BlogBooker

Low resolution pictures

From Blog to Book.

thebinac.blogspot.co.uk

Contents

1. 2014

1.1 July

Say hi to BINAC (2014-07-01 09:43)

This is BINAC.
BINAC is the world's first commercial computer ; that is it was actually made commercially and sold.
It was created by a team led by J Presper Eckert and Joseph Mauchley and turned on for the first time in February 1949.
It has 512 words of memory, 32 bits each. Input and Output is in Octal, via paper tape, though there was a hand input device for punching numbers into paper tape.
It has complete redundancy ; it was basically two machines, whose results were compared (it was quite rare for both to work).
I think it hosted the first ever game written for a proper stored program machine, a version of NIM.
They only ever made one ; it was priced at $100,000 and ended up costing $280,000, an astronomical amount in 1951. For this and other reasons, Eckert and Mauchley sold their company to Remington Rand, and it became the division responsible for UNIVAC.

Research (2014-07-02 21:33)

This is a bit of a research job, finding out how this thing works.
The Holy Grail for the BINAC is its operating manual ; I have asked about this in the few places it exists, but there doesn't appear to be a PDF copy. This was written by a fellow called Joseph Chapline, and it appears to be the first ever computer manual.
There are very occasionally some on sale, but the appear to go for $25k a time, which is a bit much really.
The most common document is a brochure http://www.studio2.org.uk/binac /brochure.pdf
This is an 9 page document with some useful stuff - it lists the whole instruction set for example - but it misses out a lot of other stuff. A BINAC has 512 words of memory, 30 bits each. Each instruction occupies 5 bits for the operation code - they were called 'orders' at this time, and 9 for the operand (an address, hence 512 words). However, each word actually contains 2 instructions - 512 instructions isn't a lot at the best of times, so by pairing instructions up, you can double the program size.
This is why the U instruction 20xxx (in octal) says 'obtain next pair of instructions from m and continue from that point' (it's an unconditional jump) and why there is an apparently pointless Skip instruction (it's for when you want an odd numbered branch)
However, digging around finds something useful. http://www.studio2.org.uk/binac/arithmetic.pdf is a mathematical paper on binary arithmetic. However, handily, the machine it uses to illustrate its sums is a BINAC. So I now know how the maths works.
This is one unusual thing about this machine. Reading another description of it, I noticed it used 2's complement arithmetic.
This was somewhat surprising. Most computers of this Era use a variety of things, but not 2's complement. One's complement is common (e.g. CDC160) which of course gives you two zeros which can cause confusion. Some

machines operate in decimal units (the Harwell machine a fellow retrochallenger is doing is like this). But this is, usefully, 1's complement.
In thinking what to do with this machine, once emulated, there are two options ; the old stuff and the new stuff. There are three 'old things' which I only know a bit about.

- Roger Mills' NIM program. Possibly the first 'game' on a real computer. Might precede draughts on Turing's Pilot ACE. I think Mr Mills is still with us (one of the problem with machines of this vintage is many of the original staff are deceased)

- C-10, a sort of machine code shorthand

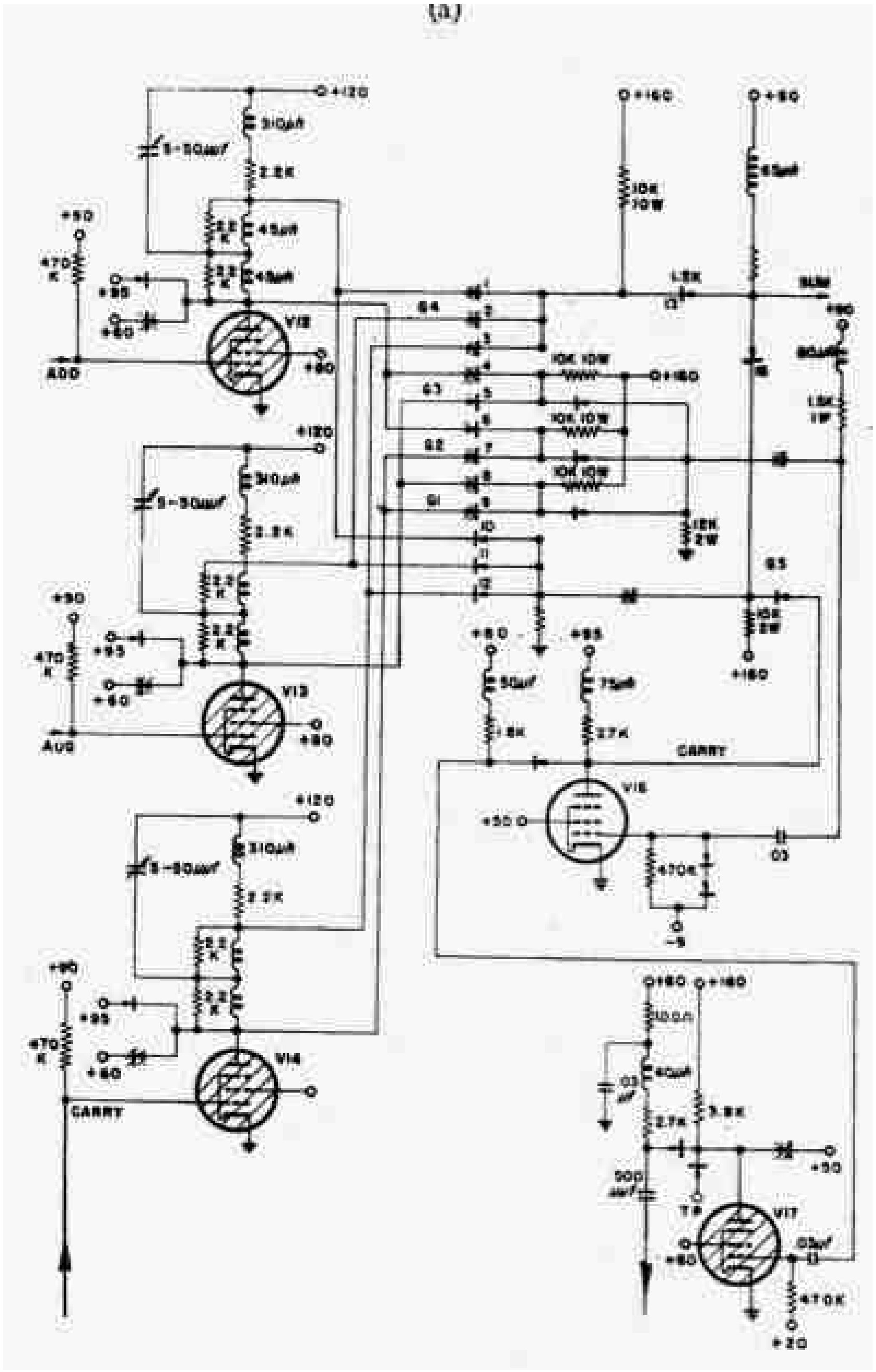
- Short Code, a second version which was extended to the UNIVAC.

I have a bit of a link to this era. I was a graduate of the University of Essex ; my personal tutor was a fellow called Tony Brooker. Close to retirement when I knew him, he was one of the people involved with these very early machines.

Github and Maths **(2014-07-03 21:00)**

I've created a github repository https://github.com/autismuk/Binac
Not much there except for some Arithmetic routines, using 31 bit 2's complement. I cheated for the Multiplication, multiplication is the same whatever, but I did implement the division algorithm from the 1950's paper so that division works as it should - for example dividing by zero does not produce an error but produces weird results.

Auerbach **(2014-07-05 20:37)**



Half Adder, with Pentodes (I think)
Or to give it its full title "*The Binac, by A.A. Auerbach, J.P. Eckert, R.F.Shaw, J.R.Weiner and L.D. Wilson,published in*

the proceedings of the I.R.E., January 1952.
I dug this out the other day ; it's great. It's full of circuit diagrams like this. No, valves are beyond me, really, but I think they operate a bit like transistor switching circuit ; this particular circuit is an single bit adder. I remember having a book about single channel radio with valves when I was about 11 (I was a deeply sad child).
The machine is entirely serial in operation, so rather than having a bank of adders it just feeds them through one after the other - you can see the carry going off the bottom and coming back on the top.
It pretty much answers all the unanswered questions, like how does the BINAC pack two instructions into a single word memory location,
More than that, it has some real BINAC code. The code shown below (well, probably) is a real BINAC program. I didn't think I'd ever actually find one. The nearest is an article "Do you want to buy a Brain ?" which is published in May 1949's "Popular Science" which has a very simple program written by Jean Bartik (a lot of these early programmers are women) to do a payroll calculation, but it's more about getting the idea across than real code. The article is really about UNIVAC, but that didn't exist in 1949, so the actual worked examples are for BINAC. There won't be much real BINAC code, you could copy memory onto an early form of magnetic tape (another first I think) but it didn't work that well, apparently.
(Note, the sequence goes ; ENIAC -> BINAC -> UNIVAC)

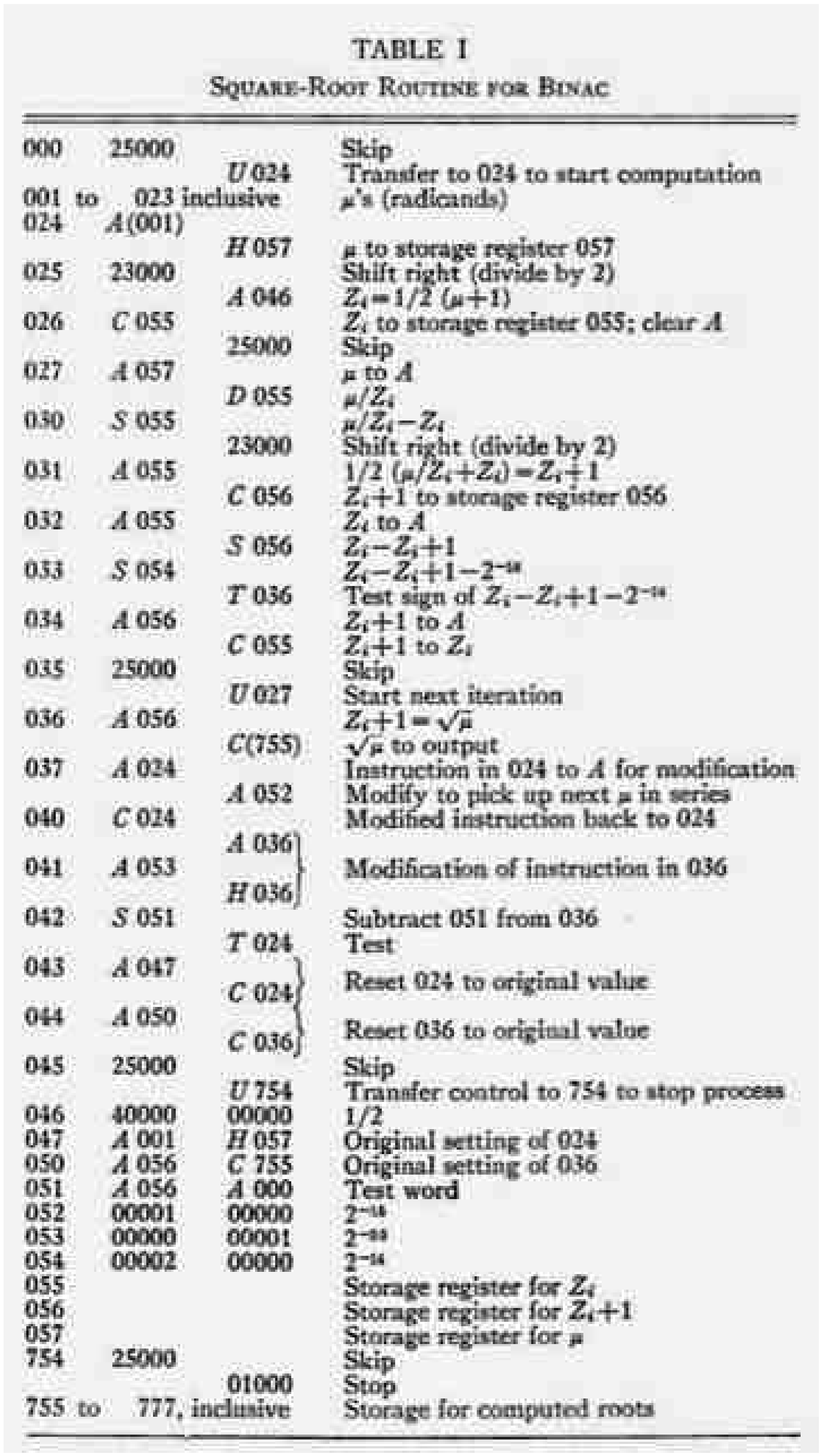


TABLE I

Square-Root Routine for Binac

000	25000		Skip
		U 024	Transfer to 024 to start computation
001 to	023 inclusive		μ's (radicands)
024	*A*(001)		
		H 057	μ to storage register 057
025	23000		Shift right (divide by 2)
		A 046	$Z_i=1/2\ (\mu+1)$
026	*C* 055		Z_i to storage register 055; clear *A*
		25000	Skip
027	*A* 057		μ to *A*
		D 055	μ/Z_i
030	*S* 055		μ/Z_i-Z_i
		23000	Shift right (divide by 2)
031	*A* 055		$1/2\ (\mu/Z_i+Z_i)=Z_i+1$
		C 056	Z_i+1 to storage register 056
032	*A* 055		Z_i to *A*
		S 056	Z_i-Z_i+1
033	*S* 054		$Z_i-Z_i+1-2^{-14}$
		T 036	Test sign of $Z_i-Z_i+1-2^{-14}$
034	*A* 056		Z_i+1 to *A*
		C 055	Z_i+1 to Z_i
035	25000		Skip
		U 027	Start next iteration
036	*A* 056		$Z_i+1=\sqrt{\mu}$
		C(755)	$\sqrt{\mu}$ to output
037	*A* 024		Instruction in 024 to *A* for modification
		A 052	Modify to pick up next μ in series
040	*C* 024		Modified instruction back to 024
		A 036	Modification of instruction in 036
041	*A* 053		
		H 036	
042	*S* 051		Subtract 051 from 036
		T 024	Test
043	*A* 047		Reset 024 to original value
		C 024	
044	*A* 050		Reset 036 to original value
		C 036	
045	25000		Skip
		U 754	Transfer control to 754 to stop process
046	40000	00000	1/2
047	*A* 001	*H* 057	Original setting of 024
050	*A* 056	*C* 755	Original setting of 036
051	*A* 056	*A* 000	Test word
052	00001	00000	2^{-16}
053	00000	00001	2^{-33}
054	00002	00000	2^{-14}
055			Storage register for Z_i
056			Storage register for Z_i+1
057			Storage register for μ
754	25000		Skip
		01000	Stop
755 to	777, inclusive		Storage for computed roots

Pointless aside.

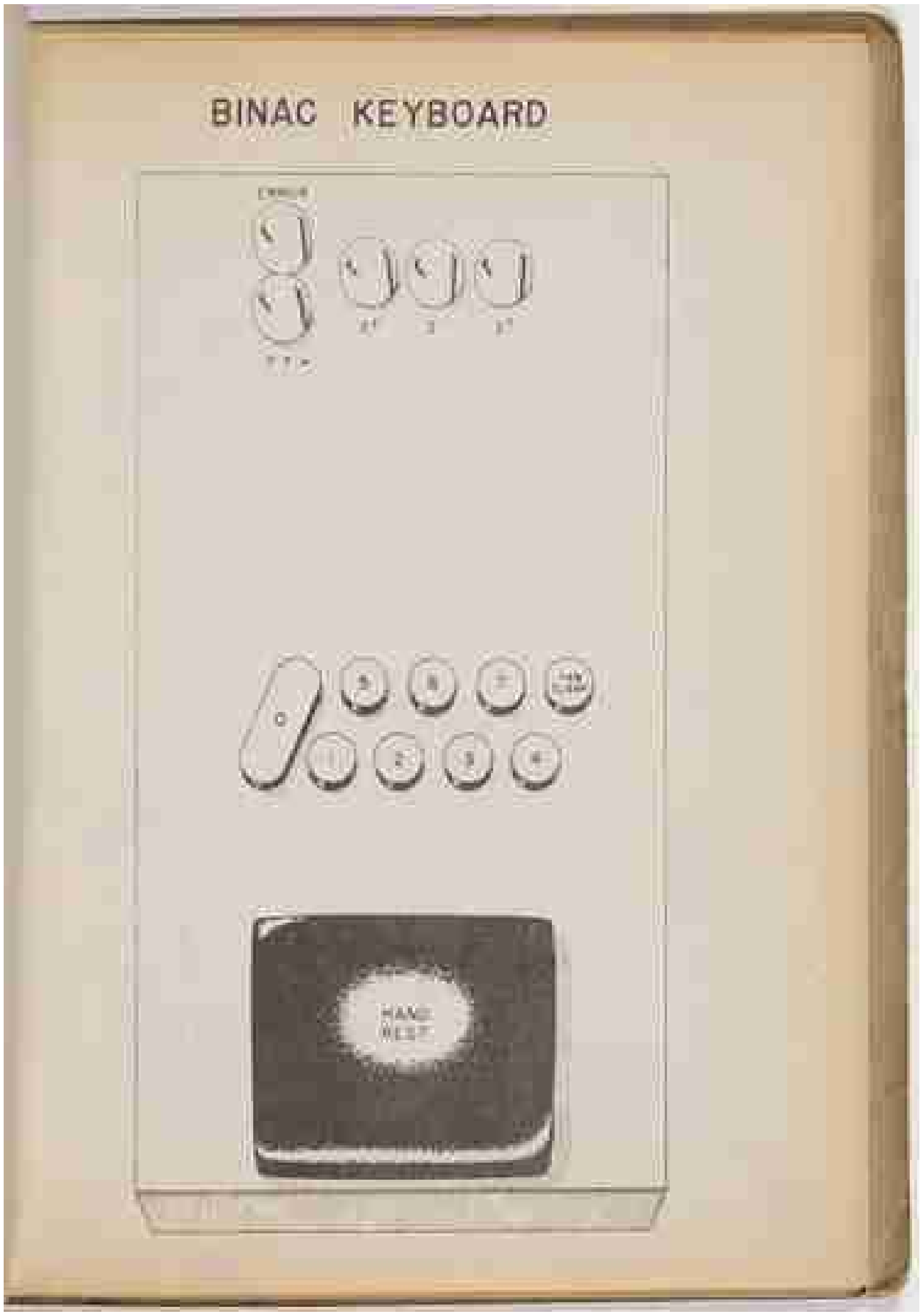
Anyone notice how dumb modern stuff is by comparison. This Popular Science article has valve circuit diagrams explaining how logic gates work, explains binary, mercury memory tanks and things like that.

I remember when I was a kid, there was a Ladybird Book (for non UK readers, this is a series of fiction and non fiction

aimed at under 11s) called ”How to make a transistor radio” and it does, with regenerative feedback, and it explains it as well, properly, like an Electronic engineer would. I still have it.
Today all people do is Powerpoints ……
I sometimes think my early days with an SC/MP Introkit - 256 bytes of RAM and a calculator hex keypad - were actually a huge advantage. Perhaps writing BINAC code, or something like PDP/8 (I used one at University, briefly) should be compulsory ?

How to train your Binac **(2014-07-06 13:25)**

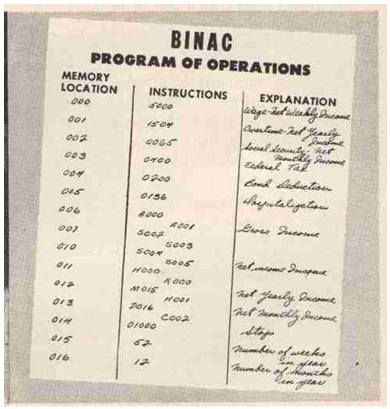
Dragons are easier, let’s put it like that.



From Chapline’s manual, the
only page I have.
This is a picture of the Binac Keypad - it is programmed in Octal. To program it you convert your program into Octal, and type it directly into memory.
It uses the Control Counter to do this - this is what BINAC calls the program counter. Typing in octal digits here copies it into a temporary register, and when you are complete it copies it into memory and goes to the next instruction.
To help it prints out the octal codes on the printer as you type it (the printer is very limited, it *only* prints Octal digits out).
This is where you do most of the work, hence the hand rest at the bottom. If you make a mistake you can press the TAB/CLEAR button and start again. (This picture is I think physically correct from what I can see from photographs).

The other commands control the operation of the computer. The main control is a rotary switch which has five positions, COMPUTE, FILL, VERIFY, EMPTY and CLEAR. There is single step switch, for executing one instruction at a time, and a breakpoint switch so you can interrupt a running program (there is a breakpoint instruction in the instruction set). You can also manually force it to stop using the ”Manual” button. Then there is a button called ’Start’, which starts the program, and a program called ’Manual Clear’ which I *think* clears the program counter.
As far as I can figure out, COMPUTE is the running the program mode, FILL allows you to enter data, VERIFY allows you to check it, I guess, but I have no idea how, INITIAL CLEAR may reset the computer completely - another guess.
Some things are difficult to work out. EMPTY is actually the printing code - if you switch it into this mode it dumps the memory between the program counter and the end of memory (777 octal) out to the printer - this is the only output available, a memory dump in octal.
This explains the odd code at the end of the square root program, where most of the code is between 024 and 047. It puts the data at the start of the program, which is easiest to change. Then once the run is complete it jumps to 754 and executes 25000 (Skip) and 01000 (Stop), so when it actually stops the program counter/control counter is at 755, which is the first word of the result - so you run the program and switch to empty, and the contents of 755 - 777 are pushed out to the printer.
As far as I can tell, there isn’t any way of changing the control counter manually, so to put code in at 754 you either type code in until you reach 754 (unlikely) or more likely you put a U754 instruction (an unconditional jump) in 000 and run that as a single step instruction , after that PC/CC will be 754 so you can switch to FILL mode and enter the code.
Some of this is described, some of it is workable out from the circuit/logic diagrams and some is sheer guesswork. I would guess Verify somehow locks the system out so you can ’single step’ through memory without actually executing anything so you can see what is in it. Initial Clear, could be a hard reset, but seems a bit pointless. Perhaps it resets the timing - because it’s a serial machine information goes round in streams which saves on valves (this has far fewer than ENIAC) but does make timing more difficult.

Code Tools (2014-07-09 20:26)



BINAC

PROGRAM OF OPERATIONS

MEMORY LOCATION	INSTRUCTIONS	EXPLANATION
000	5400	Wage-Net Weekly Income
001	1504	Overtime-Net Yearly Income
002	0065	Social Security-Net Monthly Income
003	0400	Federal Tax
004	0200	Bond Deduction
005	0156	Hospitalization
006	A000	
007	5002 A001	Gross Income
010	5004 S003	
011	H000 S005	Net income Income
012	M015 R000	
013	D016 H001	Net Yearly Income
014	01000 C002	Net Monthly Income
015	52	Stop
016	12	Number of weeks in year Number of months in year

This is the other bit of code I’ve got to run on the BINAC - this is a ’pretend’ bit from the Popular Science magazine.
To convert this into some sort of useable code, I’ve written a very simple assembler, and a disassembler, more to test the assembler than anything else, and this is now in the github repository.
Hopefully, soon, I will actually run some real code. I’ve got the Square Root code successfully assembled, and I’ve figured out how it works pretty much, so it should be a fairly sound test bed for an emulator.

---

gads

BlogBook v0.5,
LaTeX 2_ε & GNU/Linux.
`http://www.blogbooker.com`

Edited: October 16, 2015