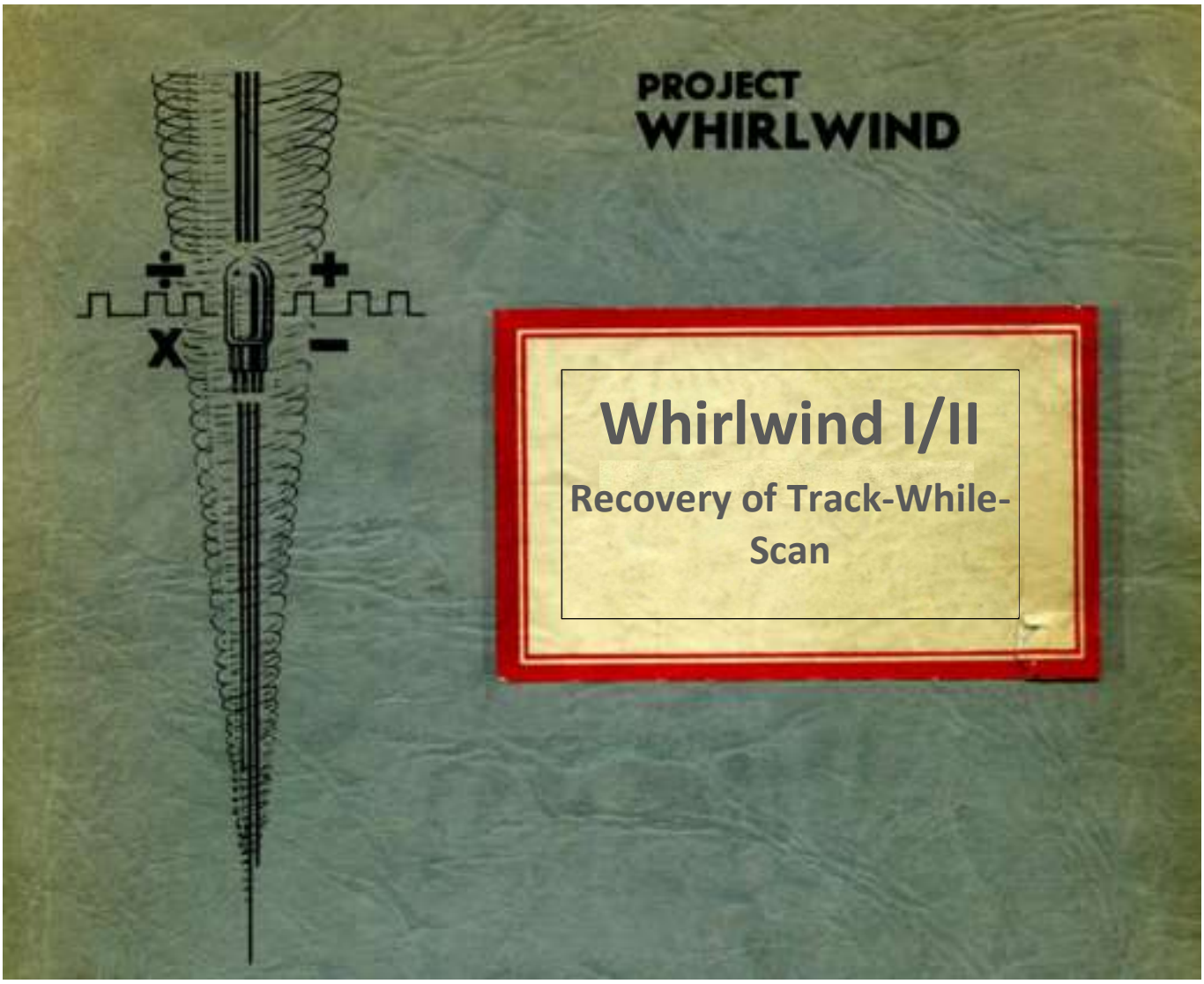


Version 1.0 Draft Notes

Guy C. Fedorkow, Apr 23, 2021

Table of Contents

1. Track-While-Scan Background

The Whirlwind project became interested in air defense in 1950[1], and by end of 1951, had enough of a working computer to run a demonstration of a key element of a real time air defense, an ability to follow several aircraft with the computer, and guide an interceptor towards a target.

[fill in more timeline from biweeklies if possible]

This was a complicated project, carried out in cooperation with the Air Force Cambridge Research Center, using the radar station at Bedford MA's Hanscom Field. Given that both Whirlwind and the radar station were essentially immovable objects, the project required development of a digital data link over telephone lines to get radar readings from the antenna to the computer.

Possible Firsts:

- This must be among the first 'real time' digital computing applications, where the computer was used to interact with real-world events as they happened, as opposed to the usual scientific calculation where they need an answer, but the result doesn't depend on keeping in sync with events as they happen.

[1] Redmond and Smith, Project Whirlwind, The History of a Pioneer Computer, pg 174.

- The program also demonstrated what would come to be known as interactive graphics, where an operator used a ‘light gun’ to select targets from those displayed on a CRT.
- I wonder if this was also a near-first for data-over-phone-lines?

For background on the Whirlwind computer, see G. C. Fedorkow, "Recovering Software for the Whirlwind Computer," in *IEEE Annals of the History of Computing*, vol. 43, no. 1, pp. 38-59, 1 Jan.-March 2021, doi: 10.1109/MAHC.2020.3048815.

https://ieeexplore.ieee.org/document/9312479

More background material on Whirlwind software recovery can be found at

https://www.historia-mollimercium.com/whirlwind/

For the impatient, you can see a short demo of the program at

https://www.historia-mollimercium.com/whirlwind/Track-while-scan-Apr-23-2021.mp4

This work has been carried out in collaboration with the Computer History Museum and the MIT Museum, and I want to express my thanks here for many helpful interactions with Debbie Douglas, Dag Spicer, David Brock, Dave Walden and many others.

1.1 Source Material

Whirlwind team member David Israel is credited with creating the software that analyzed the incoming radar data.

While it’s clear that this software went through considerable evolution, Israel did submit a complete status report on the work, filed as M-1343, as it stood at a point in time in December 1951.

http://www.bitsavers.org/pdf/mit/whirlwind/M-series/M-1343_Description_of_Basic_Track-While_Scan_and_Interception_Program_Dec51.pdf

Memorandum M-1343 Page 1 of 51

Digital Computer Laboratory
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

SUBJECT: DESCRIPTION OF BASIC TRACK-WHILE-SCAN AND INTERCEPTION PROGRAM

To: C. H. Wieser

From: David R. Israel

Date: December 3, 1951

Abstract: The track-while-scan and interception program used in the initial experiments of the 6889 Air Defense Group is described. The note covers such subjects as the source and nature of the incoming data, the logical aspects of the track-while-scan action, and the computations used for computing the interception course. A flow diagram of the computer program is presented and discussed in detail; a coded program corresponding to the flow diagram is included but not analysed.

The document is divided into several sections:

1) Introduction
2) Radar Data describes the way radar reflections are filtered and encoded for transmission from Bedford to Cambridge.
3) Track-While-Scan describes the algorithm used to follow the locations of specific aircraft as they move.
4) Interception outlines the mathematical approach to computing the correct heading (direction) for an Interceptor aircraft to fly to intercept the Target.
5) Flow Diagram and Appendix gives the complete flow diagram for the program, a brief explanation of how it works, and the actual code, written out in Comprehensive System source format.

Given all that explanation, and the actual source code, how hard could it possibly be to get the program working? <<insert double-irony symbol>> (See footnote 12).

It turned out that there were a few (ahem) difficulties in achieving the goal, despite having a working Whirlwind simulator.

a) To simulate the operation of a computer tracking aircraft by radar, one needs to simulate the radar and the aircraft. Fortunately, the description of the phone-line interface in M-1343 was careful and accurate.
b) We hadn't worked with CS-1 source code before. I have a Whirlwind source code assembler, but it uses a more modern syntax. No problem, we can now read native Comprehensive System source code[2], convert it to my more-conventional format, and assemble the result.
c) My simulator was written to implement the instruction set as it was documented in 1958. It was different in 1951. No problem, the simulator now has an Instruction Set Year parameter to say

[2] Note that this work is using the "native" CS-I WW instruction set. By the time they got to CS-II, they had a well-developed pseudo-assembler that assembled instructions for an abstract 'virtual' machine with a more programmer-friendly instruction set. This program would have been *much* easier to write in CS-II language, but I think they needed a few experiences like this track while scan program to see just how badly they needed CS-II.

which instruction set to use. This was actually not that big a deal; it's hard to change instruction sets in real machines, so they didn't do it much, and most instructions went unchanged.

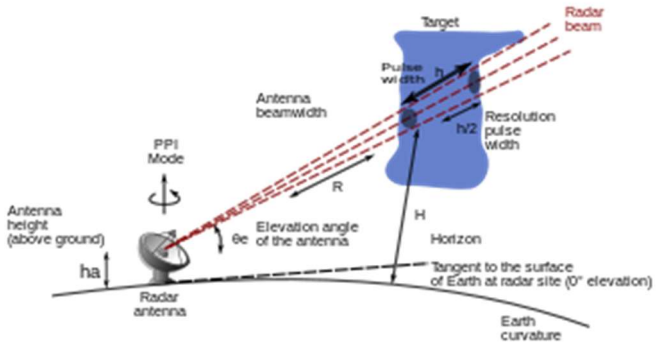
d) Debugging this kind of mathematical code is hard. My simulator now has a mechanism to insert python checker code in between execution of specific WW instructions, which can be used to check intermediate measured results against what the guy debugging thinks *should* be there.

While there's more detail below, it should be noted that this entire program fits into 255 16-bit words of memory.

1.2 What's This "Radar" Thing?

Prior to finding its true niches with police departments catching speeders and baseball pitchers practicing fastballs, radar was used to locate aircraft up there in the sky.

In the days of Whirlwind, long range radar would be centered on a radar station with a large rotating parabolic antenna. The antenna rotates in a way to sweep the entire horizon (every fifteen seconds for the Bedford radar). As it rotates, the antenna would transmit periodic pulses at a rate of (XX per sec) and listen for echo returns from each pulse. Each echo return detected indicates an object an angle given by the direction of the antenna at the point the pulse was received, and at a distance indicated by the length of time taken for the pulse to travel to the target and back. The Bedford radar was capable of detection at somewhat more than 100 miles, with an accuracy of a few degrees.



From a mathematical point of view, each echo represents a point on a polar plane, centered at the location of the radar station, with an Azimuth, or angle measured from a reference point (often due North) and a Range, or distance away from the station. Simple trigonometry can convert these polar coordinates into a cartesian x/y plane.

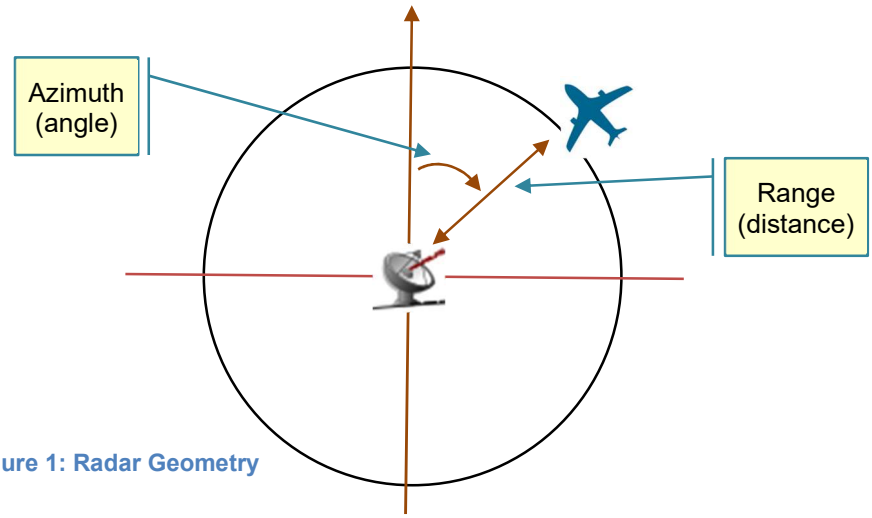


Figure 1: Radar Geometry

The Bedford radar is unable to measure to altitude, although Whirlwind later incorporated special height-finding radar as well (footnote)

Radar measurements are often degraded by “ground clutter”, i.e., stuff on the ground that also reflects the same radar signals, like hills, trees, buildings, trucks, etc, so any kind of radar processing must be tolerant of spurious reflections. There were a variety of techniques beyond the scope of this work, used at the radar station to reduce the impact of spurious reflections. (footnote)

As of this writing, the simulated radar used in this exploration generates effectively ‘ideal’ responses, without additional noise or clutter.

As noted in the document M-1343, quantization, i.e., the reduction of essentially analog measurements to digital form does introduce an inescapable form of uncertainty in locating an aircraft, something which any analysis routine must accommodate.

For much more on radar, see https://en.wikipedia.org/wiki/Radar

2. So What Did This WW Program Do?

Written in 1951, this program would have been one of the Whirlwind team’s early chances to show that their theory of using a computer to track aircraft as part of an air defense system could actually work, prior to serious work on the topic as part of the WW “Cape Cod” project, and the massive SAGE a few years later.

Document M-1343 refers to several versions of the program, but focuses on just one, a program to track two aircraft, designate one as a target, another as an interceptor, and compute a compass heading to be relayed to the interceptor pilot to set the interceptor on a course that will intersect the target. Doing this calculation once for two aircraft flying straight lines at constant speed is a straightforward problem of trigonometry, but as members of the Servo Mechanisms lab, the team knew that, even though targets may not cooperate with a static analysis, by solving the problem several times a second and giving continuous guidance to the interceptor pilot, corrections could be made along the way, accommodating a range of evasive maneuvers.

In operation, the program would present the operator with a radar-like display driven by Whirlwind, showing all the aircraft within range of the radar station. The operator could use the light gun to designate one of the aircraft as the target, and then by flipping a switch, use the light gun again to indicate the aircraft to act as the interceptor. In 1951, the WW CRT could only display dots, not lines or characters, so the setup could be configured with two CRTs, one of which would display the actual radar readings as received by Whirlwind, the other showing dots for the two tracked aircraft.

Once the aircraft had been identified, the Whirlwind program would analyze subsequent radar returns to follow the two craft in their predicted new positions, while attempting to ignore other aircraft or ground clutter that might be picked up by the radar. At the same time, the program would solve a trig problem to compute the best compass heading to keep the interceptor on track. The result was displayed in binary-coded decimal in one of the numerous sets of light indicators, to be read off by the operator and relayed by phone / radio to the pilot.

2.1 And What Does the Modern Simulation Do?

The Whirlwind instruction emulator of course executes the instructions of the original code and displays the image on the emulated CRT.[3] But to get useful results, the simulation requires a radar station and aircraft to provide input.

[3] I cheat by using two colors, one for the current radar readings, and one for the computed track positions, instead of two displays.

No one answered the phone at Hanscom AFB, so that was out. Instead, I added a new input device to the simulator, which emulates what the radar station would have sent. The radar simulation of course needs simulated aircraft to track.

To achieve repeatable results, the simulator currently presents two aircraft, launched on straight-line paths that would not result in a collision. That is, left unmolested, the two planes will fly straight tracks that don't ever put the two craft in the same place at the same time.

As usual, the light gun is emulated with a computer mouse. The operator designates the Target aircraft with a left-click, and uses a right-click to assign the interceptor.

I actually ended up optionally simulating the operator too. It was quite challenging getting repeatable results from one run to the next, so the radar simulator can also simulate the light-gun hits at pre-programmed points in the aircrafts' journeys. But fear not, the 'autoclick' can be turned off, so you, gentle reader, can play the role of defense officer and decide who's friend and foe, and when to initiate an intercept.

Finally, in this simulation, the heading computed by Whirlwind code is snatched up by simulation framework to bypass the operator, telephone and radio link and pilot, to tell the interceptor simulation directly what heading to fly.

2.2 Test Geometry

Repeatable results are critical to debug, so I set up a test case simple enough to figure out by hand, and fully automatic. Figure 2 shows the geometry and initial parameters.[4]

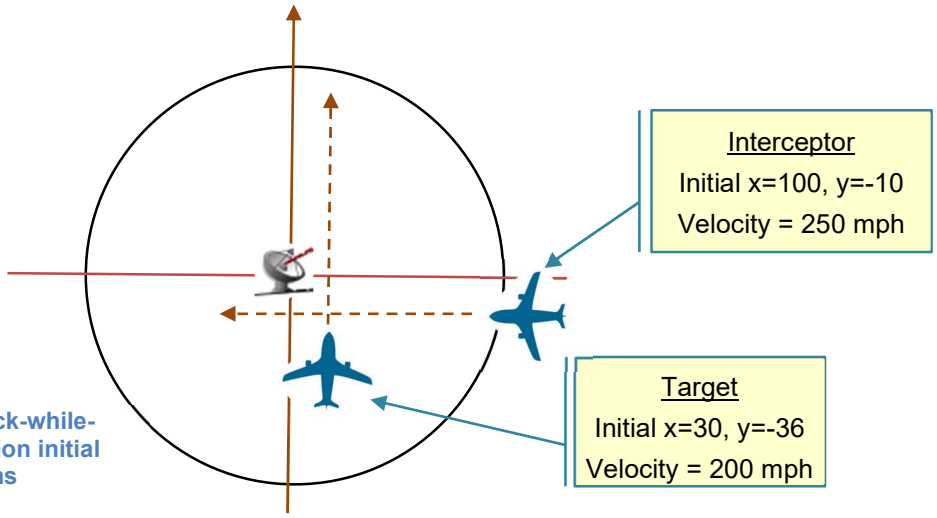


Figure 2: Track-while-Scan simulation initial test conditions

You can see a short video of the program running at

https://www.historia-mollimercium.com/whirlwind/Track-while-scan-Apr-23-2021.mp4

[4] Ok, 200 mph is pretty darn slow for a military aircraft, but I was having difficulty keeping the WW program locked on the aircraft. See Further Work below.

2.3 Paper-and-Pencil (and Spreadsheet) Solution

"Skate to where the puck is going, not where it has been."[5]

The Track While Scan keeps a record of the current location of Target and Interceptor, and their current velocities. It uses this information to estimate a heading for the Interceptor that will get it to the same point as the Target at some calculated time in the future, assuming the velocities and heading of the Target don't change.

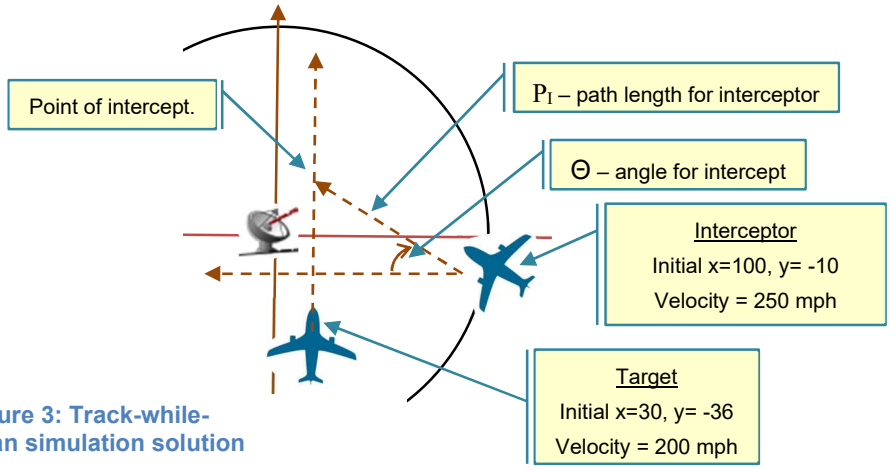


Figure 3: Track-while-Scan simulation solution

The Whirlwind program solves this trigonometric problem with an iterative algorithm that tries successively smaller time intervals until it finds the angle that will cause coincidence.

The spreadsheet below gives the solution for one point, assuming that on the very first step, at the initial conditions, I compute the heading for an intercept.

	Target	Interceptor		
initial X	30	100	miles	
initial Y	-36	-10	miles	
velocity	200	250	mph	
Angle Θ	n/a	28	degrees	
Heading		298	degrees	
Distance to Intercept	63.2	79.3	miles	Interceptor distance is the hypotenuse; extra distance for Target is Tan(angle)*base_length
Time to Intercept	19.0	19.0	minutes	

2.4 Real Time vs Really Real Time

This program would have run in real time with real radar returns, and as such, had to keep pace with the rotation of the antenna.

This need was reflected in the structure of the code, arranged so that no task would take an indeterminate interval to complete.

[5] Cf. Air Defense expert Gretzky, Wayne

Being an impatient modern kind of guy, I didn't want to wait around for the ten or twenty minutes it would take an aircraft to fly across the field of view of the radar, so I've compressed out the time in which the computer was just sitting around waiting for news from the radar set.

The simulated instructions themselves run at about real-time Whirlwind speed, i.e. at about 20 microseconds per instruction. With the wait times compressed out, the simulation runs about 25 times faster than real time.

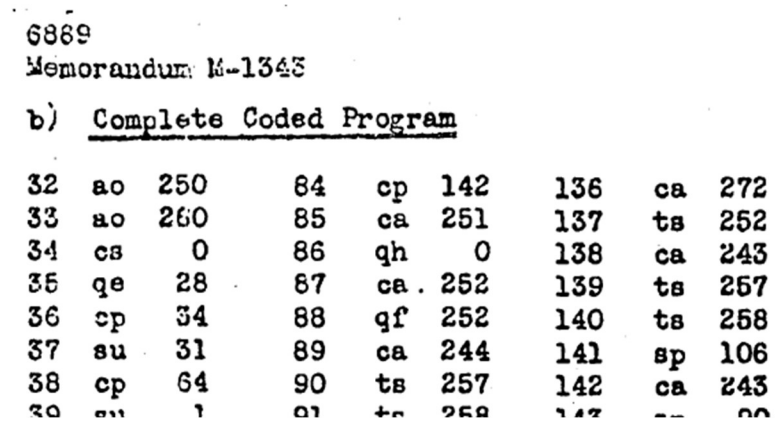
3. How was the Code Documented?

This program is unique (so far) among WW programs we've worked on so far, in that a carefully written document (M-1343) exists describing the program and its function. In fact, I have not yet found a paper or magnetic tape for the program (and given that the version described was done in 1951, I don't expect to find one; it would have become useless when the I/O instructions were revised.) But the document itself contains a complete listing of the "source"[6] code in CS1 assembler format.

As noted in Section 1), most of the document outlines the problem and the interface to the radar station, plus mathematical analysis of the problem.

The entire listing for the program is given at the end of the document on pdf page 51. While that page is enough to reconstruct the working program, without comments, variable names or paragraphing, it's very difficult to gain any understanding of how the program works by "reading the code".

Here's a snip from pg. 51:



6869

Memorandum M-1343

b) Complete Coded Program

32	ao	250	84	cp	142	136	ca	272
33	ao	260	85	ca	251	137	ts	252
34	cs	0	86	qh	0	138	ca	243
35	qe	28	87	ca	252	139	ts	257
36	cp	34	88	qf	252	140	ts	258
37	su	31	89	ca	244	141	sp	106
38	cp	64	90	ts	257	142	ca	243

Notes:

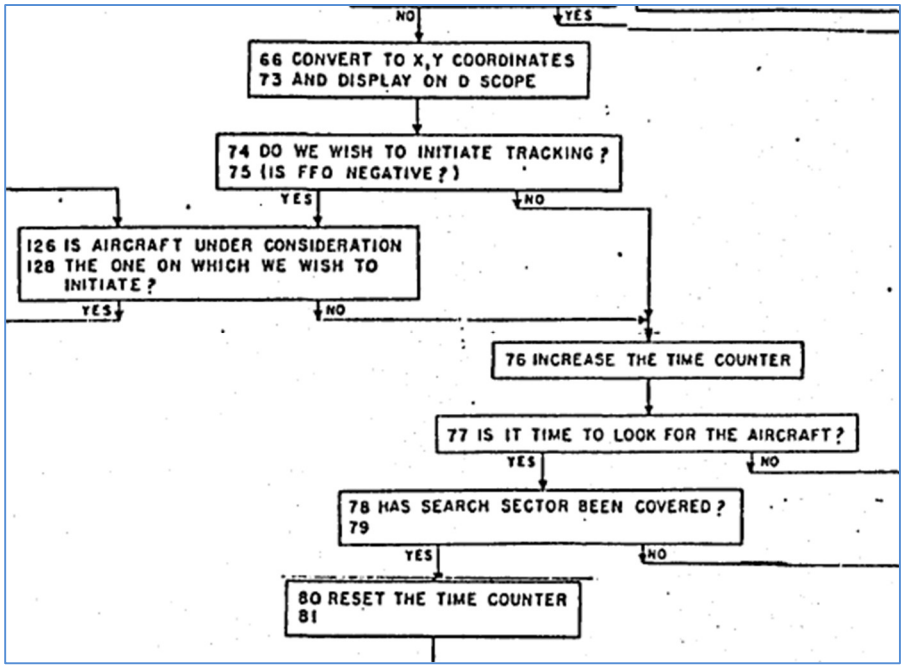
- The first, fourth, seven, etc. columns give the decimal address of each memory location.
- The second, fifth, eighth, etc. columns give the op-code to go in that address, while the next column to the right gives the memory address or numeric parameter that goes in the same word in memory.
- Addresses are all given as decimal numbers, unlike common practice of using octal for this kind of programming.

[6] Technically it's source code, in that it has two-letter acronyms standing for instruction types, but addresses are all given in numeric form. i.e., "assembly" means replacing the op-code acronym with the corresponding five-bit binary code and converting decimal numbers to octal.

There isn't even a syntax for expressing a comment or a non-numeric label in the code.[7] Numerical constants and variables are all initialized as though they were instructions which happened to have a corresponding op-code coded to the value Zero.[8]

The code totals 255 words, a number of veritable mystic significance, which must have been the limit on working memory at the time. The program does show signs of vigorous force having been applied to get it to fit the available space. For example, location 189 contains an instruction expressed in a not-totally-legal format so it could be reused elsewhere as a numeric constant.[9]

Not to worry. The part of the document that matters is pdf pg. 45, a double-page flow diagram showing how all the components of the code fit together. In this view, the program is broken into about 45 blocks, each only a few instructions long. Here's a sample snip:



Flow diagrams from von Neumann in that day were strongly mathematical, focusing on steps to solve a complex equation or chain of computations.[10]

In contrast, this flow diagram has much more focus on flow of control, highlighting the decision points for changing the flow of control (i.e., the 'If X Then Y' statements). Some of the most complex math in this flow diagram is reduced to a single box, e.g., "approximate the arctan".

[7] Ok, they did mark two instructions with asterisks, with notes following the code. But CS-I would not have known what that meant. Ironic, while one of these comments is critical (once I knew what it meant!) the other still makes no sense: "**Orders 192-195 should be disregarded". Huh? Disregarded by whom? If they're useless, why are they there? More later.

[8] i.e. the Read-In instruction had op-code zero (at the time; changed later). So if you take any constant 16-bit number and OR it with the op-code for Read-In, you get the constant.

[9] The instruction says "CP 704" at location 189., i.e., conditional branch to decimal address 704. With only 256 words of storage working, there is no address 704 to jump to. What's up? Decimal 704 is octal 1300. The machine would have ignored the 1000 octal bit with only 256 words of memory, so it's actually a jump to octal 300, or decimal 192, a valid address. But by putting the not-exactly-legal bit there, the programmer could re-use location 189 for a constant, decimal 0.8965, elsewhere in the code. This is why programmers who do stuff like this normally do it in octal, where the trick would have been obvious.

[10] Cf Burks, Goldstine, von Neumann, *An Electronic Computing Element*, 1947-1948

Each box does contain the range of addresses (in decimal) corresponding to the instructions that implement the block. E.g., instructions at location 126-128 are used to determine if the operator is trying to initiate tracking of the interceptor or the target. Text in each box is essentially commentary, with a minimum of formal notation.

The flow diagram is the godsend in decoding this program. Of course, without the listing there'd be no program to debug, but debugging just from the list of instructions would have been a much tougher job.

3.1 Code Conversion

As noted, CHM doesn't have a tape of this program, just the listing in M-1343.

No problem. The OCR mostly worked to produce a typescript.

The format, multiple op-codes and arguments all on one line was common for Comprehensive System programs, although the exact format in M-1343 is unusual, written in columns instead of the easier-to-parse row format. I assume that didn't matter, there wouldn't have been a computer able to parse or assemble the program anyway.

I converted the typescript into a single column format, and it ran it through my assembler to yield an executable binary.

Given that format, I could add comments as I figured out sections of the code one by one.

4. How Does the Program Work?

The flow diagram in M-1343 is the reference point.

Overlays in Figure 4 show which functions occur in which quadrant of the flow diagram.

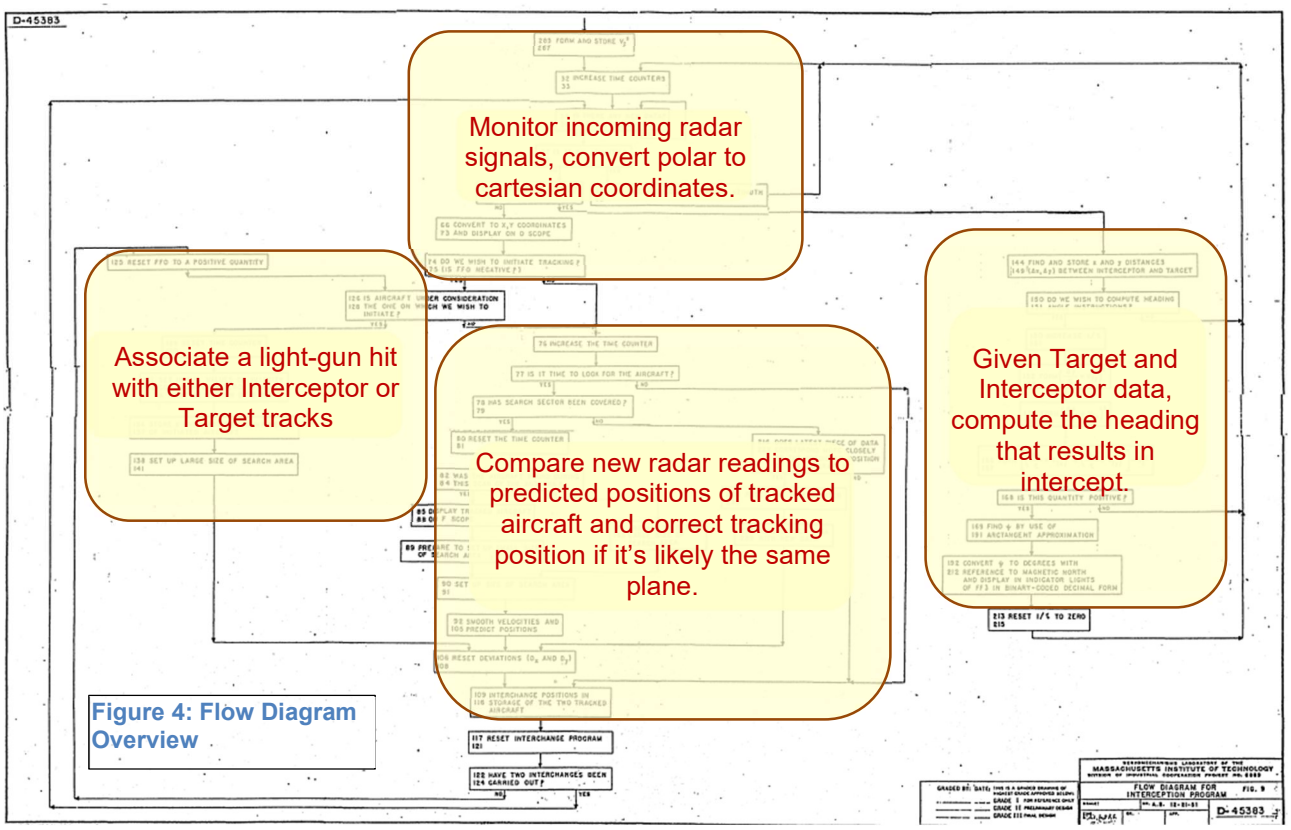


Figure 4: Flow Diagram Overview

As a real-time program, the code is arranged so that no step can take so long that they won't be ready when the next radar reading comes along.

The radar station sends a word of data every 20 msec (50 times per second) and that's the trigger that drives the simulation. Each reading is either an Azimuth (i.e., which direction the antenna is currently pointing) or a Range, a distance measurement to an aircraft at that Azimuth. Given that the antenna completes a rotation in 15 seconds, there will be a total of 750 readings per rotation. An Azimuth reading is followed by a Range of Zero if there's no aircraft seen at that angle.[11] The program can only track a few aircraft, and my simulation does not generate clutter, so most Range readings are zero.

On receiving an Azimuth reading, the code simply computes the sin() and cos() of the angle, in preparation for converting the next non-zero range reading to cartesian coordinates.

On receiving a non-zero range reading, the code checks for a light-gun hit, updates the location of the echo return on the screen, and determines if the echo return belongs to either of the two tracked aircraft (if any), by comparing all the returns within a search sector to find the one closest. Velocity is estimated and filtered for later use. The program also displays the tracked locations on a separate oscilloscope display.

Note that it does this calculation twice for each non-zero range, once using variables used to track the Target, and then again for state related to the Interceptor.

Zero-range readings (i.e., when the radar unit says "no new data", the heading angle for the interceptor is calculated. The calculation is done iteratively, with each cycle of the interaction narrowing in on a proposed time-of-intercept. In keeping with the real-time methodology, the iteration is advanced just one

[11] Note baked-in air-defense wisdom; if the attacker is actually measured at zero range, you've lost the game and better be in the bomb shelter already.

step each time a zero-range value is received, so several 20 msec intervals may be needed to complete one approximation.

At the end, the proposed heading is converted to binary-coded decimal and displayed in one of the Flip-Flop light registers.

Not bad for 255 words, eh?

5. Debug Lessons

5.1 "Math is hard"[12]

Whirlwind was a 16-bit fixed-point computer. Mathematical operations full of multiply, divide and squares such as commonly used in trigonometry are easy to overflow or underflow.

Virtually every multiply and divide is followed by a shift to change the scale factor, so the actual units used in most calculations are only in the head of the designer. We know real-world units going into the calculation –

- distance is in zero to 127 miles, shifted left by eight (check this) binary digits (i.e. it's in units of 1/256th of a mile, with a precision of one mile). After the first multiply, the scale factor will be something different.
- Angle from the radar set is in 1/256ths of a rotation, i.e. about 1.5 degrees.
- The output heading is in degrees decoded to three BCD digits.

Scale factors for everything in between is up for grabs, and very hard to figure out. And getting it right is important – Whirlwind would halt automatically with an overflow, and underflows waste resolution and introduce noise into the calculation.

One of the primary features of Comprehensive System II was a variety of floating-point formats, which would handle all the scaling automatically by dynamically calculating the scale factor and keeping it with the number as a mantissa during each calculation. But CS-II emulated floating point might well have been too slow to keep up with the real time radar data, even if it had been available at the time (which it was not).

Dear reader, allow me to gripe. The code uses a number of approximations of trigonometric functions, but there's not enough structure (and too little memory) to tell where the trig approximations end and the following calculation begins. Variables don't have names, just memory locations, and due to scarcity, variable storage is reused with abandon, e.g., up to here, location 275 has been a velocity parameter, but suddenly it contains the square of a distance. I still can't see how the sequence of instructions that apparently estimate an arctan correspond to the mathematical equation given earlier in the document.

I ultimately ended up solving most of the equations in less-optimal python code, driven by the WW simulation, to provide a "known good" set of intermediate results. From there, I could compare the answer I thought should be there with the answer that WW put there, and judge if the deviance was due to a typo, a misunderstanding on my part, or error accumulation.

Since the running code was OCR'd and corrected from M-1343, there were a few typos and ambiguities ("is that blot a 5 or a 6?"), although I'm impressed that the original seems to be relatively free of typos in material that must have been very tough to proofread.

[12] Cf Barbie. Although widely disparaged at the time, Barbie's remark has proven prophetic.

5.2 How Did They Debug This Code?

-------- Forwarded Message --------
Subject: Re: in defense of air defense
Date: Thu, 22 Apr 2021 20:31:30 -0400
From: Guy Fedorkow <fedorkow@mit.edu>
To:dave walden <dave.walden.family@gmail.com>

hi Dave,
I can tell you one thing... getting that code kind of working made reading the biweekly reports from the period *much* more informative. I had skimmed quite a few of them before, and often had a kind of a "oh poor baby, you have a bug -- suck it up, dude" response. This time, I literally read a couple of places where I thought "holy cow, no wonder I couldn't get that part to work; they couldn't either!".
So there are two spots where I will go back and look more carefully to see if the bugs they reported fixing were still in the version I've been debugging. I wonder if there's a record for the bug fix that took the longest time to apply to a piece of code :-)

I also saw more clues this time on how they did debug the stuff.
The obvious thing is that D Israel didn't just sit down and write the code and throw it on the machine. There are reports of all kinds of
experiments and modules done to test pieces of the algorithm independently. So when they did assemble the whole thing, they probably knew how most of the pieces worked already.
You're right that they must have had a core-dumper, although it dumped to paper tape I think. That would be a topic I could follow with more research -- it looks like there are many references to improvements to the Punch Out program over the life of the project. I don't know yet what they did with the tape once punched, unless it was already in Flexowriter code. This made me think that I personally rode the tail of the core-file debug methodology -- it was obviously useful to me to have a core file to do a stack trace, but I don't think I ever had to rely on a core file as the primary means to debug. That can not have been fun, at least for this kind of program where memory locations get re-used for all kinds of purposes as the calculations proceed.
And they clearly had a way to record radar test flights on analog
tape. They were anticipating the delivery of a big 16-track Ampex recorder in 1951, but apparently could record sessions already.
In fact one of the young Frank Heart's projects was to figure out how to "play back" the radar tapes onto 16mm film, I assume so they could see what the radar station would have been saying while the data was being replayed into Whirlwind. I think he tried several off-line tricks to try to get it to work with just the analog gear, but it seems ultimately he gave up and wrote WW code to drive a display.
It says they only get a dozen hours on the machine each two week period. Wow.

Anyway, that's what I've seen so far...
Thx

/guy

On 4/21/2021 4:31 PM, dave walden wrote:

They could perhaps get some way through debugging by having the hand calculation for a few useful inputs that would highlight problems and those few inputs could be used reproducibly. In iteratively writing such a program, they would probably be able to note dicey areas. Manual execution of the program seems a likely debugging step (not to the point maybe of "simulation" for all inputs. I suggest again that you ask Haigh (and Priestly) what they have learned about how people wrote and debugged code for primitive machines, or maybe this is just as good a question for the entire SIGCIS list.

On 4/21/2021 3:26 PM, Guy Fedorkow wrote:

hi Dave,
I'm sure they didn't have anything like ddt, but of course they
had
single-step and lots of lights to show what was in a few of the key registers. I think they had a core-dump by the mid-fifties, but I don't think they did when this was done.
If they had some way to generate reproducible input streams, i.e., maybe a radar run recorded on analog tape or something, then I suppose one could track a measurement through the calculation by stopping execution somehow at the 'right' place and then single stepping to see where the computer's calculation didn't agree with the programmer's estimate.
Do you think they would have had some kind of organized way to "simulate" the program on paper to see what should be in each memory location at each point?
/guy

On 4/21/2021 11:18 AM, dave walden wrote:

I will read it and get back to you. What is hard to imagine about

debugging the code? If they had something DDT like for interactive debugging or core dumps for off line debugging, that's how we debugged code in my early days with computers.

6. Next Steps

6.1.1 Technical

There are still places where I don't understand why the estimates are so inaccurate.

I suspect the code is "fragile", i.e., it works for the controlled cases I have, but it would be interesting to know how much noise and variability it really can take. My guess is 'not much'.

6.1.2 Historical Record

Now that we know more about what this program does, it's worth going back to review the relevant biweeklies to see if we can learn more about how well they thought it worked, what the test conditions were, etc.

I would also like to learn how the heck they debugged the program. Like the Apollo guys barely a decade later, I can simulate the whole thing, instrument all kinds of stuff, make up test cases, etc. But the Whirlwind guys didn't have a bigger computer in the basement to simulate the one they were building. As best I can imagine:

- They might have made analog recordings of actual radar returns, so they could replay flight paths as often as needed.
- And surely, they spent hours analyzing the geometry of individual test cases with copious pencil and paper exercises.

Again, biweeklies might offer clues.

6.2 Related Contemporaneous Work

I wish. Haven't found any yet.

The WW document logs show other documents on this topic, but none that I've found yet.

With more material, it might be possible trace the development of this technology as it evolved from this early demo to production software in SAGE.

7. Appendix

7.1 Aircraft Types

M-1301 says the following aircraft were available for flight test:

- C-47
- B-17
- B-25
- F-51

M-1334 added

- F94 https://en.wikipedia.org/wiki/Lockheed_F-94_Starfire - used as an interceptor.