Nathaniel Stott
A02386053

Term Paper Rubric

Total of 100 points possible
Points will be deducted for each criterion that is not met. Total points will be prorated to 250 as per syllabus.

- No late submissions will be accepted. Term papers must be emailed to instructor by due date. Do not upload to Canvas. Final term papers turned in without first having an abstract submitted and approved will not be graded and subsequently receive a grade of zero. Term paper **must be** submitted in Microsoft Word. No other format will be accepted. Do not email a cut and pasted version of your paper. Term Papers improperly submitted will not be graded. Order for final paper:

1) Grading Rubric (with student's name)
2) Final abstract
3) Title Page (with title, date, course name, instructor and student's name)
4) Body of Paper
5) Sources

Structure- 30 points total

- Grammar issues- run-on sentences, fragments, verb tense, etc. The writing of the paper needs to be clear and concise. If you struggle with writing skills, I suggest making an appointment with the writing center to receive the help you will need to gain the writing skills necessary for college success. (10 points)

- Writing structure- make sure your writing follows a logical order. Do not switch the style of your writing or your voice throughout your paper. Do not use first person. The paper should have a title page, introduction (not the abstract), body and conclusion. (10 points)

- The paper should be professional and be free of typographical or formatting errors, Diagrams, graphs or pictures should be used to illustrate main ideas presented (these do not count towards the page count). (10 points)

Ideas- 40 points total

- The thesis statement is clear. The final paper supports (or does not) the thesis of your paper. There should be a clear connection between the questions raised in your abstract and the support of your thesis statement. These questions should be clearly answered in the body of your paper. Do not restate the questions in your term paper. The paper should not be a biography, history or a rehash of course material

- Research should be based on current knowledge and information. Paper should be technically sound and based on scientific principles.

Format- 30 points total

- Name needs to be on the abstract and title paper. Include your revised abstract at the front of your paper.

- Paper must use APA formatting for quotes, sites and bibliography.

- Paper must have minimum six quality sources.

- Include an in-text ***citation*** when you refer to, summarize, paraphrase, or quote from another ***source***. Each source you cite in the paper must appear in your reference list; likewise, each entry in the reference list must be cited in your text.

- Quality of sources – As a guideline to what I would consider a quality source please review the following list. Be cautious about using web sites as references - anyone can put anything on a web site, and you have no sure way of knowing if it is truth or fiction. I do not allow encyclopedias or Wikipedia. Library sources are those which are from either a “brick and mortar” library or an online library.

1) Scholarly Periodicals
2) Scholarly Books (this may include textbooks)
3) Student Theses
4) Research Forums or Hotlines on the internet
5) Reputable News Media (Time, Newsweek, New York Times)
6) Serious Popular Magazines (New Yorker, National Geographic)
7) Government Publications
8) Internet versions of any of these sources

Paper Length- Deduction Varies

- Final paper needs to be 8 double spaced pages in length. A maximum of 12 font size should be used. Times Roman font is recommended. 10 points will be deducted for each missing page. Title page, diagrams, graphs, pictures, bibliography and abstract are not to be counted towards the 8 page requirement.

Abstract

A machine that could solve any problem was a growing necessity during World War II. This circumstance is where the general computer was first seen as a need. After the war, the US Army created the "Electronic Numerical Integrator And Computer" (or ENIAC, for short); it was the first general computer with nothing more than a collection of circuitry. It showed the world a whole new field of study. In the late 1940s, computer science formally emerged from mathematicians working with electrical engineers. With it came the dedication to studying computational science. The general computer and its fundamental concepts are still relevant for study today and can help us understand the field of computational science. Understanding the general computer can open the door to understanding computational science's newest fields of study, such as quantum computing and analog computers. Understanding the basic concepts that created the ENIAC can reveal how modern computers work and lead to questions about what fields of study are open in general computation.

This paper will analyze and examine the following questions:

1. What is a general computer? Why was the creation of the ENIAC a big milestone? What are the maths that underline digital computation? What are logic gates, and how do they work? How can many logic gates work together to create a machine?

2. By what circumstances and individuals influenced the modern general computer? What moved general computers to the modern age?
3. What areas in computational science are still open for study? What more can general computers do?

General Computation

PHYS 3020: Great Scientists

Professor Richard Quay

Nathaniel Stott

April 15, 2023

General Computation

Computational science studies theoretical and practical means of computing. The topic of general computation is complex and rarely discussed by most people. To understand the world inside of a computer, we will look at the roots of general computation by looking at how the ENIAC worked and then look at Boolean Algebra and Two State Logic Gates, some of today's smallest concepts and components in general computing. A general computer is programmable and reprogrammable, meaning it can do anything the instruction set tells it to do. This idea is groundbreaking because instead of making a whole new specialized machine to complete one task, the same machine could be used for many tasks. Knowing where we have been will help us understand where we are going.

The idea of the general computer began in the early 1900s. Mathematicians were curious about how many problems could be solved with formal steps (algorithms). Alan Turing developed a mathematical proof showing algorithms can solve any problem. He showed this by thinking of a box and a tape that would hold all the information for the problem. The tape would hold a series of 1s and 0s that would be the encoded instructions for the problem. The box would have a "state" and a “position” it was looking at on the tape. When the box sees the position holding a "1" or a "0", it would look up an instruction in a code book. For example, the box starts in "state 7" and looks at the first potion on the tape. The position holds a "1" so the box looks up what to do in the code book. If in "state 7" and the position is a "1," change position to say "0" and change to "state 12," then move one position to the right, but if the position is "0," then move to position number 19 on the tape. It would then continue executing instructions from the code book and move along the tape.

Once the box moved along the tape or hit a "halt instruction," the problem encoded on the tape would be changed to the answer. One would need to translate the problems from a series of 1s and 0s to something more understandable so that a human could interact with the solution. Turing's proof is the foundation for the theory of computation. This simple box with the ability to follow instructions and the ability to read and write 1s and 0s, tape that holds 1s and 0s, and codebook that tells the box what to do, can, in theory, solve any problem (though it would take a very long time). Allan Turing was an early propionate of general computation. “Turing provides the conceptual framework necessary to demonstrate how, by replicating the aforementioned procedure, the Universal Logical Computing Machine and the Universal Practical Computing Machine may be created. Computationalizing a machine is a matter of education, exactly as in the case of the human brain” (Magnani, 2022). In short, his proof shows that simple and small steps can solve any problem.

Calculations are at the heart of technological and scientific advancement. With the Manhattan Project, German Enigma encryption, and growing industries and cities during World War II, the need for computers was at an all-time high. However, finding mathematicians that could calculate differential equations by hand for artillery shell trajectories was a growing difficulty for the United States Army. Thus, the Army commissioned a group of physicists and electrical engineers to begin creating a machine that could quickly and reliably do calculations for many sectors of the war effort. It took them years to design and build the machine, but it would come to change the world. The ENIAC could calculate an artillery shell's trajectory faster than it took the artillery shell to hit the ground. The ENIAC would change the nature of war, business, and personal lives.

The ENIAC was completed in 1945, it had a cost of six million dollars in today's economy. The way it works is very alien to modern computer architecture but the important thing is that it was Turing complete, meaning that it could do anything a Turing Machine could do. The machine consists of 40 modules. Each part could be hooked up in any way that the programmers wanted. This modularity is a great advantage to having one large machine like preceding computers since each part of the ENIAC has a specific function and task. There were 20 storage modules called accumulators, meaning the computer could temporarily remember 20 variables at any given time. There was an Initializing module that would control the powering on and off of the computer as well as start the program. A Cycalling module would synchronize all the machine operations by sending a signal out to start the next computation cycle since the last cycle was completed. There was a Constant Transmitter Module that could set the accumulators to starting values or clear any of the values. There was a master programmer module that would send signals to the other modules about what operations to do next. There were several function table modules that would store constant values. “The ENIAC was built around 20 accumulators, each capable of both storing a 10 decimal-digit number (plus sign) and performing addition (or subtraction) at electronic speeds by transmitting its content (or its negative content) to a second accumulator (see Figure 1). The ENIAC units were connected by a data bus to transmit the ‘ten digits plus sign’ over parallel wires and by a control bus. Each accumulator contained minimal control logic that when prompted would allow it to either transmit or receive ‘pulse trains’ to and from other accumulators. Programming was accomplished by setting switches on the various units and wiring the connections between them using the two buses for control and data. Thus, program control for the ENIAC was distributed, not centralized” (Shelburne, 2012).

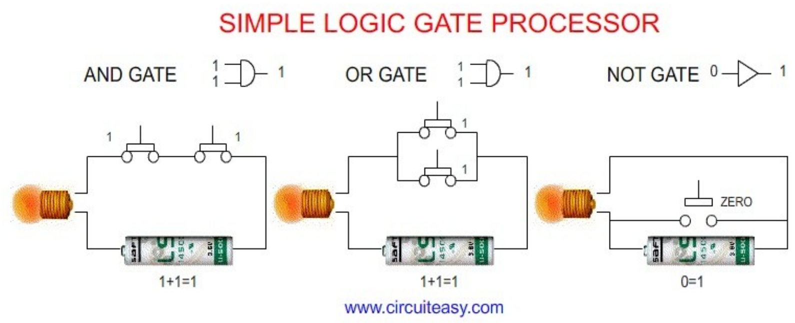
The computer stored numbers using a grid of vacuum tubes on each accumulator, where each row represented a numerical place (one's place, ten's place, and on). Each column started at 0 on the bottom and move up to 9. For example, to represent the number 256 on an accumulator, the third column of the grid would have three tubes turned on to show that there are 2 one hundreds six tubes on the second column to show there are 5 tens, and 7 tubes in the first column to show there are 6 ones. This way of storage was the first time numbers were stored digitally rather than with gears mechanically; this allowed the ENIAC to make 5,000 additions a second, since there were no physical moving parts to allow for the storage and transportation of numbers. The accumulators also doubled as adders; if a five was being stored and a four was sent to the accumulator, the accumulator would add and store nine. This technique helped simplify the design of the machine. There were also dedicated multiply, square, divide, and square-root modules.

The machine did subtraction via ten's complement. Suppose the ENIAC can only store numbers up to 1,000. Subtract 1,000 from the number to be subtracted; this will get the ten's complement of that number. Ignore the largest digit in the resulting number when carrying out the subtraction. For example, on the ENIAC, to do the calculation 400 - 256, find the ten's complement of 256, which would be 1,000 - 256 getting 744. 744 is the same thing as -256 via ten's complement, so if we add the two 744 + 256, we will get 1,000, but we ignore the largest digit and get 0, proving that 744 is the negative version of 256. So 400 - 256 can be rewritten as 400 + 744 equals 1,144 but ignore the largest digit, and we have 144. If we check, 400 - 256 equals 144, so this method works. Thus, the ENIAC could do subtraction via addition also simplifying its design. Input to the ENIAC was done through punch cards (a sheet of paper with punched out wholes to represent data), and output came out from a printer. Programming the ENIAC required wires to be connected between the modules, and data would need to be inputted. The ENIAC would do the calculation and then print off the results. While the program was running, the modules would send electrical pulses to one another as control or data signals. These pulses would be used as conditional logic, such as if a pulse-one is reserved before pulse-two, then do these steps, but if the pulse-two comes, then do these other steps. Thus, the ENIAC could do conditional branching, making it Turing Complete. It was the first computer that was Turing Complete, which is why it was such a big milestone.

Later general computers used two-state logic instead of computing with ten digits like the ENIAC. The reason is that using two numbers instead of ten vastly simplified the design on the machine. This is what made computing move to the modern age. We are used to there being ten digits in math. When we look at the number 1,457 it makes sense to us. When we see 2 + 2, we instantly know that expression equals 4; why is that the case? Numbers are symbols, and we give those symbols meaning. Moreover, we give strung-together symbols extra meaning. A notation is learned around symbols, and once we have learned that notation, it is easy to see what the number 1,457 means. The maths of digital computation can be considered very confusing to some, but it is surprisingly simple. The number 10110110001 will likely jump out as 10,110,110,001 but that is not the same notion as what the number is using. The number uses a 'base-two' or binary notation. To understand the meaning of this notation, look back to the number 1,457 in 'base-ten' or decimal notations. There are "ones," "tens," and so on for every numerical position.

The binary notation follows the same pattern but instead with ones, twos, fours, etc. Thus 10110110001 means that we have 1 one thousand and twenty-four, 0 five hundred twelve, 1 two hundred and fifty six, and so on to 0 twos, and 1 one. Adding up all these numbers will produce 1,457 in decimal notion. Thus 10110110001 in base-two and 1,457 in base-ten are the same numbers; they use different notations. Now add 1010 and 0001 in base two. The same rules from base ten still apply; adding the two would get 1011. We used the "logical OR operation" to do that calculation. In base ten, we use addition, subtraction, etc. Base two uses "OR," "AND," and "NOT" operations to achieve computations. In an "OR" operation (denoted as $\vee$), if the two inputs are both or one of the inputs is a "1," then the output is "1", but if both are "0," then the output is “0”. For an "AND" operation (denoted as $\wedge$), if both inputs are "1," then the output is "1," but if one or both of the inputs are "0," then the output is "0". With the numbers 1010 and 0001, we did a "bitwise" (one place at a time) “OR” operation to add the numbers together. The last logical operator needed to build a computer is the "NOT" operator (denoted as $\neg$). If the input is "1," the output is a "0", and vice-versa. “Each operator of the first type computes a Boolean function. The arguments of that function are the values computed by preceding operators or the values of some input variables. Each operator of the second type depends on m + 1 variables, where m is the number of program outputs, and can terminate the execution of the program” (Chashkin, 2004).

George Boole, Augustus DeMorgan, and others generalized these operators into logical rules. This logic saw applications in Set Theory and Statistics but also formalized logic for the first time. For example, the sentence "Ben is human, and humans are mortal. Therefore, Ben is mortal" is a Boolean assertion. Creating physical logic gates that follow Boolean algebra rules is surprisingly a simple task, all it takes is some knowledge of circuitry. The speed of the logic gate is dependate on how fast it switches current on and off. First, relays (magnets) were used, then triode vacuum tubes, and now transistors. Here is an image of what a logic gates circuits could look like:



(Arduino Projects, 2020)

To more easily think about logic gates and not have to think of the underlying circuitry, symbols are created to represent what’s going on underneath in the circuitry. Using these basic logic gates, we can create more complex logic gates such as the XOR Gate, also known as the Exclusive OR Gate (XOR for short). Many logic gates can be used together in order to build on one another. Using NOT, AND, and OR gates, we created a gate that will be able to add two binary numbers together, but there is an issue with this "adder." If we need to add 1000 to 1000, we will get 0000. When the XOR gate has both inputs turned on, we get an output of "0" and lose the carry bit. Circuitry that will add two binary numbers together and keep the carry bit is called a half adder. Using nothing but circuitry, we have a machine that can add two binary bits and give us the output. To add many bits together, we could simply string many half-adders together. To subtract using this system, we can manipulate the binary number to a negative representation using two's complement (similar to ten's complement). Continuing this process of combining logic gates to build ever-increasing complex systems is what a digital computer is. If we combine the multiplication, divide, and other calculation modules, we would have created an ALU (arithmetic-logic unit). We can give the ALU two binary numbers and send a signal to select what operation to be done on the binary numbers.

We could also create a system that could remember a binary number, and when a signal is sent to the system, the stored binary number would be sent out for use, and a different signal would override what number is stored. This system is called a register. Having many registers means we can store many binary numbers. Putting many registers together and being able to select one register by position is what Random Access Memory (RAM for short) is. However, this system has yet to work thus far. How do we tell the registers and ALU what to do? To fix this, let us imagine our systems are built for handling 8-bit binary numbers, which means we can express binary numbers up to 128 in decimal. Let us have an ALU and RAM but then add two registers right next to the ALU and call them registers A and B. Let us sacrifice the first 4 bits of the 8-bit number and say those first 4 bits in an 8-bit number will be called the "operation code" (opcode for short). Then let us map what each opcode means, such as 0000 means that the last 4 bits are data bits meaning they represent a number. For example, 00001110 would mean the number 14. The opcode 0001 could mean loading some data into register A. Thus 00010111 would mean loading register seven's data into register A. Opcode 0010 could mean loading data into register B. Opcode 1000 could mean adding the numbers in register A and B and then storing the result into either register A or B. Thus 10000001 would mean adding the registers and storing the result in register A (10000010 would mean storing into B). Opcode 1100 could mean to load from register A to a specified register in RAM. Thus 11000110 is to grab what data is in register A and then put it in register 6 in RAM. Finally, we need a halt opcode to tell that we are done. Opcode 1111 could do this making the last 4 bits unless.

Now we are ready to put our binary numbers into RAM. Let us put our binary numbers with the opcode 0000 (meaning they are data) in the bottom part of our RAM (higher numbered) and the other instructional opcodes at the top of RAM (lower numbered). Thus, our RAM could look as follows:

0000	00010111
0001	00101000
0010	10000001
0011	11000110
0100	11110000
0101	
0110	
0111	00001101
1000	00000001

The address is in the first column, and the information is in the second. Starting with register 0000 in RAM, the program executes as follows: load register's 0111 data into register A, load register's 1000 data into register B, add register's A data and register's B data together and store the result into register A, move the data in register A to register 0110 in RAM and halt. To make this execution possible, we would need a "Control Unit" (CU for short) that could read the data in RAM, get the opcode, look up what to do in a preset instruction log, and move things around accordingly. Together the RAM, ALU, A and B registers, and CU are a Turing Complete machine made out of nothing but logic gates and simple circuitry. All that needs to be created now are input and output modules then we have a full-functioning digital computer that follows the Von Neumann architecture created by John Von Neumann. “Fifty years ago, John von Neumann compared the architecture of the brain with that of the computers he invented and which are still in use today. In those days, the organization of computers was based on concepts of brain organization” (Kaiser, 2007). This method makes the digital computer Turing complete. “The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer. The human computer is supposed to be following fixed rules; he has no authority to deviate from them in any detail” (Turing, 1950).

However, there is a growing issue with digital computation, and future development may need to use new kinds of computation, such as analog or quantum computing. “But there is plenty of room to make them smaller. There is nothing that I can see in the physical laws that says the computer elements cannot be made enormously smaller than they are now” (Feynman, 1992). This quote from Feynman is true, making computer parts smaller have limitations and to progress further understanding must take place. Using digital computing has gotten us far in the science of computing, but for the most part, scientists and engineers have hit the bound of how small they can make transistors. Meaning that the speed of digital computing will become largely stagnate. There are few ways to get around this problem in digital computing because transistors are at the foundations of digital computing. Some have thought the solution would be to use three-state or four-state because more operations could be done per execution. However, the reason why two-state logic is the most used is that it is the most stable. For example, a logic gate could see a voltage above .5 volts as a "1," and any voltage below .2 volts is a "0". By adding more states, the gates could get confused about what signal is being sent. Thus instead of building on top of what has already been built, a whole new kind of computation will need to be created. This need is what Analog and Quantum computation are trying to fulfill separately.

Analog computers interestingly predate digital computers, but their popularity died out soon after the creation of digital computers. Today analog computing is returning to the light with the rise of Artificial intelligence. Adding two 8-bit numbers together in a digital computer takes about 50 transistors and a complex array of wires. There are many ways to calculate numbers in an analog manner, but one way is to use current and resistors. On an analog computer, one can connect two current-carrying wires to add the currents onto one wire; the current would represent a number in this method. Multiplying two numbers together using digital computation takes around 1000 transistors. While with analog computation, one can connect the two current-carrying wires and then pass the combined current wire through a resistor, then the voltage across the resistor would be "I*R," where "I" is current, and "R" is ohms. However, there are drawbacks to analog computation since currents can be of continuous values; this allows the same calculation to give differing results. Where values in the digital computation are discrete, making repeating inputs produce the same output. The benefit of analog computation is that transistors are unneeded to calculate a number, making calculations much faster. A purely analog computer can not create a general computer, but what if a general computer was built to use the benefits of both analog and digital? For example, the ENIAC computed numbers using analog and digital, converting a digital number to analog and vice versa where needed. This way, the ENIAC got speed and precision. Maybe the early developers of computation were correct in using more than one computing system in one machine. Our brains work discretely and continuously, so building this kind of computer may bring AI to its full potential.

The idea of quantum computation comes from the famous two-slit experiment. Quickly touching base on the experiment, electrons are shot toward two slits with a screen behind the slits. An interference pattern is shown on the screen without observing what slit the electrons went through. However, when there is an observation, the interference pattern disappears, and two bars are shown. This experiment shows that electrons, when there is no observer, are a wave; when there is one, they are a particle. Thus electrons are both a wave and a particle. Quantum computation uses this property in quantum mechanics to do large calculations. Quantum computers use quantum bits (qubits for short) instead of bits. While a bit can be either a "1" or a "0," an unobserved qubit can be both a "1" or a "0" with varying levels of probability. One unobserved qubit may have a 40% chance of being a "1", while another could have an 80% chance of being a "1". The qubit's state is only known once the qubit is observed or measured. A qubit, for example, could be the magnetic field of an electron coming from the electron's spin. If the field is pointing down, that could be called the "0" bit, while if it is pointing up, that would be the "1" bit. By applying energy to the electron, the direction of the field is flipped. This idea of the direction of magnetic fields is known as spin up (1) or spin down (0). In digital computing, having two bits means two bits of information are known, but in quantum computing, having two qubits would contain four bits of information. This strange property of qubits is because the two qubits can be in all possible states at the same time, such as a combination of "11", "10", "00", or "01". If n is the number of qubits, then 2^n is the number of bits those qubits store as information. Thus calculations done via quantum computation can be done exponentially faster than digital computers. “Quantum speed-up has been conjectured but not proven for a general computation. Quantum interference computation (QUIC) provides a general speed-up. It is a form of ground-mode computation that reinforces the ground mode in a beam of mostly nonground modes by quantum superposition. It solves the general Boolean problem in the square root of the number of operations that a classical computer would need for the same problem. For example a typical 80-bit problem would take about 1024 cycles (107 years at 1 GHz) of classical computation and about 1012 cycles (20 minutes at 1 GHz) of QUIC" (Finkelstein, 2007). A logic gate manipulates bits; a quantum gate manipulates qubits. The quantum gate takes in unobserved qubits and changes their probabilities without observing the qubit's state. The quantum computer needs to be kept at extremely low temperatures to keep qubits from interacting with each other. Interacting would collapse their probabilities to a single bit. A whole paper could be done on quantum computing detailing the various complexities of such a device, but there are other focuses of this paper. Quantum computing research is helping us better understand quantum physics and is opening a new field in computational science.

General computers are capable of much more than what the limits are today. Digital computers can simulate all posible drug molecules and test them in combination with other drug molecules, however it would take many years to compute. With new computing techniques drug simulation could take a matter of days. Computational science brings theories to reality. Early pioneers laid down the foundations of computational science. The beginnings of computational science were slow and costly, but it showed that the Turing Machine could be made real in various ways. This discovery has led to what general computing is today, which is mostly a series of logical statements connected and controlled to complete various computing tasks. The future of general computing will look different than what is in use today, but it will bring a new generation of computers and a new age to humanity.

References

Arduino Projects (2020). *Simple Logic Gate Processor [image]*. Circuit Easy. Retrieved 2023, Ranchi, India, https://www.circuiteasy.com/logic-gates

Chashkin, A. V. (2004). Average time of computing Boolean operators. *Discrete Applied Mathematics*, *135*(1-3), 41–54. https://doi.org/10.1016/s0166-218x(02)00293-7

Feynman, R. P. (1992). There's plenty of room at the bottom [data storage]. *Journal of Microelectromechanical Systems*, *1*(1), 60–66. https://doi.org/10.1109/84.128057

Finkelstein, D. R., & Castagnoli, G. (2007). Quantum Interference Computation. *International Journal of Theoretical Physics*, *47*(8), 2158–2164. https://doi.org/10.1007/s10773-007-9580-2

Kaiser, M. (2007). Brain architecture: a design for natural computation. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *365*(1861), 3033–3045. https://doi.org/10.1098/rsta.2007.0007

Magnani, L. (2022). Conceptualizing Machines in an Eco-Cognitive Perspective. *Philosophies*, *7*(5), 94. https://doi.org/10.3390/philosophies7050094

Shelburne, B. (2012). The ENIAC's 1949 Determination of π. *IEEE Annals of the History of Computing*, *34*(3), 44–54. https://doi.org/10.1109/mahc.2011.61

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, *LIX*(236), 433–460. https://doi.org/10.1093/mind/lix.236.433