Computing - Then & Now

George R. Trimble
T-Logic, Inc.

INTRODUCTION

Working with a computer in 1950 was significantly different from working with a computer in the year 2000. Hardware was primitive and slow. Software was non-existent. We have hardware NOW that was not imagined possible in 1950 and software that has extremely sophisticated capabilities. What I wish to do is to point out how functions were performed in those early days, how hardware has become faster, cheaper, and more reliable, how software has evolved in parallel with the hardware developments, and what has been the effects of that evolution. The functions performed are the same NOW as they were THEN but the tools we have to work with NOW are vastly superior to what we had to work with THEN.

COMPUTER CLOCKS/Y2K

The Y2K problem is a good place to begin. It is still fresh in our minds and it will have decreasing but continuing effects for some time to come. The two factors that most directly affected the Y2K problem were computer clocks and memory. As with almost any problem in the world today, it is driven by economics.

1949

None of the early computers had clocks. There had to be a source of pulses to control the timing of the computer. This source was internal and was not accessible to the programmer. These pulses were used to direct the activities of the various components, such as, memory readout, arithmetic unit sequencing, input/output operations, etc. As a result, there was no Y2K problem. Of course, none of these computers were still operating when Y2K actually occurred, making it a moot point. If a date was to be printed on an output, the program would refer to a constant in memory that was placed there by the operator and print this constant. It was an entirely manual operation.

1950s

Towards the end of the 1950 era, program interrupts were introduced into computer architecture. An interrupt driven by the clock would cause the currently executing program to be interrupted on a periodic basis. The interrupt routine which processed the interrupt would then update a location in memory that was used to represent the current date/time. The user had to enter the current date/time at the beginning of the operating day. The program could reference this memory locations and use it to date and or time stamp a printout, to time and internal function, or to perform relative date computations. Whenever the computer stopped or memory was reset it was necessary to reset the counter manually.

1960s

The next step was to build a clock into the computer that could keep track of the date and time directly rather than having an interrupt routine maintain a programmed clock. Sometimes this clock was treated like an input output device that could be set by the program and the date/time read by the program. In conjunction with this, time intervals could be measured to cause periodic program interrupts based on an elapsed time. Since most computers had power on 24 hours a day, the clock did not have to be reset manually very often.

1970s – 2000

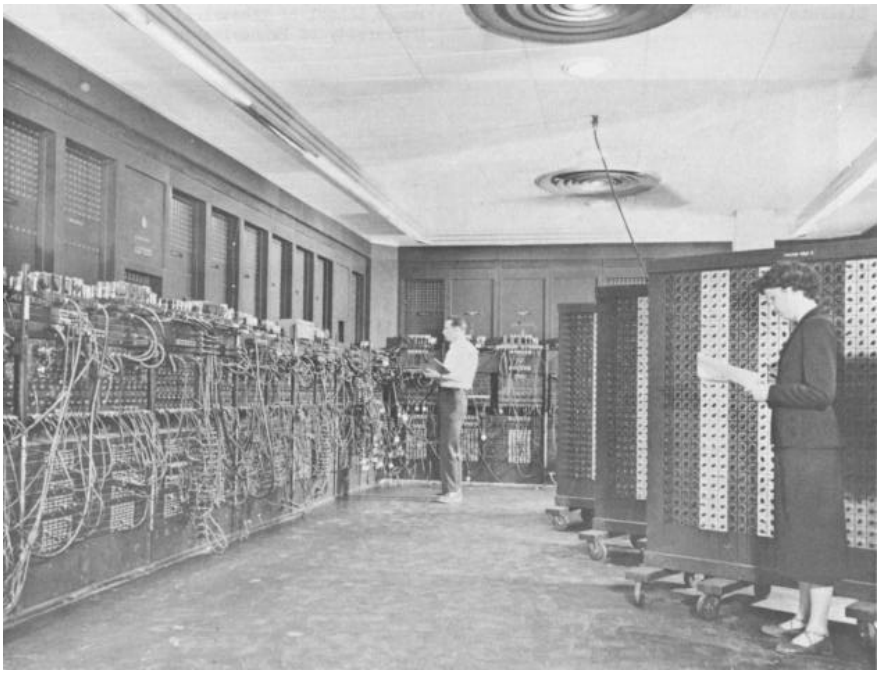
Internal clocks with their own batteries became commonplace. Once the clock has been set, it maintains the date and time of day automatically, even when the computer is powered down. When we start our PCs today, we do not have to worry about the clock. It is there for us, faithfully providing the date and time for use in programs, time stamping files, and providing audit trails recording significant events that occur. The availability of a reliable and accurate date/time stamping capability is taken for granted today. The answers to such questions as: What is the latest version of this file?, When did the operator/administrator perform maintenance functions?, are vital to the effective performance of a modern computing system.

I remember reading in the Tandy Radio Shack TRS-80 Microcomputer News, Vol. 4, 1980, a letter to the editor from a college professor complaining that every time he started his TRS-80, Model II, the TRSDOS operating system always asked for the date and time. He could find no use for this function in his programming work and resented having to enter these values. Needless to say Radio Shack did not follow his suggestion. Without this ability management of software development would be like trying to round up a bunch of baby spiders.

MAIN MEMORY

Main memory cost is probably the most significant factor in the effect of hardware on system software and applications development. Memory today is extremely cheap. The ENIAC cost a little over half a million dollars to develop in 1946. Of this amount, roughly half was the cost of memory. Just for calculation purposes assume that the memory cost $200,000.

FIGURE 1 – ENIAC ACCUMULATORS/FUNCTION TABLES



The ENIAC was a pure decimal machine. It consisted of 20 “accumulators”, each accumulator having ten decimal digits. Each digit position used 10 flip-flops in a linear array to form a ten stage ring counter (i.e., 10 bits per decade.) Thus, the ENIAC had 200 digits of storage, plus signs for over 2,000 bits of storage. This means that the cost per bit of storage was approximately $100 per

bit. At that cost you obviously had to make the most of every bit. Today you can buy chips with containing multi-Mega-Bytes for a little over $100. Specifically, I priced one chip for which the cost was $0.0000005 per bit (32MB for $135), a reduction by a factor of nearly 200 million over the cost of ENIAC's memory. As a result, memory cost in today's computers is almost inconsequential.

Many of the applications which resulted in the Y2K problem were developed in the 1970 to 1990 time frame. Even in 1975, the cost of a bit of memory was approximately seven cents, significantly more than today's memory cost. As a result, much effort was expended to minimize the amount of memory used. This was a conscious management/programmer decision to attempt to use the hardware in the most economically expeditious manner. It was not, as so often reported in the popular press, an attempt to achieve job security, or to ignore the future consequences of the decision, or as sometimes reported, just plain stupidity. Memory was expensive. An application which required the use of extensive dates (banking, subscription the fulfillment, insurance, etc.) could easily use several thousand bytes of main memory just for dates. At seven cents per bit, or $1.12 dollars per date (2 bytes), this could amount to several thousand dollars. The choice was the between having four digit years or having two digit years and using that saved memory to provide more application functions. In most cases, the latter choice was made.

DECIMAL/BINARY – WORD SIZE

The "bit", or binary digit, is universally recognized as the elementary unit which is processed by a computer. How bits are used to represent information is another matter. Today we recognize the byte, a group of eight bits, as a kind of unit. Memory sizes are measured in bytes, transfer rates are measured in bytes per second. The word Byte has become one of the most frequently used words in the English language and many others as well.

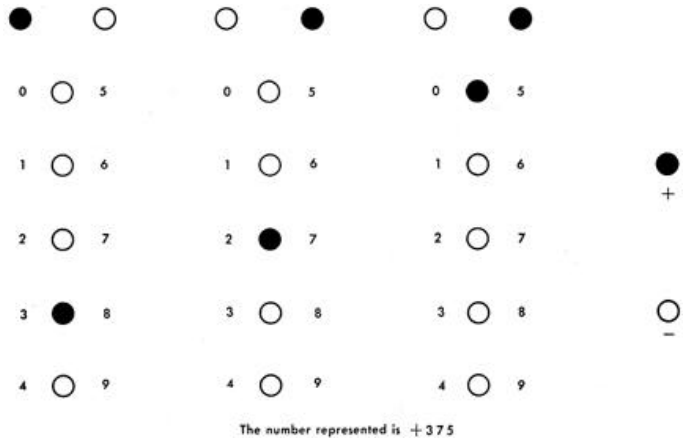
The speed of a machine is significantly characterized by the number of bits it processes in parallel. Before bytes became a common term, a significant characteristic of a machine was its word length. Word length was and still is used to characterize the number of bits that are processed in parallel by a computer, however, its meaning has become fuzzy because the "width" of the arithmetic unit and the "width" of the internal bus used to transfer data in the computer may not be the same. In the early computers the arithmetic unit width, memory width, and data bus width were all the same. With the introduction of PCs, these widths were not necessarily the same. The original chips were 8-bits wide, as were the memories used with them. Then came 16-bit arithmetic units but still retaining 8-bit data busses and 8-bit memories. Consequently, it took two memory accesses to get the 16-bit processed by the arithmetic unit. Variations like this were what resulted in the "DX" and "SX" versions of PCs.

The early computers had internally compatible "widths" but varied greatly in their word sizes. For example, the EDVAC, a computer that used mercury delay lines for its main memory, had a 44-bit word length. It also was a four-address machine, e.g., three operands and the next instruction location. The ORDVAC had CRT storage with a 40 bit word length and stored two instructions in a word. It was a one address machine. The IBM 701 had a 36 bit word length with CRT storage, and was one address.

Data representation or data encoding has varied greatly. As described above, the ENIAC is a decimal machine in the strictest sense of the word. The IBM 650 was also a decimal machine but it used the bi-quinary code to represent decimal digits. It was somewhat like an abacus with its "heaven" beads and its "earth beads". The "bi" part was two binary digits, one representing a

zero and the other representing a five. The “quinary” part was five binary digits representing 0, 1, 2, 3, or 4. The sum of the two parts made up the number, as is illustrated in the figure.

FIGURE 2 – BI-QUINARY REPRESENTATION OF DECIMAL NUMBERS IN THE IBM 650



The 650 had a magnetic drum as the main memory, with access times measured in milliseconds.

Representing alphabetic information was awkward at best. The IBM 701 36-bit word could conveniently be divided into six segments of 6-bits each, or six characters. This made it possible to have upper case letters, the ten digits and some special characters. Although restrictive, it was the best that could be done at the time. The IBM 650 used two decimal digits to represent one alphabetic character. This gave the possibility of both upper and lower alpha characters. Largely because of the IBM System/360 a 32-bit word length became popular, with four 8-bit bytes per word.

TODAY – almost universally eight bit bytes is the standard. The preponderance of business applications has dictated the interpretation of the eight-bit bytes as characters, however, they can be manipulated as pure binary for engineering and scientific calculations or the numerical calculation part of business applications.

INPUT / OUTPUT

Having evolved out of a tabulating machine background, much of the input/output equipment came from that environment. Punched cards, both IBM with rectangular holes, and Remington Rand with round holes, were common. Both the programs and the data were on punched cards. A large application could involve tens of thousands of cards, which were wheeled into the computer room on carts for entry into the card reader. Woe be unto you if the cards should be dropped. It was necessary to use a sorter, a mechanical device that could resort the cards into the correct order, before the job could be run. Greater woe unto those super confident individuals who did not have sequence numbers on their program cards so that there was no way to resort them except manually, based on a knowledge of their content. The chips punched out of the cards became part of the material, including ticker tapes, that was dropped on heroes parading down Broadway.

Paper tape was used on several smaller accounting machines as well as on teletype (and ticker tape) machines. As a result, the early computers used this as an input/output medium. High speed (1,000 cps) readers were sometimes a problem. The take-up reel frequently could not keep up with the tape and a solution to this problem was to simply let the tape go into a waste basket and rewind it manually later. This itself could be a problem when the tangles in the tape became knotted. Besides this, the tapes broke occasionally or the oil on the hands of someone who handled the tape could cause the tape to become transparent to optical readers and thus misread. It was an inexpensive medium, however.

Tabulating machines were used for their printing capability. They could only print upper case characters, which was at least compatible with the use of 6-bit character representation. Control of the tabulating functions was by means of plug-boards, discussed below. The great bulk of

their tabulating capability was bypassed or used only for formatting of the output by routing the output data to the desired print position. Format control also included a "printer control tape", a 1½ to 2 inch wide paper tape that had 12 channels into which holes could be punched at positions corresponding to each line on a page. The tape was as long as the physical page of the paper and was glued to form a loop. Channel 12 was most commonly used to define the beginning of a new page. The printer output program would cause a signal "Eject to new page" to be sent to channel 12 to space up to a new page. Obviously the tape had to be aligned properly with the printer paper's perforations. Other channel might be used to define the line on which page totals were to be printed, sub-totals, or other similar summaries of data.

Tabulating printers were gradually replaced by line printers designed to work with computers and print speeds went from 60 or 100 lines per minute to several thousand lines per minute. At these speeds the paper handling became a problem.

When minicomputers became popular in the 1970s, typewriters were widely used as output devices, in addition to line printers. The wide availability of teletype machines made them a candidate for printed output. A variety of serial character printers, including dot matrix printers were developed. Finally, the Hewlett-Packard laser printer set the standard that still prevails today, although the inkjet printer has also become popular as it's quality has improved.

An 8" Floppy Disk was used by IBM for loading micro-programs into it's System/360, which was introduced in 1964. It migrated to become an input/output device for several of the early personal computers developed in the mid 1970s. It was used on the Radio Shack TRS-80 in 1979. The 8" floppy evolved to the 5¼" floppy on the early PCs and then to the 3½" diskette on IBM PS/2 PCs. That is universally used today but is supplemented by several higher capacity discs today, such as the 100MB Zip and 1GB disks from Iomega and other such companies. A problem with the proliferation of devices is obsolescence. It is almost impossible to find an 8" floppy drive today. Even finding a 5¼" drive can be difficult as I recently found out. Not only hardware but software as well. Be careful with your digital photos since the software to display them today may be difficult to find in ten or twenty years.

PROGRAM CONTROL

Tabulating equipment had a significant impact on the control of early computers. Plug panels were used on virtually every kind of tab equipment to control its functions and to route data. Standard card readers were used as input devices and the plug panels were wired to select which columns of the card were to be read. Similarly, plug panels were used to direct the output to the desired columns of a card punch or to the desired print positions on a tabulating machine. Many of the early electromechanical computers used plug panels to control their functions. Even the ENIAC used plug wire control in its initial version, as can be seen in the picture of the ENIAC on the left side at the bottom of the accumulator panels.

A simple example of plug panel wiring can be seen with the Digital Equipment Corporation's Computer Lab – see FIGURE 3. This device is intended to teach logic circuitry in an introductory engineering course on computers. The rectangular boxes on the diagram represent flip-flops (two state electronic circuits) while the half moon shaped figures are NAND circuits. Most of the elements shown are two input NAND circuits but some are three and four input NAND circuits.

Electrical signals are either high, that is, 5 volts, or low, that is, 0 volts. If both inputs of an AND circuit are high, the output is high. If one or both inputs are low, the output is low. The small circle at the half Moon end is an inverter. The inverter changes a high signal to a low signal and a low signal to a high signal. This makes it a NAND circuit, which is what is on the

Computer Lab. This reverses the sense of the output so that it is low only if both inputs are high. The following table illustrates the possible states:

A	B	Output
Low	Low	High
Low	High	High
High	Low	High
High	High	Low

Using plug wires the output of switches 1 and 2 can be connected to the indicator lamps 1 and 2 and to the two inputs of a two input NAND circuit respectively. The lamps will indicate the status of the switches and of the NAND circuit output.

Although this is a fairly simple example, it illustrates the concept of plug panel wiring. Actual panels have from 3 or 4 hundred plug positions to 1500 positions. A fully wired large plug panel weighed 30 or 40 pounds.

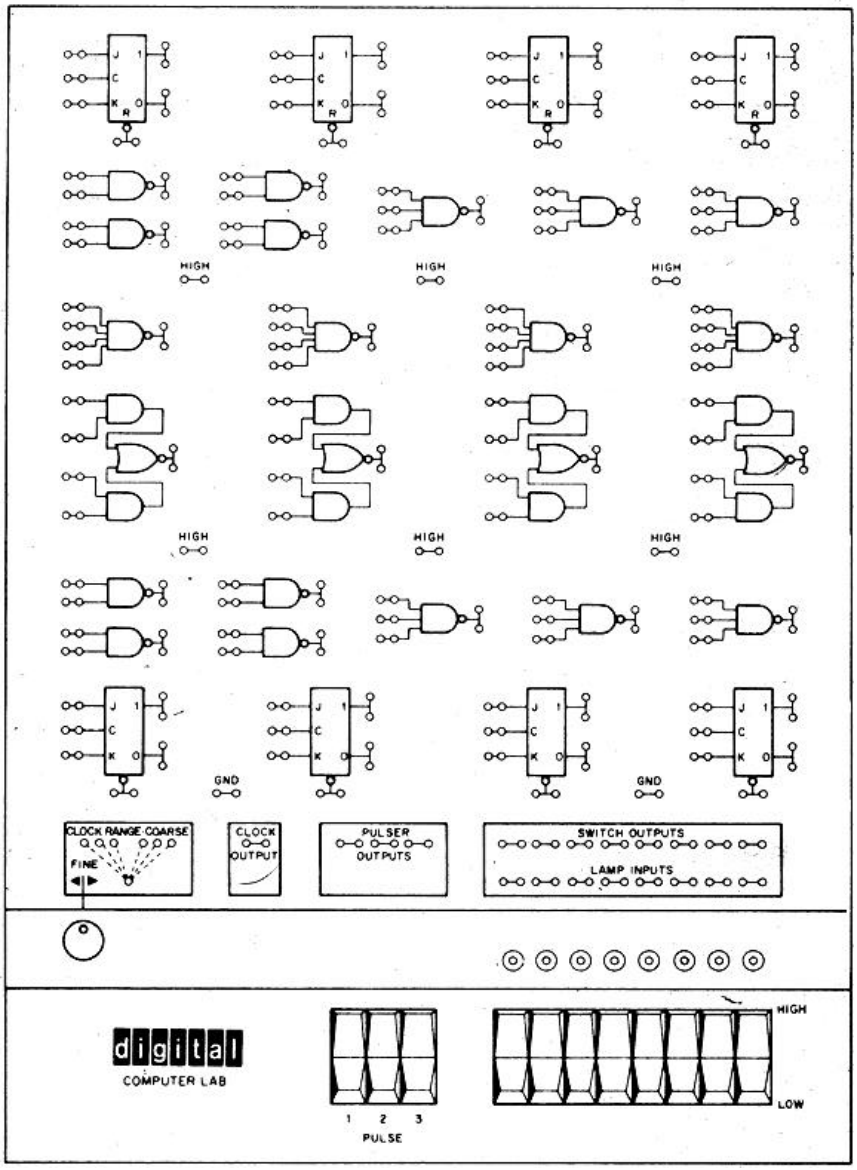
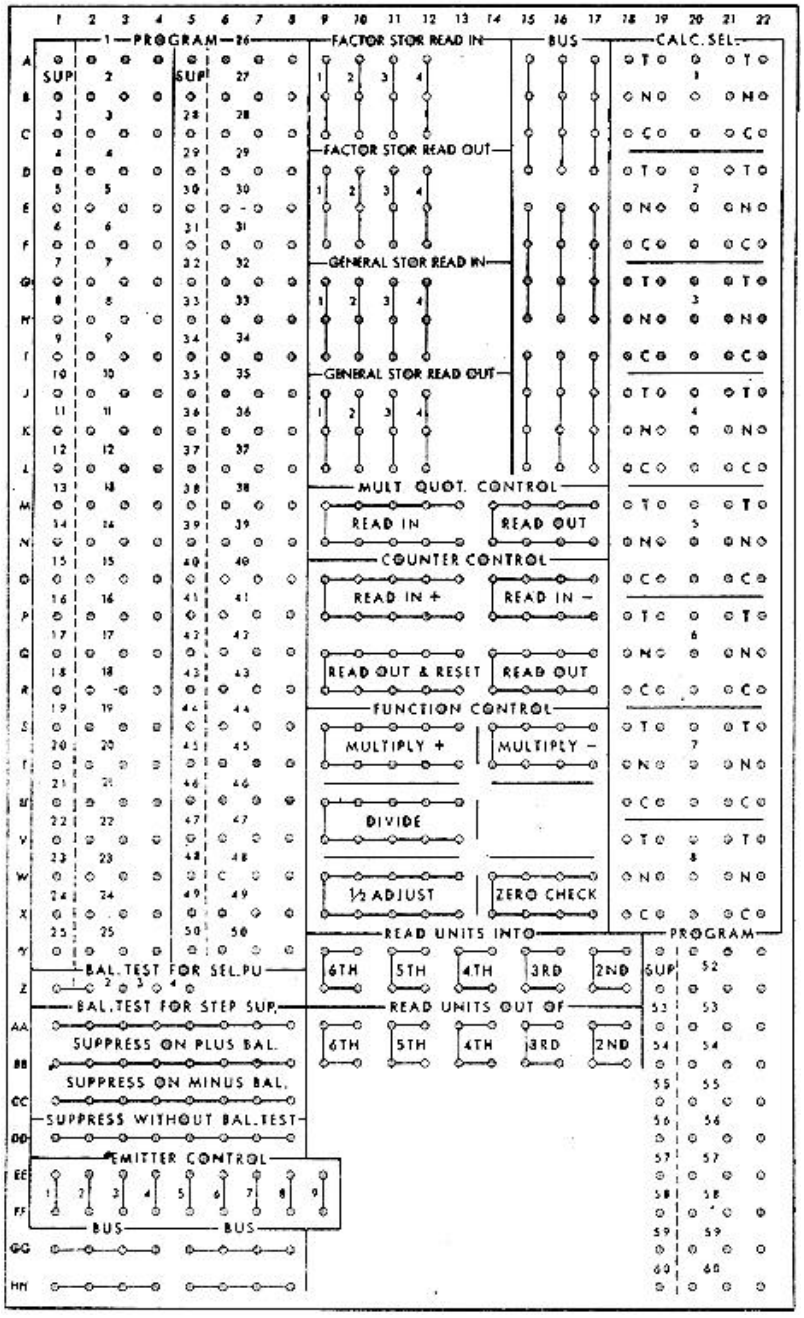


FIGURE 3 – DIGITAL EQUIPMENT COMPUTER LAB

The IBM 604 was a plug board controlled calculator of the early 1950s. It's plug board is shown in FIGURE 4. As can be seen it had about 750 plug board positions. It was controlled by pulses emitted from the PROGRAM hubs, of which there were 60. These pulses were routed to control read in or read out of the registers, multiplication, division, etc. Programming consisted of planning and wiring the panel. Each different application had its own plug panel. Changing the program for an different application involved putting the plug board for the next application in the machine. The 604 had a card reader and card punch. Output punched on cards was run through a tabulating machine to print the results

FIGURE 4 – IBM 604 PLUG BOARD



.

The IBM Card Programmed Calculator, Model I (CPC I) consisted of the 604 connected to a tabulator, a card read/punch unit, and a storage device that held 16 words of data in electromechanical storage (relays) that was referred to as the "ice box". Systems people developed General Purpose Boards for the 604 that interpreted pseudo-instructions punched on IBM cards to perform various calculations. In this way programs could be written using the pseudo-instructions so that the same control panel was used for many different applications. It was not a stored program computer but was a work horse of the industry for several years, particularly for engineering and scientific computing.

Remington Rand developed a similar line of equipment, but neither could provide the flexibility and speed that true stored programmed machines could provide.

The concept was interesting in that, in a way, it was the precursor of micro-programming which was used by IBM in its System/360 line, as well as by several other manufacturers.

The original design of the ENIAC used plugable data busses as a means of controlling the machine. Referring to FIGURE 1 again these cables can be seen along the bottom of the accumulators. In effect, it was necessary to re-cable the machine for each new application. This process was lengthy as well as being error prone and was changed by using the Function Panels as a means of control. Interpolation was used extensively to calculate trigonometric and other functions required by an application. ENIAC had four function panels, each one having 1200 ten-position rotary switches which were set to the table values needed for the interpolations. The change made control reside in the switches. Each pair of switches became an instruction to the ENIAC. Setting up for a new application then became the task of setting the switches to contain the program for the application. This process took about two hours and was also prone to error, but was faster than recabling. It was still not stored programming.

PROGRAMMING

In the beginning there were no program generators, no compilers, no macro assemblers, no basic assemblers. There was only absolute machine language. To write a program, it was necessary to code each instruction exactly as the machine would interpret it. As an illustration, consider the program in FIGURE 5 which was written for the IBM 650 to calculate square roots by an iterative process. The first column represents the location of the instruction in memory. The second column is an instruction mnemonic, but is provided only for the convenience of the programmer. The third, is the actual operation code required by the machine. The fourth and fifth columns are the data address and next instruction address respectively (the 650 is a two address machine.) The use of the second address will become apparent in a moment.

Normally instructions would be executed from sequential locations. Since the main memory of the 650 is a magnetic drum, this would mean at least a full drum revolution for every instruction execution. The second address is used to specify the location of the next instruction.

By judicious coding, the program can be speeded up significantly. Looking at the first instruction, located in 0139, the data is placed in 0142 and the next instruction in 0148. The instruction in location 0148 refers to data in register 8001, which is immediately available. The next instruction is located in location 0105. Since there are 50 locations around the

circumference of the drum in a single track this is actually only 7 locations away. This process is repeated for every instruction. Each different instruction has a different set of rules for placement of the data and the next instruction so the programmer has to know these rules as well as the absolute operation codes. Specifically, in this case the optimized program executes in .152 milliseconds as compared to .337 milliseconds for the sequential version of the program, a factor of 2.

FIGURE 5 – MACHINE LANGUAGE PROGRAMMING

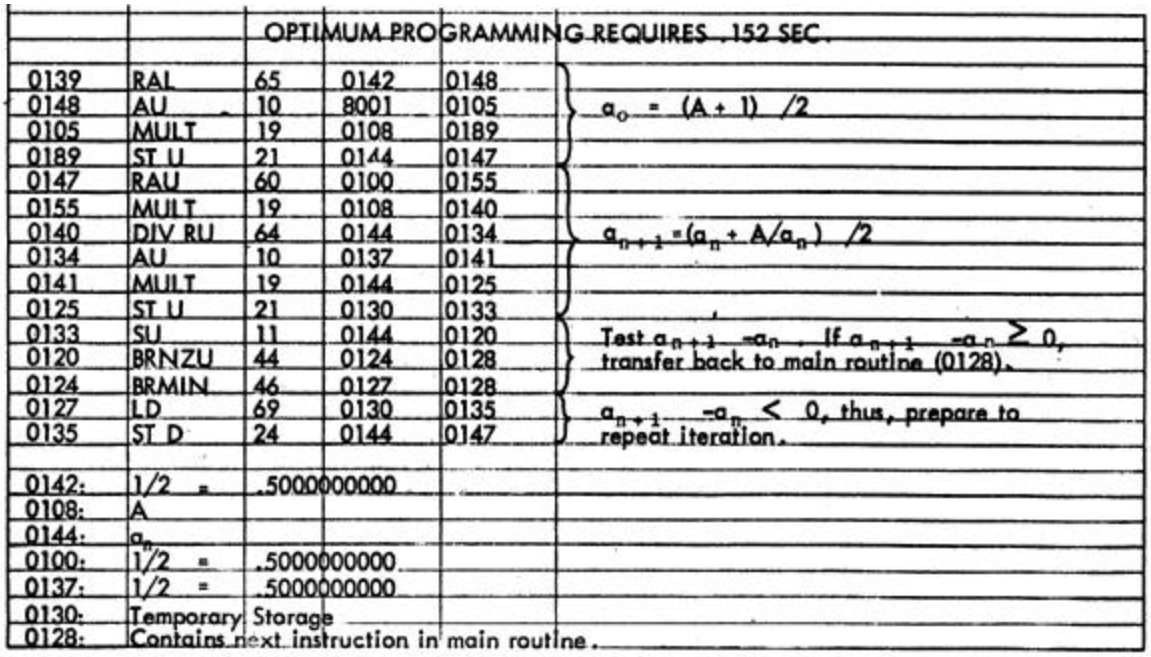
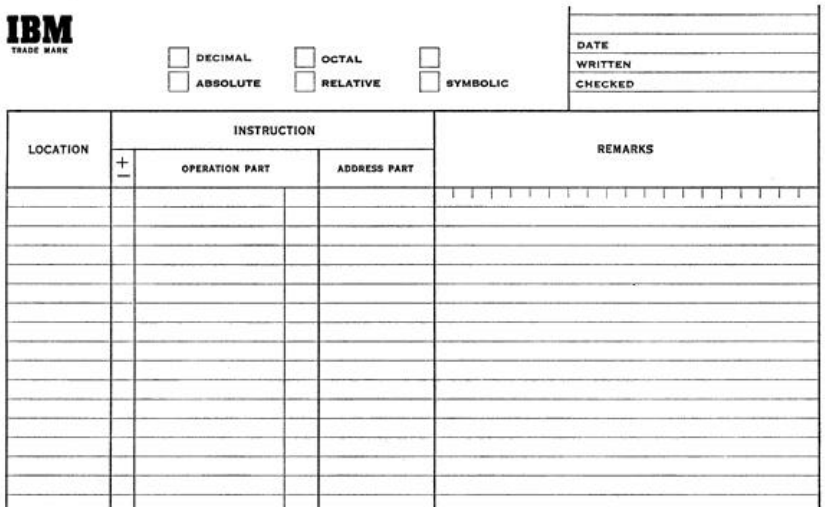


FIGURE 6 illustrates a general purpose coding sheet that services several of the variations of the elementary assembly languages that followed machine code. The simplest was to use mnemonic operation codes but absolute addresses. Even here, pure binary was rarely used. Octal was the most prevalent form of absolute addresses. This was because the 36 bit word length of the IBM 701 was divisible by three so that 12 octal digits (three bits per octal digit) could be used to represent the entire word. In addition, the 701 address field was 15 bits, which is also divisible by three. Hexadecimal, which is fairly common today, did not become popular until the 16-bit mini-computers proliferated.

Relative Addresses were used to define a location relative to the program counter, i.e., the location of the instruction itself. The assembler could use a relative address, say +15, to calculate the address of the data. Symbolic assemblers used alpha numeric symbols to define both data and instructions. Even here, relative addressing, relative to a defined symbol was commonly used initially because of memory restriction. Symbolic addresses meant that a Symbol Table had to be built on the first pass of the assembler and absolute addresses assigned to any remaining undefined addresses on a second pass over the program. Symbol tables took memory, which takes us back to the memory cost problem. A program might have all symbolic addresses, say for 500 symbols, at say 6 characters per symbol, or alternatively, it might be written using only 25 or so symbols and use relative addresses based on those symbols. This latter approach would save 2,850 characters from the symbol table. This is just one example of how system design and programming techniques were influenced by hardware costs.

After full Symbolic Assemblers came Macro Assemblers where frequently used sequences of instructions could be integrated into a program by a macro call with parameters to specify the parameters used in the macro. As memory became cheaper and more available, this, and other facilities became common features.

FIGURE 5 – GENERAL PURPOSE PROGRAMMING SHEET



STARTING A SYSTEM / PROGRAM LOADING

Starting a system today is simply a matter of turning on the power and sitting back until the operating system has been loaded. Modern systems include hardware and built-in software to perform a start-up, initial program load, and starting of the operating system. ROM/BIOS makes this possible on PCs and on larger machines it is frequently called IPL. Early machines did not have such sophisticated capabilities. It was necessary to manually boot the machine by keying in several instructions which could be manually executed. These instructions would load a small boot loader for an input device. Data entered from that device would then load a bigger, more powerful loader that would load the program.

For an example of such a process, consider the IBM 650 and its control panel illustrated in FIGURE 7. The 650 had 10-digit words. Across the middle of the panel are 10 rotary switches used to load data. On the bottom row, switches 3 – 6 are used to enter data into the program counter, which is four digits in length. The operator would set the data (or instruction) to be entered into the 10 data switches, then the address of where the data is to be stored in the lower 4 switches, store the address in the program counter, and finally store the data in that location. This would be done for 8 or 10 instructions. When all of the instructions had been entered, the address of the first instruction in the sequence would be entered into the program counter and executed. These instructions would read a card that had a bigger loader on it, which would then be executed, and would read in more cards containing the application program and start executing it. This process had to be performed every time you did a cold start of the machine.

Debugging of programs was also performed at the console. The program could be stopped at any point and the program counter displayed in the ADDRESS lights in the bottom section and the contents of a register displayed in the upper set of lights (both in bi-quinary format.)

Even though the 650 has been used to illustrate these function, the same principle applied to other computers of the era. The IBM 701, for example, had 36 two position toggle switches that were used in the same manner. The operator essentially worked in binary, not even octal. The Digital Equipment PDP-8 had 12 toggle switches for the same purpose

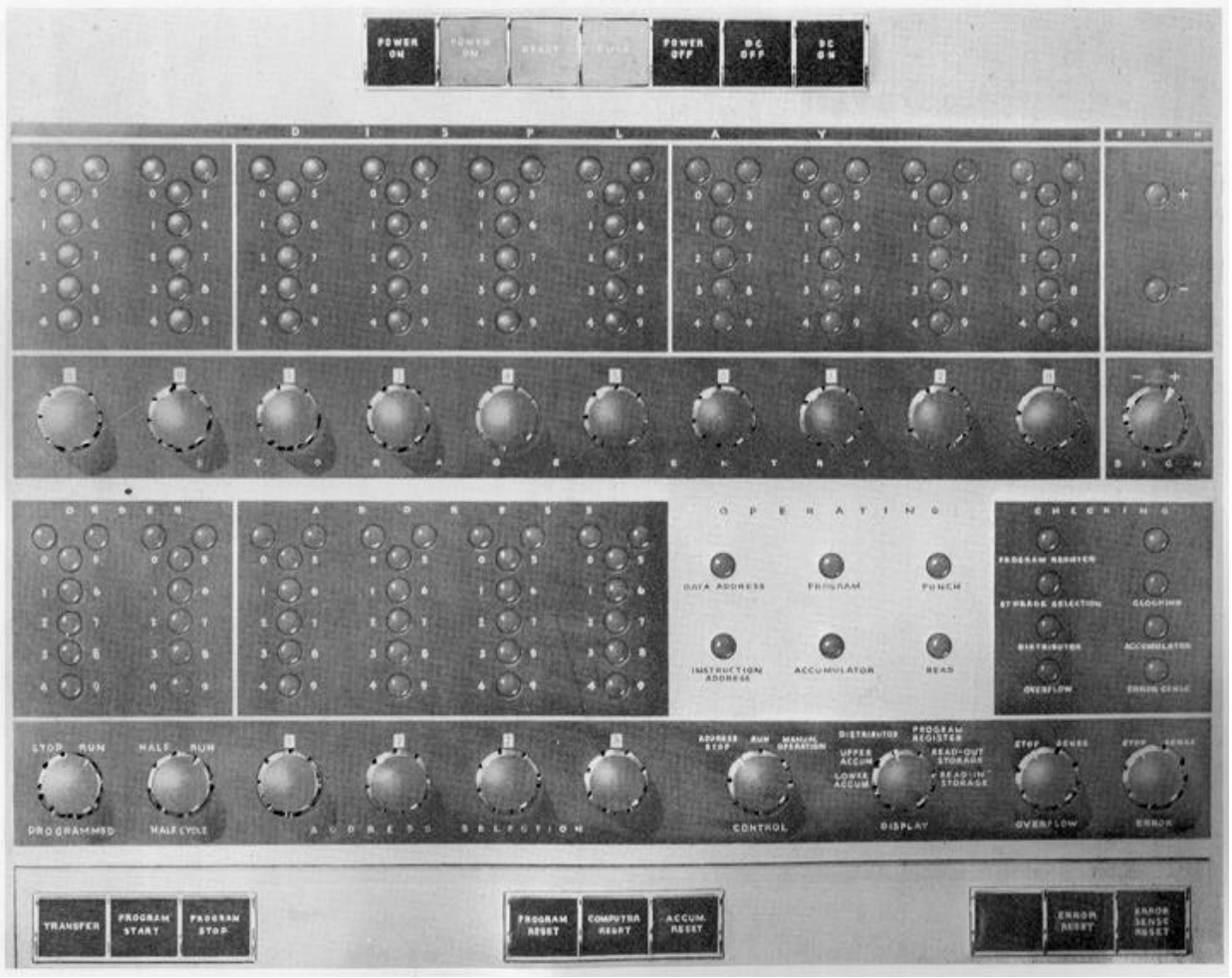


FIGURE 7 – IBM 650 CONTROL PANEL

.

OPERATING SYSTEM

There were no operating systems on these machines. The next user scheduled to use the computer brought their card decks (or paper tapes), manually started the system, and fed their cards into the card reader. When the job was done they took their punched card output and/or printed output and turned the computer over to the next user.

The next step in the evolution of this process was to convert the card decks to magnetic tape beforehand. The job would be run from magnetic (the job stream). Eventually, jobs got queued on disk, were given priorities, were run under a multi-programmed operating system, and operating systems were born.

Today we have multi-programmed, interactive, on-line operating systems with sophisticated debugging and support and capabilities we never even dreamed about in 1950.

COMMUNICATIONS

Changes in data communication have been just as dramatic, if not more so. The earliest means of moving data from one location to another was to physically transport the media. That was done using the USPS if there was no urgency. To expedite delivery we frequently used airline pilots to

carry magnetic tapes. Either an employee or a courier service would be used to carry the tape to the airport and give it to the pilot who would then give the pate to another courier service at the destination and deliver it to the recipient.

The ability to electronically transport files or batches of data was developed by punched card to punched card transmission over telephone lines; sometimes punched card to printer transmission. Then came magnetic tape to magnetic tape transmission. Finally came direct computer to computer file transfer.

TODAY – we need no discussion. Commercial applications have interactive screen-by-screen, field-by-field, character-by-character processing. The Internet provides almost unlimited access to Web Sites, computing centers, individual PCs, with almost transparent file and data transfer. We are approaching computational Nirvana.