The peripheral device would have been in stand-by status waiting for something to do. When the operator placed it in ready status, an interrupt would have been issued signaling this change in status. The handler for the interrupt would have passed control to the conversion program associated with the device that caused the interrupt. When all of the devices needed for the conversion to proceed, were in ready status, the program to perform the media conversion would have sensed this state and would have automatically started.

This level of multi-programming is known as SPOOLing. SPOOL is an acronym for Simultaneous Peripheral Or OffLine.

Programs such as these would take up very little processor time. Most of the execution time would be spent waiting for mechanical devices to move. Hence, it was easy for a small computer to keep many conversions going without running out of processor capacity.

Once this basic level of multiprogramming has been achieved, it is a fairly small step to permit any programs for which adequate memory (or virtual memory) is available to execute concurrently. However, in this instance, additional hardware support is needed so as to prevent interference between the various running programs. Various means of providing this added level of protection have existed. These forms of multiprogramming are commonly seen in timesharing systems.

Appendix I. Blocking and Buffering

Two efficiency problems occur with magnetic tape and some other media that are ameliorated with the use of Blocking and Buffering. The problems and the use of Blocking and Buffering in solving them are described in this appendix.

When recording data, it takes a magnetic tape some time to get up to speed and to stop after the recording is complete. The same is true for reading. To accommodate these needs to speed up and to slow down, each record on the tape is surrounded by what is called an "inter-record gap". This gap is about a half inch long. Hence, the tape is not all recorded data, but a bunch of records separated by gaps. If the records are laid down on the tape at 200 characters per inch, a tape record created by a card-to-tape converter to record the contents of an eighty column card (a standard IBM card) would occupy 80/200 = 0.4 inches. Therefore, a complete tape of such records would contain only 0.4/(0.4 + 0.5) * 100 = 44% recorded information and 56% inter-record gaps. The situation with 120 character print lines is somewhat better, but still not ideal—0.6/(0.6+0.5) * 100 = 54% recorded data and 46% inter-record gaps. In general, about half of a tape consists of inter-record gaps when the recording density is 200 characters per inch. As recording densities increase, this problem becomes more and more severe.

The solution to the problem is to use blocking, which means records are not read and recorded singly, but several at once. This, of course, means that programs written to use unblocked data must be changed to permit them to use blocked data. Although this is true, the difference is small and easy for the user to deal with.

Most computers, and the IBM 700-series and the GE 600-line in particular, have a feature called index registers that permit a user to refer to data relative to a starting address contained in the index register. Hence, if you write a program that refers to a single set of contiguous data, you could modify it to refer to many sets of contiguous data by simply making reference to each set relative to a different starting address and changing the starting address each time it changed from one set to the next. So if I write a program that refers to **A**, **B** and **C** and each of these is one word long; I can change it to a program that refers to a set of such triplets by replacing each reference to **A,X**, **B,X** and **C,X** where the addition of the "**,X**" means "relative to the contents of index register **X**". If I now arrange to set the index register to the starting location of the triplet

before I perform the reference, I will have achieved the desired modification.

Because this problem is widespread and most programs find the use of blocking to be beneficial, standard subroutines are typically supplied that perform the input and output of blocked records so as to relieve each programmer from the need to consider it in detail and to guarantee interchangeability of blocked records produced by different programs. Hence, if I do **not** use **blocking**, I might record one of my triplets with the instruction **Write A, 3*6,1** where **A** is the address where the triplet starts, **3*6** is the length of the triplet in characters (in a 36-bit word-length machine) and **1** is the number of the tape handler upon which the triplet is to be recorded. Later, I may reread the same record and address its contents as **A**, **B** and **C** as before.

However, if I do **use blocking**, I perform an instruction like **Open for Output Trip, 3*6, 1, 5, X** early in the execution of the program. This announces to a blocking subroutine I wish to create a file with a blocking factor of **5** named **Trip** on tape handler **1** upon which this program will write records of length **3*6** characters. At the same time, the subroutine will set a specified index register, in this case **X,** to contain the location in which to place the first triplet for output. As the values of **A, B** and **C** are determined by the program, they will be stored in **A,X**, **B,X** and **C,X.** When a triplet is to be written to tape the program will perform an instruction like **Put Trip.** The Put subroutine will see if enough space is available in the block it is creating for another triplet; if so, it will advance the index register to the next available space and return control to the program that issued the Put. If this Put is for the last available space in the block, the subroutine will write the entire block to tape **1**, reset the index register to the location of the first triplet in the block and then return control to the program that issued the Put after the write to tape **1** is complete.

By using this technique, the effective speed of tape handlers can be increased and the capacity of individual tapes can be substantially increased. Some comparisons of capacities in percent of tape utilized versus blocking factors and recording density are shown in the following two tables.

Percent of Tape Utilized with 80 Character Records

Recording Density (Chars per inch)	Blocking Factor = 1	Blocking Factor = 5	Blocking Factor = 10
200	44%	80%	89%
400	29%	67%	80%
800	17%	50%	67%

Percent of Tape Utilized with 120 Character Records

200	54%	86%	92%
400	38%	75%	86%
800	23%	60%	75%

By multiplying these numbers by the instantaneous character transfer rate while the tape is moving (at 75 inches per second) the maximum attainable data transfer rate of each of these combinations can be calculated. The following two tables give these results.

Maximum Average Transfer Rate (Chars/sec) with 80 Character Records

Recording Density Chars/Inch	Blocking Factor =1	Blocking Factor = 5	Blocking Factor = 10
200	6600	12000	13350
400	8700	20100	24000
800	10200	30000	40200

Maximum Average Transfer Rate (Chars/sec) with 120 Character Records

Recording Density Chars/Inch	Blocking Factor = 1	Blocking Factor = 5	Blocking Factor = 10
200	8100	12900	13800
400	11400	22500	25800
800	13800	36000	45000

The other efficiency problem has to do with the sequencing of events required so as to permit input/output operations to occur concurrently with central processor operations. This simultaneity requires the hardware to be capable of operating its central processor and its input/output devices concurrently (which was the case with the 700-series and the GE 600-line), but it is also necessary to organize the program in such a way it will take advantage of this potential. Assume the hardware will permit simultaneous operation of the processor and not more than one magnetic tape handler at a time and consider a simple program performing the following sequence repeatedly:

Read record from tape 1
Compute using input from tape 1
Write results to tape 2.

As it is written, it is not possible to utilize simultaneity. The **compute** cannot proceed until the **Read** is finished because it has no input to use and the **Write** cannot begin before the **compute** is finished because it has no results to write. Also, the next **Read** cannot begin until the last **Write** has been completed because we have assumed only one tape can be in operation at a time. The program would work as written, but lots of time would be spent waiting for tape handlers to complete the tasks they have been commanded to perform.

To solve this problem, we expand the subroutine that performs blocking and ask it to also perform what is called buffering. With this capability we will always read input in advance of the time of its use and provide extra storage in memory to hold new outputs while earlier outputs are being written to tape. To make this work, we write the program in a slightly different way. It would now look as follows:

```
Open for Input Inp, L1, 1, 5, X, B
Open for Output Outp, L2, 2, 5, Y, B
        Start Loop
                Get Inp
                Compute
                Put Outp
        Repeat Loop until EOF(1)
Close Inp
Close Outp
```

This is a bit more complex than what we have looked at before, but it accomplishes a great deal more. The two **Open**s announce to the I/O subroutine what files are to be involved, the length of the records in each (**L1 and L2**), the tape handlers on which the files are to be mounted (**1 and 2)**, the blocking factor for each file (**5**)**,** the index registers to be used in referencing the data within the records (**X and Y**) and that the files are each to be buffered (**B**). When the **Open for Input** subroutine is executed, it will read a block of input into one of the input buffers and return control to the program that issued the **Open**. The **Open for Output** subroutine will allocate two blocks of storage to be used to hold outputs in memory preparatory to being written to tape 2, will set index register **Y** to the beginning of the first block and return control to the program that issued the **Open.** The **Loop** will then be traversed repeatedly until the end-of-file occurs on the tape 1 handler (**EOF(1)**).

The first instruction within the **Loop,** the **Get**, will cause the I/O subroutine to set index register **X** to the location of the beginning of the next input record to be processed. If this is the first record of a new block, it will also

start a **Read** operation to bring the next unread block from tape to the buffer whose records have just been processed. It will then return control to the program that issued the **Get.** Whatever computations are to take place will do so in the **Compute** line making reference to inputs using index register **X** and references to outputs using index register **Y**.

When the data for a complete record has been stored, the **Put** instruction will be executed. This will cause the I/O subroutine to increment the address in index register **Y** to the next available record space. If room is available in the current block for another record, the new value in the index register will be an address in the current block. If the current buffer is full, the index register will be set to the first address in the alternate storage buffer and a **Write** instruction will be issued to tape handler 2 to record on tape the contents of the full block. After these actions, control will be returned to the program that issued the **Put**. These events will be performed repeatedly until no more input exists to be processed. This condition will be signaled by **EOF(1)** (meaning end-of-file on handler 1) becoming true.

The result of this approach is to have a continuous supply of input available to the **Compute** line and to always have space in a buffer block for its output. If this can be achieved, the program can progress at the rated speed of the central processor or the tape handlers (whichever are limiting) and never need to slow down to await action on the other type of device. Of course, whether or not this is achieved depends upon the balance between the amounts of central processor work versus I/O work involved in the particular program. However, by using blocking and buffering, it is possible to eliminate many of the repeated waits upon I/O that would otherwise be encountered repeatedly.

This description of blocking and buffering has been presented using the example of magnetic tape as the I/O medium. However, the same logic works with a little modification on magnetic disk. The big difference is that on disk, the block size is normally an inherent property of the device. Each surface of a disk storage unit is formatted to contain a certain number of concentric tracks. The amount of storage a track may contain is fixed and constant. Each track is further subdivided into sectors. Again, the number of sectors is fixed and constant. If one writes a record shorter than a sector to disk storage, it will be written in the beginning portion of the sector and the rest of the sector will remain unused. Hence, one wants to write records to disk that are as nearly as possible about one sector in length. In other words, one wishes the block size on disk to be the sector size or a little under. With this understanding, the information given above in relation to magnetic tape can also apply to magnetic disk.

Appendix J—GE-645 Addressing

This appendix describes how the GE-645 with 18-bit addresses was able to reference more than 256k unique memory locations. In brief, the GE-645 expected the users and the Operating System to employ virtual memory.

The real address space of the GE-645 was the same old 256k set of addresses used on the GE-625/35. The virtual memory was a much larger space, which consisted of 256k such spaces, each referred to as a segment. [57] So the virtual address space was a two dimensional array: the first dimension consisted of segments each of which was addressed in the manner of the GE-625/35, the second dimension consisted of a segment number that ranged from 0 to as high as 256k – 1. Each user process had such a virtual memory potentially as extensive as this for its exclusive use.

The address within a segment was formed in the same way that any address was formed for the real memory of a GE-625/35 machine. It consisted of the address resulting after all address modifications, such as indexing and indirection had been performed and was referred to as the "effective address". Formation of the effective address was the first step in forming the virtual memory address.

To complete the formation of the virtual memory address, it was augmented by a segment number that was stored in one of two 18-bit registers: the PDR or Procedure Descriptor Register if this was an instruction access or the DDR or Data Descriptor Register if this was a data access. There were eight DDRs. The DDR to be used in a particular instruction was specified by three bits of the instruction word. With these two values—the effective address and the segment number from one of the Descriptor Registers—the virtual memory address was fully specified.

But how could many users be concurrently provided with their own unique virtual memory spaces? This was done with the introduction of another 18-bit register—the Descriptor Segment Base Register. As the name suggests, this register contained the address that led to the beginning of a segment called the Descriptor Segment. A Descriptor Segment contained one entry for each segment in the virtual memory available to a single user. The Multics File System constructed the Descriptor Segment for each user process as it confirmed the user's right to access the segment. Multics could change the virtual memory from one user process to another by simply resetting the contents of the Descriptor Segment

Base Register. The value in the Descriptor Segment Base Register led to the real memory address of a Descriptor Segment.

But how was a virtual memory address translated to a real memory address? The answer is that each entry in the Descriptor Segment contained the address of another table, the Page Table for that Segment. Only part of a segment, a Page, needed to be stored in real memory while the process that used the segment was in execution, though more than one page might have been in real memory if space were available. Pages, in the GE-645, are 1024 words in size. The Page Table for a segment contained one entry for each page of the segment and the location of each page in real memory was contained in the Page Table entry, if the page was in real memory. The page table entries also contained bits that indicated (among other things) if each page was or was not contained in real memory.

If a reference were made to a word in a page of virtual memory that was in real memory, the processor would construct the real memory address of the reference by using the location of the page from the segment's Page Table and the offset within the page from the effective address. If the page containing the reference were not in real memory, the process making the reference would be taken out of execution and control of the processor would be given to the File System to start retrieval of the needed page from secondary storage. After this action had been started, control of the processor would be given to another process, which was perhaps previously removed from execution because of a missing page.

But how about the Descriptor Segment, was it necessarily contained completely in memory while its process was in execution? The answer is no. The Descriptor Segment was paged just like any other segment; only its active pages needed to be in memory while its process was in execution.

Figure J-1 summarizes the process by which a real address was formed in the GE-645. To simplify the description, let us assume the first time through that the pages referred to in the discussion are all present in real memory. We will then go through a second time describing the actions to be taken when any of those pages is missing.

Whenever a user process was going to be given control of the processor, Multics would have created a Descriptor Segment for the process in virtual memory. The Descriptor Segment would contain a single entry for each segment the process was entitled to access. Like all segments, the

Descriptor Segment would be paged and would have a page table, which would have contained one entry for each page of the segment that might currently have been stored in real memory. If a given page were in real memory, its page table entry would provide the real memory address in which it started. If it were not in real memory, a bit in the page table entry would be set to indicate that fact.

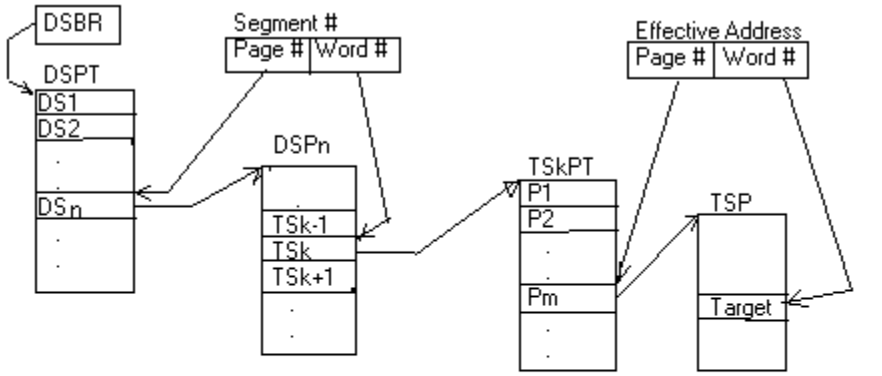

Figure J-1. 645 Address Formation

Multics would set the Descriptor Segment Base Register before giving control to the user process. That register would contain the address of the Descriptor Segment Page Table that would also have been constructed by Multics. By setting and resetting the Descriptor Segment Base Register, Multics could establish or change, with the execution of a single instruction, the pieces and parts of real memory each user process was permitted to access.

Each segment to be referenced, called here the target segment, was known within a program and within the Multics File System by a name, but within the hardware, it was known by a segment number. This segment number, ranging from 0 through 256k – 1, was the number of the target segment's entry in the Descriptor Segment. The user program would specify the segment number to be used by placing it in either the Procedure Descriptor Register or one of the eight Data Descriptor Registers. The segment number from the Procedure Descriptor Register would be used if this were an instruction reference; the number from one of the Data Descriptor Registers if this were a data reference. The segment number would be separated into two parts: the high-order eight

bits would specify which page of the Descriptor Segment Page Table contained the descriptor for the target segment, the low-order ten bits would specify which entry in that page contained the entry for the target segment.

These various references can be seen graphically in the four leftmost rectangles of Figure J-1. Note that it is important for the page size to be a power of 2 so that the processor can easily isolate the page number and word number. (If the page size were not a power of 2, it would be necessary to perform a division to determine the page number and word number—an untenable alternative.)

Once the target segment descriptor was located, it would provide the real memory address of the target segment Page Table. Now the effective address was split into a Page number and Word Number just as the segment number was split. The page number was used as the offset into the target segment Page Table, which contains the real memory address of the target segment page origin. Finally, the word number was added to the target sector page origin to give the real memory address of the target. These last steps in the address formation process are shown graphically in the right-hand three rectangles of Figure J-1.

This all works very well but would be a complete waste if all pages of all segments were located in real memory all of the time. The true value of the scheme was realized only if a fraction of the pages of each segment was in real memory at any one time. First, note that a page table was not over 256 words in length—one word for each of the 1k pages in a 256k segment. Hence, there was no need to consider paging page tables.

If a new process were being started by Multics on behalf of an authorized user, it would create a Descriptor Segment Page Table for the process and initialize a Descriptor Segment to contain certain segment descriptors that corresponded to segments that all processes had such as one or more segments for a command processor. Now if the user of this process entered a new command that needed to use one or more segments not now part of his or her process, the segment descriptors for these new segments would be added to the descriptor segment for the process. Of course, such additions would be made to the descriptor segment only if the File System determined that the user was entitled to use them.

Hence, descriptor segments could be built up as the needs of processes became apparent and justified. Now let us suppose that a user made reference to a segment for the first time so that its descriptor was not contained in the descriptor segment for the process. The File System

would add a descriptor for the new segment to the descriptor segment and would create a page table for it. However, the File System has no way to know which of the pages of the segment would be referred to by the user process, so it would set a bit in each of the page table entries indicating that the page was not present in real memory.

When the user process made a reference to a word (target) in one of the segment's pages, the processor would detect (by examining the state of the "page absent" bit in the page table) that the page was not in real memory and would execute a "missing page fault". When the fault occurred, the contents of all of the registers for the process would be saved and control would be passed to the File System. The File System would begin to retrieve the page requested by the process and would then give control of the processor to another process, which, it was hoped, would be ready to continue execution. The page that caused the missing page fault would eventually be located and placed in real memory. Its page origin would be entered in the segment's page table, the "page absent" bit would be reset and control would be returned to the process that was interrupted by the missing page fault at the point of its interruption.

While interactions such as this were occurring, the File System was busily controlling the flow of pages between real memory and secondary storage. (In the GE-645, secondary storage was a high-speed magnetic drum known as the "fire-hose drum".) The type of processing described here went on both for the management of application segments and for descriptor segments. Pages of each type and their page tables needed to be constructed, moved between primary and secondary storage and have their descriptors updated on a continuing basis. Actually there were many status bits in the descriptors that assisted in the proper management of the virtual memory mechanism. These were all maintained in a harmonious interplay between the processor and the Multics Operating System Software.

Appendix K—Utility Programs

In general, a need exists for various programs to perform small but essential tasks concerned with running programs and the management of their media. These are referred to as Utility Programs. Some of these are described in this appendix.

K1. Loaders

Stored-program computers, in their pure form, have one possibly obscure weakness—no way is provided to get programs into their memories. In a bare von Neumann computer, the program will execute beautifully (after all of its bugs have been removed) once it is located in internal memory, but how does it get there? The answer is it is loaded into internal memory by a program called a loader.

But how does the loader get there? Each machine has its own special feature that loads a small snippet of code into the memory. This small piece of code then brings in a bit larger piece of code and so forth until a complete loader is in memory and capable of loading any desired program. For obvious reasons, this process of getting the loader loaded is referred to as "bootstrapping".

The bootstrap process is generally started from the operator's console. The operator is capable of executing a single wired read instruction from some device. This wired instruction is executed and then control of the processor is given to the area into which the read has occurred. In the RAYDAC, a rotary switch on the console allowed the operator to choose one of the tape handlers from which the first read would be executed and a button that would cause the read to be executed. The read instruction would bring a block of information into the buffer memory for the selected tape handler and then control would be transferred to the first word of the buffer. The block of information read would (hopefully) contain the beginning of the bootstrap process and would permit a complete loader to be loaded that would, in turn, load the desired program.

On some machines with card readers, a start button on the console caused a card to be read into a fixed location and then control of the processor would be transferred to that location. Again, this would provide enough instructions and data to be read into memory to allow the bootstrap process to proceed. The card would contain enough instructions and data to permit the bootstrap process to load a complete

loader that would then load a program from the cards following the first "startup" card.

In early machines, most loaders could only load instructions and data contained on the input medium along with the loader and its bootstrap. If any library subroutines were to be used, these had to be incorporated into the program at compilation or assembly time. This was necessary because certain addresses in the subroutines and the calling program needed to be set so all of the parts of the resulting program could pass values to and from one another. At some point in the mid 1950s the use of relocatable loaders became common. With a relocatable loader, it was possible to defer this process of address adjustment until load time.

Use of a relocatable loader often speeds up the loading time because more of the program is read in from the high-speed medium upon which the library is stored (magnetic tape or disk) as compared to the slow medium upon which the user's program might be stored. But perhaps more importantly, the user is assured to always have the latest version of every subroutine included in his program. So if a more accurate or faster version of a subroutine were put in the library between one program execution and the next, the user would get the new version from the library instead of continuing to use the old version bound into his program.

K2. Memory Dumps

After writing a program and assembling or compiling it, it is necessary to subject it to the "acid test" of the computer. On the first several tries, it will in all likelihood crash. In other words, it will contain errors that will cause the computer to cease operation because some attempt has been made to execute a meaningless instruction. Many errors may result in such a situation: transferring control of the computer to a word containing data, dividing some number by zero, getting in a never-ending program loop and addressing memory outside the address range available to the program are but a few possibilities. When a crash occurs, you can depend upon a good computer operator to record the value of the instruction counter and perhaps the values in important computer registers at the time of the crash. But that is often not enough.

A memory dump gives a snapshot of the computer's registers and memory at the time of the crash. It permits the programmer to reconstruct the situation including the state of all data and instructions. This type of information is vital in diagnosing the source of the problem.

K3. Media Conversion

It is often necessary to transfer trays of cards to magnetic tape or vice versa or magnetic tapes containing reports to be printed to paper. All of these must be possible with various possible choices of blocking factors and possible tape formats. The programs that perform these tasks are referred to generically as media conversion programs.

K4. Sorts

It must be possible to reorder the contents of magnetic tape files of various formats in any desired sequence. Sort generators or parameterized sorts provide this function.

K5. Trace Programs

If the path a program is following is difficult to determine during debugging, a Trace Program will provide invaluable information as to the reason for the errant behavior. A Trace Program causes each instruction of a program to be executed in sequence and the results of the execution to be recorded. The trace produces a complete record of each instruction executed as follows:

- The location of the instruction
- The form of the instruction at the time of execution
- The values of operands at the time of execution
- The results of the execution
- The values of all computer registers after the execution

This wealth of information usually permits the mystery of errant paths of execution to be revealed.

Appendix L. Personal Background

I was born on April 27, 1926 in Stockton, California. I have dim recollections of people debating whether to vote for Hoover of Roosevelt in 1931. I was raised in the Great Depression; nevertheless my father was always employed. We led a humble life, but we were always adequately sheltered and clothed and better than adequately fed. My only sister and I were always well treated and avoided any major traumas.

My educational background began at Victory School in Stockton, California. After graduation from Victory School in February of 1940, I continued my education at Stockton High School—the only high school in town at the time. I graduated from Stockton High in February 1944.

In grade school I found most of the material—Reading, Writing, Social Studies, Art, English—boring. However, I enjoyed Arithmetic, Singing and Manual Training (what is now called Industrial Arts). The Bank of Italy (later renamed the Bank of America) had a savings program for students and conducted a weekly, student-operated bank at school during which students could do banking business. I was chosen to be a teller at the bank.

I joined the school orchestra at the beginning of my fourth grade year. I started out playing the bass drum and then switched to the bass horn (tuba). The orchestra conductor came to the school once a week for practice. He also had a band made up of students chosen from throughout the city that practiced every Monday night. I was chosen to be in the band and played in it until I graduated from grade school.

Also while in grade school I became a member of the League of Curtis Salesmen. That was the name given to young people who sold the Saturday Evening Post, Lady's Home Journal and the Country Gentleman—the three publications of the Curtis Publishing Company. I sold 50 to 70 Posts per week, 15 to 20 Journals and 5 or 6 Country Gentlemen per month. I made only a few cents per copy, but I learned a little about selling and conducting my own mini-business.

My Dad had introduced me to the use of some tools before I took Manual Training, but I enjoyed the approach taken at school for its comprehensiveness. We went to El Dorado School, which was farther from our house than Victory and a more up-scale institution, to take the

course during my 7th and 8th grade years. We had one semester each of woodworking, wood lathe, sheet metal and metal lathe. This provided a scope my father could not have provided and which gave me a feel for a wide range of skills.

In high school, I continued to find most courses to be boring. This lack of motivation was punctuated when I flunked English and got a D- in typing. My Typing teacher said she would give me a D- instead of an F if I promised never to take the course from her again. However, I continued to do well in mathematics through Trigonometry and played in the Band for all four years and enjoyed every minute of it.

The high school band had a little internal organization that helped the director get all of the instruments and equipment where they had to be before performances and back to the high school afterward. The organization consisted of a Sergeant, a Lieutenant and a Captain selected each year by vote of the band members. The officers had the privilege of wearing special chevrons on their uniforms that distinguished them from the other band members. In my second year in the band I was elected Sergeant; in my third and fourth years I was elected Captain. I gained some organizational and leadership experience from these involvements.

From the time I was quite young, I can remember going to my dad's office and being fascinated by the calculator he always had on his desk. I believe it was a Monroe. It occupied a space about 16 inches by 16 inches on the desk and had a ten by ten array of buttons to enter values (the digits zero through 9 for each digit of a ten digit number). It also had a carriage across the top much like a typewriter carriage except it contained displays of numbers. One display was of the accumulator—it displayed the answer if you were adding numbers or the product if you were multiplying. Another was a display of the buttons depressed on the keyboard and still another was a counter that told how many operations had been performed since the machine was last cleared. You could use the counter, to see if you had entered the right number of addends if you were adding a column of numbers, for example. The counter was also needed in multiplication and division.

The carriage also had a little handle on the right side you cranked to perform an operation—addition, subtraction, multiplication or division. For addition, you cranked the crank clockwise; for subtraction, you cranked it counterclockwise. (Multiplication and division were more complicated.) A way was provided to clear all of the displays, but I don't remember what it was—probably another crank. So to add a column of numbers,

you would first clear the machine. This made all of the displays zero. Then you would enter the first addend and turn the crank clockwise once, and then you would enter the next addend and turn the crank clockwise once, and so forth until all of the numbers had been entered. The sum would appear in the accumulator at each step.

The machine had one other control that permitted it to multiply and divide. This was a little knob at the front below the keyboard. When you turned the knob clockwise, the carriage would shift one decimal place to the right; when you turned it counterclockwise, the carriage would shift one decimal place to the left. The significance of this shifting was that it caused the addend, stored in the keyboard to be multiplied or divided by 10 before it was added to or subtracted from the accumulator. It also caused the accumulator to be incremented/decremented by ten or one-tenth instead of one when a one was entered in the keyboard. Hence, if you wanted to multiply 45 by 21, you would enter 45 in the keyboard and crank the crank on the carriage once clockwise. This would add 45 to the accumulator and place 1 in the counter. You would then turn the knob in front one notch clockwise. That would cause the next operation to affect the accumulator one decimal place to the left of where it previously had, or, in other words, it would cause 450 to be added for each clockwise turn of the crank. You would then crank the crank on the carriage twice clockwise. That would add 900 and produce the product, 945, in the accumulator and would increment the counter by 20 so that it showed the multiplier, 21. You could perform a similar process in reverse to accomplish division. My Dad always let me fool around with the machine and I found it fascinating.

As a boy, I had the usual array of career goals including fireman, pilot, streetcar driver (we still had street cars) and others. However, at some point, my Dad suggested I might want to look into Chemical Engineering. Being an engineer himself, it was natural for him to have a bias toward engineering occupations and he was very fascinated and impressed with the advances being made at the time in Chemistry. I heeded his advice and kept Chemical Engineering as my career goal for many years.

In my junior year at high school, I took Chemistry. I absolutely loved it. It was exciting and fascinating to me. Better still, my enjoyment of Chemistry spilled over to my other subjects and I started doing better in all areas of study. I was even selected to become a member of the club whose membership was reserved for the best students (I don't remember the name of it).

If this was a positive turn of events, then my senior year was even better. That was the year I took Physics. I had not previously known Physics existed or what it was about. When I was exposed, I was hooked. If it was possible to find anything I loved more than Chemistry, then it was Physics. I was blessed to have taken my first semester from Mr. Corbett, who was an old gentleman who had taught Physics for years and loved it and passed on his love of the subject to his students. Every day his students looked forward to his class because he made it entertaining and participatory. He presented the "nuts and bolts" of the universe before your very eyes and made it all seem like fun. Of course, I got an A in the course.

Unfortunately, he retired at the end of the first semester. An okay teacher replaced him in the second semester and I got another A, but I will always be grateful I had the old man for my introduction. I was also hired as the laboratory assistant for the second semester so I had an opportunity to get more fully involved in the course and its presentation than I had before. The result of all this was that I changed my career objective from Chemical Engineering to Physics.

World War II was raging during my years in high school. Before I graduated from high school, I enlisted in the Army Specialized Training Program. As a member of this program, I knew I would be sent to some university in April of 1944. It turned out to be the University of Idaho, at Moscow, Idaho. After I graduated from high school in February 1944, to fill the time until April, I audited a course in Music Appreciation at Stockton High School. Also, at about this time, I was invited to play the tuba for Stockton Symphony Orchestra and continued to do so until I went away to the service.

Attendance at the University of Idaho took place as scheduled and I enjoyed the army life on the campus. I also did very well in all of my courses, especially math and chemistry. I got high grades. I don't know if they were all A's but many of them were. The only problem was the program was cancelled during our first quarter and that was the end of it. We were all sent to the Presidio of Monterey to be inducted into the regular army after completion of the quarter.

While at Monterey, I had a chance to try out for the Army Band. I marched with them one afternoon and played one of their sousaphones, but I guess I didn't measure up because nothing ever came of it. I was assigned instead to the Army Air Corps. From Monterey I went to Basic Training at Buckley Field outside of Denver and from there to Cryptographic Technician's School at Chanute Field, Rantoul, Illinois. I

ended up with an MOS (Military Occupation Specialty) of 805, Cryptographic Technician. I spent the rest of my tour of duty as a cryptographer, first in Greenland and then in the Azores.

My experience as a cryptographer served me well. In addition to being able to apply my cryptographic skills, I finally learned to type, and the communication equipment of the day became very familiar to me. Everywhere I was stationed, they had CW (constant wave, Morse code) radio and radio-teletype communication. These were the state-of-the art at the time. I became quite familiar with Teletype operation and the use of paper tape as a storage medium. I also got used to relying on machines to assist me in doing my job.

After military service, I attended Stockton Junior College from September 1946 to June 1948. I had been told I could get the same courses there as I could get at a university; it would be less expensive, and I could stay at home. It was true I could get courses with the same names, but I was to discover later the quality of the course content was not the same. However, I was more than adequately compensated by having found and married my wife during my sophomore year at Stockton JC.

While I was at Stockton JC, I had three friends with whom I was very close: Ralph Cowen, Peter McCurdy and Stanley Bacon. Ralph had been my friend through most of my high school years, Peter played snare drum in the high school band and had also been in our group at the University of Idaho (and also introduced me to my wife to be), and so we continued our friendship. I had also known Stanley from high school. He played in the high school orchestra and owned a wide array of electronic equipment he used to play recorded music at school dances and other activities. We were all taking similar courses and sometimes studied together. The big tough course was calculus and it was one of my strongest subjects. They all came to me for help and I, of course, got A's in the subject.

We first heard about computers during this period. Once or twice during the Stockton JC days, newspaper articles appeared about computers. We all read them and were dazzled. Machines capable of performing thousands of calculations per minute were described. They used electrical switches called relays or even in some cases vacuum tubes to get their work done. We tried to imagine how these spectacular machines might be built.

In September of 1948, my wife of one year and I went to Berkeley so I could attend the University of California. My junior year was difficult for

me because I was faced suddenly and unexpectedly with the inadequacies of the course materials to which I had been exposed at the junior college compared to those the Cal students had been provided. I loved the courses I was taking, but I had to work very hard to keep up. For the first time in my life, I took a mathematics course I was unable to cope with. It was Vector Analysis and I flunked it. I took it from an old man with whom I did not at all communicate. This was a severe blow to my ego but I retook the course and got an A.

During the summers of my freshman and sophomore years, I worked as gopher/delivery boy for Atwood Printing Co. During my junior and senior summers, I worked as a Rod and Chainman on the survey crew for the City of Stockton.

In any case, in June of 1950, I received a baccalaureate from the College of Letters and Science at the University of California, Berkeley, with a major in Physics. I had a good solid B average; I don't remember the numerical grade point average. Most of my classmates were going on to graduate school somewhere, but nobody had bothered to tell me a bachelor's degree in Physics was good for nothing. So when it came time to get a job, very few potential employers responded. However, I received an offer from the U. S. Navy to work at the Naval Air Missile Test Center at Point Mugu, California as a GS-4 Civil Servant. I don't remember the name of the position, but I do remember it paid $2475 per annum and I was glad to get it.

Appendix M. Evolution of Computer Technology

During my thirty-two years in the Computer Industry, I had the opportunity to observe a passing parade of technology. The changes were truly remarkable in many cases and worth reviewing. They are described below in figures that consist of parallel time lines and also in text.

Figure M1 shows the evolution of storage media present on systems I worked on. The lines are to represent the passage of time with the earliest times being at the top. Delay lines were not a very successful medium on the RAYDAC, but they were about all that was available in the early days and were also used on Univac I. That machine was around and in use for quite some time, so I have showed the life of delay lines to have continued until approximately the end of Williams Tube use. Williams Tubes were relatively fast but not very reliable, so they didn't last very long. Magnetic cores were quite good and improved over time. They would have lasted even longer had they not been superseded by semiconductor memories. Although I never worked on a machine during my career that had semiconductor memory, I have since retirement and they are certainly far superior to any of their predecessors.

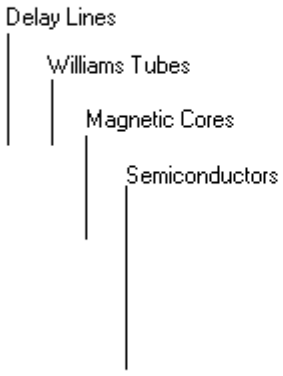

Figure M1. Storage Media Versus Time

Figure M2 shows the evolution of peripheral storage devices. The earliest was probably paper tape, though punched cards also came along very early. I don't think paper tape survived for long after the availability of

punched cards and associated equipment. Magnetic tape came along quite early. It was available on Univac I, RAYDAC and SWAC. It has also been a medium with great longevity. Over the years, it has improved in every dimension: recording density, recording format and tape speed (especially in rewind). It has also come out in various forms—not only in reels but also in cassettes and on cards.

Magnetic drums also came out fairly early and they were useful and reliable media in general, but were replaced by magnetic disks for many purposes about half way through its long life. The big move to magnetic disk came with the IBM-305 RAMAC. From then on, disks have gotten smaller but their capacity and reliability have increased remarkably. The recording densities now possible are truly amazing and they seem to continue to improve.

A big step forward in the development of disk technology was the introduction of removable disks. These first took the form of "disk packs"—each of them looked like a covered cake dish—that could be installed or de-installed from a disk handler just as a magnetic tape could from a magnetic tape handler. As time passed, this idea was revisited with the use of floppy disks on personal computers.

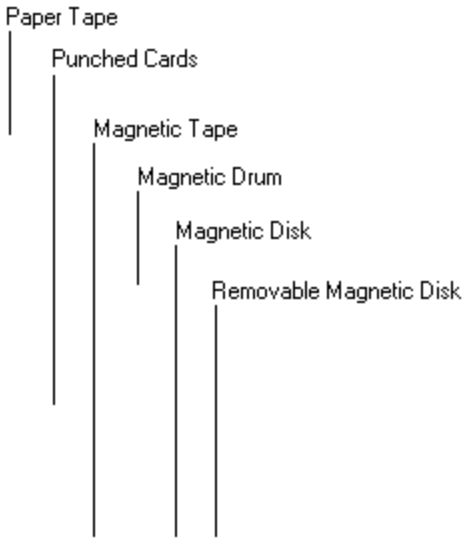

Figure M2. Peripheral Storage

Figure M3 shows the evolution of bi-stable devices over the years. These are the circuit elements that permit computers to perform arithmetic and logic. It started out with relays being the only available bi-stable devices.

After a short time, these were replaced with vacuum tubes. On the RAYDAC, we had about 18,000 special electron tubes that had gold filaments for improved reliability.

Vacuum tubes were the best thing available at the time, but their reliability was not outstanding and they used a very large amount of energy. This produced much heat that had to be disposed of using a very large amount of air conditioning (more energy gone). Then transistors came along, which were a big help, but not the final answer. The final answer, so far, has been semi conductors and they are still with us.

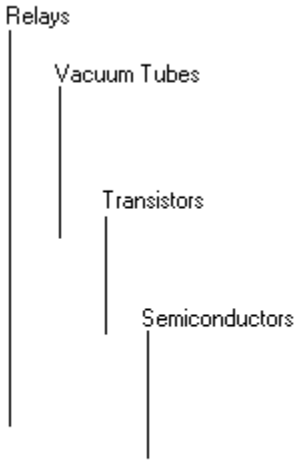

Figure M3. Bi-Stable Devices Versus Time

Figure M4 shows the evolution of certain interior décor features. These relate mostly to modes of addressing. The earliest and simplest was fixed word length. This simply means that in the first computers the engineers decided how many bits would be stored into or retrieved from the internal memory at one time. In the case of the RAYDAC, this was 30 bits, for Univac I it was 40 bits (I think) and for many later machines, like the IBM 700-series and GE-600 series, 36 bits. This was convenient for the engineers, but not particularly good for the programmers or the customers. To more nearly match the needs of the users, variable word length machines were developed. These included the IBM-702, 705 and 1401. The programmer convenience was improved as expected, but performance suffered.

The introduction of interrupts was a major improvement in computer capability. Previously, the computer would just execute one instruction after another in the sequence the programmer had indicated. If an error occurred, the users had to test for it using a special instruction and then go to a special program provided to handle that particular error. Often programmers didn't test for the occurrence of all the possible errors and hence, the computers sometimes produced incorrect results without anyone's knowledge. With the introduction of interrupts, a special class called "faults" came along that caused the processor to execute its next instruction from a fixed location if an error of a certain type occurred. This location was different for each type of error and when the fault occurred and the transfer resulted, the contents of all of the processors registers were also stored in a fixed location, as a function of the type of error that occurred. In this way, it was possible to be sure all errors were responded to in some way. In some instances, the fault could be handled by a standard program supplied by the system; in others, the user programmer would specify a location to which control would be transferred so the user program could handle the fault itself. In other cases, the program would simply be aborted with a record of the cause of the abort.

However, interrupts were caused not only because of errors, but also in connection with executing input/output instructions. Any one of many interrupts may occur depending upon the I/O device being used. Some examples are:

- A previously started I/O operation has completed:
    1. Correctly
    2. With an error condition (which error would be returned as a status condition).
- An I/O operation sent to the IOC to be started, but cannot be started because:
    1. It was a card read and the card hopper is empty.
    2. It was a write to printer and the printer is out of paper.
    3. It was a punched card and the card hopper is full.
    4. It was to any device and the power was off.

Segmentation and paging in one form or another were introduced first with the crude, fixed fences as in the GE-625/635 that provided no paging and then with the elegant virtual memory concepts of the GE-645. Many variants of this scheme appeared after they were introduced in the Ferranti-Atlas, Burroughs- B5000 and GE-645.

Finally, and this occurred after my retirement, byte accessing within word accessing became common. This combines the best of both the

program efficiency of variable word length and the memory access efficiency of fixed word length and is utilized in most modern machines.

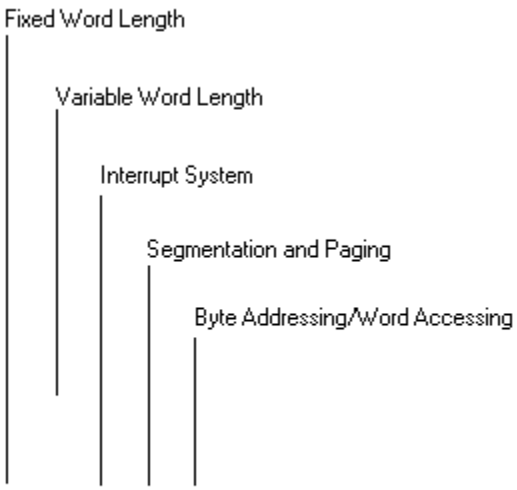

Figure M4. Evolution of Décor Features

Figure M5 shows the various choices of parallel printer available over time. A parallel printer is one that displays an entire line at a time as compared to a serial printer like a typewriter that displays a single character at a time. The speed advantage of the parallel approach is obvious. In the very early days, printers were called tabulators and they printed by raising a whole row of sticks arrayed side-by-side, each to the correct height, and then pressing the entire array against the paper. The sticks each had an upper-case alphabet and the decimal digits on it and the letter or digit to be printed on each line by each stick was determined by the height to which the stick was raised. These were used mainly by punched card equipment. The number of columns they would print was limited and the speed was less than 100 lines per minute.

The next generation of printers replaced the sticks with wheels, and the letter or number to be printed was determined by the angle to which each wheel was rotated before the entire assembly was pressed against the paper. These printers could print 120 columns at up to 150 lines per minute and were quite reliable. However, they could only print the upper case alphabet, the decimal digits and some special characters.

In an effort to obtain higher print speeds, wire printers were offered like modern dot matrix printers where the dots were printed by wires striking

the paper selectively depending upon the character to be printed in each column. These printed 120 columns and in the same character set as the wheel printers, but much faster. IBM offered one wire printer with a speed of 500 lines per minute and another with a speed of 1000 lines per minute. However, their great speed was a mixed blessing because they had very low reliability.

I found the next generation of printers very appealing though I never worked anywhere that had one. They were the electrostatic printers. They worked by projecting an optical image of a line to be printed onto a selenium cylinder. The projected optical image was converted to an identical electrostatic image in the process. The cylinder—in continuous motion—then passed electrically sensitive ink where the electrostatic image became covered with ink. Finally the inked image was pressed against paper to form the final hard copy. This is exactly the way a Xerox machine works except the source of the image in a Xerox machine is a piece of paper instead of a projector which is getting its signals from a computer or other electronic controller. Several of these were offered in the late 1950s and they printed up to 5000 lines per minute. Their shortcoming was they would not print multipart forms and many computer shops were wedded to such forms.

Finally, the laser printers and Ink Jet printers came along. These are the high quality printers available today for personal computers at very reasonable costs. They have all come out since I retired. I have shown this on the figure by showing their time line as a bold line.

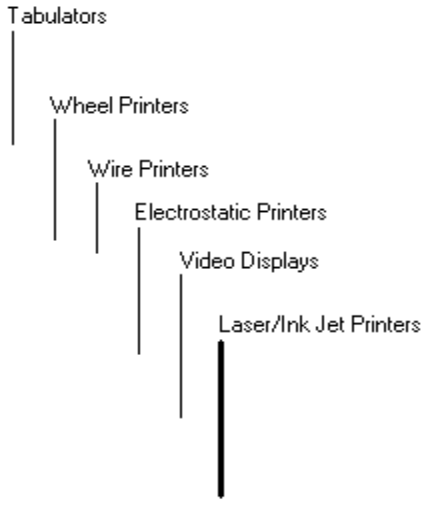

Figure M5. Parallel Print/Display Devices

Figure M6 shows a general evolution of programming languages. When I started, we had nothing but octal or hexadecimal in which to write our code. This was arduous and error prone and soon was embellished by the introduction of crude assemblers like the one I developed at Point Mugu permitting programs to be assembled from previously written and tested subroutines. (These capabilities were later incorporated into relocatable loaders.) These subroutines were generally written with relative addresses that started each subroutine at address 0. These addresses were then relocated as necessary by the assembler.

The first step toward what might be legitimately called a computer language was the introduction of symbolic addressing. In this type of language, a programmer could name the variables of his problem using names that had meaning in the application. For instance, he could call First Name FRSTNM, and year-to-date pay YTDPAY, and so forth. The symbols were in upper case and generally limited in length to 6 or 8 characters. In some cases, as in SCRIPT, it was possible to use alphabetical symbols for program references, but data references were in a kind of decimal notation like A1.22.02 or 14.17.03. In general, the leading characters before the first decimal, referred to a block of storage, the second two digits referred to a single entry in the block and the last two digits were for filling in things you had forgotten on the first numbering. The better symbolic assemblers made obsolete the use of the type of relative assembler I mentioned above by providing the ability to incorporate subroutines from a subroutine library. The programmer needed merely to supply the assembler with the names of the library subroutines to be incorporated along with the main program's code.

With the advent of FORTRAN and BASIC, it was possible to write mathematical expressions. That ability has been included in many, if not most, languages after these initial ones.

Starting with Grace Hopper's work with the A0 compiler at Remington Rand a whole sequence of English-like programming languages developed leading eventually to COBOL. These enjoyed a rich and useful life and are, I assume, still in use in some places.

Finally, generalized programs such as 9PAC and the others from Hanford were developed. This approach has been greatly exploited in personal computers. For example, word processors are generalized routines where the problem solved is text recording, editing and formatting. Excel is a generalized system for creating, editing and displaying spreadsheets.

Access is a generalized system for creating, updating and displaying databases.

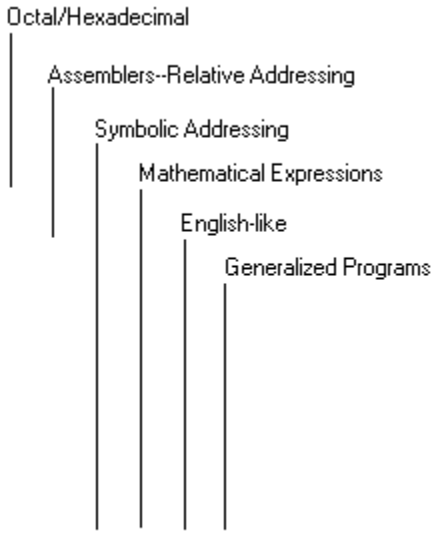

Figure M6. Programming Languages

Figure M7 shows the evolution of input/output methods. With the first computers the only methods of addressing devices and their contents were those supplied by the hardware. So if one wanted to address tape handler 3 or printer 2, one was required to specify tape handler 3 or printer 2.

Blocking and buffering were probably the first enhancements available that affected the content of files. They have been explained in Appendix I and subroutines were provided to support them from early in the evolution of I/O technology.

Mass storage devices came next. Like other devices of their time, it was necessary to address them and their contents by using the addressing schemes provided by the hardware. This worked, but produced code that was difficult to read and was often inflexible if it was necessary to make changes (as it always was).

Operating Systems (or perhaps some of the more elegant job monitors) permitted the introduction of symbolic device designations. In this case, a program could refer to Tape 1 and Tape 2 and when execution occurred have these files mounted, for example, on tape handler 3 and tape handler 7. Better yet, it would be possible to refer to them functionally as

TIMECARDS and PAYCHECKS. In that event, TIMECARDS and PAYCHECKS files could be assigned to any pair of tape handlers (such as 3 and 7) available at the time of the program's execution. These assignments would be made by the Operating System and would be made based upon its knowledge of the status and utilization of the system devices. The fact that the files were magnetic tape files would be announced on control cards accompanying the program.

A natural extension of symbolic device designations was the notion of device independence. One might want to assign a print file to magnetic tape on one execution, for example, and directly to the printer on the next. With device independence, merely changing a control card between executions would make such changes—the program would be the same in each case and would be unaware of the change.

File Systems were a natural companion of timesharing systems. A user at a terminal wanted to be able to get to his or her data quickly and easily without being concerned with devices or other system details. The services of a File System permitted users to refer to their files by name and to catalog them for current and future use.

Methods of structuring the contents of ones mass storage files appeared in about the same time frame as File Systems. Chief among these to my knowledge were Index Sequential and IDS (Integrated Data Store, developed by Charlie Bachman of General Electric).

Note that any or all of these methods carried forward into the future. They each continued to have an application at some level of the system.

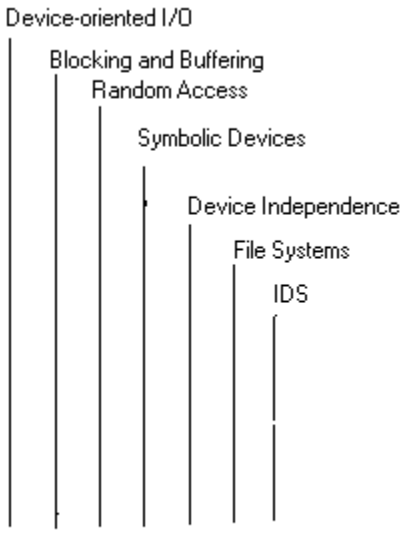

Figure M7. I/O, File Systems and Database Managers

Figure M8 gives a brief depiction of the evolution of Operating Systems. Before Operating Systems existed, human operators performed all of the functions that controlled the system. Humans still take charge in case the automatic controls fail to function. Job Monitors conducted the first attempts at automatic control. However they were short lived because Multi-Programming and Multi-Tasking Operating Systems rapidly superseded them.

Multi-Programming and Multi-Tasking Operating Systems could not take place until hardware systems with some sort of segmentation and an interrupt system were available. With this event, it was possible for the Operating System to take control of the scheduling, allocation and de-allocation of resources to different jobs and/or users. This is what occurred with GECOS and MULTICS. With MULTICS and other timesharing systems the entire panoply of system resources was brought to the user from his/her terminal in a way similar to what the modern personal computer user enjoys today.

Of course, all of this led to the creation of the Internet and to Intranets. Although I have made good use of the Internet, I do not pretend I had anything to do with its creation, hence, its time-line is shown with a bold line in Figure M8.

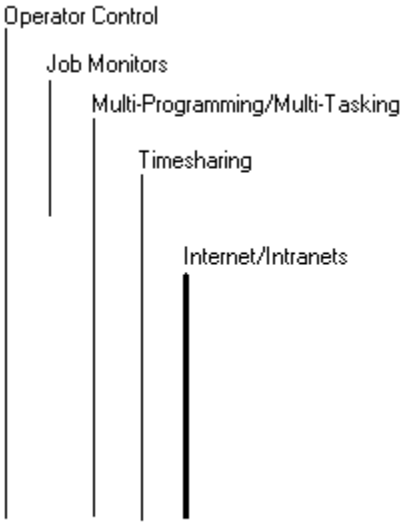

Figure M8. Monitors and Operating Systems

GLOSSARY

Access time	The time it takes to store or retrieve a word to/from memory. In a disk storage unit, it is the sum of the latency time, the seek time and the transfer time. See Appendix C.
AEC	Atomic Energy Commission. The U. S. Government agency that oversaw Nuclear Energy activities prior to the U. S. Department of Energy.
Application Program	A program written by or purchased for use by a user permitting him or her to solve the problem or perform the function, which he wishes (that is, his or her application) on a computer.
ALU	Arithmetic And Logic Unit. See Appendix A.
ARPA	Advanced Research Project Agency of the U. S. Department of Defense.
Assembler	See Assembly Program
Assembly Program	A program used for building other programs, which uses symbols to represent the operations and operands of the program to be created instead of the language of the machine that will execute it. See Appendix E.
Atomic Energy Commision	The predecessor of the U. S. Department of Energy that had responsibility to control all developments and materials having to do with Atomic Energy.
Backing Store	A form of memory that is usually larger but slower than the internal memory. Backing stores are usually used to swap blocks of storage out of and into the internal memory to give the illusion that it is larger than it actually is, as in Virtual Memory.
Batch Processing	A mode of computer use in which the user or a computer operator representing him or her is in the computer room mounting and dismounting magnetic tapes, loading and unloading card hoppers, and manipulating printers in order to get the user's computer program run. This is in contrast to the timesharing mode of processing, which see.
BCD	Binary coded decimal.
Binary	Having to do with base two numbers. See Appendix B.
Binary Coded Decimal	A code representing each decimal digit as a group of four bits: 0 = 0000, 1 = 0001, 2 = 0010, 3 = 0011, 4 =

	0100, 5 = 0101, 6 = 0110, 7 = 0111, 8 = 1000, 9 = 1001.
Bit	Binary digit. A single digit in the base-two number system having the value zero or one.
Byte	An eight-bit storage group, which is now the international standard. Bytes can be used to represent all the decimal digits; upper and lower-case letters; many special characters as well as foreign alphabets.
CPU	Central Processing Unit. This is the ALU and CCU combined.
CTSS	Compatible Time Sharing System. See Note [G].
Differential Equation	A differential equation is an equation connecting certain independent variables, certain functions (dependent variables) of these variables, and certain derivatives of these functions with respect to the independent variables. Two types of differential equations exist: ordinary and partial. The following are examples of these: $\frac{dy}{dx} = y - x$ $\frac{\partial z}{\partial x} + \frac{\partial z}{\partial y} = yz$ The first of these is an ordinary differential equation. The variable x is the independent variable and y, the dependent variable, is a function of x. The quantity on the left of the equal sign is the derivative of y with respect to x. In other words, it tells how much y changes for a small change in x. The second of these is a partial differential equation. The dependent variable changes as a function of two independent variables, x and y. The quantities on the left of the equal sign are the partial derivatives of z with respect to x and y, respectively. [See Simultaneous Differential Equations.]
DOD	United States Department of Defense.
Drive	This word is used interchangeably with "handler" in the case of magnetic tape and disk devices. Hence, a magnetic tape drive and a magnetic tape handler are the same thing.
File System	A set of capabilities permitting a user to refer to his or

	her files by name, and perform a host of other functions concerning the storage and maintenance of files. Some of these functions are: File security, permitting only authorized users to access each file; File traffic control, protecting various users from interfering with one another while updating files; File Recovery, restoring files to previously states, known to be correct, in case of hardware or software failure; Maintaining Catalogs of User Files; Storing and retrieving user files upon request of authorized users.
Fixed Point	A form of representation in which every number has the same number of decimal places—that is, the decimal point location in each number is fixed. See Appendix B.
Floating Point	A form of representation in which the number is represented in the form of a mantissa and an exponent. Hence, the number N, might be represented in the form $N = M * 10^E$, where M is the mantissa and E is the exponent, and $1 > M >= 0.1$. See Appendix B.
FSO	Honeywell Federal Systems Operation.
Hexadecimal	Having to do with base sixteen numbers. See Appendix B.
IAO	Internal Automation Operation. A department within General Electric.
INA	Acronym for the Institute for Numerical Analysis of the National Bureau of Standards located on the Campus of UCLA.
Latency time	The time, on a rotating storage device, it takes for the device to rotate from its angular position when a command is issued until it reaches the addressed record. See Appendix C.
Loader	A program that is used to move other programs from some input medium, such as paper tape, cards or magnetic tape to the internal memory of a computer prior to execution of the loaded program.
Minimum Latency Programming	A programming technique used on properly equipped serial memory computers to minimize the storage latency time. See Appendix D.
Multiprogramming	A mode of utilizing the computer resources in which more than one program is loaded into the internal memory and take turns using the processor. This is in contrast to uniprogramming, which see.

Octal	Having to do with base eight numbers. See Appendix B.
Operand	One of the quantities in an arithmetic expression. For example, in the expression A + B = C, A and B are operands. Individually they are known as "augend" and "addend", respectively; but collectively they are "operands". Also, any values used as inputs to operations performed by a computer.
Page	In the context of this book, a page is a 1024-word subdivision of a virtual memory segment or file. This is the meaning in the GE-645 and may have other more general meaning in a broader context. In this narrow context, a page is a 1024-word unit of virtual memory swapped into and out of memory as needed for processing.
Paging	A method of memory addressing in which the total memory address space is partitioned into groups (pages) of a fixed size (such as 1024 words) and the origin of addresses in each page may be set to any value between zero and T/1024-1, where T is the total, in words, of the available real memory. The origin of each page is specified in a page table maintained by the operating system and the CPU performs the translation between the address in an instruction and the effective address used for memory access by appending the page origin from the page table to the intra-page address from the instruction.
Parity	A simple method of providing a check for data integrity. Parity may be either odd or even. If the parity is even, then the parity of a correct string of bits = 1 if the number of bits equal to 1 in the string is even. If the parity is odd, then the parity of a string of bits = 1 if the number of bits equal to 1 in the string is odd.
Parity Checking	When parity checking is performed, the parity of the bit string is appended to the string whenever the string enters the computer system. The parity is then recomputed whenever the string is used or moved to see if any change has occurred since the string entered the machine. On magnetic tapes, each frame of data on the tape (on old-fashioned tape handlers) consisted of six information bits and one parity bit. In this way each frame could be checked for change whenever it was read. Odd parity is

	generally preferred because it distinguishes a zero from a null transmission that might occur, for example, because of a broken wire.
Pipeline	This is high-speed storage within the CPU of a very fast computer to fetch instructions ahead of their use. This process reduces the number of times when the processor is stopped while waiting to gain access to the next instruction it is to execute. The pipeline is made to contain the next L instructions that might be executed on each of the possible execution paths, where L is a look-ahead amount in number of instructions.
PPU	Problem Preparation Unit of the RAYDAC. This piece of peripheral equipment permitted data and programs manually entered via a Teletype Unit to be recorded on magnetic tape for reading by RAYDAC magnetic tape handlers.
Processor	See ALU.
Pure Procedure	A pure procedure is a set of computer instructions intended to be run without being modified. Many users may share a single pure procedure if a means is provided for each user to have his or her own data upon which the pure procedure may operate. It is also possible for pure procedures to call themselves, which in some instances is advantageous.
Relocatable Loader	A loader that is capable of adjusting the addresses of the programs it is loading while performing the normal loading process.
Seek time	The time it takes to position the read/write head on a disk storage unit in a storage or retrieval operation. See Appendix C.
Segment	An amount of storage overlaid with an address space that is unique to this storage and is not accessible via the addresses of any other address space. In this book, segments will usually be GE-645 segments that have a content less than or equal 256K 36-bit words.
Segmentation	The process by which a computer is capable of providing addressing within segments.
Simultaneous Differential Equations	Two or more differential equations in which two or more dependent variables and the differential equations are concurrently satisfied. For example, suppose two functions, x and y, exist, both of which are functions of the independent variable t and the

	following two differential equations also exist: $\frac{dx}{dt} = 2y + x, \qquad \frac{dy}{dt} = 3y + 4x$ These two equations can be solved for x and y as a function of t. [See Differential Equations.]
System Program	A computer program (usually supplied by the computer vendor) provided to support users' application programs. System programs are often invisible to the user and supply functions in more user-friendly forms than those supplied by the computer hardware. For example, I may address the file containing this book as "D:\Career\Book.doc" instead of "Record 3 of Track 129 of Disk 1 on I/O Channel 5". Various system programs provide this and many other capabilities.
Timesharing	A computer system in which the users gain access to the machine by way of remote terminals. This is in contrast to a "batch system" in which the user (or a computer operator representing him or her) is in the room with the computer and manually mounts and dismounts tapes and loads and unloads card hoppers and tends printers to cause a computer job to be completed.
Uniprogramming	Running one program at a time. Completely finishing one program before beginning the next, in contrast to multiprogramming where several programs cohabitate the computer and take turns using the processor.
Virtual Memory	An array of storage that can be operated upon by normal computer instructions, but is addressed by an address space much larger than the real address space. The storage that does not fit in the real memory is retained in a secondary memory such as disk or drum. The virtual memory may also have other capabilities that in conjunction with File System software provide for protection of user files and management of the real internal memory.
VMM	Virtual Machine Monitor. See VMM chapter above.
Word	The amount of information in bits retrieved from or stored in internal memory in a single access.
Word length	The number of bits retrieved or stored in a single memory access.

NOTES

[A]

As we seriously got into the business of RAYDAC programming, in 1952, other people were busily trying to advance the art of programming at the same time. [58] Most notably Grace Hopper was making great progress. She had started out as a programmer on the Harvard Mark I in 1944, and by now was employed by Remington Rand and working on the UNIVAC. In this year, she wrote a paper entitled *The Education of a Computer* in which she described the idea of a compiler. This paper took up the idea of reusable programs and paved the way for the work in high-level languages she would pursue for many years to come.

By the end of 1952, the name UNIVAC was becoming synonymous with "computer" as Frigidaire was with refrigerators and Kleenex was with facial tissues. Following up on this popularity, CBS used the UNIVAC at the Philadelphia factory to forecast the winner in the 1952 presidential election. The UNIVAC predicted Eisenhower would win when only 5% of the vote had been counted. However, the network failed to believe the results until after midnight Eastern Time. Elections have never been the same.

In the same year, John von Neumann completed the IAS computer at the Institute for Advanced Study at Princeton University.

[B]

IBM introduced the 650 in 1954. Its internal memory was magnetic drum and it performed its arithmetic and addressing in decimal. It was intended to fill the capability gap existing below the 700-series machines. Smaller users that couldn't afford a 701 or a 704, could upgrade from punched card equipment by utilizing a 650.

[C]

Two users groups had been formed in 1955—the first of their kind to exist. One was SHARE for IBM-701 and 704 (and later 709) users; the other was USE for UNIVAC users.

[D]

As 9PAC was moving along and I was doing a lot of traveling, computer technology was moving ahead at a rapid pace. The year 1958 was a signal year in that regard [59]. Control Data Corporation, recently founded under the direction of William Norris,

entered the market with the fully transistorized CDC 1604. Seymour Cray was the chief architect of the machine. Jack St. Clair Kilby conceived and proved his idea of integrating a transistor with resistors and capacitors on a single semiconductor chip, which is a monolithic IC. The SAGE system for Air Defense was put into service at McGuire Air Force Base in New Jersey and the first effective air traffic control system was made operational for the north-eastern U. S. John McCarthy developed concepts of the LISP programming language for manipulating strings of symbols, a non-numeric processing language.

[E]

IBM introduced two small computers in 1959[60]: the 1401 and the 1620. They were both character-oriented machines. The 1401 was targeted for use by small businesses, colleges and universities and had memory sizes ranging from 4 to 16 k characters (where k = 1024) and could be equipped with a full array of peripheral devices (except magnetic disk memory units?). The 1620 was targeted for use by small scientific users. It had memory sizes ranging from 20 to 60 k characters. It was like a timesharing terminal without much software support. It had no logical place to fit at Hanford, but Fred Gruenberger loved it and thought he would like to have one of his own.

Also in 1959, General Electric delivered 32 ERMA (Electronic Recording Machines—Accounting) systems to the Bank of America in California. The machines employed Magnetic Ink Character Recognition (MICR) as a means of capturing data from checks. It was the leading edge of the wave of technology that would lead to ATM machines and personal electronic banking.

[F]

Since 1952[61], Grace Hopper, at Remington Rand and later Sperry Rand, had been developing a series of programming languages. These culminated in the development of FLOWMATIC in 1960. The languages that were developed made increasing use of natural-language-like phrases for use in business data processing. In the same year, IBM responded by producing COMMERCIAL TRANSLATOR with the same objective.

COBOL (Common Business Oriented Language) was defined as the first standardized business computer programming language. The standard was created by CODASYL (Conference on Data System Languages) under the leadership of Joe Wegstein of the US Bureau

of Standards. In a more mathematical vein, a committee also developed ALGOL 60. Although ALGOL was not widely implemented, it became the conceptual basis of many programming languages thereafter.

[G]

In 1961 MIT put in operation the CTSS [62] (Compatible Timesharing System). This was the first timesharing system. It was implemented on a modified 7090 with bounds registers to isolate user programs from one another and a timer for limiting the execution duration of separate users. In the same year, the first integrated circuits became available commercially from Fairchild Corporation.

[H]

While our opportunity to develop a Y computer disappeared in 1962, the Ferranti Atlas became operational. It contained such advanced features as virtual memory, paging and pipelined instruction retrieval—all features to be included in many computers of the future.

[I]

A great achievement was accomplished in 1963—the definition of ASCII, the American Standard Code for Information Interchange. It took some time after its definition for the code to come into common use, but this was the necessary first step.

[J]

In 1964, CTSS was still the only operational time-sharing system. However, the Dartmouth Time Sharing System [63] was nearing operational status. This system was implemented on a GE-235 and featured BASIC as the principal programming language. The system was used for student program development. This was also the year in which Douglas Englehart invented the "mouse" and IBM announced the System/360, the first IBM family of compatible machines.

[K]

In 1965, Digital Equipment Corporation introduced the PDP-8. It was the first true minicomputer.

[L]

In 1963, the B5000 was introduced. It had a form of addressing called segmentation. In this case, multiple address spaces were available each of which was called a segment. This capability was

used to great advantage in the creation of compilers by Burroughs. When I arrived in Phoenix, everyone was talking about the recently announced Burroughs B-5000. It included a whole bundle of innovations in addition to segmentation: multiprogramming and multiprocessing, use of stack segments for process communication, use of high-level language in implementation of system programs and use of Polish notation in the parsing of language statements. The uninitiated reader may be troubled by the inscrutability of these attributes, but suffice it to say this machine involved a GIANT LEAP in both hardware and programming technology and we needed to be able to compete with this class of machine.

[M]

In 1966, a joint project between IBM and SHARE defined a new programming language intended to be useful for both scientific and business data processing even as the System/360 machines were intended to be. The language was called PL-1 and was also intended to be a high-level system development language.

[N]

The first third generation computers were delivered in 1967. These used integrated circuits as their primary electronic elements.

[O]

In 1968, Edsger Djikstra's paper "GO TO Statement Considered Harmful" (Communications of ACM, August 1968) was published and was a key event in the striving for more effective methods of software development.

[P]

During 1969 ARPAnet was begun. This was a network of computers that was a harbinger of the Internet. Ritchie and Thompson at Bell Labs started their operating system--a "watered down" version of Multics. They called it Unix.

[Q]

In 1970 and 1971 two important technological developments came along. Intel Corporation produced the first computer on a chip—a microprocessor called the Intel 4004. In the same period, Alan Shugart of IBM used an 8-inch floppy disk in the DISPLAYWRITER product.

[R]

The first digital microcomputer for personal use was made available in 1972. It was the MITS (Micro Instrumentation and Telemetry Systems) 816. In 1973, Robert Metcalfe created Ethernet, the basis for a "local area net" at Xerox PARC.

[S]

In 1974, the personal computer trend really started to gain momentum. Scelbi Computer Consulting of Milford, Connecticut offered a machine as did Jonathan Titus—the Mark 8. Gary Kildall introduced the first operating system to run independent of the platform. Also, the first ATM machines appeared. In 1975, Edward Roberts, William Yates and Jim Bybee produced the MITS Altair 8800. Bill Gates and Paul Allen wrote their first product for the Altair—a BASIC compiler. Also, IBM introduced its first PC, the 5100.

[T]

In 1976 and for the rest of my career, all of the important activities were with personal computers. In 1976, the Apple II came out. In 1977 Commodore Pet and the Radio Shack TRS-80 joined in. In 1978, Visicalc was introduced and the first spreadsheet programs came into use. Word was beginning to spread of the convenience of these small machines. Wordstar word processor was added in 1979. Perhaps the biggest breakthrough came in 1980 when Alan Shugart introduced the Winchester hard drive. IBM introduced the PC in 1981 and provided DOS as its operating system. In the same year, Commodore introduced the VIC-20 and sold a million units.

[1] http://kbs.cs.tu-berlin.de/~jutta/time/

The chronology in the Introduction is based upon this reference.

[2] http://www.turing.org.uk/bio/part3.html

This idea is illustrated by the following quote from this web site: "The Universal Turing Machine exploits what was later seen as the 'stored program' concept."

[3] http://www.turing.org.uk/bio/part3.html

The paper referred to is *On Computable Numbers with an Application to the Entscheidungsproblem, Aug. 1936.*

[4] http://ftp.arl.army.mil/~mike/comphist/96summary

[5] *Ada, A Life and a Legacy*; by Dorothy Stein, MIT Press Series in the History of Computing, 1985

[6] http://www.epemag.com/zuse/part5.htm A description of Plankalkul and reference to other Zuse works.

[7] *The Preparation of Programs for an Electronic Digital Computer, with Special reference to the EDSAC and the use of a Library of Subroutines,* by Maurice V. Wilkes, David J. Wheeler and Stanley Gill.

[8] http://www.awc-hq.org/lovelace/1997.htm This is a brief biography of Betty Holberton.

[9] http://www.cbi.umn.edu/oh/display.phtml?id=78 In addition to providing an interesting discussion of early work on the EDSAC and some of its contributors, this interview seems to attribute the use of flow diagrams for program design to von Neumann.

[10] http://members.iinet.net.au/~dgreen/ Data on first computer deliveries in 1952 and later are based upon data from this URL.

[11] www.cs.unr.edu/~ban/Dclist.html A list of early digital computers in the order of their development dates.

[12] http://www.stanford.edu/group/mmdd/SiliconValley/Valley/valley.rtf
This contains (toward the end) an excellent description of the debate between advocates of analog versus digital computers in the 1950s. It also indicates that the Hurricane Computer (RAYDAC) was considered for use in the SAGE system.

[13] http://www.computer.org/history/development/
The chronology of events in the Introduction and Chapters 3 through 13 and Notes is taken from this reference.

[14] http://www.thocp.net/timeline/1952.htm Fred Gruenberger is credited with writing the first computer manual in 1952.

[15] Digital Computer Programming by Daniel D. McCracken (1957)

[16] *Generalization: Key to Successful Electronic Data Processing,* by W. C. McGee, Journal of the Association for Computing Machinery 6 (January 1959): 1-23.

[17] Private communication from E. C. Roddy to R. C. McGee. Email dated June 24, 2003.

[18] See Note 15.

[19] http://www.cc.gatech.edu/gvu/people/randy.carpenter/folklore/v6n1.html
The first 1103A was delivered to Lockheed, Sunnyvale in September, 1956. It had an interrupt capability. I have been unable to find any earlier commercial machine with interrupts.

[20] Private communication from Charles Bachman to R. C. McGee. Email dated January 1, 2004. I am indebted to Charlie's wife Connie for the dates from her personal diary that Charlie was able to provide in this email.

[21] *A Re-evaluation of Generization,* Datamation, July/August 1960, pp. 25-29, by R. C. McGee and H. Tellier

[22] The history of the General Electric Computer Department up to my arrival in Phoenix in 1961 is taken from *GE: King of the Seven Dwarfs* by Homer R. Oldfield.

[23] http://www.cbi.umn.edu/oh/pdf.phtml?id=92
An oral history by Fernando Corbató, his early days at MIT including the development of CTSS and a brief history of Multics and its development. In this history, credit is given to Jack Dennis for the address development hardware design of the GE-645. I would have thought that Ted Glaser should have gotten at least some recognition.

[24] http://www.mit.edu:8001/afs/net/user/srz/www/multics.html
Multics home page--the entry to an enormous collection of Multics facts and information.

[25] See Note [13].

[26] See note [H]

[27] http://www.cbi.umn.edu/oh/pdf.phtml?id=104
This is a personal history by Jack Dennis in which he is given credit for recommending that both paging and segmentation be provided in the GE-645. He also indicates he disapproves of the way the MULTICS developers used the hardware provided.

[28] Private communication from Charles Bachman to R. C. McGee. Email dated July 30, 2003.

[29] See note [26].

[30] *GE: King of the Seven Dwarfs* by Homer R. Oldfield Page 215.

[31] http://www.scholzroland.de/VPStuff/MYHIST.htm

[32] http://starfish.osfn.org/~mikeu/h316/history.shtml

[33] IBM 7090 PROGRAMMING SYSTEMS, SHARE 7090 9PAC
Part 1: INTRODUCTION AND GENERAL PRINCIPALS, Manual J28-6166
Part 2: The File Processor, Manual J28-6167
Part 3: The Reports Generator, Manual J28-6168

[34] Private email, Glenn Otterbein to R. C. McGee, 8/27/2003.

[35] Email, Fred Ourenn to Glenn Otterbein,8/25/2003.

[36] Email, Fred Ourenn to R. C. McGee, 9/14/2003.

[37] "Evolution of Database Management Systems" by Fry and Sibley, Computing Surveys 8 (March 1976).

[38] Email to R. C. McGee from Phillip Frana dated 11/21/2003 included the following references:
Introduction to RPG II, 2nd edition (Rochester, Minn.: International Business Machines Corporation, 1971).

IBM-System/3: Disk system, RPG II, and System Addditional Topics Programmer's

Guide (Rochester, Minn.: International Business Machines Corporation, 1970).

IBM-System/3: RPG II Auto Report Feature Reference Manual, 2ne edition, Technical Newsletter No. SN21-7691 (Rochester, Min.:International Business Machines Corporation, 1973).

IBM-System/3: RPG II Disk File Processing Programmer's Guide, 2nd edition (Rochester, Minn.: International Business Machines Corporation, 1974).

Solomon Martin Bernard, System/3 Programming: RPG II (Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1972).

[39] Postley, John A, Mark IV: Evolution of the Software Product, A Memoir". IEEE Annals of the History of Computing, Vol. 20, No. 1 (Jan-Mar 1998): 43-50.

[40] http://www.softwarehistory.org/history/informatics.html

[41] http://ca.com/acq/sterling

[42] Bauer, Walter F., "Informatics, An Early Software Company", IEEE Annals of the History of Computing, Vol. 18, No. 2: 70-76

[43] Private communication: W. C. McGee to R. C. McGee. Email dated 11/26/2003 says, 'Concerning other generalized programs, a survey of some 20 of these programs is given in my article "Generalized File Processing" in M. Halpern and C. Shaw (Eds.), Annual Review of Automatic Programming (Pergamon Press), 1969. Many of these came later than the HAPO programs, but some must have been in development concurrently.

[44] http://www.corsairsfreehold.com/nuke.htm

[45] http://picturethis.pnl.gov

[46] http://www.pnl.gov/main/welcome/

[47] http://www.pnl.gov/main/welcome/history.html

[48] "The need for speed", Tri-City Herald article from Glenn Otterbein to R. C. McGee, 6 September 2003.

[49] http://www.honeywell.com/about/page1_1.html

[50] http://feb-patrimoine.com/Histoire/english/chronoa11.htm

[51] http://www.bull.com/servers/gcos7/

[52] http://perso.club-internet.fr/febcm/english/gecos_to_gcos8_part2.html

[53] http://www.bull.com/servers/gcos8/

[54] http://www.20minutesfromhome.com/twminpages/BULLprofile.html

[55] http://www.multicians.org/history.html

[56] Personal email, Paul Kosct to R. C. McGee, dated 10/12/2003

[57]http://www.spies.com/~aek/pdf/honeywell/multics/AL39-01C_multicsProcMan_Nov85.pdf

[58] See note [13].

[59] See note [13].

[60] See note [13].

[61] See note [13].

[62] See note [13].

[63] See note [13].

Index