Building the World's First Computer

Charles Babbage's "Analytical Engine"

Len Shustek

4 February 2024, version 0.5

Introduction

Charles Babbage, as historian Doron Swade likes to say, is equally famous for two things: inventing the digital computer, and failing to build it.

Babbage began his first design for a computer, which he called the Analytical Engine, in 1834, at the age of 43. He worked on that and subsequent designs for most of the next 37 years, until his death in 1871. During that time he created "Plans" for at about 35 different Engines, each with varying degrees of details worked out. Unfortunately none of the Plans was a complete design with a set of drawings from which parts could be built and a computer assembled.

We are left with a rich collection of ideas, drawings, notes, mechanical designs, timing diagrams, and descriptive symbolic "Notations" for an entire family of unfinished computers that no attempt was ever made to build.

The challenge we wish to pose is: Can we, 185 years later, build a fully functional mechanical computer of the kind that Babbage envisioned?

Contents

Executive summary

The dawn of the computer age is generally situated in the middle of the 20th century, at about the time of World War II. Although we crave heroic individual inventors for important technological advances – like Bell for the telephone, and Edison for the lightbulb – the situation for the computer is much more complicated.

If, however, we travel back to the middle of the 19th century, we discover a lone genius named Charles Babbage who devoted his life to inventing calculating machines. The most innovative of those machines was a programmable digital computer that he called an Analytical Engine.

The problem with anointing him on that basis as the sole inventor of the computer is that he never built it. We don't know if his ideas were sound, or if his computer would have worked.

The project we wish to undertake is to build a working version of Babbage's Analytical Engine, or at least something close enough to it that will demonstrate to the world that he fully deserves to be considered the inventor of the computer.

The Engine we propose to build is an entirely mechanical monstrosity, perhaps 12 feet long, 6 feet wide, and 10 feet high, made of gears, wheels, levers, and cams. It is programmed with "software" that is encoded by punched holes in rigid cards that are sewn together into continuous chains. It is animated by a single rotating shaft, which we would ideally like, in the spirit of Victorian technology, to be powered a steam engine. Useful computations will take 10 to 30 minutes, and the results will be observable afterwards as 30-digit decimal numbers shown on 7-foot tall columns of digit wheels.

This project is likely to take 3 to 5 years and cost 2 to 4 million dollars. The final year is the assembly and debugging phase, which we propose be staged in public view as a "living" museum exhibit.

Assuming we achieve success and have a reliable operating Analytical Engine, it would then become a permanent, regularly demonstrated, and no doubt wildly popular exhibit.

This is an ambitious project, but the obligation to create a just record of history motivates us to undertake it.

Posterity, and Charles Babbage, deserve it.

Part 1: the project and its background

Related Babbage machines

In addition to designing computers, Babbage also designed mechanical calculators he called "Difference Engines". These were special-purpose non-programmable machines that compute and tabulate values of polynomial equations that can be used to approximate a wide variety of mathematical functions. Like all traditional calculators, they are fundamentally different from computers in that they don't follow instructions we call "software". The Analytical Engine, in contrast, would have been of an entirely new and different class.

From 1823 to 1842 Babbage made an attempt to build his first Difference Engine using funding provided by the only venture capitalist of the day: His Majesty's government. The project ended as a complete failure for a variety of reasons, including that Babbage had become more interested in his new idea for a computer.

Between 1846 and 1849 Babbage took time off from working on the computer to design an improved calculator, Difference Engine #2. He completed the design in full and meticulous detail, but he made no attempt to build it.

Some 140 years later, from 1985 to 1991, Doron Swade at the Science Museum in London led a bold project to build Difference Engine #2 exactly according to Babbage's plans, with fabricated parts that have mechanical tolerances no better than what Babbage's machinists could have achieved. It was a spectacular success. The machine works perfectly, demonstrating both that the design was correct and that Babbage could have built it. We can only conclude that the reasons for his failure were managerial and organizational, not technical.

Is the same true of his computer, the Analytical Engine? With the right management and funding, might we have had a computer a century earlier than we actually did?

The Science Museum ultimately built two copies of Difference Engine #2 as well as the integrated printer that was also designed by Babbage. The first copy stayed there and is run by curators only occasionally. The second one was loaned by its private owner to the Computer History Museum in California, where it was demonstrated by volunteer docents daily for 8 years, to the amazement of hundreds of thousands of visitors. In 2016 it was moved to the owner's personal site and is now also rarely operated.

On the prospects for building a complete Babbage Analytical Engine

There is a long-term vision for a project that would choose one of Babbage's 35 plans for Analytical Engines, complete the many parts of that design that he had left undone, and build it as an historically accurate incarnation of one of his imagined computers.

This may someday be possible. But building an engine that adheres strictly to any of Babbage's Plans is a massive undertaking and will be very difficult. Here are some of the issues:

- Only the later engines, starting with Plan 27, have conditional branches and loops that qualify them as modern-style general-purpose computers. Earlier designs do not, and so are less interesting as precursors of what was reinvented in the 20th century.
- Babbage prized both generality and high performance, with the result that all of those later Plans are frighteningly complex.
- The design for the chosen Plan would have to be completed in a way we might imagine him doing. There are substantial subsystems (like the interaction between user-level instructions and the microcode, and the transmission of barrel[1] stud movements to the rest of the machine) as well as many important smaller mechanisms that are not designed. It is not just a few minor details that are missing.
- The complex mechanisms he did design have never been built or simulated, and some of them (like the selection mechanism for table-driven division) are devilishly hard to understand. We would have to convince ourselves, by reasoning or by simulation, that they would work correctly.
- For historical authenticity there might be pressure to use only materials and perhaps even methods that would have been available to Babbage, which increases the difficulty.

It will be a challenge. In 2012 Doron Swade said of the magnitude of such a project[2], "I reckon on 15 to 20 years to construct, and between 20 and 40 million dollars".

[1] The "barrels" are rotating cylinders with vertical columns of preset studs that control operations in the machine during a cycle, which is either 15 or 20 basic time units. The barrel rotates to a new position for each cycle by an amount that may depend on the data being manipulated. Depending on the Plan, Babbage proposed from one to five barrels, each of which can be viewed in modern terms as an independently executing microprogram.

[2] "Mission Impossible: Constructing Charles Babbage's Analytical Engine", Dartmouth University, 8 May 2012, https://www.youtube.com/watch?v=FFUuN-ZRLz8

The proposed project

As an alternative -- or perhaps as a prelude -- to choosing, completing, and constructing an historically accurate instantiation of one of the full Babbage engines in the course of a 15 to 20 year project, I propose that we build a "Babbage-inspired" smaller Analytical Engine. It will still be a fully-functional mechanical digital computer run by software, but it will be much simpler because it implements only a subset of the many features that are in Babbage's highly developed Plans.

In 1840 Babbage briefly considered such a machine, designated as Plan 26, but he was discouraged by the poor theoretical performance and did not pursue it. Again at the end of his life he considered -- and actually started constructing -- a different simplified engine that he called "the Model", which might be considered his final Plan 35. The incomplete fragment he built is at the Science Museum in London, but no drawings of what he intended for the complete machine survive.

The functionality of our proposed small engine is in line with both of those. Our goals are threefold:

- to gain experience with constructing a mechanical computer using Babbage-designed mechanisms that might inform later attempts to build one of the complete Babbage engines
- to help justify the assertion that Babbage is the inventor of the computer
- to create an engaging public museum exhibition of a programmable computer that could have been built a century before it was actually first achieved

We are not overly concerned with performance, because we can choose specific demonstration problems that execute in a reasonable time given the characteristics and generality of the smaller engine. We wish instead to emphasize the novel and prescient aspects of his instruction sequencing and microprogram control, which predated the reinvention of electronic equivalents by over a hundred years.

Will this be an "authentic" Babbage Analytical Engine? No, and yes.

No, in that we are not implementing a specific Babbage design, and we are not using only materials that he would have used. It will not look in its details exactly like an engine he could have built, and it will not have some internal features that were in many of his designs.

Yes, in the sense that it is it is an honest amalgam from several of his designs. It will use only architectures and mechanisms he designed, and it will functionally be a faithful subset of many of his unbuilt computers. It fully inherits the spirit of his ambitions.

We will be careful to describe this not as “The Babbage Analytical Engine” (as if there were one), but as a “Babbage-inspired”, “Babbage-like”, or “Babbage subset” computer. See Appendix 1: proposed exhibit placard.

Related projects

There is a complementary formal organization already in place. Plan28 (www.plan28.org) was established in December 2011 as UK non-profit charity #1145043 whose stated mission is "to build a Babbage Analytical Engine for historical and educational purposes." It has four volunteer trustees (Doron Swade, John Graham-Cumming, Tim Robinson, Johnathan Rosser Histed) and no employees. The reported total gross income has been less than £1K per year since inception. Its 2013 appeal to accumulate financial pledges on www.justgiving.com has been suspended. As of March 2023 several attempts to solicit additional participation in the activities of Plan28 had not been successful

After 12 years, the major accomplishment of Plan28 has been the incredible six years of work that Tim Robinson (a docent at the Computer History Museum in California) has done as a volunteer, cataloging and analyzing the 7000 pages of documents that Babbage left behind relating to the Analytical Engine. Tim is writing what will be the definitive scholarly exposition on the evolution of plans for the Analytical Engine during Babbage's lifetime.

Tim's work is monumental, and provides important background for understanding Babbage's designs. But it does not contain within it a proposal for a modern construction project. No resources have been applied to that.

The Plan28 project is committed to choosing, completing, and faithfully building one of Babbage's full designs, and doing it in the UK. Our goal is to build a reduced design, and to do it in the US. The projects are compatible and harmonious, but different and independent.

Project phasing, timeline, cost, venue, and funding

Doron Swade's estimate of 15-20 years and $20-40M for the full engine build was based, as he says, "on the classic and traditional engineering fudge factors, guess work, divination, [and] astrological prediction".

At this point we can do no better than that for the proposed smaller engine. We have only uninformed wild guesses, of which this is the first one:

phase	for what	time	money
1	architectural design, including register-level simulation	6 months	$0
2	detailed mechanism design, including some mechanical connectivity simulation and prototype construction; develop test and evaluation plans	1 year	$0
3	CAD part drawings, including limited motion simulation of some subsets	1 year	$250K
4	part fabrication and procurement	6 months	$1M
5	assembly and debugging	1 year	$250K
	totals	**4 years**	**$1.5M**

The cost for parts was guesstimated thusly: The most numerous part is the digit wheel. We have a quote of less than $2 each for 3D-printing them in China out of nylon. We need about 1700, so $3400. Quadruple that cost to account for metal shafts and surrounding supports, so $14K. Assume we need 4 times as many parts for the rest of the engine (so 6800, of which some are custom and some are stock) that cost $150 each on average, and the total comes to $1.03M[3]. This is, of course, a gross and premature estimate.

The two chunks of $250K are for professional salaries. The rest of the labor, including management, comes from volunteers. We might be able to get some contributed professional labor from related companies, like Autodesk.

Clearly these completely fabricated estimates cry out to be quickly and periodically refined as the design becomes more specific.

[3] Note that the most numerous part – the figure wheels – are an insignificant part of the total!

The first four phases require no venue, other than modest storage for parts as they arrive during phase 4. Assembly would most preferably happen as a year-long public exhibit at the Computer History Museum in California, at no cost for the space. The machine would then, if successful, become a permanent museum exhibit and be operated daily by trained volunteer docents.

I expect that raising $1M to $2M ($2M to $4M if we are off by a factor of 2) for this project from a small number of interested individual donors will not be difficult. Corporate sponsorships are also possible and would be welcome.

This is a difficult but definitely achievable undertaking that will be a major contribution to the history of computing and therefore of our modern age.

Part 2: technical details

We here codify two sets of propositions for the construction of the small engine:

1. guiding principles for making design decisions, including those that will allow the engine to qualify as “Babbage-inspired”, or “an amalgam of Babbage’s designs”
2. the design decisions, based on those principles, that determine the detailed characteristics of the Engine we will build

These are strawman proposals. After they have been debated, modified, and accepted, we can move on to the next phases of the project: detailed design, simulation at several levels, part drawings and specifications, prototype testing, fabrication, and assembly.

The principles

1. The top-level goal is to exhibit the practicality of a general-purpose programmable mechanical digital computer that could have been built using Babbage-designed mechanisms in the 19th century. It is not, as was motivating Babbage, to build a machine that will actually be used for utilitarian table-making, or for novel mathematical research.
2. We are not picking one of Babbage’s extremely complex Plans to implement. None of them were complete, and some of the Plans closest to being complete violate other of these principles.
3. We are instead building an Engine that is “inspired by” Babbage’s designs. We will pick from among his algorithms and mechanisms, from whichever of his Plans they were in, to create an achievable modern Plan that has a high likelihood of success.
4. We will include all the features necessary to demonstrate that the engine is a fully general programmable computer in the spirit of those we build today. That includes the ability to create program loops, and to execute conditional transfers that are based on computed results.
5. Maximum performance is not an objective, as it was for most of Babbage’s Plans. Our objective is “reasonable” performance that is suitable for demonstrations.
6. We place high value on simplification, subject to the “reasonable” performance objective.
7. We place high value on avoiding the use of untested or novel mechanisms that might increase the risk of failure.

8. We place high value on having only one kind of each distinct part type, not multiple variants. So, if we can: only one kind of Figure Wheel, only one kind of barrel pickoff lever, etc.
9. All things being equal, we would prefer to make more copies of a smaller number of parts that we can develop high confidence in, rather than to use unique, unproven, or complex mechanisms.
10. We will primarily use architecture and mechanisms that had been proposed by Babbage for some Plan, or variations of them that we could easily imagine him devising given the appropriate motivation. We are rarely if ever allowed to invent new mechanisms, and only if the need is compelling, and only if they are in the style of those he devised.
11. We will not be constrained to use only engineering processes that were available to Babbage. We unapologetically use modern tools for simulation, design, and manufacturing.
12. We will not be constrained to use only materials that were available to Babbage. Plastic parts fabricated using 3D printing, for example, are fair game.
13. We wish the resulting Engine to be demonstrated regularly in a museum environment, so we will need to choose and achieve appropriate quantitative targets for performance, life expectancy and reliability.
14. Our focus is on building the calculating engine. We will leave the building of printing[4] and/or plotting apparatuses, both of which interested Babbage, to future generations.

[4] Note that the printer designed by Babbage for Difference Engine #2 and built by the Science Museum London in the 1990s would work equally well, with minor changes in the interface, for the Analytical Engine.

The design decisions and the rationales therefor

1. Numbers have 30 digits.

- Babbage's designs varied from 25 to 60 digits per number, stored as a set of rotating "digit wheels" that are stacked vertically on a common axis with the units position at the bottom. That may at first seem like an excessive number of digits, but because there was no floating point number representation, long numbers are required to support a large dynamic range.
- 30 digits is more than adequate for demonstration purposes. We could even do with fewer.
- Increasing the number of digits in numbers has a surprisingly modest impact on the details of the design. But it does increase drive force requirements, change the framing system, increase the overall height, and increase cost. We would do that only if further analysis provides the justification.
- The Difference Engine #2 built in the 1990s had 31 digit numbers, so we would be in good company. We would feel comfortable about the complexity and utility anywhere in the 20-40 digit range.

2. Use sign-magnitude representation for numbers in the store.

- Babbage proposed this representation in all mature versions of the Engine.
- The sign is stored on an additional (31st) Figure Wheel, with any even number representing positive, and any odd number representing negative.
- This is convenient for user interaction, and for multiplication and division.
- It does complicate addition and subtraction, which will sometimes require extra cycles to convert to 9's complement form suitable input for the adder, and to convert negative numbers back to sign-magnitude for storage.

3. Use 9-cycle whole-zero operations for Giving Off[5] transfers

- Starting with Plan 28, Babbage proposed a clever "half-zero" idea which almost doubles the speed of transfers by turning wheels in either direction during Giving

[5] "Giving Off" is the process of rotating all the digit wheels of a number axis so that they become zero, while at the same time adding the original values to the digit wheels of another axis. When using "whole-zero" it takes one cycle of 15 time units: 1 time unit for receiving the directive from the microprogram barrel, 2 for putting the axes into gear, 9 for the rotation, and three for engaging the locks that prevent any further movement of the wheels.

Off. But it almost doubles the number of wheels, and it adds complexity to the Anticipating Carriage. We do not embrace that tradeoff.

4. Do addition using an anticipating carriage

- Babbage was justifiably proud of his clever design that performs all the carries for a multi-digit addition in parallel, in a small amount of time that is independent of the number of digits in the values being added.
- We accept the complexity of this mechanism both to improve performance for a common operation and to honor the genius of its inventor.
- We do not implement his "hoarding carriage" for consecutive additions that accumulates up to 10 carries for later inclusion into the sum.

5. Multiplication and division are implemented without using a table of multiples of the multiplicand or divisor

- Most of Babbage's mature Plans precomputed nine multiples of the multiplicand or the divisor, in the interest of increased speed during the computation of the product or quotient.
- That scheme requires nine additional "Table Axes" to be built, and several cycles of additional overhead time at the start of a multiplication or division. It represents a significant increase in parts and complexity.
- In many cases it also results in a significant performance improvement. But it actually reduces performance for some simple cases where the overhead of constructing the tables dominates.
- To support our decision not to use tables, I did timing simulations using random operands of various lengths. See Appendix 2: multiplication and division timing charts for graphs that elucidate and quantify the cases where the tables help. The conclusion is that we can minimize the performance hit of not using tables if we avoid long multipliers and long dividends with small divisors.
- We will select demonstration programs so that the lack of precomputed tables is not a serious issue. Babbage, in his quest for general applicability of the engine to a wide range of problems, did not have that option.
- That said, it is historically important that we be able run the program to compute Bernoulli Numbers whose trace Ada Lovelace documented in Note G of her 1843 translation of Luigi Menabrea's paper on the Analytic Engine, and in that case we will suffer from not using the tables. See Appendix 3: calculating Bernoulli numbers.

- Babbage eliminated the tables twice: once for the short-lived Plan 26 in 1840, and again for what Tim Robinson calls Plan 35 in 1869-1870. Babbage called that his “model”, and used the design to actually start construction, for the first time, of an Analytical Engine. It was still incomplete at the time of his death in 1871, but the fragment is preserved in the collection of the Science Museum in London: https://collection.sciencemuseumgroup.org.uk/objects/co62245/babbages-analytical-engine-1834-1871-trial-model-analytical-engine-mill

6. Multiplication will generate only a single-precision 30 digit result

- Many of Babbage’s Plans generate a double-precision result from multiplication that is stored on two axes, so that (1) all product digits are always retained, (2) overflow cannot happen, and (3) the result can be used as a double-precision dividend for division by a single-precision divisor.
- But doing so requires complexity which we think is not worthwhile for our smaller engine.
- We will instead detect overflow during single-precision multiplication and signal it to the operator, which is a scheme that Babbage proposed to use in several other cases.
- This is not an uncommon limitation for later computers. Having a single-precision multiplication result matches the model of integer arithmetic implemented a century later by most programming languages. When you multiply two integers in Fortran or C, you don't get a double-precision result that can then be used as the double-precision dividend for a division. It is true that in that environment there is the option of switching to floating point for greater range. But their integers don't hold the equivalent of 30 decimal digits, so the need for floating point is more pressing. And anyway the early floating point formats, like for the IBM 704, had a maximum single-precision exponent of only 2^{128} (10^{38}) so they were not that much better than 30-digit integers. We would be quite happy if our Analytical Engine had computational capability similar to the IBM 704 from 1954.

7. Implied decimal points are implemented

- In some Plans Babbage allowed the user to specify an implied decimal point that was “d” digits to the left of the least significant digit.
- The source of “d” isn’t always clear. In Plan 13 it is assumed to have been set manually by the operator on a special figure wheel of a particular Mill axis.

- Implementing this requires additional barrel microcode complexity, and additional time for division to generate more quotient digits. For multiplication, the required right shift of the product might overlap with other operations and take no extra time. There is no impact on addition and subtraction.
- We initially decided against implementing this. But the desire to execute, unmodified, the famous 1843 Lovelace program to compute Bernoulli Number B7 caused a reversal of the decision.
- The alternative is to require the programmer to scale up operands appropriately, and to use the explicit shift ("step up" and "step down") instructions when necessary.

8. Square root is not implemented

- This was a feature in early Plans, but by Plan 16 even Babbage had discarded it, complaining that "It doesn't do cube roots."

9. Approximate multiplication and division is not implemented

- In some Plans Babbage proposed versions of multiplication and division that were faster because they operated on fewer digits. He thought they would be useful for certain mathematical approximation methods, which typically do not require full precision in the early stages.
- If this requires more than a trivial modification to the full-precision algorithms, it is not worth the additional complexity. But if it is simple, and if it speeds up notable multiplication and division, it may be worth doing.

10. We do support pipelined streamed addition/subtraction

- In some Plans Babbage allowed for a succession of variables coming from the store to be added to an accumulating value in the mill. All, some, or only the final value of the sum is written back to the store, as directed by the cards.
- Babbage didn't, unfortunately, provide modern-style registers -- or even a single accumulator – to retain data in the mill between operations. Hence computing the sum of a sequence of values without streamed additions is painfully slow, since each intermediate value has to be written back to the store and then reread. A simple addition or subtraction of numbers in the store takes 6 cycles of 20 time units each, or about 19 seconds. Adding ten numbers would take 190 seconds.

- With pipelined streaming, adding ten variables takes about 44 seconds instead, a 4.3x improvement. That is a significant enough savings for a common operation to warrant the extra complexity.
- This is also a stellar demonstration of "microcode" in Babbage's design that is distributed among independently operating barrels.

11. The store

- The store will contain 25 axes. Each axis has 31 cages[6], and each cage holds two digits, for a total storage of 50 signed 30-digit numbers in sign-magnitude format.
    - ENIAC, which after its 1947 conversion was the first stored-program electronic computer to solve serious research problems[7], had only 20 10-digit numbers. We will have over twice that in both metrics.
- The store is arranged linearly, with transfers to the Mill effected via straight geared racks.
- Readout from the Store is not destructive, because returning the racks to their home position after "giving off" and transferring the value to the Mill will restore it to the Store axis. (There is a suspected timing problem in Babbage's design for this that will need to be resolved.)

12. On the engagement of axes

- We will investigate having axes engage with racks and with other axes by being raised or lowered, which also selects one of the two numbers. There is then no need for additional axes with pinion gears.
- This was proposed by Babbage in Plan 28, although we are not also adopting that Plan's half-zero scheme for giving off.
- The feasibility of doing this will depend on the geometry of the machine layout in the various areas. In the Mill, additional columns with small pinion gears may be necessary to engage figure wheel axes that can't be put close enough to another to allow lifting to effect engagement with their intended partner.
- Raising and lowering axes to effect engagement may also complicate the support structure that keeps digits wheels separated by a constant distance.

[6] A "cage" contains two or more adjacent digit wheels from unrelated numbers at the same decimal position on an axis. See Digit wheel parameters.

[7] See the description of Klara von Neumann's Monte Carlo simulation program for hydrogen fusion as described in "ENIAC in Action" by Haigh, Priestley, and Rope, MIT Press 2018.

13. The user-level instruction set

As Tim Robinson says in his treatise, “This is perhaps the least well elaborated aspect of the mature Engine, and one in which he [Babbage] showed much indecision.”

We will need to define the format and encoding for Operation, Variable, and Number cards, which together form what is today called a program. Each type of card is fed as a separate connected loop to one of three card-reading prisms.

Focusing on the Operation cards, we propose the following for what we would today call the “machine language instruction set”:

1. algebraic addition
2. algebraic subtraction
3. algebraic multiplication, producing a 30-digit result
4. algebraic division, using a 30-digit dividend and divisor
5. ascertaining if a variable is zero
6. ascertaining if a variable has a positive or negative sign
7. reading a number card into storage
8. stepping up (multiplying by 10)
9. stepping down (dividing by 10)
10. turn cards forward (forward jump; see S/2/2.339 in the Babbage archive)
11. turn cards back (backward jump; see S/2/2.339)

Except for the change to single-precision for multiplication and division, these are all operations that were proposed by Babbage for Plan 27.

Pipelined streamed addition or subtraction doesn’t require an additional instruction type because it is encoded on the normal addition or subtraction operation card. We can choose one of the two possible schemes Babbage described:

1. The operation card (or a Store location) contains an index which indicates the number of operands to be operated on, each fetched using a new variable card. When all the operands have been processed, a second addition or subtraction operation card then orders the result to be written to the store. The semantics of addition and subtraction instructions are thus context sensitive.
2. A new operation card is read with each variable card, and it indicates when the accumulated result should be written to the store. In this case multiple partial results could be written.

The critical new features in Plan 27 were the tests (5 and 6), which make the Analytical Engine a universal Turing-complete computer. The design, however (quoting Tim Robinson), "is not very clear regarding the actions that will be taken depending on the result of the test." It is most likely that small indices on the operation and variable cards indicate how many of those types of cards should be skipped, and whether in the forward or backward direction, if the condition being tested is satisfied.

14. Digit wheel parameters

The digit wheel is the basic storage unit for one decimal digit. Its geometry affects everything in the engine.

- considerations for the digit wheel size:
    - In all of Babbage's designs through Plan 25 the diameter of the digit wheels were 4", although some isolated wheels on certain axes in the Mill were as large as 8".
    - In Plans 27 and 28 he changed to 3" wheels for the store and table axes.
    - In plan 30 he reduced it further, to 1-1/4".
    - Smaller wheels are better because the mass (proportional to the square of the diameter) is reduced, and because it allows axes to be placed closer together, which in turn reduces the required size and mass of racks and the supporting framework.
    - But smaller wheels are worse because they require more precision, and even small accidental motions can cause derangement.
- considerations for interleaving the digits of multiple numbers on one axis:
    - All number columns in the Mill and the Store must have the same inter-digit spacing.
    - The inter-digit height required by the anticipating carriage mechanism leaves room for 2 or more digit wheels from separate numbers to be stored at each of the 30 digit positions along an axis.
    - Plans 1 to 13 used 3-5/8" high cages that hold digits from 4 different numbers.
    - Plans 14 to 27, thanks to a more compact mechanism Babbage devised for the anticipating carriage, used 2-7/8" high cages holding digits from 2 different numbers.
    - Plans 28 used 1-5/8" cages with 2 numbers.
    - Plans 30 and beyond proposed using pressure die casting or metal stamping to achieve cage heights of 5/8" or even smaller.

- We propose to adopt the Plan 27 intermediate wheel size and its corresponding cage height, digit spacing, and digit speed. The complete set of wheel parameters is thus:
    - DP ("diametral pitch") of 10, which means 10 teeth per inch of diameter
        - That sets the size of the teeth and the width of locking bars
    - 3 inch diameter wheels, so 30 teeth/wheel
    - 3 copies of 0..9 on each wheel, which means a 0.314 inch "unit of space" between digit positions, and one tooth per digit position
    - 2 inch/second maximum surface speed, which means a 0.157 second "unit of time" for single digit position movements
    - preliminary investigation indicates that the cage height could be as small as 2.3 inches
    - 2 digits per cage
    - a 30-digit column plus sign that holds two numbers is then about 6 feet high
        - There is mechanical overhead, mostly underneath the digit columns, that will occupy an additional 3 or so feet.
        - For maintenance it may be necessary to allow enough ceiling height to pull an axis out the machine vertically. That could require about 16 feet of ceiling clearance above the floor.
- The same digit wheel configuration is also used in the Store, which therefore holds twice as many numbers as it has axes.
- In the Mill, some of the axes may only need or require only 1 digit per cage.
- In general we will endeavor to use this as the only figure wheel format in the engine, consistent with principle #8.

15. Powering the engine

In the interest of making the museum experience as historically authentic as possible, it would be gratifying to be able to power the machine with a live steam engine.

The steam engine would need to be of the condensing kind -- not to improve efficiency, but so that steam is not vented indoors. The boiler would be modern, electric[8], and located outside the demonstration area.

[8] Like, for example, https://catalog.electrosteam.com/item/process-steam-generators/lg-10-40-electric-steam-generators/lg-10-h

It is not clear if we can pull this off. Steam-operated equipment is generally subject to extensive safety, inspection, and even training/certification regulations. Since our boiler would be less than 400,000 BTU/hour it would be exempt from many of the regulations, but it would still likely require a waiver of normal building codes to secure a permit for installation.

Providing power in the 2-5 HP range should be easy, but at this point we don't know what will be required.

Using compressed air to drive an air motor could be an appropriate and less problematic substitute for a steam engine. Practical electric motors were invented around 1840 and were not in general use until the 1880s, so using one of those, if we had to, would be an anachronism.

16. Units of measure

- Although most modern mechanical designs are done in metric, we will use U.K. imperial units in deference to the 19th century convention of Babbage's time. It is, thankfully, still common enough as to not cause raised eyebrows from our suppliers or fabricators.
- Drawings will be dimensioned in inches and decimal fractions of an inch.
- Unlike Babbage, we get to take advantage of standardized parts:
    - For screws and bolts we will use imperial diameter gauge (4, 6, 10, 12, 1/4", 5/16", 3/8", 7/16", 1/2") with pitch in standard threads per inch (40, 32, 24, 20, 18, 16, 13).
    - We will use the now ubiquitous 1918 standard UNC/UNF (Unified National Coarse/Fine) 60° thread profile, not the 1841 Whitworth 55° profile that Babbage would have used.
    - For stock metal we will buy inch-denominated parts, as in *0.125" x 1" x 72" 1144 alloy steel rectangle bar.*
    - We will try to use only standard 20° pressure angle involute spur gears, chamfered on the face edge when necessary for axial mating.

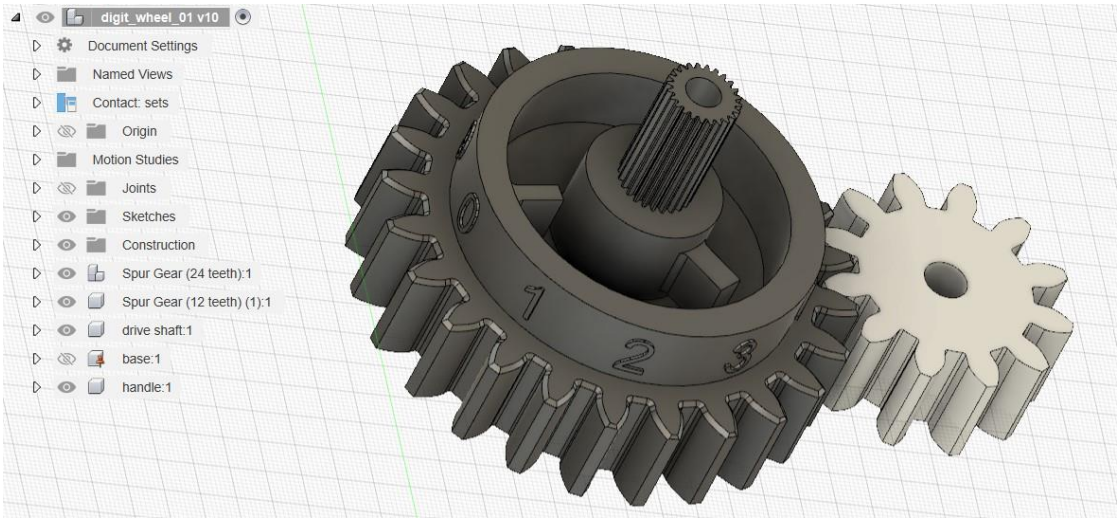
17. Design decisions yet to be made:

- the plan view specifying the arrangements of axes, racks, microcode barrels, etc.
- the microcode barrel structure: how many, the distribution of functions, and the assignment of studs to operations
- the detailed microcode barrel programs

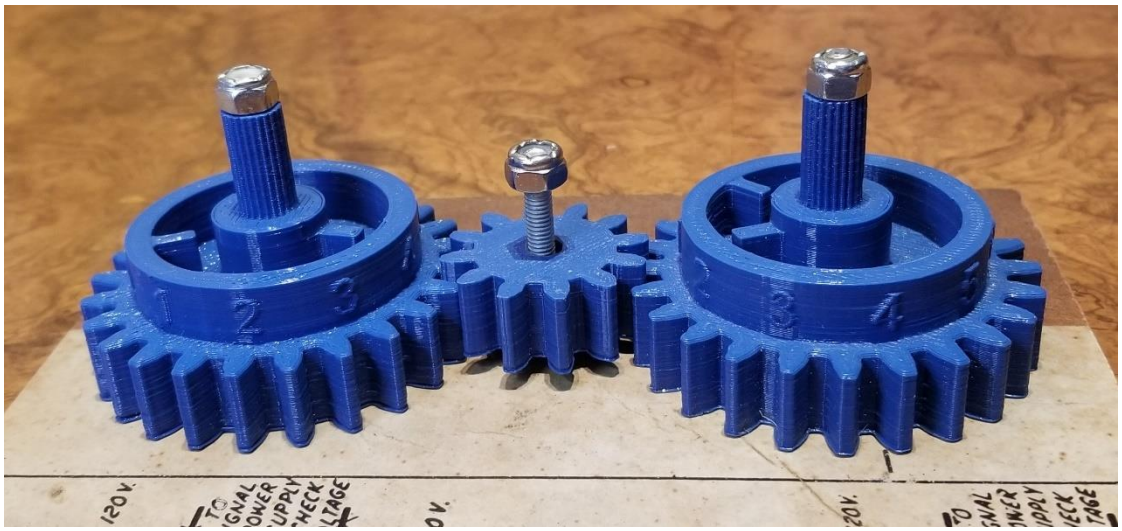
- the linkages for transmitting the bits in the current microcode word to the parts of the machine controlled by it
- an outline of the support structure
- the choice of material for each part
- the details of providing power
- encodings for operation, variable, and number cards
- any I/O we want aside from the programming cards as input and the bell as output
- provisions for modern high-tech auxiliary equipment, like cameras for reading out the values currently in the Store or in the Mill
- features that should be added specifically for debugging and status monitoring
- *more?*

The CAD system

For 3D printing experiments I first used Autodesk Fusion/360, which is the modern alternative to the venerable and groundbreaking 40-year-old AutoCAD.



I then fabricated prototype designs out of PLA polyester with a LulzBot TAZ 6 desktop 3D printer, https://lulzbot.com/store/taz-6. The results are surprisingly good, even with no after-print finishing.

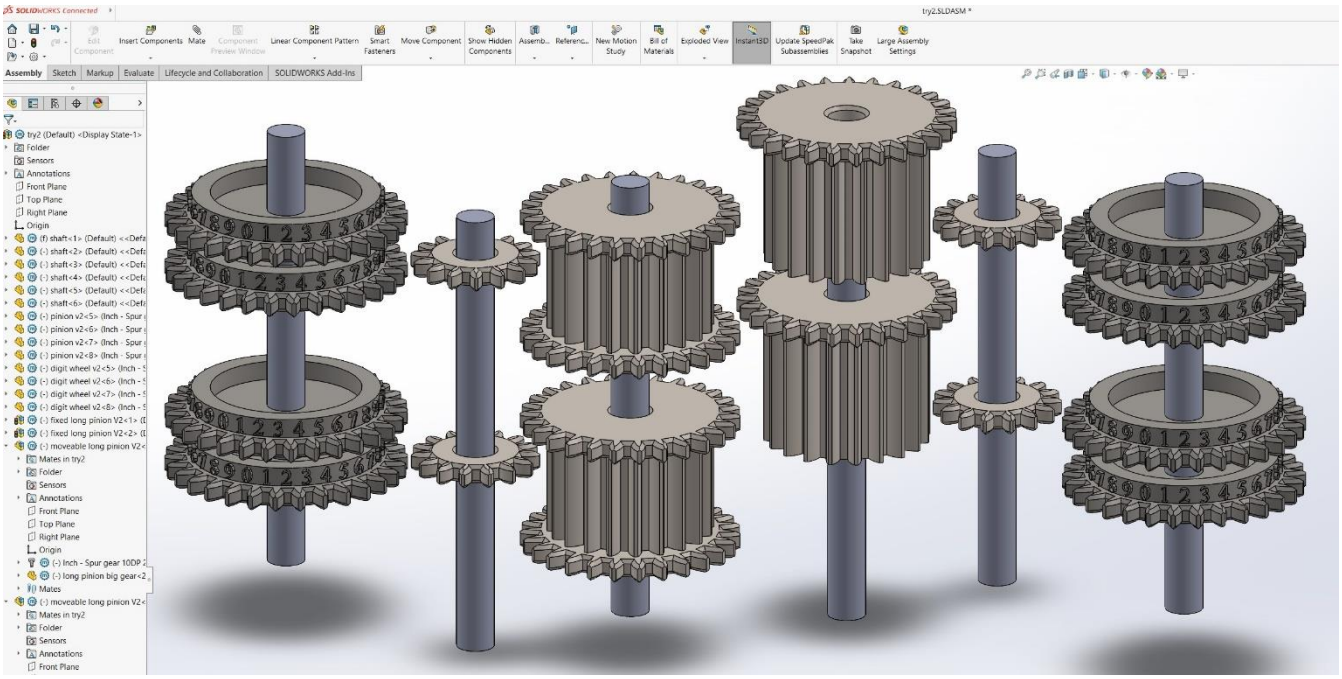


If we decide on using 3D-printed plastic parts like these for the finished engine we would switch to a more durable plastic, like Nylon or one of the new Igus abrasion-resistant composites[9], and choose appropriate surface finishing techniques. We may also want to use a two-nozzle head

[9] https://www.igus.com/3d-print-material/3d-print-filament

that would allow us to print some features, like the raised lettering on the digit wheels, in a contrasting color.

I have since switched to experimenting with SolidWorks Connected from Dassault Systèmes, because it is generally considered a more professional CAD system and has better simulation facilities. Currently under development is a proposed digit stack configuration, a version of Babbage's clever anticipating carriage, and parts of the Mill.



Simulation opportunities

In evaluating the machine that is implied by these design decisions, we are privileged to have a tool Babbage didn't: a computer that can simulate a computer that has yet to be built. We can take advantage of that at many levels:

1. An instruction simulator of the programmer-accessible instruction set and the machine operation focuses on the preparation of operation/number/variable cards and the resulting output. The details of instruction execution are not modeled, although rough timing may be. It is useful for experimenting with and debugging programs.

   This was the level of John Walker's web-based JavaScript "Analytical Engine Emulator" that he wrote over six years ago: https://www.fourmilab.ch/babbage/contents.html. It is not currently an active project, and it doesn't use the exact instruction set we will implement.

   I have recently implemented an assembler, disassembler, and simulator in Python for The user-level instruction set we are proposing. It generates the Operation, Variable, and Number card stacks that the engine would use. Test programs written and simulated so far are for the computation of Bernoulli B7, B5, and B3, Fibonacci Numbers, and the Greatest Common Divisor using Euclid's algorithm. See Appendix 3: calculating Bernoulli numbers and Appendix 4: additional simulated example programs.

2. A timing simulator understands the algorithms used for internal operations with just enough detail to compute how long things take. It does not model the mechanical components. It can be used to predict the performance of the machine and help in choosing between alternative schemes.

   I have written a timing simulator like that in Python for the basic operations and used it for the analyses in Appendix 2: multiplication and division timing charts and Appendix 3: calculating Bernoulli numbers.

3. A component simulator models the various functional components and internal operations of the mill and store. This is similar to the Register Transfer Level (RTL) architectural description of modern electronic computers.

   Components include number wheels with and without a carriage mechanism, axes with one or more sets of wheels, axes with only interconnecting pinions, microcode barrels, racks, card reading stations, etc. Operations include engagement of axes to other components, giving off a column of numbers with or without stepping, advancing a barrel, propagating carry prediction, etc. It models only function, not the geometry of individual mechanical parts or how they are interconnected. It is useful for understanding and validating the algorithms and their detailed internal sequencing.

I have written such a simulator in Python that so far implements about one third of a complete Analytical Engine. It includes an assembler and disassembler for the microcode programs that are coded into the position of studs on the barrels. The simulator operates at the resolution of the 0.157 second “basic unit” of time, and attempts to mimic the inherent parallelism that the mechanical computer will have. The only test so far has been of a barrel microprogram that implements unsigned multiplication. See Appendix 5: component-level simulator.

4. A mechanical connectivity simulator models the movable mechanical components and how they interact with each other to create chains of motion. It does not model the physical shape or location of components, although it does keep track of their position within a range of possible articulation. It can detect movement which is insufficient or excessive to achieve a desired outcome, but it does not in general detect interference between parts. It is useful for debugging the mechanical details of the algorithm implementations.

   Tim Robinson has written a simulator like this in Ruby, which he successfully used to model details of Babbage’s Difference Engine #2. He is now applying it to some of the mechanisms in the Analytical Engine.

5. A detailed machine simulator uses the dimensioned part drawings in a CAD system to model the motion of the actual parts we want to build. It would verify both correct operation and the lack of interference. This is useful to give us confidence that the built parts will work when assembled.

   I have hinted at this direction using a "Motion Study" for an Autodesk Fusion/360 model that shows a figure wheel giving off its value. Here is a link to the video:

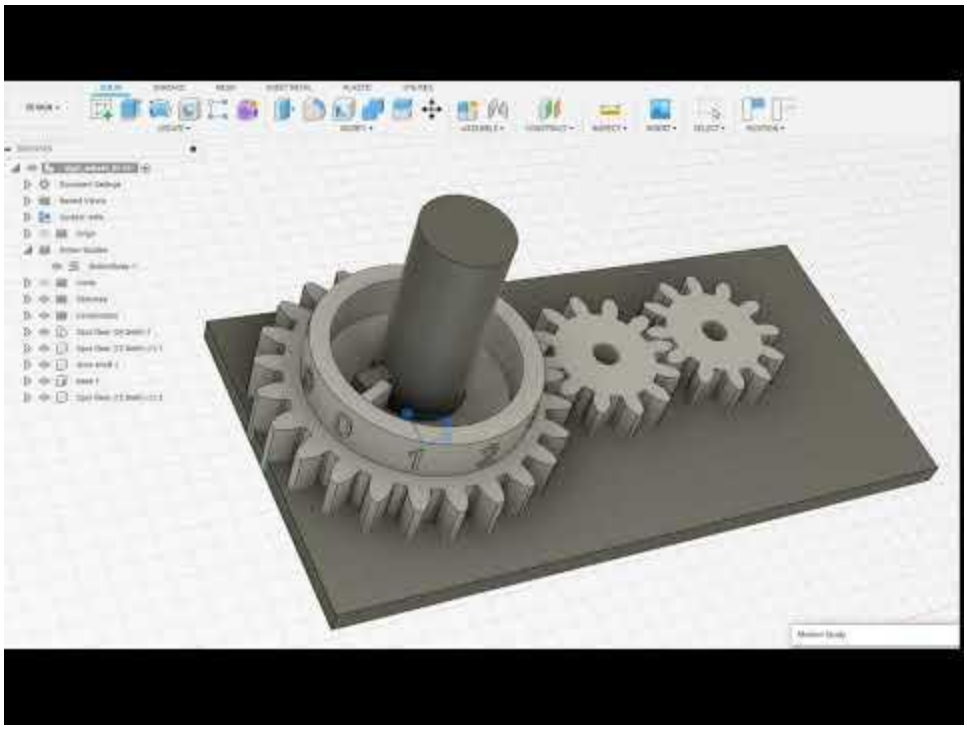


Figure 1: click to play

We don't yet know if Fusion/360 or even SolidWorks is an appropriate tool for simulating the larger assemblies. But regardless of the CAD system, the whole of the AE can probably not be modeled and we will need to choose only specific subsystems to simulate.

6. A dynamics of motion simulator adds to the detailed machine simulator a model of the forces and stresses on components. This is a feature of many of the advanced CAD systems, often using finite element analysis techniques.

Appendices

Appendix 1: proposed exhibit placard

What is this machine?

Charles Babbage – a mathematician, philosopher, inventor, and mechanical engineer – created 35 incomplete designs for mechanical digital computers he called "Analytical Engines", from 1834 until his death in London in 1871. He made no attempt to build, or even to complete the plans for, any of them.

This fully working mechanical computer is a simplified amalgam of mechanisms taken from several of Babbage's historical designs. It is not "one of Babbage's Analytical Engines", but rather "a Babbage-inspired Analytical Engine".

We included the following features that were in many of Babbage's designs:

- a central processor (the "Mill") that is separate from memory (the "Store")
- user-level program control using software stored on operation and variable cards
- program loops and conditional transfers based on computed values
- internal control using multiple parallel microprograms
- geared decimal digit wheels stacked to create 30-digit numbers
- addition and subtraction using an "anticipating carriage" for carries and borrows
- multiplication and division as primitive hardware operations
- a settable implied decimal point
- pipelined accumulation of sums

We omitted the following advanced features that are in many of Babbage's designs:

- table-based multiplication and division
- double-precision and reduced-precision results
- multiple-position shifts

Although Babbage never built any of his calculating engines, his incomplete designs provide clear evidence that he was a genius.

This operating computer, built 185 years later, provides proof.

Aside: "185 years" is the time from Plan 27 (1842) to 2027. That gives us 4 years to build it!

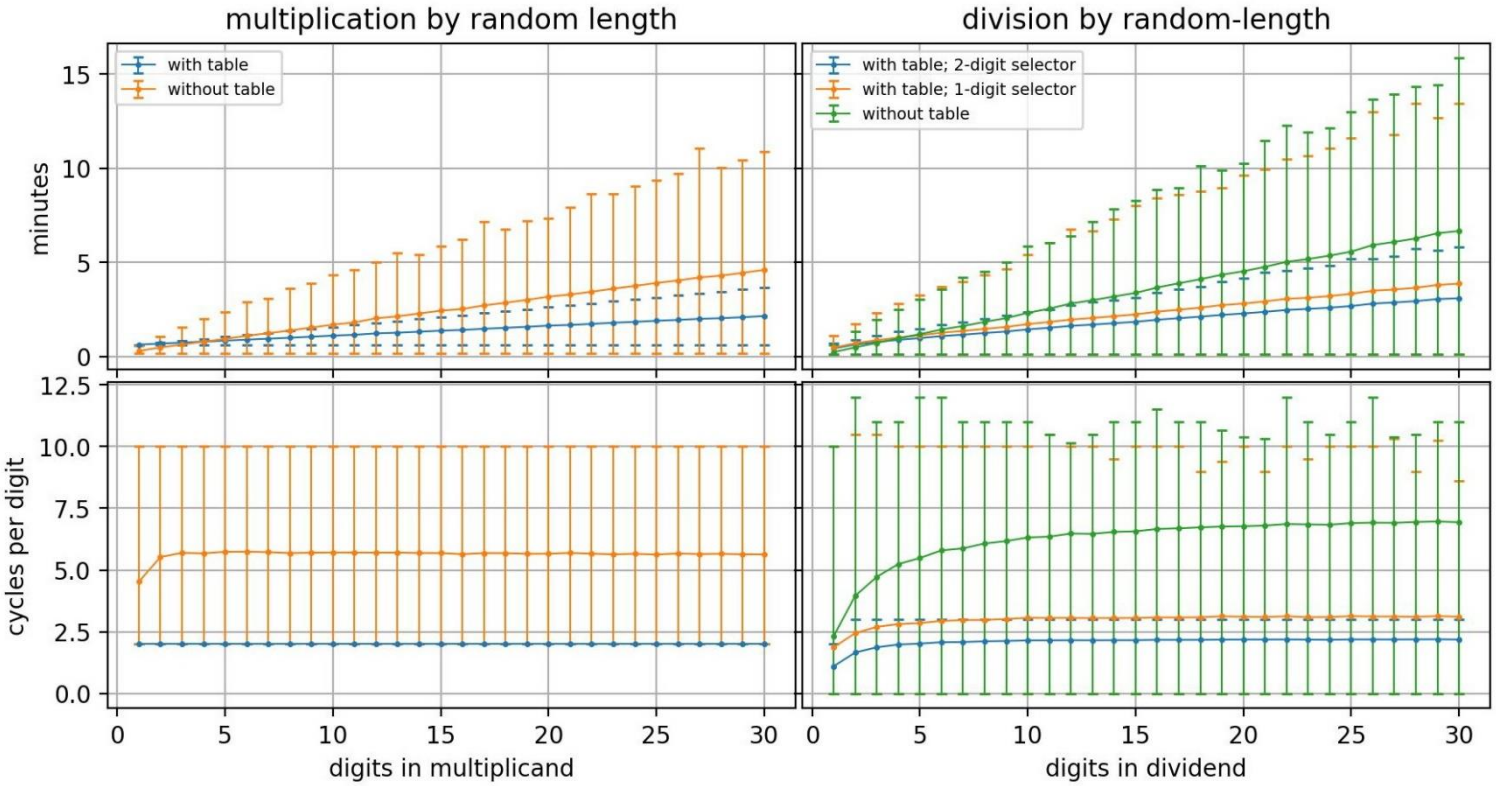
Appendix 2: multiplication and division timing charts

A simulation in Python was done to understand the advantage of precomputing the nine multiples of the multiplicand or the divisor. The charts below show the average, minimum, and maximum of the total time[10] and the cycles per digit when multiplying or dividing by a random number whose length is first randomly chosen from 1 to 30.

For extremely short multiplicands and dividends, generating the table makes the operation take longer than the simple no-table algorithm. The initial overhead is just not worth it.

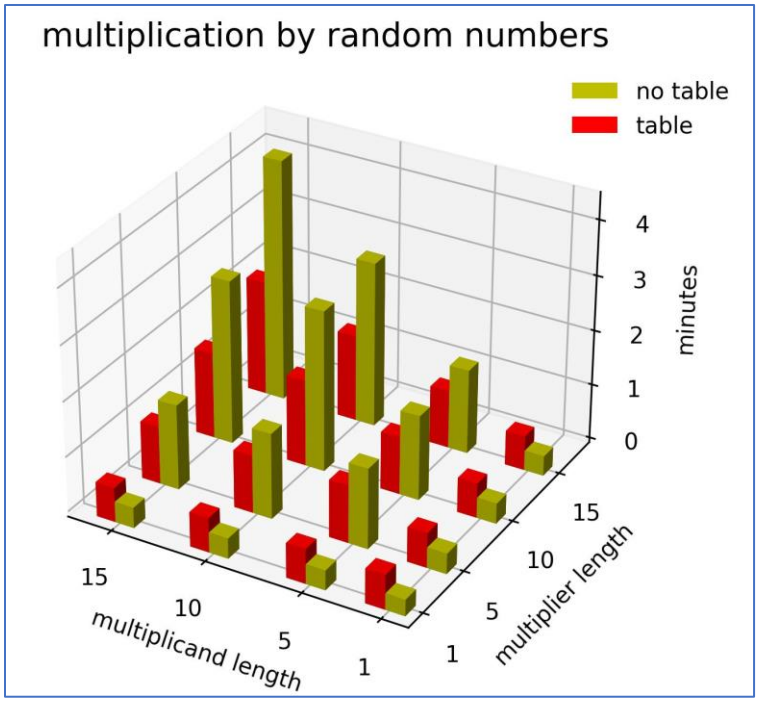
For moderately sized multiplicands and dividends, the average times are not much different. But for a few values, the non-table version is quite a bit slower.

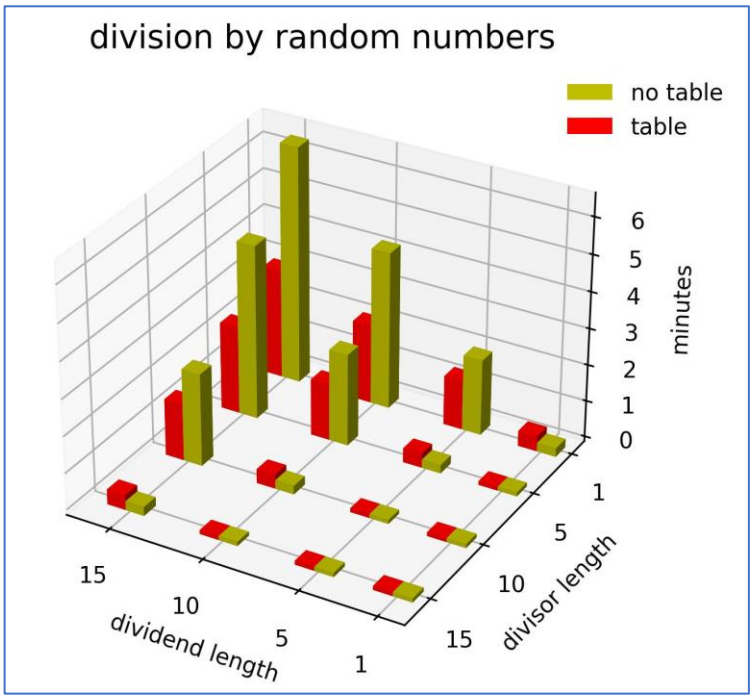
Only for large multiplicands and dividends is the table-based version almost always significantly better.



[10] The timing simulator uses several approximations that still need to be verified or made more precise, including a fixed 20-unit cycle time, and other assumptions about how many cycles certain setup and overhead operations take.

For another visualization of multiplication and division timing, I computed the average time to multiply or divide 1000 random numbers for various combinations of operand lengths. The conclusions are the same: for short multipliers, and for divisors that are long relative to the dividend length, the pre-computed tables generally don't confer a huge advantage and sometimes makes it slower.





Appendix 3: calculating Bernoulli numbers

The most famous and most detailed example of a proposed use for the Analytical Engine, described in Note G of Ada Lovelace's 1843 translation and expansion of Luigi Menabrea's paper, was for the computation of Bernoulli numbers. It would be a sublime expression of historical justice if we could demonstrate that computation with our engine.

Lovelace and Babbage never explicitly developed what we today would call the "program" for Bernoulli numbers. Instead, the paper contained a "trace" of the internal state of the machine as the computation proceeded. We need therefore to work backwards from that to the instructions that will induce that behavior. Stephen Wolfram[11] has done a helpful analysis of Lovelace's trace, although he underestimates the time it would have taken to execute. Wolfram, and other[12,13] useful pages, point out an error, probably typographical, in the fourth line of her trace.

The Lovelace computation used this recurrence relation:

$$B_n = -\sum_{k=0}^{n-1} \binom{n}{k} \frac{B_k}{n+1-k}$$

for n>0, with B_0=1.

The example she works out in detail is for the calculation of B_7, assuming that B_1, B_3, and B_5 have previously been calculated and are in the Store. The computation illustrated in the trace is this, where n=4 for the computation of B_{2n-1}:

$$B_7 = -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2} + B_3 \cdot \frac{2n(2n-1)(2n-2)}{2 \cdot 3 \cdot 4} + B_5 \cdot \frac{(2n(2n-1)(2n-2)(2n-3)(2n-4)}{2 \cdot 3 \cdot 4 \cdot 5 \cdot 6}$$

which she denotes as

$$B_7 = A_0 + B_1 A_1 + B_3 A_3 + B_5 A_5$$

The form of that expression and the trace are more general than is needed to compute just B7, because, as she explains, she is trying to demonstrate part of an imagined more general program that would compute an arbitrarily long sequence of the Bernoulli numbers:

> "…we suppose the engine to be *in the course* of computing the Numbers to an indefinite extent, from the very beginning; and that we merely single out, by way of example, *one amongst* the successive but distinct series of computations it is thus performing."

---

[11] "Untangling the Tale of Ada Lovelace", https://writings.stephenwolfram.com/2015/12/untangling-the-tale-of-ada-lovelace/

[12] "Note G", https://en.wikipedia.org/wiki/Note_G

[13] "What Did Ada Lovelace's Program Actually Do?, https://twobithistory.org/2018/08/18/ada-lovelace-note-g.html

The published trace for computing B_7 is generated by executing a total of 36 instructions. It includes two identical iterations of 11 instructions that have been unrolled to create a linear program, since it was designed to be running on the Plan 25 engine that did not include conditional branches or loops.

The trace includes three superfluous instructions to maintain a counter that isn't actually used. We could remove those, and we could also make minor performance improvements by reusing instead of recalculating some intermediate results.

But for historical authenticity I simulated the calculation of B_7 using a program that conforms *exactly* to the Lovelace trace, except for the correction of the typographical error in step 4. The program below looks like assembly language for the Analytical Engine, but is actually Python code that is executed by the instruction-level simulator described earlier.

```
    decimals=6
    scalefactor = 10**decimals
#initial Store variable values come from Number Cards that contain constants
    num (V1, 1 * scalefactor)            # 1
    num (V2, 2 * scalefactor)            # 2
    num (V3, 4 * scalefactor)            # n=4, to compute the 4th number, B(2n-1), or B7
    num (V21, int(1/6 * scalefactor))    # B1, a previously computed Bernoulli number
    num (V22, -int(1/30 * scalefactor))  # B3, a previously computed Bernoulli number
    num (V23, int(1/42 * scalefactor))   # B5, a previously computed Bernoulli number
#program: computes B7 = -A0 - A1B1 - A3B3 - A5B5
    mul (V2.R,  V3.R, [V4,V5,V6])   #step 1: 2n
    sub (V4,    V1.R,  V4)          #step 2: 2n-1
    add (V5,    V1.R,  V5)          #step 3: 2n+1
    div (V4,    V5,    V11)         #step 4: (2n-1)/(2n+1)
    div (V11,   V2.R,  V11)         #step 5: (1/2)((2n-1)/(2n+1))
    sub (V13,   V11,   V13)         #step 6: -(1/2)((2n-1)/(2n+1)) = A0
    sub (V3.R,  V1.R,  V10)         #step 7: n-1  counter=3
    add (V2.R,  V7,    V7)          #step 8: 2    copy constant
    div (V6.R,  V7.R,  V11)         #step 9: 2n/2 = A1
    mul (V21.R, V11.R, V12)         #step 10: A1*B1
    add (V12,   V13,   V13)         #step 11: A0+A1*B1
    sub (V10,   V1.R,  V10)         #step 12: n-2  counter=2
    sub (V6,    V1.R,  V6)          #step 13: 2n-1
    add (V1.R,  V7,    V7)          #step 14: 2+1 = 3
    div (V6.R,  V7.R,  V8)          #step 15: (2n-1)/3
    mul (V8,    V11,   V11)         #step 16: (2n/2)*((2n-1)/3)
    sub (V6,    V1.R,  V6)          #step 17: 2n-2
    add (V1.R,  V7,    V7)          #step 18: 3+1 = 4
    div (V6.R,  V7.R,  V9)          #step 19: (2n-2)/4
    mul (V9,    V11,   V11)         #step 20: (2n-2)*((2n-1)/3)*((2n-2)/4) = A3
    mul (V22.R, V11.R, V12)         #step 21: A3*B3
    add (V12,   V13,   V13)         #step 22: A0+A1*B1+A3*B3
    sub (V10,   V1.R,  V10)         #step 23: n-3   counter= 1
    sub (V6,    V1.R,  V6)          #step 13: 2n-1
    add (V1.R,  V7,    V7)          #step 14: 2+1 = 3
    div (V6.R,  V7.R,  V8)          #step 15: (2n-1)/3
```

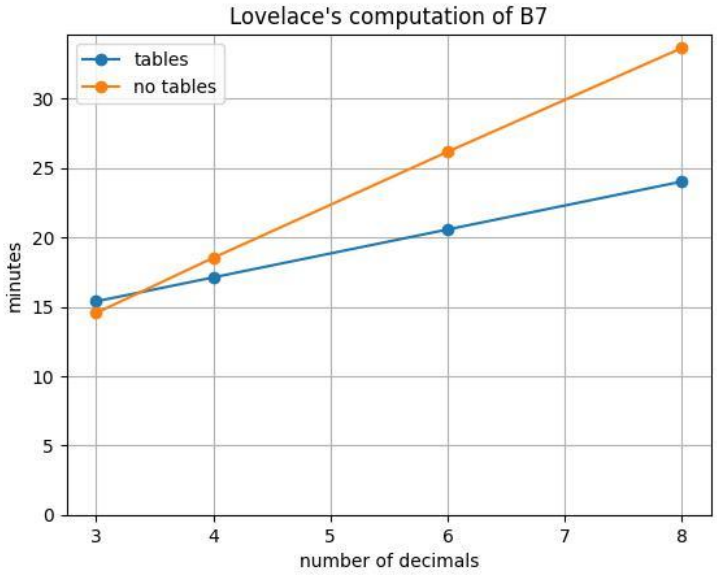
```
mul (V8,    V11,   V11)         #step 16: (2n/2)*((2n-1)/3)
sub (V6,    V1.R,  V6)          #step 17: 2n-2
add (V1.R,  V7,    V7)          #step 18: 3+1 = 4
div (V6,    V7,    V9)          #step 19: (2n-2)/4
mul (V9,    V11,   V11)         #step 20: (2n-2)*((2n-1)/3)*((2n-2)/4)= A3
mul (V23.R, V11,   V12)         #step 21: A3*B3
add (V12,   V13,   V13)         #step 22: A0+A1*B1+A3*B3
sub (V10,   V1.R,  V10)         #step 23: n-4   counter = 0
add (V13,   V24,   V24)         #step 24: A0+A1*B1+A3*B3+A5*B5 = B7
add (V1.R,  V3,    V3)          #step 25: n=5  ready to compute B9!
stop ()
```

```
43 instructions were executed in 26.8 minutes, using 6 fraction decimals, without tables
B7 is 0.033332
```

Note important constraints on this and all programs for the Analytical Engine:

- There are two types of access encoded on the variable cards: destructive readout (which Lovelace called “Zero Supply-cards”) that leaves the accessed variable zero, and non-destructive readout (“Retaining Supply-cards”) that restores the original value. We denote the latter by appending “.R” to the variable name.
- Variables used as destinations must be zero, or else the result of the computation will be added (digit-by-digit without carries!) to the current value and will be useless. Destination variables generally become zero because their last use was as an operand accessed with destructive readout, so it is important for many of the variable references that they do not include “.R”.
- Since the engine only does fixed point calculations, all the numbers are assumed to have an implied decimal point that is d positions from the right end. For running on the real engine, that number will need to be set by hand on a special figure wheel of a particular axis, which is a procedure that Babbage endorsed. It is used in multiplication to shift d digits off the right of the product, and in division to continue cycling for an additional d digits.

The program above was executed with d=6 digits to the right of the decimal point. I have also measured how long it takes for different values of d, with and without using precomputed tables for multiplication and division.



For this calculation we clearly take a performance hit for not using tables, but the results are nevertheless encouraging. This is definitely a program we will want to run sometimes.

But computing B_7 probably takes too long to use for the regular public demonstrations. So I also modeled computing two of the earlier Bernoulli numbers by truncating and simplifying the formula and the program, although for generality it is still written assuming n is a variable.

With n=3, this equation computes B_5:

$$B_5 = -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2} + B_3 \cdot \frac{2n(2n-1)(2n-2)}{2 \cdot 3 \cdot 4}$$

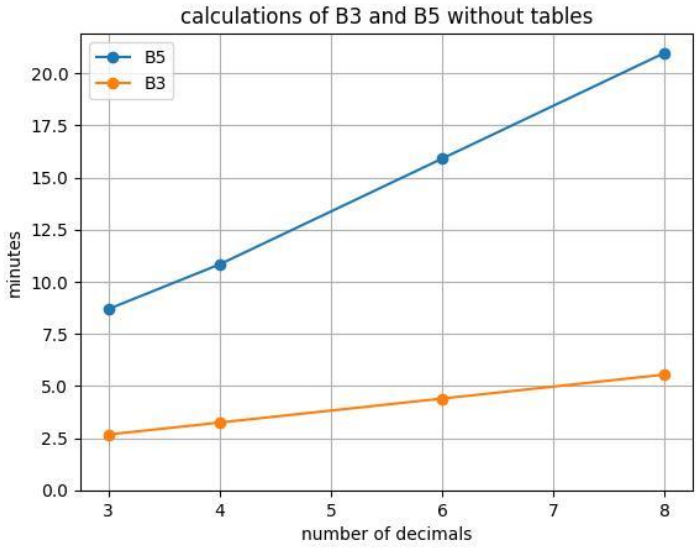
which simplifies to

$$B_5 = -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot n + B_3 \cdot \frac{n(2n-1)(2n-2)}{3 \cdot 4}$$

With n=2, the simplified version for B_3 is:

$$B_3 = -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot n$$

Here is the timing for computing B_3 and B_5.



Either computing B_3, or computing B_5 with a small number of digits to the right of the decimal point, would make an eminently practical daily public demonstration.

Of course we will create many other demonstration programs (like for the Fibonacci series) that take less time. We will also want some demonstration programs (like for the Greatest Common Divisor) that show off the conditional branching and looping.

Appendix 4: additional simulated example programs

These example programs are composed of Python statements that mimic an assembly language for the proposed Analytical Engine. They create the Operation and Variable cards that are then input to the instruction-level simulator.

Compute Fibonacci Numbers

```
num         (V2, 1)           #constant
num         (V3, howmany)     #how many numbers to compute
num         (V4, 0)           #the previous Fibonacci number
num         (V5, 1)           #the current Fibonacci number
sub         (V7, V3, V7)      #move -count to V7 as loop counter
label("loop")
add         (V4, V5.R, V6)    #compute next number
add         (V5, V4, V4)      #move current to previous
add         (V6, V5, V5)      #move next to current
add         (V7, V2.R, V7)    #increment loop counter
jmpn        (V7.R, "loop")    #jump if not done
stop        ()
```

```
52 instructions were executed in 8.3 minutes, using 0 fraction decimals
The last of 10 Fibonacci numbers is 89
```

Compute the Greatest Common Divisor of two integers using Euclid's algorithm

```
#compute the GCD of V1 and V2
label("loop")
sub         (V1.R, V2.R, V3) #compare a to b
jmpz        (V3.R, "end")
jmpn        (V3,"b_GT_a" )
sub         (V1, V2.R, V1)    #a = a - b
jmp         (V3, "loop")
label("b_GT_a")
sub         (V2, V1.R, V2)    #b = b - a
jmp         (V3, "loop")
label("end")
stop        ()
```

```
28 instructions were executed in 3.8 minutes, using 0 fraction decimals
The GCD of 27 and 6 is 3
```

Appendix 5: component-level simulator

The engine is simulated at a resolution of basic "time units", 15 or 20 of which comprise a "cycle" of the engine. The activities during a cycle are controlled by the current "verticals" on the microprogram barrels, each of which can be thought of as microprogram instruction word.

The model is of an interconnected assembly of hierarchical "components" that have internal state. Examples of components are:

- digit wheels
- stacks of digit wheels
- axles that go through one or more interleaved stacks of digit wheels
- anticipating carriage for an axle
- barrels, and the studs on barrels

Components are created with simple Python function calls:

```
A =  Axle("A",2)  #a simple axle with 2 digits per cage
B =  Axle("B",2, withcarry=True) #an axle with anticipating carriage
```

Barrel studs are defined, and given semantics that describe their effect on the engine, using calls like these:

```
create_stud("GIVE_G_TO_C",   lambda barrel: mesh(barrel, G, C))
create_stud("SHR_C_TO_D",    lambda barrel: mesh(barrel, C, D, shift=-1))
create_stud("ADD_A_TO_B",    lambda barrel: mesh(barrel, A, B))
create_stud("SUB_F_FROM_E",  lambda barrel: mesh(barrel, F, E, subtract=True))
```

The “microprogram” on the barrels is then specified by a series of function calls that resemble traditional assembly-language statements with optional labels. The output is a matrix that indicates where on the cylindrical barrel the studs need to be placed. Here is an example program fragment that does unsigned multiplication:

```
mulpgm = program("multiply program", studnames) #create and assemble the program
mulpgm.vertical("outerloop",  SHR_C_TO_D, GIVE_C_TO_E)
mulpgm.vertical(              SHL_D_TO_F, GIVE_D_TO_G)
mulpgm.vertical(              SUB_F_FROM_E, GIVE_A_TO_D, CYCLE20)
mulpgm.vertical("innerloop",  GIVE_D_TO_A, DECR_E, CYCLE20, IF_RUNUP_EC,  "doadd")
mulpgm.vertical("doadd",      ADD_A_TO_B, GIVE_A_TO_D, CYCLE20,  "innerloop")
mulpgm.vertical(              SHL_A_TO_D)
mulpgm.vertical(      GIVE_D_TO_A, ADD_G_TO_E, GIVE_G_TO_C, CYCLE20, IF_NORUNUP_EC,  "zero_e")
mulpgm.vertical("zero_e",     ZERO_E,  "outerloop")
```

The simulated execution can produce various levels of diagnostic tracing information, or just the results at the end:

```
multiplying 123456 by 123456
done in 1547 time units, or 4.05 minutes
product is 15241383936
```

Appendix 6: document change log

V0.1	29 Aug 2023	L. Shustek	Proposal for a complete but simplified engine
V0.2	7 Dec 2023	L. Shustek	Reframe it as a “prototype Babbage-like” engine that anticipates a subsequent complete Plan xx build
V0.3	30 Dec 2023	L. Shustek	Changes from T Robinson; add explanatory footnotes, add Bernoulli example; implement implied decimals; add unconditional jumps; add link to simulation video; adjust Fusion/360 price; editorial changes from D. Wilborn; more about my simulators.
V0.4	8 Jan 2024	L. Shustek	Restructure into two halves to be more accessible to readers who are not part of the project; add the Executive Summary and sections describing more context for the project.
V0.5	4 Feb 2024	L. Shustek	Changes from G Saviers, T. Robinson; wordsmithing/reordering; don’t call it “prototype”, because we use that to refer to test pieces; more detail on component simulator; position this as a separate but collaborative project relative to Plan28.org.