C L A S S N O T E S F O R

CS 333

COMPUTER SYSTEM ORGANIZATION

Chapter 1: Introduction and History

FALL, 1974

By D. J. Kuck

June, 1970

Designer's Difficulties

Struggling with the design and construction of the world's most powerful computer has never been easy. And in many ways the nature of the struggle has been constant through time. It has taken at least four or five years to get every major new machine going. Typically, financial crises arise, regardless of whether the undertaking is in a university or industrial setting. And the speedup over the fastest previous machine has never been much more than factor of ten, often much less. Still the cumulative results from the mid 1940's to 1970 have resulted in an impressive speedup factor of 10^6.

Just as impressive, but more bewildering is the growth in complexity of computer organization. Early machines contained a few thousand relays or vacuum tubes, but modern ones are approaching 10^6 transistors. One of the designer's main trade-off problems has always been between the number of parts he uses and the speed of each individual part. Since for a fixed cost he always wants as fast a machine as possible, he can choose a simple organization with very fast parts or a more complex organization with slower parts. The fewer the parts the higher the reliability, but faster parts cost more than slow ones and producing them may be very difficult. The designers of the most powerful machines have always pushed both reliability and cost to their limits. One reason for this is that from the early 1950's on, there have usually been two or more groups in competition to build the next big machine.

For the moment we can leave the definition of "most powerful machine" at the intuitive level of "fastest and biggest." But modern machines have several goals in addition to these traditional ones. From the standpoint of operating cost, maximum "throughout" is desired. In other words, a computing center manager would like to collect fees for as much of his machine time as possible. This becomes a difficult matter when complex operating systems and input/output equipment are used, since these may consume a good deal of overhead time. Another goal which is becoming more difficult to achieve is low "turnaround time" for users.

When many individuals are attempting to use a common central facility, the system response time may get very long. To a large extent these newer problems are related to the software provided for big machines. Thus the modern design of super machines must really be the design of a hardware-software system.

Earlier we remarked that certain machine design difficulties have not changed in time. The overall design of systems has in fact become more complex due to the introduction of software design questions on top of hardware or logical design. No large machines has been a one man show. Thus the designer-builder interface has very often been the source of much difficulty. These difficulties include personality clashes, technical disagreements, failures to communicate, etc. Interfacing the designers and implementers of software is no easier than with hardware people and indeed seems to be very much harder. Furthermore, now the hardware and software designers must talk to each other. Currently, large machine projects may involve literally hundreds of professional people. Usually, the more, the worse.

Finally, in our jeremiad of big system design, the bitterest pill of all for imaginative designers is the "design freeze." Having kept open all options as long as possible, the designers must make their final decisions and stop designing. The several year construction period which follows is similar to a gestation period in that changes in the design are virtually impossible and if attempted may prove fatal. In reality, of course, there are always some mistakes in the design and as many of these as possible are removed. These changes often cause major expenditures of money and sometimes degrade the machines' performance.

In this introduction we shall quickly sketch the history leading to modern digital computers. We do this for several reasons. First, in spite of their great number of parts, computers are quite simple in functional terms and it is interesting to learn when various ideas were first proposed or implemented. It is also revealing to note how few really big innovations have occurred. Finally, we cannot resist telling the story of Charles Babbage.

The World's First Computer Designer

Although present machines are direct descendents of ideas of the mid-1930's, Babbage designed his Analytical Engine, the world's first general purpose digital computer, nearly 150 years ago.[1] He also built a prototype of the world's first special purpose digital computer, his Difference Engine, which he evidently first thought about in 1812 -- ten years after the invention of the steamboat! The ideas that he and a few colleagues had about computers and programming over some 30 years are overwhelming. They touched on a great many of the ideas used in modern computers. Nor were his thoughts limited to computers, as we shall see later.

Not surprisingly, Babbage had to face many of the above mentioned difficulties that present day designers encounter. Several of these proved so overwhelming that he never finished anything but a prototype of the Difference Engine. His major problem seems to have been a too ambitious plan -- a block over which every designer must stumble at least once. This led to financial problems and difficulties with his chief engineer.

Babbage himself wrote down few details about his machines and it was said that his lectures about machines were pretty much incomprehensible. Fortunately, an Italian army officer named Menabrea, who sat through a series of lectures Babbage gave in Turin in 1840, published a good account of the Analytical Engine. This was later translated into English and, at Babbage's suggestion, annotated by his colleague Ada Augusta, Countess of Lovelace. On reading this paper as well as several by Babbage one is depressed by the relatively small progress made by thousands of modern computer scientists. Or, to be more correct, one is annoyed by how often the same problem is discovered, worked on, solved, and breathlessly discussed in the current literature.

Babbage had been motivated as early as 1812 to consider a machine which could evaluate polynomials by the method of differences. He was annoyed by the fact that human computers of astronomical and other tables were usually

people of some intellectual accomplishment but that such computations really required only mechanical skills. He was also bothered by the large numbers of errors occurring in published tables as well as errata in errata sheets. So between 1820 and 1822 he built a six decimal digit Difference Engine capable of evaluating any second degree polynomial. Initial conditions were placed on wheels by hand. Spurred by his success with this project he obtained Government funds for a 26 digit, sixth degree Difference Engine. This was a very much more complex machine. It was to have automatic rounding, provision for double precision arithmetic, various alarm (interrupt and completion) bells, as well as a method for engraving copper plates for printing the computed results. The latter would preclude transcription errors. Concerned about inherent mechanical errors, Babbage arranged various roller and conical bearings that would jam if certain mechanical tolerances were exceeded. If completed, the Difference Engine would certainly have revolutionized the tabulation of mathematical functions. It must also be noted that Babbage was developing a complex design notation for communicating his ideas to his engineering and construction people.

This project dragged on for 10 years until 1833 consuming 17,000 pounds of English government money and perhaps as much of Babbage's own fortune. During this period Babbage engaged in a series of fund raising activities and became increasingly at odds with his chief engineer Clement. Evidently he proposed many design changes but the exact details of the collapse of the project do not seem to have been recorded. In any case, by the early 1830's he was only interested in obtaining funds for the construction of his newest idea, the Analytical Engine. Before discussing its details, we shall set these events in historical perspective by noting the following. The chronometer of Harrison, which was the first one adequate for precise longitudinal transoceanic navigation, was produced in the 1760's after a very long and trying experience. It took Harrison 3 years to produce a copy of his first successful model. Interchangable parts were not to come for some

time. In fact Whitworth, who later introduced standard screw threads among other things, lost his job with Clement when the Difference Engine project collapsed. Babbage worked at a time which was sparked with great inventions -- the steam locomotive in 1825, the electric generator in 1831, the reaper in 1834, the electromagnetic relay in 1835, Daguerreotype in 1839 and telegraphy in 1844. Of course, no thought of an electrical machine was possible then. But one is impressed by Babbage's courage to attempt so complex a mechanical device given the state of the art at the time.

Babbage's machines were all designed to be driven by a hand crank, but in one of his accounts of his first inspiration he quotes an early conversation with John Herschel. They were checking some tables and Babbage said "I wish to God these calculations had been executed by steam," to which Herschel replied "It is quite possible." Herschel, Babbage, and George Peacock had been friends as Cambridge undergraduates, where they formed the Analytical Society. Later Herschel became a famous astronomer and Peacock a leading algebraist at Cambridge. Babbage later had many discussions of his machines with these men and many of the leading scientists of the day. LaPlace, Bessel and Jacobi (not to mention the Duke of Wellington) all had extensive discussions with him.

It is fascinating to note that Boole and DeMorgan were both contemporaries of Babbage, but no interaction between them has been noted concerning machine design. However, Ada Augusta Byron -- the poet's daughter -- studied mathematics under DeMorgan for many years. Mrs. DeMorgan notes than on an early occasion, she took Ada to visit Babbage and that Ada quickly understood what was going on. Some years later as Lady Lovelace, she translated Menabrea's paper on the Analytical Engine and collaborated with Babbage.

The Analytical Engine that Babbage designed in the 1820's and 1830's was spectacular, even by the standards of the 1950's. His design methods and his ideas for the machine's organization and use demonstrate Babbage's genius.

The immense complexity of what he hoped to build demonstrates his kinship with many of todays designers. By pushing funds and technology to the limit -- and often too far past the limit -- he faced a long series of frustrations.

The Analytical Engine was to be a fifty decimal digit machine. Its "store" or memory was to hold 1000 of these words (about 165,000 bits) in decimal form. These words could be written from or read to the "mill", or arithmetic and logical unit, via some mechanical linkages. The whole system was under the control of a process which was described on two sets of punched cards. One set, the "operation cards" contained the series of operations to be performed. The other set, called "variable cards" indicated which store locations were to be operated on by the operation cards. Babbage was quite familiar with the Jacquard loom which was controlled by a sequence of punched cards. In fact, the punched card idea dated back to the early 1700's, although Jacquard's famous loom was not developed until 1804.

While the Analytical Engine did not have a stored program, it was able to perform various kinds of condition tests and then branch on the outcome. In particular it could move its card sequence forward or backward a fixed distance. Furthermore, there was an index register and index adder available for loop control; to quote Menabrea, "When the number n has been introduced into the machine, a card will order a certain registering apparatus to mark (n-1), and will at the same time execute the multiplication of b by b." This is in a discussion of evaluting b^n. Note that the indexing arithmetic was apparently carried out in parallel with the multiplication. The index register was evidently not used to index through memory, however.

The arithmetic unit was designed to perform fixed point, fifty digit calculations at the following speeds: add or subtract in one second, multiply or divide in one minute. To achieve such speeds Babbage devised, after years of work, a parallel addition algorithm with anticipatory carry logic! He was

very proud of that accomplishment. As in the Difference Engine, Babbage provided for multiple precision operations, automatic mechanical fault prevention and detection, and automatic rounding and overflow detection.

Babbage was bothered for some time about the provision of standard function values e.g. log x, sin x, to the machine. Finally he concluded that either the recomputation of such numbers, essentially via a subroutine, each time they were needed or their provision from external cards would work. He was willing to let the decision rest on operating experience. His table look-up procedure was arranged as follows. The machine's operator would be provided with drawers full of such cards punched with both x and f(x). When a bell rang the operator would read a dial and pick out the corresponding card. The machine would check to see that the correct card had been supplied by testing the argument and if an operator error had occurred a louder bell would ring. He was quite proud of this idea because the problem as well as his solution had evidently perplexed Bessel, Jacobi and others for some time.

When reading Babbage, Menabrea, and Lovelace one is amazed and delighted to see how far the questions of mechanical computing were explored. It is tempting to read things into their statements from time to time. On some occasions they are exasperatingly brief and sometimes they are ambiguous or they mildly contradict each other. Such matters as the self checking mechanisms which would jam when too much mechanical error accumulated are hard to understand and the writers said they would not attempt a complete explanation. On the matter of parallel arithmetic operations they make several passing remarks. We quoted Menabrea above about index calculations. At another point, in his summary, which seems to indicate the importance of the idea, he is discussing the speed of the machine and says, "Likewise, when a long series of identical computations is to be performed, such as those required for the formation of numerical tables, the machine can be brought into play so as to

give several results at the same time, which will greatly abridge the whole amount of the processes." This seems to be a clear statement of parallelism between arithmetic operations!

Babbage and Lady Lovelace both discuss programming questions, but she exhibits her own great insight in her notes on the Menabrea paper. She was quite concerned about languages for expressing programs. One was a kind of assembly language notation on large charts. These were translated from another notation very much like compiler assignment statements. All variables were denoted by V_i where i indicates the storage location from 1 to 1000. To avoid the confusion of writing $V_1 = V_1 + V_2$ she introduced another index and wrote ${}^{m+1}V_1 = {}^{m}V_1 + {}^{n}V_2$ to indicate that the right hand side values were the mth and nth values to occupy their respective storage locations. Her machine level language was a kind of zero address operator language, although a separate operand stream was specified to the machine. Thus, to evaluate

$$x = \frac{d'm - dm'}{mn' - m'n}$$

$$y = \frac{dn' - d'n}{mn' - m'n}$$

she would use these three operation cards $6(\times)$, $3(-)$, $2(\div)$ where commas separate the cards. Note that the common subexpression in the denominator is evaluated just once. Locations were supplied by a three address scheme using three variable cards, two for the arguments and one for the result.

She finally suggests a loop notation using the $\sum$ sign to denote loop control. She also allows for an index variable and nested loops! Her notes contain several quite complex programs but she and Babbage were not bothered by long programs. In fact they were both heartened by the fact that Babbage owned a Jacquard tapestry which had required over 20,000 cards for its production. She does remark that from the standpoints of time required and ultimate accuracy, some numerical results would be impossible to attain in any practical sense.

We noted above that during the course of the Difference Engine project, Babbage had received 17,000 pounds from the Government. He had spent perhaps as much of his personal inheritance from his banker father. Thus, by the time he was deeply involved with the Analytical Engine, sources of funds were scarce. Evidently Lady Lovelace and her husband were fairly well heeled and were both interested in horse racing as was Babbage. So at one point they devised betting procedures, evaluated them on the prototype Difference Engine, and lost a good deal of the Lovelace fortune.

On another occasion Babbage studied game playing (including chess) on the Analytical Engine and designed a tic-tac-toe machine. He proposed to put several of them on the road with admission charges. Perhaps he had heard of Mälzel's "automatic chessplayer" which was revealed to contain a man. One is also reminded of Mälzel's collaboration with Beethoven which resulted in "Wellington's Victory" but no machine. In any case, Babbage dropped this plan.

Viewed on the whole, Babbage's life was a very interesting and creative one; his computing activities formed only one facet of his career. We conclude with a short discussion of some of his other interests. He carried on a life-long battle with street musicians - hauling them into court on several occasions. As a result, his home was the scene of frequent retaliatory concerts. Being much interested in the heart beat and respiratory rates of all animals, he took every opportunity in his travels to measure these. On one occasion he had himself sealed inside a 265° F oven for about five minutes to study the effects on himself. Railroads, a new invention, were a great interest and he is credited with many ideas including the invention of the first recording speedometer as well as the first cowcatcher. A contribution of which he was very proud was a notation for describing the motion and "logic" of his mechanical drawings for his Engines. Earlier in his life he and his Analytical Society friends had been instrumental in getting English mathematicians to drop Newtonian notation for the calculus in favor of that of

Leibniz. We shall end this discussion with an abbreviated list of other writings and work: an operations research type study of the post office system; meteorological and tree ring observations, electricity and magnetism, a light house occulting system widely adopted, various other signaling schemes and a study which convinced him that the Analytical Engine could play chess with a "3 or more" move lookahead.[4] In short, while Babbage may occasionally have been in error he was seldom at a loss for ideas about a subject.

He was Lucasian Professor of Mathematics at Cambridge for nine years, but bitterly remarked that that was the only honor conferred on him by his own country. Babbage's entire life was filled with the frustration of having few of his ideas appreciated and even fewer adopted. Toward the end of his life a friend noted, "He spoke as if he hated mankind in general, Englishmen in particular, and the English Government and Organ Grinders most of all." In his book "The Exposition of 1851" he expressed his feelings quite clearly when he wrote, "Propose to any Englishman any principle or any instrument, however admirable, and you will observe that the whole effort of the English mind is directed to find a difficulty, a defect, or an impossibility in it. If you speak to him of a machine for peeling a potato, he will pronounce it impossible; if you peel a potato with it before his eyes, he will declare it useless because it will not slice a pineapple. Impart the same principle or show the same machine to an American or to one of our Colonists and you will observe that the whole effort of his mind is to find some new application of the principle, some new use for the instrument." In 1871, the London Times noted in his obituary that he lived to be almost 80, "in spite of organ grinding persecutions."

Actually Babbage lived to see some small successes for his ideas. Inspired by a published account of his Difference Engine, a Swedish printer, George Scheutz, and his son, Edward, built a machine. Scheutz spent a good deal of his own money and had some government support. In 1854 he exhibited

in England his fourth order, eight digit difference machine with a printing output mechanism. Babbage and his son received Scheutz warmly and after a good deal of publicity the machine was sold to the Dudley Observatory in Albany, New York. Whether or not it was much used seems to be in question. In any case, a copy was made in 1863 and the British Government used it to compute actuarial tables for the newly emerging life insurance business - a topic on which Babbage had discoursed in earlier times.

Babbage's son, H. P. Babbage continued to work on the Analytical Engine and after his father's death managed to construct some working parts of the mill between 1880 and 1910. At a demonstration this machine computed and printed a table of twenty digit multiples of π.

In the 1880's another interesting forerunner of modern computer equipment was under development. Working at the U. S. Patent Office, Herman Hollerith, an engineering graduate of Columbia, constructed a punched card tabulating machine. By 1890, Hollerith machines were in use at the U. S. Census Bureau for processing returns of the 1890 census. Hollerith later went into business for himself, manufacturing a variety of card processing equipment. He was quite successful and as we shall see below, his company became a basic building block in the modern computer industry.

In 1892, young Thomas J. Watson launched his sales career on a horse drawn wagon, peddling sewing machines, pianos, organs and caskets out of Painted Post, New York.[2] Before long he had moved to Buffalo, and Rochester and became a star salesman for the National Cash Register Company of Dayton, Ohio. His record having been observed by J. H. Patterson, the head of NCR, Watson was elevated to various positions and by 1914 was more or less the number two man at NCR, which by then was the largest cash register company in the U. S. His position in the company and the company's position with respect to competition caused Watson some difficulty.

First, Patterson was a manager who ruled with an iron, if somewhat bizarre, hand. His executives had to engage in various Patterson designed regimens (e.g. prework group horseback riding and special foods) and were fired for various kinds of real or imagined insubordination. Occasionally instead of firing someone Patterson would provide him with a "fresh start" by moving the entire contents of his office out on the front lawn, dousing it with kerosene and touching a match to it. So, after almost twenty years with NCR and the survivor of many earlier purges, Watson was fired by Patterson in 1914.

The foremost market position of NCR was due in large part to Watson's efforts, but this was his second difficulty. Some months before his firing, a number of top management NCR people including Patterson and Watson had been taken to court for a number of illegal business practices. They had essentially eliminated all competition in the new and used cash register business by strong selling, price cutting, industrial espionage, personal harrassment and their ultimate weapon, the "knockout machine." This was a flimsy copy of a competitior's machine which would be sold cheaply as the real thing and soon break down. Watson was at the time of his firing appealing a fine and one year jail sentence. In spite of this, Watson asked Charles R. Flint for a job.

Flint was a New York tycoon who had invested in practically everything, and in 1911 had formed one of the early conglomerates of diverse product manufacturers -- the Computer-Tabulator-Recording Company, otherwise known as C-T-R. This included a number of companies making equipment that could be called business machines, and included Herman Hollerith's Tabulating Machine Company. When Flint proposed Watson to the Board as manager of C-T-R, there were some raised eyebrows, but Flint prevailed. Later the jail sentence and other litigation disappeared. Watson moved rather slowly at first, but became C-T-R president and by 1924 was solidly in command. In 1924 he changed the name of the company to International Business Machines.

In many ways, Watson ran IBM as Patterson ran NCR. He was once referred to as a "benevolent despot", but he was more rational and if not intellectually inclined, he did enjoy and have good intuition about making money. IBM flourished and by the mid 1930's Watson was the highest paid person in the U. S.

Watson's interest in developing new products as a way to higher profitability caused him to support various new machine development activities within the company. He also enjoyed talking with people inside and outside IBM about possible uses of his equipment. Thus, when he was telephoned by a young education professor at Columbia, Benjamin D. Wood, in 1928, Watson said he could spare an hour for a lunch meeting. The meeting went well and Watson stayed until 5:30 listening to the problems and ideas Wood presented. In short, Wood had been developing intelligence tests for college students and had 35,000 to process. With a room full of girls and some equipment he had designed, the processing of these tests was costing at least $5.00 each. He explained how these tests and similar material could be processed for perhaps 10 or 20 cents using IBM equipment -- perhaps with some modification. Two days later Wood had a room full of IBM equipment at his disposal, free of charge. His predictions were correct and he continued to offer suggestions to Watson including one that

the mechanical parts should be eliminated in favor of all electrical equipment. This association led to a line of IBM equipment for education, and Wood remained an IBM consultant for many years. More important, the equipment attracted the attention of other Columbia faculty and students. An astronomy graduate student, Wallace Eckert, talked to Wood and Watson. This later led to another gift to Columbia, the T. J. Watson Astronomical Computing Bureau. One of Watson's top engineers, Clair D. Lake, built a special machine for the Bureau. It was the first machine which could multiply and it also had a sequencing mechanism. It was used for the computation of astronomical and navigational tables -- the latter were very important in antisubmarine warfare in the North Atlantic in the late 1930's. Later, Eckert joined IBM as the first director of the T. J. Watson Laboratory which was located near the Columbia campus.

Eckert's earlier astronomy calculations had attracted a good deal of attention and among his visitors were Harlow Shapley, astronomy professor at Harvard University and James B. Conant, the president of Harvard. Shapley discussed the Columbia work with Howard Aiken who was teaching mathematics in Harvard's Graduate School of Engineering. Aiken had known about the state of the art in computing and had been thinking about building a more complex machine. Shapley prompted Aiken to visit Eckert at Columbia and later to discuss his ideas with James W. Bryce of IBM. Bryce had been one of IBM's key inventors for thirty years and as a result of these discussions Watson put up a million dollars to build a machine for Aiken.

Although, Watson had a reputation for occasionally trampling on everyone close to him -- including the Columbia professors -- Aiken had shopped around and found no one but IBM capable of building his machine (whose details will be discussed later). Aiken also had a strong personality. Watson apparently did not involve himself much with the project until the machine was finished.

At that point he decided it should be enclosed in a special glass and stainless steel case; Aiken strongly disagreed. Watson won that round as he always had within the company. Watson had been honored by many organizations and nations and expected that his gift of a million dollar machine plus another $200,000 for operating it would bring out the best in Harvard. When Watson arrived at Harvard for the dedication he found that it was Aiken and not Watson who was to get the credit for the machine. After raising a ruckus which included a threat to take the machine away, Watson was calmed down by President Conant who then made a speech at the dedication.

Watson died in the mid-1950's and was succeeded by his son as president of IBM. The company has continued to build punched card equipment and other machines.

Modern Machine Beginnings

Three men ushered in the modern digital computer era in the 1930's. They were Howard H. Aiken of Harvard University, George R. Stibitz of Bell Telephone Laboratories and Konrad Zuse of the Technische Hochschule in Berlin. Collectively they designed and built a number of relay machines and by the 1940's, each had completed a general purpose programmable digital computer. They all apparently worked independently of one another, although Aiken used the engineering talent of IBM to build his machine, in particular three men were his coinventors: B. M. Durfee, F. E. Hamilton and C. D. Lake, who had designed a good deal of earlier IBM equipment. By 1946, J. P. Eckert (no relation to Wallace Eckert) and J. W. Mauchly of the Moore School of Electrical Engineering at the University of Pennsylvania had successfully completed ENIAC, the first electronic digital computer. This attracted the attention of John von Neumann who, as a consultant, with Eckert and Mauchly proposed EDVAC, the first stored program computer. This design was modified and embellished by a number of people and by 1950 there were more than a dozen big machine projects under way. By 1950, so many of the ideas used in current machines had been proposed and experimented with, that it will take us a good deal of space to outline some of the details. It is, of course, impossible to pin down who had each idea first but we shall attempt a rough chronclogical ordering based on various published documents.

Zuse[3] evidently began first (he had his first ideas in 1934) but his influence outside Germany was probably the smallest of the pioneers. Unfortunately, most of his early work was destroyed during the war. His special purpose relay machines Z1 and Z2 were built between 1936 and 1940. Z3 was a general purpose machine which operated under external program control. It had a 64 word data memory and the numbers were of binary floating point format: 22 bits with 14 mantissa, 7 exponent and one sign bit. The machine contained 2600 relays and was built between 1934 and 1941. During the war Zuse developed two special purpose control computers, one which continously sampled 100 points for process

control. Following the war, Zuse built Z4 and then went into business, commercially manufacturing Z5 and subsequent machines. As we shall see, Stibitz was almost an exact American parallel of Zuse, although a few years behind him.

At Bell Labs, Stibitz built his Model I or "complex computer" between 1938 and 1940.[4] It was not a programmable machine, it simply performed complex arithmetic on numbers presented via a teletype keyboard. Its main claim to fame is that Stibitz demonstated the first remote terminal system (keyboard and printer) to an American Mathematical Society meeting at Dartmouth in 1940, using the machine which was in New York City.

Subsequently Bell Labs built several other relay machines, including an interpolator and a ballistic computer each of which had a few internal registers for data storage. Between 1944 and 1947 Stibitz and S. B. Williams built the Model V system which was a general purpose two processor machine. This machine contained 9000 telephone relays and 50 pieces of teletype equipment occupying 1000 square feet of floor space. The speeds of each processor were: 300 millisecond for addition, 1 second for multiplication, about 5 seconds for divide or square root, and .07 seconds for a register to register transfer. Earlier Stibitz machines had used an excess three binary number system, but for this machine Stibitz invented and used biquinary decimal numbers for several reasons. It made self checking, conversion to decimal, and implementation in relay circuits relatively easy. The numbers were floating point with seven decimal digits and an exponent of magnitude less than 20. Each processor's internal memory was 15 relay registers. The entire system consisted of two such processors and three I/O positions, all interconnected. Each I/O position could handle a number of I/O devices. Thus one job could use both processors or two separate jobs could be run together. Furthermore, the machines could, on completing one job, switch to another I/O position. Thus, set up time by a human operator could be masked. Also the tape motion time to access a new job could be masked and by preparing a number of jobs

on several paper tapes the machine could be run overnight, unattended.

The machine was programmed using a simple three address symbolic language, taking advantage of the fact that the 15 registers were named by letters of the alphabet. Loops could be programmed by making paper tape loops With typical Bell System concern for reliability, the machine had various self checking features and high reliability was achieved. The chief cause of difficulty was dirty relay contacts. Various lamps would indicate to an operator where the difficulty was if the machine stopped. On an unattended run, the machine could abort one job and proceed to try the next one if a fault occurred. Two of these machines were built, one for the National Advisory Committee for Aeronautics (Langley Field, Va.) and one for the Ordnance Department of the Army (Aberdeen Proving Ground, Maryland).

Bell Telephone Labs constructed a Model VI system in the late 1940's which was installed at their Murray Hill, N. J. Laboratory. This machine was in several ways an improved version of the Model V. First, it had a number of remote terminals from which jobs could be submitted to the machine via telephone lines. Second, when a job failed for some reason, the machine would automatically restart it and try once more. A sticky relay might work the second time. If not it would go on to the next job as did Model V. These two features made the system appear to be very much like a modern machine with a remote entry batch processing operating system.

Another interesting feature of Model VI was the ability to wire in subroutines. Provisions were made for up to 200 such subroutines. They could call each other and be nested down to four levels. Since the program was otherwise on external paper tape, this speeded up the operation of the machine and made the programmer's life easier.

Models V and VI were both "asynchronous" machines. That is, they had no controlling clock; when one step of an operation was over it caused the next

step to begin. This design philosophy has been tried with varying success in some modern high speed machines.

In contrast to the Bell Labs approach, Aiken and the IBM group designed a synchronous computer which was operated at a 300 millisecond cycle.[5] This machine was designed and built between 1937 and 1944. IBM became involved in 1939 and the work from then until completion was carried out in their facilities at Endicott, N. Y. The machine was operated at Harvard University, and was known either as the Automatic Sequence Controlled Calculator or the Harvard Mark I. Mark I was 8 feet high, 51 feet long and 6 feet deep. It was a decimal, fixed point machine using a 23 digit plus sign, word. It could store 72 such words in 10 position counter wheels and had an additional 60 number storage facility in manually set dial positions (what would now be called a read only memory). Its speeds were add or subtract in 300 ms., multiply in 6 seconds, divide in 11.4 seconds, and it could evaluate several special functions in about one minute. These latter were so slow that faster, lower accuracy subroutines were often used. The machine could also perform double precision or half word operations.

Instructions were externally stored on 24 hole paper tape and were in two address format. Initially it could conditionally jump to one of two external tape routines based on the range of an argument. This was later changed to a branch to one of several tapes based on a more general transfer on minus instruction.

Programming for maximum speed could present interesting challenges. All operations shared a main bus and during the exection of a long operation the programmer could initiate shorter commands such as addition or certain I/O operations. A hardware interlock prevented these "interposed operations" from conflicting with the longer ongoing operation. Evidently this technique was used a great deal. Mark I was the first large scale machine to be completed and was first used to compute various tables and later used to solve systems of algebraic and differential equations. After it was broken in, Mark I was quite reliable,

reportedly available 95% of the time in 1950, and it was in use for 15 years.

While we have gone over the period of early development in a very quick way, it is clear that spectacular progress was made. Zuse, Stibitz and Aiken had broken ground for events that in the subsequent five years would yield the "modern" digital computer. While their mechanical realizations were great feats of engineering, their ideas were mainly rediscoveries of things that were well known to Babbage exactly 100 years earlier. For their implementations alone, however, they would have earned Babbage's respect, as he wrote in "The Life of a Philosopher" in 1864, "If, unwarned by my example, any man shall undertake and shall succeed in really constructing an engine embodying in itself the whole of the executive department of mathematical analysis upon different principles or by simpler mechanical means, I have no fear of leaving my reputation in his charge, for he alone will be fully able to appreciate the nature of my efforts and the value of their results."

The Second Wave

The improvements introduced in the next wave of machines included electronic parts, large internal memories, stored programs, index registers, and magnetic tape and drum secondary storage. By the early 1950's the typical machine could multipl in a few milliseconds and had 1024 words of primary memory. We shall attempt to point out the most important steps in terms of the people who made them and the machines they built.

In 1943, Mauchly and Eckert undertook the design of what turned out to be one of the physically largest computers made before or after that time.[6] ENIAC was sponsored by the Army Ordnance Department and was intended to integrate ordinary differential equations for the generation of ballistics tables. It was finished at the Moore School in February, 1946. The machine was configured in a U-shape but overall it was about 100 feet long and 8 1/2 feet high. It contained 18,000 vacuum tubes and 1500 relays and consumed 150 kw of power. Each register in the machine used 550 tubes and was about 2 feet wide and 8 1/2 feet high! In spite of its gargantuan dimensions the machine was very fast and quite reliable.

ENIAC was a ten digit fixed point decimal machine with a parallel arithmetic unit which performed at the following speeds: add in 200 μs, multiply in 2.8 ms., and divide in 6 ms. It also had a square root unit and was capable of double precision operations. Its internal memory consisted of 20 registers, each of ten digits. It was able to do I/O and arithmetic simultaneously and had an 800 card per minute reader. Nevertheless, computations were often I/O bound and while its raw speed was a factor of 1000 over Mark I its overall performance may have been closer to a speedup of two or three hundred. The machine was externally programmed by attaching various portable "function tables" which would be arranged by the programmer. These external tables could also be used as a read-only data memory. The machine was capable of conditional jumps although this feature evolved in time. The time to set up the machine for a particular calculation

ranged from a half hour to a day. In 1947 its "up time" was estimated to be 20% but by 1950, measured over a one month period, the hardware was available 85% of the time; when set up time and program hangups were included, 67% utilization was measured. After completion, the machine was moved to the Aberdeen Proving Ground and various improvements were made. John von Neumann was instrumental in making the programming easier and faster via external boards, wires, and switches.

Having been attracted by ENIAC, von Neumann became a consultant to the Moore School group and began to study the question of machine design. In 1944, Eckert had written a memo suggesting the use of a magnetic drum or disk as the primary memory of a machine. The use of a variety of memories for radar systems had developed during World War II. Crawford had written a thesis at MIT in 1942 suggesting a magnetic disk or drum in this context, and a variety of acoustic delay line memories were in use by radar people at the time.

In 1945, von Neumann wrote a memo as an ENIAC consultant discussing a stored program machine. This important idea, due perhaps to Eckert, Mauchly and von Neumann, led to a new project to build EDVAC. This was to be a machine of much more modest size than ENIAC, but with a larger internal memory and slightly slower arithmetic. While it spawned a great many other machines and ideas, EDVAC was not the first stored program machine to become operational. The project was begun in 1946 and the machine was not operational until 1952. During this period, Mauchly and Eckert left the Moore School to form their own computer company and von Neumann launched his own project at Princeton taking with him several other Moore School people.

In any case, EDVAC was a binary, 44 bit, fixed point machine with a bit serial arithmetic unit. This required only 3500 tubes to achieve average speeds of 850 μs for add, and 2.8 ms for multiply. It had a mercury delay line memory which contained 1024 words of data and program. This was organized as 128 delay

lines each containing 8 words. This memory led the designers to choose a four address instruction format, two for arguments, one for result and one for next instruction, since any of these could be anywhere in the 1024 word circulating memory. The machine had two arithmetic units; the second used for checking the first.

England Pulls Ahead

Following a visit to the Moore School, Maurice Wilkes of Cambridge University started a project at Cambridge at the end of 1946. This led to EDSAC, the first stored program machine to be completed, in 1949.[7] EDSAC was quite similar in design to EDVAC although somewhat slower. It had a 1.5 ms add time, an average 6 ms multiply time and required a few hundred ms for division. Its memory characteristics were much like those of EDVAC described above. The overall machine had about 3000 tubes and dissipated 15 kw. Wilkes was quite interested in questions concerning the programming and use of the machine. Among other things, he developed a large subroutine library for EDSAC users.

Others had preceded Wilkes in England with thoughts about automatic computers. Alan M. Turing had published his famous paper in 1936 and J. R. Womersley at the National Physical Laboratory had begun to think about real machines in 1945. By 1947, Turing and others had joined him to begin a project which led to the construction of ACE, the pilot model being completed in 1950. The Ace pilot had only about 1000 tubes but achieved an add time of 32 μs on 32 bit words. Its small component count made it very reliable. Shortly after the NPL activity began, the Telecommunication Research Establishment began to study the problem. This led to the development of MADM at Manchester University, the project being moved there in early 1947 with continuing support from the Telecommunications Research Establishment.[8]

Delay lines had a rather long latency; since they operated at a few megacycles and contained several hundred bits, it could take a millisecond to access a word. Thus a random access, large, cheap memory device was sought. At

Manchester, F. C. Williams developed the "Williams tube" which filled this bill. His first tube worked in 1947 and was used in a prototype machine by June of 1948. This was a cathode ray tube with bits stored on its face. They could be capacitively sensed and access time was a function of electron beam switching and sensing times only. Thus, the first large random access memory was available. In 1948, the Manchester group, which also included T. Kilburn, demonstrated a 2000 rpm, head per track, magnetic drum and used this as backup to Williams tube primary memories in 1949.

Using this memory hierarchy, they issued I/O instruction for blocks of data from the drum and stole processor cycles to access the main memory. They built another prototype in 1949 that had an interesting new feature which they called the B-tube. Using the B-tube, they said, "...instructions, and in particular their address section, could be modified in their effect without being modified in their stored form." Thus appeared the world's first index register. With these important innovations as background, they designed MADM in 1949 and it was finished in 1951. This was a one address, binary machine with 40 bit, fixed point operations. Its arithmetic speeds were: addition in 1.2 ms and multiplication in 2.16 ms. 1600 pentodes and 2,000 diodes were used. The Williams tube memory consisted of 512 words stored in 8 tubes, together with a 150,000 bit drum.

We remarked earlier that magnetic recording on disks or drums had been suggested at least as early as 1942. The first successful machine to use a magnetic drum was built in 1947 by A. D. Booth at the University of London. It was called SEC and had 256 words of 21 bits. The arithmetic unit employed only 250 tubes and had a 1.6 ms add time.

Meanwhile, Back at Princeton

Just a year after his EDVAC report, von Neumann and two co-workers, Arthur W. Burks and Herman H. Goldstine, published another report.[9] This was June, 1946 and they were all at the Institute for Advanced Study (IAS) at Princeton University; Burks and Goldstine had both been at the Moore School for some time and had been involved with ENIAC. Their new report was entitled "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," and it was a detailed, clearly argued discussion of many details of machine design. In 1947 Goldstine and von Neumann wrote an accompanying document on the analysis and coding of problems for the machine. These documents led to the construction of the IAS machine which was completed in 1952. Julian H. Bigelow was the chief engineer in charge of the IAS machine. This project became the focal point of computing activities in the U. S. The project was funded by the Army Ordnance Department, with contributions from the Air Force, the Office of Naval Research and the Atomic Energy Commission.

The IAS machine was completed in June, 1952 and was a rather compact unit; excluding the I/O gear its dimensions were 8X8X2 feet. It contained 2300 tubes (many double triodes) and 40 Williams tubes each containing 1024 bits. Thus the memory contained 1024, 40 bit words each being interpreted as one fixed point number or two instructions. The machine had a one address order code with 10 bits of address per instruction. The memory access time was about 25 μs and excluding this, the average arithmetic times were: 15 μs for addition, 400 μs for multiplication and 1 ms for division. Many engineering innovations were included; among them a word parallel memory access feature not included in the Manchester machines. The arithmetic unit also operated in parallel and the machine was asynchronous.

This machine and project were quite important from several standpoints. First, the excellent engineers who built the machine had a number of rather good recent inventions to use. Second, von Neumann and his staff thought very imaginatively and broadly about how to use the machine. Finally, their reports and visitors caused this machine's reputation to be widely known. A number of copies of the machine were built.

In parallel with the IAS activity, the Servomechanisms Laboratory of MIT began to build a machine. One original motivation was the problem of real time aircraft simulation. The Whirlwind I project began in 1947 under Office of Naval Research sponsorship and was directed by Jay W. Forrester. Very high speeds were achieved in the 15 bit (plus sign) parallel, fixed point arithmetic unit: add in 8 μs, multiply in 24 μs. When memory fetch time was included, both operations averaged 180 μs. Whirlwind was a synchronous machine with a 2 megacycle clock for the arithmetic unit. It was also a stored program machine. The machine was operational in 1951.

One important outcome of the MIT activity was in the primary memory area. Initially, Whirlwind had a 1024 word, 16 bit, modified Williams tube memory. Under Forrester's direction, alternative memory devices were being studied. The MIT group was in close competition with an RCA team headed by Jan Rajchman. At least by virtue of consent decrees some ten years later, MIT won the race. (The settlement included royalty-free rights to RCA and a $13 million license from MIT to IBM.) In 1953 they had installed in Whirlwind a 2048 word coincident current magnetic core memory. This memory had a 1 μs read time and an 8 μs write and cycle time and the cores were about 80 mils OD. The machine also had a cathode ray tube for output display with a computer controlled camera attached.

Thus by 1953, Whirlwind I with its core memory, and the IAS machine were both in operation. These two machines are regarded by many people as the first of the "modern" digital computers. They had combined some ten years of engineering development by a number of other groups together with their own inventions and excellent engineering. The influence of these machines was widely felt in both university projects and the newly emerging electronic computing industry.

A New Industry Begins

We mentioned earlier that one of the reasons that EDVAC was not completed earlier may have been the departures of von Neumann and his people to the IAS project as well as Eckert and Mauchly to form their own company. In December, 1947

the Eckert-Mauchly Computer Corporation was founded with financial backing from a multimillionaire. The firm designed and built BINAC for Northrop Aircraft under an Air Force contract. It was an EDVAC-like machine with a delay line memory and about a one millisecond arithmetic speed. BINAC was demonstrated in August, 1949.

At the time, their only commercial competition was from IBM which was selling various combination electronic and electromechanical devices. These included the Selective Sequence Electronic Calculator (SSEC), the 604 Electronic Calculating Punch, and the Card Programmed Calculator (CPC) all introduced in 1948. The CPC actually grew out of an experiment in which a 604 and an accounting machine were joined by people at Northrop. None of these was a stored program machine, and it looked as if the Eckert-Mauchly Corporation had a clear field. Based on their BINAC experience they designed a new machine, UNIVAC, and began taking orders at $250,000 per system.

At that point their fortune changed. Their financial backer was killed in an airplane crash at about the time they realized that the $250,000 UNIVAC price tag was too low to make a profit. Seeking funds they talked with people at the T. J. Watson Laboratory in New York. The technical people there were enthusiastic about UNIVAC but evidently on Watson's decision, the Eckert-Mauchly talks were terminated. James Rand of Remington Rand then discussed the matter with Eckert and Mauchly and subsequently took over their company.

At the time, Remington Rand had a line of desk calculators as well as various punched card equipment. Unlike IBM, Remington Rand used a 6 row, 90 column card. While IBM equipment had been primarily designed for "business applications" it had found its way into many "scientific" uses. Remington Rand equipment seems to have retained the flavor of "business equipment" only, at that time.

The first UNIVAC was delivered to the Bureau of the Census in June of 1951. UNIVAC was a synchronous machine and had a delay line memory of 1000 (not 1024) words of 12 decimal digits. The serial arithmetic unit operated at about 1 millisecond and the numbers were binary coded decimal in excess three format. Magnetic tapes were used as secondary memory and special buffer registers were provided for data entry to primary memory. UNIVAC was quite successful and 48 systems were built (sale price was $750K, although they were also leased).

In 1952 Remington Rand bought out Engineering Research Associates of Minneapolis. ERA had been a pioneer in commercial magnetic drum manufacture and had designed their 1101 and 1102 computers around their drum. The UNIVAC name had numbers attached to it for later Remington Rand machines and still later the 1100 numbering scheme was resurrected.

IBM finally saw the light and in 1950 began a project which led to the IBM 701 by the end of 1952. The 701 was a 36 bit fixed point, synchronous, parallel machine with a 2048 word Williams tube memory. Its speed was about 40 μs for addition and 400 μs for multiplication or division. This was the beginning of a long series of 700 and 7000 series machines. It also signalled the end of the open field for Remington Rand. With Watson's aggressive sales background and widely established sales network, IBM quickly moved in. Eventually nineteen 701 systems were sold and many other machines followed.

Thus by 1953--just nine years after the completion of Mark I--Whirlwind I and the IAS machine were leading the research front and UNIVAC I and the IBM 701 were both commercially available. In 1970 there are some 70 companies in the business of computer manufacturing. IBM has about 70% of the market and its nearest competitor, Honeywell with its newly purchased GE division, has about 8%.

SUMMARY

We will close this Chapter with a synopsis of the history of machine organization up to 1953 and a few remarks about what followed. At this point the reader has surely noticed that a large fraction of the "big ideas" of modern machines were in use by 1953. In fact a good many of them were thought about by Babbage, 100 years earlier. Babbage had proposed a machine organization with a memory, arithmetic unit, control unit and I/O facilities. He invented a parallel arithmetic unit with anticipatory carry logic and an overflow alarm. He also used an index register for loop counting and it worked in parallel with the arithmetic unit. Between them, Babbage and Lady Lovelace proposed a good many programming ideas which were similar to those in current use. Unfortunately, they were a hundred years ahead of the technology.

In fact the vacuum tube and Eccles-Jordan flip-flop circuit were both invented in the first quarter of the 20th century but were not employed until 25 years later in ENIAC. After the feasibility of large general purpose computers had been demonstrated using electric relay and mechanical technology, the events of World War II caused the US and British governments to provide the funds for a good deal of computer research and development. The earlier radar efforts certainly provided many engineering and technology ideas.

By 1953, most of what Babbage had proposed was implemented. Machine speed was the main thing that would have surprised Babbage. He proposed a one second add and a one minute multiply. In fact several tens of microseconds were all that addition required and multiplication was about an order of magnitude slower. The clever memory hierarchy ideas of the Manchester group as well as the notion of a stored program would have impressed, if not surprised, Babbage.

The computer scientist of 1970 should give pause to notice the wealth of innovations which had been demonstrated by 1953. The multiprocessor with remote job entry at Bell Labs, the 8 μs core memory at MIT, the proposal of

microprogramming by Wilkes in 1951 -- any of these sound like current subjects.

Many topics had been sharply debated in the 1940's, including synchronous vs. asynchronous operation, bit serial vs. word parallel arithmetic, decimal vs. binary and fixed vs. floating point number representation. Several of these subjects are still debated -- or "settled" by providing both. It should be noted that asynchronous operation as pioneered by Stibitz and followed through the IAS machine, has largely disappeared. The extra control hardware and time required for "reply backs" between elementary operations became unreasonable as machine speeds increased. It is also interesting to note that while early machines (Zuse and Stibitz) had floating point hardware, it had largely disappeared by 1953 -- not to return for several years. von Neumann had been instrumental in this, arguing that proper scaling was easy if one sufficiently understood his problem; otherwise he shouldn't be computing in the first place. His argument contained one genuinely unfortunate flaw -- few users since have understood their calculations as von Neumann understood his. In any case, the "philosophy of machine design" papers written in the 1940's often read in part as if they had been written last year.

Not that all ideas had been proposed by 1953. Some inventions big and small that came after 1953 will close this chapter. The transistor and integrated circuit certainly provided the biggest technology changes and with them came remarkable system speedups. Memories with extra tag bits, indirect addressing, and phased or interleaved banks were to follow as was modern paging hardware. This led to complex multiprogramming and time sharing systems. Fancy terminals have greatly aided some users. Faster arithmetic algorithms and pipelined arithmetic units as well as program look ahead have contributed to faster computation. Stack machines have led to a variation in addressing as well as fast compilation. As we said at the beginning, things have become much more complicated and hardware and software organization have become deeply intertwined.

In 1953, software was in a rather simple and pure state. Symbolic assemblers were common and high level languages were being discussed. Fortunately no one had thought about software operating systems.

FOOTNOTES

1. [11] contains most of the available Babbage references. Also [5] contains a fair amount about Babbage.

2. Much of this material was obtained from [14].

3. [1], pages 359 and 367, contains accounts of the work of Zuse. In [9] on pages 508 and 650 one can read further details including an article by Zuse himself.

4. [2], pages 1 and 69, contains articles about the activities at Bell Labs. [12], page 41, is a very good discussion of the Bell Labs Machine. On page 91 of [12] there is an interesting philosophical paper by Stibitz.

5. [12] is a complete description of the machine.

6. ENIAC is discussed in [12] page 31 and [1] page 97.

7. EDSAC is discussed in [3], also in [18] by Wilkes who was the designer of EDSAC.

8. MADM and its preceding developments are discussed by the designers in [5] page 117.

9. A complete discussion of the IAS machine design is contained in [7].

10. See [16] for a discussion of Whirlwind.

Library/ Code

BIBLIOGRAPHY

DCL [1] "Mathematical Tables and Other Aids to Computation," 1947, The National Research Council, Vol. II, pp. 13-20.

DCL [2] "Mathematical Tables and Other Aids to Computation," 1949, The National Research Council, Vol. III, pp. 21-28.

DCL [3] "Mathematical Tables and Other Aids to Computation," 1950, The National Research Council, Vol. IV, pp. 29-32.

In the early days MTAC published many papers about computing machinery as well as computations. In each volume the section called Automatic Computing Machinery contains a number of interesting tidbits.

X6.30 B644a DCL [4] Booth, Andrew D., Booth, Kathleen H. V., 1953, Automatic Digital Computers, Academic Press, Inc.

Some historical references as well as a fine discussion of the state of the art at the time.

X2.3 [illegible]72f DCL [5] Bowden, B. V., 1953, Faster than Thought, Sir Isaac Pitman and Sons, Inc.

This is a collection of papers, many by Englishmen, many of whom were building their own machines which are described in the book. It is a very good source about the state of the art at the time.

X6.21 En33h DCL [6] Engineering Research Associates, 1950, High Speed Computing Devices, McGraw Hill Book Company, Inc.

This book discusses mainly American equipment and goes back to very early days, surveying desk calculators and so forth. Perhaps due to its early publication date several of the numbers quoted about electronic machines are wrong.

X6.0 G578r DCL [7] Goldstine, Herman H., Neumann, John von, and Burks, Arthur W., 1947, Report on the Mathematical and Logical Aspects of an Electronic Computing Instrument, Institute for Advanced Study.

This is the famous report. It describes what the IAS machine was supposed to do and was written before the machine was built.

Library/
Code

X2.3 .[illegible]c DCL

[8] Hartree, Douglas R., 1949, Calculating Instruments and Machines, The University of Illinois Press, University of Illinois at Urbana-Champaign, Champaign, Illinois.

X2.3 H675d DCL

[9] Hoffman, Walter, 1962, Digital Information Processors, John Wiley and Sons, Inc.

This book is a collection of papers, many by European people and in that respect is of special interest. Zuse's activities are reported twice, once by himself.

DCL

[10] Institute of Radio Engineers, Proceedings of IRE, October, 1953, Vol. 41, No. 10.

This was the first special edition dedicated to computers and contains a number of interesting early papers.

X2.2 B113c DCL

[11] Morrison, Philip, and Morrison, Emily, 1961, Charles Babbage and his Calculating Engines, Dover Publications, Inc.

Contains most of the available Babbage references, is an excellent source book and is very entertaining.

[illegible].84 Sy 68 ENG L

[12] "Proceedings of a Symposium on Large-Scale Digital Calculating Machinery," 1948, The Annals of the Computation Laboratory of Harvard University, Harvard University Press, Vol. XVI.

An earlier computer conference was held at MIT but this was the first big one with the proceedings published. It contains a large number of interesting papers which describe the state of the art at the end of 1946.

ONR DCL

[13] Office of Naval Research, July, 1950, A Survey of Large-Scale Digital Computers and Computer Projects.

This is a very nice snapshot of how things were in 1950. Some of the speeds are estimates since the equipment was not functional at the time.

X2.2 R616t DCL

[14] Rogers, William, 1969, Think, Stein and Day.

The first half of this book provides a very interesting historical introduction to IBM. The latter parts are less interesting and less well written. Unfortunately dates are frequently not given. This is an unauthorized biography of Watson. For the official party line read Belden.

Library/
Code

[15] Rosen, Saul, March, 1969, "Electronic Computers: A Historical Survey," Computing Surveys, Vol. 1, No. 1, pp. 7-36.

[16] Serrell, R., Astrahan, M. M., Patterson, G. W., Pyne, I. G., May, 1962, "The Evolution of Computing Machines and Systems," Proceedings of the IRE, pp. 1039-1058.

This is a fairly comprehensive paper which tells a number of details about early machines as well as ones in the 50's. It contains a large bibliography.

510.84
W42s
ENG L

[17] Weik, Martin H., June, 1957, A Second Survey of Domestic Electronic Digital Computing Systems, Report No. 1010, Ballistic Research Laboratories.

Chapter 2

Processor Design

2.1.1 Overall Design Questions

A computer system designer must solve one problem in many forms and at many levels: What is the least expensive way to provide a given function to the user of the machine? The function may be a low level detail or it may be an overall system characteristic. The function may be of an entirely logical nature or it may involve the speed with which something is accomplished. The function may be stated in terms of a user's problem or in terms of the computer itself. The possible functional specifications are endless so let us turn to the question of cost. Except in rare cases the designer must do things as cheaply as possible, subject to the functional constraints specified for the system. Cost savings may be made by using less expensive parts and by reducing the number of parts. This often involves a number of trade offs, particularly because cheaper parts are usually slower and judiciously adding more parts generally speeds things up. Since overall costs are usually all that matter, cost and speed trade offs may be made between various units of the overall system. In any case, it is usually bad practice to include features merely because they are exotic (although some machines may appear to contradict this). Functions should be of justifiable use to the customer and the overall cost should be as low as possible.

Given these rather obvious remarks, the question remains: how does one go about designing a computer system? We shall attempt to answer that question in a fairly general way by discussing a number of computer functions

and how they are interrelated. We shall attempt to discuss general principles and then relate them to some real machines. Our overall approach will be from the inside out; we shall start with the arithmetic unit, primary memory and the control unit. These will be followed by overall system discussions.

2.1.2 Arithmetic and Logical Unit

If we regard consideration of the arithmetic and logical unit as the first design problem, then a number of decisions at this level will be reflected throughout the system. In practice the various parts of the machine affected would be considered simultaneously. Here we shall restrict our attention to one part at a time.

In terms of cost and speed we must concern ourselves with the kind of circuits used as well as how many parts are required. Circuit parameters of interest are switching speed, fan-in and fan-out limitations, power dissipation, noise immunity, reliability and cost. Interrelations between parts required by the functions desired must be compatible with such things as layout on boards with respect to number of wiring levels, cross talk, cooling, and repairability.

The user requirements specified may be rather vague. Problems in different contexts tend to place a variety of demands on the arithmetic unit--some problems requiring one thing and others something else. In any case, large numerical computation tends to be the most severe burden for the arithmetic unit so we shall discuss some details of this.

First, one must decide which operations are to be performed. Addition, subtraction, multiplication and division are typical, although there are many variations on these. In the future, more complicated functions

may be built into machines e.g. trigonometric functions, log, exp, or n-ary summation. Initially we shall restrict our attention to addition.

First we must decide if we are going to actually add or just do a table lookup to get the result. This latter strategy has sometimes been employed (cf. IBM 1620) in slow, small machines; large, high speed machines usually have the ability to compute the sum of two numbers using some kind of sequential logic. The form of the numbers turns out to be quite important in a number of respects. By form we mean the number of digits, the number system and whether or not some kind of explicit exponent is used.

The number of digits dictates the word length of memory as well as the arithmetic unit and its registers. This can be quite important in terms of overall system cost. Users can often give estimates of the required word length in terms of the maximum round-off error tolerable for certain calculations. It is usually desirable to make the word length a multiple of the character size (byte) used in the computer system and this has usually been either 6 or 8 bits in binary machines. In the early days some internal decimal machines were built (e.g. IBM 650) although these are quite rare now and we shall restrict our attention to internal binary machines. For numerical calculating the range of 32 to 64 bits has been common. The possibility exists of choosing a standard word length and then providing arithmetic operations on double or half words. This has often been done to try to satisfy a wider class of users.

Choosing a word length typically requires choosing both an exponent and fraction size in most modern machines used for numerical computation. von Neumann argued against floating point hardware and built a 40-bit fixed

point machine. Later, most companies built floating point machines with 40 or fewer total bits to the chagrin of many numerical uses. Typically, from 6 to 16 bits of exponent are provided. As more complex numerical computations are performed, users are less happy with normalized arithmetic. Several unnormalized or significance arithmetic schemes have been proposed and implemented.

Finally, the number system chosen can greatly influence the speed and gate count of the arithmetic unit. The well-known polynomial representation is commonly used, although a redundant form of this has quite desirable properties. Also the residue number system has interesting properties which are useful in a theoretical way as well as for some applications.

We shall attempt to discuss several of these issues and to contrast some of them with others. The reader should be forwarned that no pat answers are forthcoming. Some fairly detailed results are available but the choices between alternatives must be dictated by individual design requirements.

2.3 Number Systems

2.3.1 Polynomial Numbers

Numbers may be coded in a variety of ways. For example, the polynomial number

$$p(r,k) = \sum_{i=0}^{k-1} d_i r^i, \qquad 0 \le d_i < r,$$

represents a k digit integer with radix r. If r = 10 we have a decimal number, e.g.

$$p(10,4) = 3 \times 10^3 + 7 \times 10^2 + 1 \times 10^1 + 9 \times 10^0 = (3719)_{10}.$$

We shall use the radix subscript notation when necessary to avoid ambiguity. As another example if r = 2 we have a binary number, e.g.

$$p(2,3) = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (101)_2 = (5)_{10}.$$

Finally, if r = 16 we have a hexadecimal number, e.g.

$$p(16,3) = 1 \times 16^2 + 9 \times 16^1 + 15_{10} \times 16^0$$

To avoid confusion, the substitutions $A = 10_{10}$, $B = 11_{10}$, $C = 12_{10}$, $D = 13_{10}$, $E = 14_{10}$, $F = 15_{10}$ are often used. Thus for our example, $(19F)_{16} = (415)_{10} = (0001,1001,1111)_2$. Since four bits (binary digits) can be used to represent the 16 possible coefficients required in a hexadecimal number, an easy conversion from binary to hexadecimal may be made. In the last example this can be seen by simply reading off groups of four bits in the binary form and rewriting them as hexadecimal coefficients of appropriate powers of 16.

Because of the ease of building physical devices with two stable internal states, a radix of some power of two is often chosen for computer number representation. Binary, octal, and hexadecimal are common choices.

The above numbers were all integers, but real numbers are easy to write as polynomials by letting the summation range over negative as well as non-negative values. Thus

$$p(r,k,j) = \sum_{i=-j}^{k-1} d_i r^i, \quad 0 \leq d_i < r$$

is a j + k digit real number, base r.

Examples of decimal and binary real numbers are

$$p(10,3,2) = 9 \times 10^2 + 0 \times 10^1 + 4 \times 10^0 + 7 \times 10^{-1} + 3 \times 10^{-2}$$

$$= 904.73$$

and

$$p(2,2,3) = 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = (10.101)_2$$

$$= (2.625)_{10}.$$

Note that they have j digits after the decimal and binary point, respectively.

2.3.2 Signed Digit Numbers

The polynomial numbers of the last section used non-negative digits, only. There are good reasons to allow each digit to have its own sign, as we shall see in a later discussion of arithmetic operations. Many possible definitions of signed digit numbers could be given, but we choose the following.

A signed digit polynomial number is given by

$$sp(r,j,k,\max|d_i|) = \sum_{i=-j}^{k-1} d_i r^i, \quad -\max|d_i| \leq d_i \leq \max|d_i|$$

where $r > 2$,

and

$\frac{r+1}{2} \leq \max|d_i| \leq r - 1$, if r is odd,

$\frac{r}{2} + 1 \leq \max|d_i| \leq r - 1$, if r is even. For example, if we choose $r = 10$ and let $\max|d_i| = \frac{r}{2} + 1 = 6$, we have $sp(10,2,3,6) = \sum_{i=-2}^{2} d_i 10^i, \quad -6 \leq d_i \leq 6 =$

$$3\times10^2 + (-6)\times10^1 + (-3)\times10^0 + 6\times10^{-1} + 2\times10^{-2} = 300 - 60 - 3 + .6 + .2 = 237.62.$$

The sign of such a number is the sign of the highest power nonzero digit, and negative numbers are formed by changing the sign of each digit. Thus no explicit sign is required. Note that the algebraic value of a signed digit polynomial number is zero if and only if all $d_i = 0$.

2.3.3 Residue Numbers

The residue number system uses an implicit definition for each number rather than the explicit polynomials of the previous sections. Before discussing this system we will review a few definitions.

We say that a is congruent to r, modulo m, and write

$a \equiv r \pmod{m}$

if for integers a, m and r there is an integer k such that

$$a = r + mk.$$

In this congruence relation, r is called the residue and m the modulus of the number a. We shall concern ourselves only with the least positive residue $r_{\ell p}$, defined by $0 \leq r_{\ell p} < m$. Thus if m = 2 and a = 5 we have

$$5 \equiv 5 \pmod 2, \quad k = 0$$

$$5 \equiv 3 \pmod 2, \quad k = 1$$

$$5 \equiv 1 \pmod 2, \quad k = 2$$

$$5 \equiv -1 \pmod 2, \quad k = 3$$

and $r_{\ell p} = 1$. Clearly $r_{\ell p}$ is unique for any a and m. Finally we recall that two integers are relatively prime if their greatest common divisor is 1.

The residue number system represents an integer as a concatenation of the least positive residues of that integer with respect to a set of relatively prime moduli. For example, let 2 and 3 be the moduli, then we can represent the integers $0,\ldots,5$ as follows:

N	$r_{\ell p} \pmod 2$	$r_{\ell p} \pmod 3$	Residue number N
0	0	0	00
1	1	1	11
2	0	2	02
3	1	0	10
4	0	1	01
5	1	2	12

In general, if we are using k moduli $m_1,\ldots,m_k$, then we can represent $\prod_{i=1}^{k} m_i$ distinct numbers in the residue number system. If we had chosen moduli that were not relatively prime this would not be true.

2.4 Machine Representation of Numbers

2.4.1 Precision and Machine Radix

A great variety of formats have been proposed and used to store and operate on numbers inside computers. If machine numbers have a word length of w digits (excluding sign), we say that integers may be represented with w digits of precision. It is important to distinguish "precision" in this sense from the meanings of such words as "accuracy" or "significance". Thus numbers may be represented to 20 digits of precision. But if the measuring device from which they were obtained was only accurate to 3 digits, only 3 digits of the 20 are accurate. The other 17 may have been "extrapolated" by a meter reader.

We now consider the meaning of the word "radix" in computer terms. A machine is built of elements, each having a number of different internal states. Let us say that each element can represent v values. Almost all current machines are built using physical devices with two stable states. These may be assembled into v value elements with v = 2 or with some other value of v, say v = 10. Thus, while most machines are made from two state physical devices, a number of current machines have binary (v=2) as well as decimal (v=10) arithmetic capabilities. When we wish to clearly denote a machine radix in terms of v value elements we shall write r_v instead of r.

2.4.2 Fixed and Floating Point Numbers

Integers are stored in most binary machines as a sign bit and w digits of integer. Thus, the range of w digit integers in a radix r_v machine is

$$-r_v^{\,w} < i(r_v, w) < r_v^{\,w}$$

with both plus and minus zero included.

Numbers in this form need not be regarded as integers. Obviously the radix point may be assumed to be anywhere in the number. Or it may be assumed

to be a fixed number of zeros to the right or left of the word. Wherever it is assumed to be, it is fixed by the programmer (as in slide rule computation). This number representation in computers is thus called either integer or fixed point form. A fixed point number which is not an integer clearly has the range

$$-r_v^{w+s} < fi(r_v, w, s) < r_v^{w+s}$$

where s (a signed integer) is a scale factor assumed by the user.

Since the late 1950's most big machines have provided arithmetic units which operate on integer as well as floating point or real number forms. Such forms usually represent a signed fraction and a signed exponent. Assume two signs plus $w = e + f$ digits are used, where e is the number of digits of the exponent and f is the number of digits of the fraction. Suppose we have a machine with radix r_v. An exponent e_1 of e digits and a fraction f_1 of f digits may take the forms

$$e_1 = fi(r_v, e, s_1)$$

$$f_1 = fi(r_v, f, s_2).$$

In most machines $s_1 = 0$ so exponents are regarded as integers and $s_2 = -f$ since the radix point is assumed to be at the left end of each word.

Thus $e_1 = i(r_v, e)$

and $f_1 = fi(r_v, f, -f)$.

are assumptions that we shall make unless otherwise noted in our subsequent discussion.

Up to this point we have discussed forms of the fraction and exponent but we have not mentioned the base to which the exponent is raised. This is often referred to as the radix of machine numbers by users and we shall denote it by r_b. In many machines $r_b = r_v = v$, but this is not always so. It is also popular to choose $r_b = r_v^k$ for some small integer k. For example in the IBM 360 floating point

operations $v = r_v = 2$, but $r_b = 2^4 = 16$, and it is referred to as a hexadecimal floating point machine. If $r_b = r_v^k$ then our distinction between radicies may seem pedantic because collections of k digits in r_v may be regarded as digits in r_b (recall our earlier discussion of conversion from binary to hexadecimal).

Now we can express floating point machine numbers (with $s_1 = 0$ and $s_2 = f$ as above) as

$$r_v^{-f} \times r_b^{-(r_v^e - 1)} \leq f\ell(r_v, r_b, e, f) < 1 \times r_b^{(r_v^e - 1)}$$

or $$f\ell(r_v, r_b, e, f) = \pm 0$$

or $$-1 \times r_b^{(r_v^e - 1)} \leq f\ell(r_v, r_b, e, f) \leq -r_v^{-f} \times r_b^{-(r_v^e - 1)}$$

Note that the intervals represented contain only a finite number of reals. We shall adopt the notation that if $r_v = r_b$, both will be denoted by r. For example if r = 2 we have

$$2^{(1-f-2^e)} \leq f\ell(2, e, f) < 2^{(2^e - 1)}$$

or $$f\ell(2, e, f) = \pm 0$$

or $$-2^{(2^e - 1)} < f\ell(2, e, f) \leq -2^{(1-f-2^e)}.$$

We say that the precision of a floating point number is determined by f and its range is determined by r_b and e.

2.4.3 Normalized and Unnormalized Numbers

The computer stored form of a floating point number is not necessarily unique. Thus we have

$$fl(2,e,f) = f_1 \times 2^{e_1}$$

$$= 2^{-a} f_1 \times 2^{a+e_1}$$

$$= fl(2,e,f)$$

if $\|a+e_1\| \leq e, \|f_1\| \leq f$, and $\|2^{-a} f_1\| \leq f$, where $\|x\|$ is the number of digits in x.

For example, if e = 3 and f = 5

$$fl(10,3,5) = .03210 \times 10^{003}$$

$$= .32100 \times 10^{002}$$

$$= .00321 \times 10^{004}.$$

To make the stored form of floating point numbers unique, some standard form may be chosen. Very often this is the normalized form of a number which we shall denote by $nfl(r,e,f)$. This means that if the number being represented is non zero, the first digit to the right of the radix point is non zero. By properly adjusting the exponent, any non-zero floating point number can be normalized as we did above using an adjustment factor a. If the radix point is assumed to be at the left end of the fraction, then clearly we obtain maximum precision for fractions using normalized forms.

It is not always the case that users want to do normalized floating point calculations. Hardware and software aids for performing unnormalized or significance arithmetic are often provided. In this case some adjustment a is used so that the normalized number is shifted a digits to the right. Roughly speaking such a number may be said to have a significance of f-a digits. The point of providing significance arithmetic is that often the user starts out with numbers of less significance than the f available on his machine. Also the significance of all his numbers is usually not the same. In such cases it may be very misleading to compute using the full f digits of the machine and to deliver an f digit result. Rather, the significance of

the result should be expressed as a function of the significance of the input data. Machines with significance arithmetic features provide proper adjustment for each operation.

2.4.4 Multiple Precision Representation

Whatever word length is provided by machine designers will prove inadequate for some users. Thus multiple precision hardware or software is often built. If n word precision is provided, then n memory locations must be fetched per operand. In multiple precision floating point operations it may seem desirable to use an exponent of the same size as that used for single precision. However it is often the case that only an f digit arithmetic unit is available. Thus, each word in the multiple precision representation is used as an f, e pair. The exponents are then adjusted to reflect the position of each component in the longer number.

2.5.1 Floating Point Arithmetic Definitions

The following definitions should be intuitively clear.

Let $f\ell_1(r,e,f) = f_1 \times r^{e_1}$ and $f\ell_2(r,e,f) = f_2 \times r^{e_2}$, then

$$f\ell_1(r,e,f) \pm f\ell_2(r,e,f) = \begin{cases} (f_1 \pm f_2 \times r^{-(e_1-e_2)}) \times r^{e_1}, & \text{if } e_1 \geq e_2 \\ (f_1 \times r^{-(e_2-e_1)} \pm f_2) \times r^{e_2}, & \text{if } e_1 < e_2 \end{cases}$$

$$f\ell_1(r,e,f) * f\ell_2(r,e,f) = f_1 \times f_2 \times r^{(e_1+e_2)}$$

$$f\ell_1(r,e,f) / f\ell_2(r,e,f) = (f_1/f_2) \times r^{(e_1-e_2)}.$$

A number of difficulties may arise in terms of machine representation of the results of these arithmetic operations. In the case of add or subtract the exponent of the result is the same as one of the original exponents but one of arguments must be shifted (adjusted) a distance equal to the magnitude of the difference of the exponents. This can cause digits to flow off the right end of a number or machine register and we shall call it fraction underflow. In case both fractions have a high order 1, 1 is progagated off the left end of the number or machine register and we shall call this fraction overflow.

In the case of multiplication and division the exponents are added and substracted, respectively. In case a positive exponent gets too large we shall refer to it exponent overflow, and exponent underflow will mean that a negative exponent exceeds the e digits in magnitude.

These various kinds of exceptions should always be used to trigger an alarm to the user. Provisions should be provided to allow him to take appropriate action. With most modern compilers and operating systems, the actions can be taken automatically. For example, certain values should be saved for the user to study and the job may or may not be continued.

2.5.2 Machine Addition

First let us restrict our attention to addition of nonnegative integers. For example

$$n_1(r,k) + n_2(r,k) = \sum_{i=0}^{k-1} d_{1i} r^i + \sum_{i=0}^{k-1} d_{2i} r^i$$

$$= \sum_{i=0}^{k-1} (d_{1i}+d_{2i}) r^i.$$

Since each digit is required to be less than r, $d_{1i} + d_{2i}$ must be regarded as a pair of digits, commonly called a sum and carry digit, so $d_{1i} + d_{2i} = rc_{i+1} + s_i$ where $c_{i+1} = 1$ if $d_{1i} + d_{2i} \geq r$, otherwise $c_{i+1} = 0$.

It will become important below to decide precisely what we mean by "the addition of two numbers". Is it sufficient to generate only the c_{i+1} and s_i digits for all i? Or must we worry about propagating the carry across the result? Note that

$$\begin{array}{r} 316 \\ +\ 253 \\ \hline 569 \end{array}$$

can be evaluated with all zero carry bits whereas

$$\begin{array}{r} 316 \\ +\ 694 \\ \hline 1010 \end{array}$$

requires a carry to propagate across all positions. Apparently the latter process is much slower than the former. On the other hand, the generation of c_{i+1} and s_i in each position would seem to take the same time for each position and if these could be saved for the next addition perhaps an overall time saving would be possible. We shall return to these questions later.

Note that the second sum above overflowed in the sense that two three digit numbers led to a four-digit one. With a fixed word length machine this would require the raising of some kind of alarm to cause appropriate action to be taken.

Now let us consider the addition of nonnegative floating point numbers. For example

$$fl_1(2,2,3) + fl_2(2,2,3)$$

$$= 101. \times 2^{10} + 011. \times 2^{01} \quad (=5_{10} \times 4_{10} + 3_{10} \times 2_{10} = 26_{10})$$

$$= 101. \times 2^{10} + 001.1 \times 2^{10}$$

$$= 110.1 \times 2^{10} = 6.5_{10} \times 4_{10} = 26_{10}.$$

This addition process required an extra step before the addition to equalize the exponents and align the binary points of the two arguments. In particular, the smaller exponent was set equal to the larger one and the fractional part was properly shifted to compensate. Notice that the process underflowed the three digits allowed for the fraction part of the floating point numbers.

2.5.3 Normalized Floating Point Addition

We will now consider the process of floating point addition assuming normalized arguments. Let

$$nf\ell_1(2,3,5) = 10100 \times 2^{011}$$

$$nf\ell_2(2,3,5) = 10100 \times 2^{001}$$

The addition process requires equal exponents so we must first align the fractions and adjust the smaller exponent. Thus we have

$$nf\ell_1 + nf\ell_2 = 10100 \times 2^{011} + 00101 \times 2^{011}$$

$$= 11001 \times 2^{011} = nf\ell_3.$$

In general we may overflow the left end (by at most one digit) and this requires a post addition adjustment or renormalization step. This may cause a digit to drop off the right end of the word. If we have the choice of losing a digit at the right or left ends, clearly we must choose to drop the low order digit (otherwise the result would be nonsense). There is a choice of simply dropping the lost digit called truncation, or adding 1/2 to the highest order digit about to be dropped called rounding, and generally rounding is preferable. The errors introduced by these processes are called truncation error and round off error, respectively.

While we have discussed the error introduced by post addition normalization, the preaddition alignment may also introduce error. For example, let

$$nf\ell_1(2,3,5) = 11100 \times 2^{011}$$

$$nf\ell_2(2,3,5) = 10111 \times 2^{001}$$

Then

$$\begin{aligned} 11100 \times 2^{011} &= nf\ell_1 \\ + \underline{0010111 \times 2^{011}} &= nf\ell_2 \\ \underline{1}0000\underline{11} \times 2^{011} &= nf\ell_3 \end{aligned}$$

The fraction underflow digits underlined at the right are somewhat in doubt, being the sum of the low order digits given for $nf\ell_2$ and assumed low order zeros for $nf\ell_1$. The underlined digit at the left is the fraction overflow discussed above. To finish the addition we must shift right to renormalize the fraction and subtract one from the exponent to adjust it. Finally we shall round by adding an appropriate 1. Thus we obtain

$$\begin{aligned} & 1000011 \times 2^{011} \\ & 1000011 \times 2^{100} && \text{shift and adjust} \\ + \quad & 1 && \text{round} \\ & 10001 \times 2^{100} && \text{result} \end{aligned}$$

In decimal notation

$$nf\ell_1 = 7,\ nf\ell_2 = 1.4375, \text{ and } nf\ell_3 = 8.5$$

and our machine addition process has introduced a round-off error of + .0625. Note that if we had truncated instead of rounding, $nf\ell_3 = 8$ and the truncation error would be - .4375.

Generally error may be expressed in absolute or relative terms. Both of our above examples were absolute error, ϵ_a, the actual value of the error. Perhaps of more interest is the relative error, ϵ_r, expressed as the ratio of absolute error to the correct value (or approximately to the computed value). Thus for the above example, the relative error due to round off is

$$\epsilon_{rr} = \frac{\epsilon_a}{nf\ell_1 + nf\ell_2} = \frac{.0625}{8.4375} = .0074$$

while the relative error due to truncation is

$$\epsilon_{rt} = \frac{\epsilon_a}{nf\ell_1 + nf\ell_2} = \frac{.4375}{8.4375} = .052.$$

2.5.4 Floating Point Multiplication

We shall briefly consider floating point multiplication, emphasizing the steps before and after the actual multiplication. Assuming normalized numbers, let

$$nf\ell_1 \times nf\ell_2 = (f_1 \times r^{e_1}) \times (f_2 \times r^{e_2})$$

$$= f_1 \times f_2 \times r^{e_1+e_2} = nf\ell_3.$$

Normalized fractions may be multiplied with no possibility of overflow. At the same time the exponents may be added and this process overflows if $|e_1+e_2| > e$. In this case the user should be notified that he has exceeded the machine's capacity. The product of two f digit fractions will generally be of length 2f and this means that extra register length should be provided in the arithmetic unit. The user may want to save both the high and low order bits of the product. More likely he will simply want to save a rounded single length result. Note that the rounding process must be followed by a renormalization and exponent adjustment step.

The time consuming process of multiplying the fractions may be done in various ways. Typically some kind of repeated addition loop is executed. Thus one operand is shifted and added to itself under the control of the other operand.

2.6 Bit Level Design Options

To this point we have discussed a number of elementary ideas about computer numbers and arithmetic. With this as background we shall turn our attention to some overall questions about computer arithmetic. To make the discussion tractable we shall limit ourselves mainly to floating point addition. As we mentioned earlier the design of a machine must be regarded from the user's point of view as well as the designer's. The designer wants an "inexpensive" unit. Users have a great variety of performance desires. We shall discuss performance and cost in terms of a number of parameters. Our objective is to give the reader some ideas of an overall nature rather than to discuss specific designs. Thus we shall present analyses of: shifting as a function of radix; precision and accuracy as functions of radix; roundoff as a function of word length and number of arguments; overall speed as a function of number representation, hardware characteristics, and word lengths.

2.6.1 Optimal Choice of r_v

We shall consider several aspects of the choice of radix. First is the optimization of r_v in machine structure terms.

Assume that a variety of physical devices are available at various costs. Their speeds may be assumed to be equal or cost may be written as a function of speed. In any case, assume that an entire machine is made of various "black boxes" which are radix r devices. Also assume that the cost per bit of such devices can be expressed as (here $r_v = r$)

$$c_{bit} = \alpha r^{\beta}.$$

We must qualify this assumption because memory and processor costs per bit are usually quite different due to different technologies. This could be reflected in α. Furthermore, it is probably true that if we simulated radix r "black boxes" using radix 2 hardware, the cost per bit of memory and processor would have quite different β values. If the assumption is valid for some part of a machine, we proceed as follows. Let some number $N = r^n$ be represented by n bits in various radices. Then

$$\log_e N = n \log_e r$$

and the cost of N (storage, processing, etc.) may be expressed as

$$c_N = \alpha r^{\beta} n$$

$$= \alpha r^{\beta} \frac{\log_e N}{\log_e r}.$$

To minimize this cost with respect to r

$$\frac{dc_N}{dr} = \alpha \log_e N \frac{\beta r^{\beta-1} \log_e r - r^{\beta-1}}{(\log_e r)^2} = 0.$$

Thus

$$\beta \log_e r = 1$$

or $$r = e^{1/\beta}.$$

Since the second derivative is positive we have an expression for a minimum cost radix. For example, $\beta = 1$ implies that the bit cost is proportional to the radix value in a linear way. In this case $r = e$ and binary or ternary arithmetic are nearly optimal. In fact a binary radix is optimal in case $\beta = \frac{1}{\log_e 2} = 1.44$.

2.6.2 Choice of r_b

Next we consider the choice of radix in determining the range of floating point numbers. Given some fixed hardware representation for numbers, the e digit exponent and radix r_b allow f digit fractions to be scaled up and down. r_b is built into the logic of the arithmetic unit, while e and f determine the machine's word length. The choice of r_b affects the precision and range of normalized floating point numbers. The following table contains some illustrative examples (assuming $r_v = 2$).

Number	$r_b = 2$		$r_b = 16$	
	f	e	f	e
1/16	.1	-011	.0001	000
1/8	.1	-010	.0010	000
1/4	.1	-001	.0100	000
1/2	.1	000	.1000	000
1	.1	001	.0001	001
2	.1	010	.0010	001
4	.1	011	.0100	001

It is immediately clear that fractional parts of hexadecimal numbers may have leading zeros and still be normalized. On the other hand, when a shift is necessary, four binary digits are lost per hexadecimal digit. Thus, we incur a larger loss of precision per shift with hexadecimal. It should also be clear that fewer different values of exponent are required for the same range using larger r_b. Thus another question is, what r_b is most efficient of total word length use? Another obvious question given

N digits is, what sizes of e and f should be used? We shall deal with each of these matters below.

2.6.2.1 Addition Shift Distances in Practice

First we consider the loss of precision due to shifting using higher r_b values. Specifically we shall discuss hexadecimal and binary. D. W. Sweeney [ ] has analyzed floating point addition in a number of scientific codes. By tracing about 10 million instruction executions, he observed that an overall average of about 10% of the instructions executed were floating point additions. We shall reproduce only a few of his findings. In particular we are interested in preaddition alignment shifts and post addition normalization shifts. The values in the table represent the number of shifts of a particular distance expressed as a percentage of all cases measured. The numbers added were not necessarily of like sign and a few unnormalized operations were included.

	Shift Distance	$r_b = 2$	Shift Distance	$r_b = 16$
alignment	0	32.64	0	47.32
	1-4	34.61	1	26.02
normalization	overflow	19.65	overflow	5.5
	0	59.38	0	82.35
	1-4	14.51	1	7.24

As expected, we observe more zero shift cases with higher r_b. In fact, normalization shifts for hexadecimal numbers only occur about 18% of the time. Comparing the sum of the alignment shifts from 0 to 4 for binary and from 0 to 1 for hexadecimal, slightly favors hexadecimal. Of course the binary shifts occur in increments of one bit of precision loss. Similar sums for normalization are almost equal.

Viewed another way, we can observe that the sum of alignment percentage for distance 0 and 1 with base 2 are slightly less than the percentage of distance 0 shifts for base 16. Similarly the distance 0 and 1 normalization shift percentages in binary are slightly more than the distance 0 shifts for base 16.

2.6.2.2 Distribution and Number of Values as a function of r_b

We first study the number of different values representable using various bases and the distribution of these values. Notice that when floating point numbers with $r_b = 2$ are required to be in normalized form, only half of the possible values representable with f bits are used (just those with a leading 1). When one leading zero is allowed ($r_b = 4$) then 50% more values are representable and so on.

Given e and f bits of exponent and fraction, respectively, there are 2^e different exponents and 2^{f-1} different normalized fractions representable. (We are assuming here that $r_b = r_v = 2$.) Thus the total number of representable values is 2^{e+f-1}. Since the largest fraction representable is approximately 1, the largest binary number representable is approximately 2^{2^e}.

Now if $r_b = \beta = 2^p$, numbers have the form $f_1 \times \beta^{e_1}$. To estimate the number of values less than the maximum binary number ($r_b = 2$) we observe that for some k (assuming $f_1 \approx 1$)

$$\beta^k \approx 2^{2^e}.$$

Thus $k \log_2 \beta \approx 2^e$. Now the number of values less than β^k is approximately

$$(2^{f-1} + 2^{f-2} + \ldots + 2^{f-\log_2 \beta})(k+1).$$

Thus we can write

$$\frac{\text{number of } r_b = \beta \text{ values less than } 2^{2^e}}{\text{number of } r_b = 2 \text{ values less than } 2^{2^e}} = \text{representation ratio}$$

$$= \frac{2^{f-1}(1 + 2^{-1} + \ldots + 2^{-(\log_2 \beta - 1)})(k+1)}{2^{f-1} 2^e}$$

$$= \frac{2^{f-1}(1 + 2^{-1} + 2^{-2} + \ldots + 2^{-(\log_2 \beta - 1)})(k+1)}{2^{f-1}(\log_2 \beta)k} =$$

$$= \left(\frac{k+1}{k}\right)\left(\frac{1 + 2^{-1} + 2^{-2} + \ldots + 2^{-(\log_2 \beta - 1)}}{\log_2 \beta}\right).$$

If $\beta = 16$ and $e = 8$, then we have $k = \frac{2^8}{\log_2 16} = 2^6$

and representation ratio $\approx \frac{1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}}{4} \approx .47$

By a similar analysis it may be shown that there are about 1.88 times as many hexadecimal values as binary values representable using fixed e and f. Thus we conclude that about half of the hexadecimal values are in the range of the binary values and about $\frac{3}{2}$ are outside the binary range.

2.6.2.3 r_b, f, and accuracy

The accuracy with which some form of floating point numbers represents the real numbers may be studied by examining the intervals

between the floating point numbers. Thus, if $f\ell_i$ and $f\ell_{i+1}$ denote a pair of adjacent representable floating point numbers, then

$$\frac{f\ell_{i+1} - f\ell_i}{f\ell_i}$$

is a relative interval measure of accuracy. It turns out that this has r^{i-f} as its maximum value. In [9], for this as well as other accuracy measures, the question of floating point number representations is studied. It is shown that for fixed N = e + f, the choice of $r_b = r_v$ always provides as much accuracy and more exponent range than some $r_b = r_v^k$. In other words, while f may be made larger at the expense of e with $r_b = r_v^k$, the tradeoff with accuracy is not a good one.

That this is true is not hard to see by studying the exponent range, E, as a function of f and i, where $i = \log_{r_v} \beta$, $f\ell = f\beta^e$, and N = e + f. Thus we have

$$E(f,i) = i(r^{N-f}-1).$$

Assuming that $f \geq i$, for the same accuracy we study the ratio of exponent ranges of an $r_b = r_v$ number to an $r_b = r_v^i$ number:

$$\frac{E(f-i+1,1)}{E(f,i)} = \frac{1(r^{N-f+i-1}-1)}{i(r^{N-f}-1)} = \frac{r^{i-1}}{i}\left(\frac{r^{N-f}-r^{1-i}}{r^{N-f}-1}\right) > \frac{r^{i-1}}{i} \geq 1$$

2.6.3 Rounding and Truncation

Assume we have a computed floating point number with f digits of fraction to be retained plus some low order digits which must be disposed of. The lowest order digit of the f digits represents r^{-f}, so the digits to the

right represent a quantity whose value is less than r^{-f}. Regardless of the process used to dispose of these low order digits, the error on each step is less than r^{-f}. Intuitively it seems desirable to minimize this error on each step. If it is necessary to introduce a positive error on some steps and a negative error on other steps, it would also seem intuitively desirable to try to minimize the algebraic sum of these errors; that is to minimize the bias in disposing of the extra digits.

First we consider the error due to simply dropping and forgetting the extra digits; this is usually called truncation error. With an f digit fraction, the truncation error ϵ_t is

$$0 \leq \epsilon_t < r^{-f}.$$

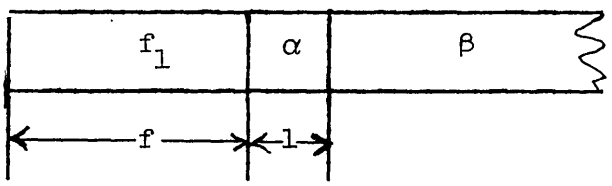
The bias introduced by truncation is the sum of these errors over many steps. If the average error is one half of the maximum then the bias over n additions is

$$b_t = \frac{nr^{-f}}{2}.$$

An intuitively better procedure is rounding the f bits to be saved using the high order bit of those to be disposed of. The error so introduced is usually called round off error. In this case the error ϵ_r is

$$0 \leq \epsilon_r \leq \frac{1}{2} r^{-f}$$

or at most one half that of ϵ_t. This may be seen by considering a floating point number



where $\|f_1\| = f$. To round f_1 we add $\frac{r}{2}$ to it in the α position. If $\alpha\beta = 0$ then clearly no carry is generated and by dropping $\alpha\beta$, no error is introduced. If $\alpha < \frac{r}{2}$ and β is as large as possible, then no carry is propagated and β is dropped, introducing an error of at most $\frac{1}{2} r^{-f}$. If $\alpha \geq \frac{r}{2}$ then a one carries to f_1 thus adding r^{-f} to f_1. In this case the smallest that $\alpha\beta$ can be is $\frac{r}{2} \cdot r^{-(f+1)}$. Thus we introduce a maximum error equal to the amount added to f_1 minus the least amount lost by dropping $\alpha\beta$, i.e.

$$\epsilon_r \leq r^{-f} - \frac{r}{2} \cdot r^{-(f+1)}$$

$$\leq r^{-f} - \frac{r^{-f}}{2} = \frac{r^{-f}}{2}.$$

In the case of $r = 2$, the error introduced by this process is $-\beta$ if $\alpha = 0$ and $2^{-f} - \frac{2^{-f}}{2} - \beta$ if $\alpha = 1$. If $\beta = 0$ then the bias is $0 + \frac{2^{-f}}{2} - 0 = 2^{-(f+1)}$ If $\beta \neq 0$, then for each β_i we can find a β_j such that $\beta_i - (2^{-(f+1)} - \beta_j) = 0$. Thus, if we assume that all values of β are equally likely, the total bias is just $2^{-(f+1)}$.

Let us consider the possibility of reducing the round off bias to zero. Consider the following table

Number Presented			Rounded Result	$2^f \times$ error	
2^{-f}	$2^{-(f+1)}$	$2^{-(f+2)}$	2^{-f}		
X	0	0	X	0	bias = $\frac{1}{2}$
X	0	1	X	- 1/4	
X	1	0	X + 1	1 - 1/2 = 1/2	
X	1	1	X + 1	1 - 3/4 = 1/4	

Here the second and fourth numbers may be paired to introduce zero bias (cf. discussion above) and here (as in general) the third case (10) introduces the bias. If it were possible to detect the case $\beta = 0$, $\alpha = 1$ and round this in only half of the cases a zero bias rounding procedure would exist. For example, some random bit could be used to take the choice in the $\alpha = 1$, $\beta = 0$ case.

A scheme which is easier to implement than rounding and not much more difficult to implement than truncation is the jamming of a 1 into the last bit position. The error and bias of this are between those of rounding and truncation.

2.7 Addition Speed vs. Gate Count vs. Number Representatives

A simple way of adding two numbers is to add two digits at a time, generate sum and carry digits and go on to the next pair of digits. Such schemes are generally referred to as serial by digit and were often used in early machines [10]. To speed up the addition process one naturally considers adding several digits at once. In general this leads to questions about propagating carry digits. Using the residue or signed digit representation, carry propagation is not a problem as we saw earlier, but these are both "unusual" number systems and we shall deal with them later. Another question that comes up is the possibility of adding n numbers together at once and considering the speed and cost of this process compared with adding two numbers.

By the early 1960's a number of fast parallel addition algorithms were in common use. A number of alternatives for binary addition are compared in [9] by Sklansky and summarized in Figures 11, 12, 14, 15, and 16 and Table I there. Sklansky shows an n bit serial adder with 7 gates and 4n gate delay time steps. He also has a full ripple carry adder with 7n gates and 2(n+1) time steps. Several look ahead carry units are described including a full look ahead conditional sum adder with $3n(2+\lceil \log_2(n+1) \rceil)$ gates and $2(1+\lceil \log_2(n+1) \rceil)$ time steps. It is assumed that all gates have a fan in of 2. Sklansky also proposes and contrasts three criteria for performance.

At about the same time, Mac Sorley [5] surveyed various binary arithmetic algorithms. While he does not give functions describing their speed and gate count, his Table II contains numbers which compare several schmes for n = 50 and n = 100. From this one can infer that his full ripple algorithm requires 8n gates and 2n time units while his full look ahead algorithm requires $2\lceil \log_2 n \rceil$ time units and less than $2n\lceil \log_2 n \rceil$

gates. MacSorley also discusses a number of multiplication and division ideas. Another paper appeared in late 1961 [14] which also compares a number of addition schemes and proposes that several are "better" than Sklansky's earlier conditional sum technique. Table II of this paper compares several schemes. In this paper as well as [5], the nation of "gate" seems to be less well defined than in [9]. In [16] Lehman again compares a number of schemes.

In any case, [14] led to an exchange of correspondence in April, 1963, beginning with [15]. A number of assumptions are discussed at some length in this correspondence. Sklansky discusses some bounds on add time independently of any particular circuits but which do include fan in and fan out considerations.

We can roughly summarize the state of the art for binary addition in the early 1960s as follows

Adder Type	Gate Count	Time Units
Bit Serial	7	4n
Full Ripple	8n	2n
Full Look-Ahead	$2n\lceil \log_2 n \rceil$	$2\lceil \log_2 n \rceil$

This leads to the obvious question: Can one demonstrate an addition circuit faster than $2\lceil \log_2 n \rceil$ steps at any cost? One should also be prepared to consider unusual number systems at this point.

Winograd [11] studied the time required to perform addition under a rather general set of assumptions. We shall particularize things somewhat in the present discussion. Roughly speaking, Winograd's definitions are wide enough to include most known number systems and addition algorithms, except signed digit addition. One must be concerned about the encoding, adding, and decoding of numbers to ensure that the addition is "really performed" by the addition algorithm and not by the encoding and

decoding process. In any case, his Theorem 1 is proved for gates with unit delay and a fan in of f.

Theorem 1 The time T to add two n digit numbers is

$$T \geq \lceil \log_f 2n \rceil$$

From this he obtains for k arguments:

Corollary 2 The time T to add k, n digit numbers is

$$T \geq \lceil \log_f kn \rceil$$

Winograd also constructs a multiplication scheme which approaches this bound as shown in his Theorem 2. However, the technique uses residue numbers and so overflow is not detected in the time given. In [12] Winograd discusses (the time required for multiplication as well as) the time required to detect an overflow in the addition of two residue numbers. In Theorem 9 he shows that the overflow detection time is $T \geq \lceil \log_f 2n \rceil$. Winograd summarizes his results in a simple way in [13].

Comparing these results with the full lookahead scheme mentioned earlier it is clear that Winograd's lower bound requires about half the time of a full look-ahead adder. But overflow detection requires the same time so nothing is saved. The question remains, however, can Winograd's bound be approached by some scheme with overflow detection?

Brent [4] considers this problem and establishes that a kind of carry look-ahead adder can be constructed which for large n approaches Winograd's bound. Furthermore, his Theorem 1 outlines a scheme for constructing the adder with order of $n \log_f n$ gates, although he does not exhibit the scheme. This is favorable improvement on the full look-ahead numbers we tabulated earlier.

We remarked earlier that Winograd's formulation of this speed, cost problem did not include the signed digit number system. It was shown by Avizienis in [2] and discussed in more generality in [1] that addition could be performed in a fixed amount of time independently of n, the number of digits. [1] also discusses the number of gates required for various schemes, but the redundancy required complicates direct comparison with the binary cases discussed earlier.

Avizienis also discusses the addition of k numbers and derives a time $T = \lceil \log_{\frac{f}{2}} \lceil \frac{k}{2} \rceil \rceil + 1$, $f \geq 4$, as well as some gate count functions.

When signed digit arithmetic is performed, it is assumed that all numbers are encoded before the calculation begins. Then signed digit arithmetic is performed. Finally the numbers are decoded, a process which propagates the last carries.

2.8 Multioperation Speedups

We have seen that arithmetic operation speeds may be reduced to a function of the speed of parts from which the arithmetic units are built. We have also seen that lower bounds may be established on the times required to perform arithmetic in various number systems. Do these observations imply that the speedup of computers has been reduced to waiting for faster parts from which new machines may be built? Obviously not, since we may consider operating on many pairs of numbers simultaneously. Recall that Babbage planned to do arithmetic and indexing at once and that Menabrea suggested performing more than one arithmetic operation at once.

If we restrict ourselves to the addition and multiplication operations, we can now regard an arithmetic processor as a collection or combination of multiplier and adder units. Suppose we have an arithmetic processor containing two adders and a multiplier and wish to evaluate (a+b)*(c+d). Then the two sums can be formed simultaneously. Thus the arithmetic processor would appear to be able to add twice as fast as each adder can in fact add. In the CDC 6600 this idea is implemented, cf. Ch. V of [8].

It is also possible to achieve faster arithmetic by what is called pipeline processing. If some operation requires T time units, then by cutting the logic into K stages and connecting them through registers, it is possible to introduce a new pair of operands every $\frac{T}{K}$ time units. Similarly, results emerge from such a pipeline at the rate of one result per $\frac{T}{K}$ time units. This idea is used in the 360/195 as well as the CDC STAR and TI ACS machines.

Another approach to speedup by machine organization is to sequence many simple (one of each operation at a time) arithmetic processors

from one program. If we have to add k pairs of numbers and we add them all at once then it takes T time units, but the effective speed per addition is $\frac{T}{k}$ time units. This speedup is analogous to that in the pipeline case, but in practice k may be made much larger for parallel than for pipeline machines. This is the approach being taken in the construction if Illiac IV [3].

Just as we studied the maximum speed of addition (and multiplication) for single arithmetic units, more complex function's speeds can be studied for multiarithmetic function processors e.g. the cases discussed above. As a model of the most general case of these, let us consider an unlimited number of adders and multipliers which operate simultaneously. Each operation (add or multiply) takes one time unit and the processors can communicate their outputs to any other processor in zero time. We also ignore memory times.

How fast can such a machine multiply two matrices or evaluate a polynomial? Two $N \times N$ matrices may be multiplied in $1 + \lceil \log_2 N \rceil$ steps, instead of the usual $2N^3$, by the following scheme. We must form N^2 inner products, each of dimension N. Consider N^3 multipliers each of which performs one multiplication on the first step. On the second step we start to form the sums for the inner products. After one addition using $\frac{N^3}{2}$ adders, we have $\frac{N^3}{2}$ results. On the second addition step we use half of these adders to obtain $\frac{N^3}{4}$ results. After $\lceil \log_2 N \rceil$ such steps we have N^2 results, namely, the elements of the product matrix.

It has been shown by Pan [7] that 2n operations are required to evaluate a polynomial of degree n. Thus, for a serial machine, Horner's Rule is optimal. However, it is easy to see that the form

$$p_n(x) = a_0 + a_1x + a_2x^2 \ldots + a_nx^n$$

requires only $\lceil \log_2 n \rceil$ steps to evaluate all powers, one step to multiply by the coefficients, and $1 + \lceil \log_2 n \rceil$ steps to sum the terms. Thus, by introducing some "redundant" operations we can obtain the result in $2(1+\lceil \log_2 n \rceil)$ time steps. This is a crude upper bound for a multiarithmetic unit machine because some additions can be performed before the final multiplications are performed. A lower bound for a multiarithmetic unit machine is $1 + \lceil \log_2 n \rceil$, following Pan, but it is not obvious how to achieve this. In [6] Muraoka shows how to approach it. Improvements of Muraoka's result may be found in [17] and [18].

[1] Avizienis, A., ACM Symposium on Theory of Computing, May 5-7, 1969, "On the Problem of Computational Time and Complexity of Arithmetic Functions", pp. 255-258.

[2] Ibid., IRE Transactions on Electronic Computers, September, 1961, pp. 389-400. "Signed-Digit Number Representations for Fast Parallel Arithmetic". [On Reserve in DCL Library]

[3] Barnes, G. H., et al., "The Illiac-IV Computer", IEEE Trans. of Computers, C-17 (August, 1968), pp. 746-757.

[4] Brent, R., IEEE Transactions on Computers, August, 1970, "On the Addition of Binary Numbers", pp. 758-759.

[5] MacSorley, O. L., Proceedings of the IRE, January, 1961, "High-Speed Arithmetic in Binary Computers", pp. 67-91.

[6] Muraoka, Y., "Parallelism Exposure and Exploitation in Programs", Ph.D. Thesis, University of Illinois. [On Reserve in DCL Library]

[7] Pan, V. Ya., "Methods of Computing Values of Polynomials", Russian Mathematical Surveys, 21 (January-February, 1966), pp. 105-136.

[8] Thorton, J. E., "Design of a Computer, The Control Data 6600", Scott, Foresman, and Company, 1970. [On Reserve in DCL Library]

[9] Sklansky, J., IRE Transactions on Electronic Computers, June, 1960, "An Evaluation of Several Two-Summand Binary Adders", pp. 213-226.

[10] Wilkes, M. U., "Automatic Digital Computers", New York, Wiley, 1956. [This was the omitted Reference 18 in Chapter 1.]

[11] Winograd, S., Journal of the Association for Computing Machinery, Vol. 12, No. 2 (April, 1965), "On the Time Required to Perform Addition", pp. 277-285.

[12] Ibid., Journal of the Association for Computing Machinery, Vol. 14, No. 4 (October, 1967), "On the Time Required to Perform Multiplication", pp. 793-802.

[13] Ibid., Sci. Am., September, 1968, "How Fast Can Computers Add", pp. 93-100.

[14] Lehman, M. and N. Burla, IRE Transactions on Electronic Computers, Vol. EC-10, December, 1961, No. 4, "Skip Techniques Carry-Propagation in Binary Arithmetic Units", pp. 691-698.

[15] Sklansky, J., IRE Transactions on Electronic Computers, Correspondence, Vol. EC-12, April, 1963, No. 2, "Ultimate-Speed Adders", pp. 142-147.

[16] Lehman, M., Information Processing '62, "A Comparative Study of Propagation Speed-Up Circuits in Binary Arithmetic Units" pp. 671-676.