COPY

orig to LOC

Topical Outline of the Saphire Interview of John Backus San Francisco, Calif. December 15, 1967

* Note - This is number TC-46 of a series in the IBM Oral History of Computer Technology.

Page

This is Interview TC-46 in the IBM Oral History of Computer Technology, Larry Saphire interviewing John Backus December 15, 1967, in San Francisco, California.

S. John when did you first come to work for IBM?

B. In September of 1950.

S. Were you out of school or did you have a previous job?

B. Well I was in school actually. I was just about to get my Master's degree at Columbia in mathematics.

S. And what did you start working on in IBM?

B. I was hired by Rex Seiber, who hired me as a programmer for the SSEC.

S. I take it at that time like everybody else you didn't know what programming was, much less how to do it.

B. That's right, yes. I was delighted not to have to become a teacher though.

S. How did you go about learning about the SSEC?

B. Oh just from the other people that were already there. They started out working on one job and I picked it up really. But there were no instructions.

S. What did you pick up?

B. Well how to program, what the instructions for the SSEC were.

Also over a considerable period I acquainted myself with the problem that I was assigned to and that was computing the position of the moon which was a fantastically complex Fourier series of 1,500 terms with all kinds of little corrections and what not.

S. Did you have to be a mathematician to do that problem?

B. Not really, just common sense. If you know the machine instructions, why you can begin to program if you have some sense. That's all you need. Of course you do it very badly and then you learn from somebody that they have discovered how to do a certain kind of thing very easily. But I rapidly got interested ..... oh one of the things that interested me very much was how to .... well the machine had a very short error-free interval that it would run and then you had to stop the machine and find out what was wrong by flipping toggle switches to see whether two storage units had the same kind of contents or not. So you'd run for three minutes and be stopped for five while you figured out what was wrong and got it restarted. And so one of the things I was interested in right from the start was how to speed that up. We got a system going, Ted Cott and I, we devised a system of automatic error detection and error correction that involved peculiar features of the SSEC. What you'd do is you'd code a sub-routine which was on a paper tape loop and the normal practice was to ... there were actually two tape loops for one sub-routine that

would go simultaneously and the tradition was to have one doing the calculation and the other doing the calculation again in duplicate and simultaneously and then at frequent intervals checking and comparing these results and stopping it if it was wrong. And the method we used was to just run right to the end without any checking, doing it all in duplicate without comparing anything, compare the results at the end. If they were the same, to go on to the next routine or whatever and go back to the main routine and everything was all right. If it was wrong, this would cause a relay to be set and that would change the reference of a lot of codes that were being used so that during the next pass, it would go through and actually check each instruction and would now refer, if you designed it that way, to another set of tapes that were the error detection tapes for this sub-routine. They would have a set of instructions that would then check the results of that single pair of instructions that you'd done and if that was correct, it would send you back to the sub-routine tape, do another pair of instructions or another few sets of instructions, after which an altered code would send you back to the error detection tapes and so on.

S. So after figuring out what kind of an error checking code you had, you had to punch out the tapes and get them on to the machine.

B. Yes, when you coded them .... coded the problem you had to code

all the sub-routine tapes and the error correction tapes and set up a lot of plugboards that were part of the machine that would affect this error detection mechanism.

S. Was this error detection mechanism there to start with?

B. No, it was just a way of using the facilities of the machine. The machine was enormously flexible. Most of the signals that the machine generated and used in its operation were available in plugboards and you could do all kinds of things. Like each instruction had a two digit code that indicated which tape station the next instruction would come from. But those codes were not fixed. They went through a decoder and you could arrange it so that that code referred to any particular tape station and in fact you could have it go through a selector so that at one time, when the selector was in one position, the code would refer to one station and when it was in another position, it would refer to another. That was the facility that we used in this error detection scheme.

S. So you were really slowing down the computer but by slowing it down, you eliminated errors which slowed it down even more.

B. What actually happened was the way the thing would work with this system was the machine would go through the tape, the sub-routine, if it then made a mistake, it would go through the second time but the altered rhy thm ...it worked out that rhythm was rather important with a lot of relays. It would

often go through successfully, the next time because it would do this one instruction and the rhythm would be entirely altered by the error detection mechanism operating on the second go-round. Then it would conclude successfully the second time and would go on.

S. And you knew that it was concluded successfully how?

B. Because it just didn't stop.

S. I see.

B. It was so arranged that if it went through and checked everything and everything checked, it then just went on.

S. So you had enough confidence that the machine was checking correctly.

B. In fact it was arranged to actually operate as follows. It went through once. If it made an error, it would go through once again just straight and then if it made an error a second time, then all the error detection tapes would be brought in and sometimes it would get through the second time on a fast run.

S. The second time was Seiber's method of error detection, wasn't it? I remember he told me something about that.

B. Oh just running it twice. Yes.

S. In fact running the problem in parallel or almost.

B. Oh no, his fundamental method of checking for errors was used all along in this thing and that was that the entire machine was sort of divided .... the memory was divided into two halves and everything was kept in duplicate and every step was done in duplicate. That was his basic operating philosophy and we fell in line and followed that. The difference between his method and ours was simply that we didn't take the time in a normal sub-routine execution to do all check calculations because it was important to do it frequently in his arrangement because the machine was going to stop and then you had to sort of manually check to see on which line it had made an error and this involved flipping lots of toggle switches. And if you only did one check at the end of a sub-routine it would take you all night to find out where the error had occurred. But with our method, since we had this automatic method for looking for errors once one was made, we didn't have to put in a total of a lot of checks that were always done as you did in his method. You only had one check at the end of a sub-routine. It might be a very long sub-routine with no comparison calculations. We were doing everything in duplicate just as he had done. But there weren't a lot of these comparisons being made all through this sub-routine. So when the thing did end and an error was made, it was determined that something had gone wrong because the results didn't compare and then you'd try

again and if it then went wrong, then all of this automatic mechanism went into effect to find exactly on what line the error was made, not what block of lines but when it did make an error on its final error detection run, then it stopped you and told you exactly which line was wrong so that you had none of this toggle switching thing to do.

S. That was where the time savings came in. You spent a lot more time computing.

B. No, you spent less time computing. But only when there was an error did you spend this time. In the .... you did less computing because .... you did far fewer comparisons than you did in the old method. But then when an error was actually in effect, then there was a lot more computing. But that computing was doing at high speed what you ordinarily had to do at a very slow speed manually. Its purpose was to do what was being done manually, namely to find which line was going wrong. Then when you knew that, generally if it made an error, it had to make the same error three times or an error in the routine three times in order for it to stop, so that generally when it did stop it was a solid error and you could see exactly what device in the machine was wrong and the engineer would change it and it was easy then to restart.

S. Did you have to go back through the whole sub-routine at that point?

B. Well in the restarting procedure you're not dependent upon the situation. The main thing was if you knew what line it was, then you'd look at the coding and you could see exactly where was the appropriate place to start and what had to be done before you could do so.

S. In those days did you have to hover over the machine most of the time?

B. Oh yes. I mean in those days the operation of the machine was much more fun than programming now, because you spent a long time programming. Then you spent weeks preparing these tapes and gluing them together and doing all this stuff.

S. Those were the tape loops.

B. Right. The sub-routines, you coded them and key punched them and then you ran the key punch card through a strange machine that prepares punched holes in tapes from the cards. Then you'd cut that tape off and you glued it together in the back room...

S. To make a loop.

B. Yes to kind of make a loop. Then you numbered them all and hung them on tape stations. Then once you were ready to go and the machine was ready, you took over the entire machine night and day. You had it full time. There was no interchangability of problems at all. It took weeks to get a

problem running, let alone to do it. Then it would run for months, very unlike present day operations.

S. With these loops, did you have to keep substituting them or could you put a lot on at the same time?

B. Oh there were 66 tape stations and you could put all of the routines that the problem was going to use on the machine at one time. It was programmed then to transfer from one loop to another. It had a main routine. And the main routine, when it had done certain things, would then call in the sub-routine.

S. Was this system of error detection adopted for all programs afterwards or did you just use it personally?

B. I guess it wasn't adopted for all programs afterward. I don't know. As I recall, I'm a little foggy about it now, I think I used it on the problems that I was associated with after that. Another interesting feature of that machine that we made use of was that we actually took one of the storage cells which was a relay storage thing, and essentially rewired .... what was his name, Wayne Brooks? made a special storage cell that was essentially just a plugboard where you could fiddle with the relays and all, and we made use of that in this first problem I did on the moon ~~position~~ problem to wire it in such a way that you'd ~~need~~ feed an argument into it and instead of getting what you put in out again, you could read out an

instruction that had part of that argument involved in it in a complicated way and that was to speed up operations. It did speed things up quite considerably.

S. Well was this when you first started getting ideas of ways to speed things up .....

B. This was just a little deal that presented itself as a way to eliminate a few steps in a very repetitive sub-routine that was used an awful lot. You see, that program ran for many months after we got it running.

S. Was the machine down frequently?

B. Well down, no, not too frequently but it was just that it was constantly making intermittent errors.

S. How long did you work just programming the SSEC?

B. Let's see, from about the time I joined the company until about 1951 or 2 I guess, when they got the 701. I'd have to look up the records.

S. But then you moved on to the 701.

B. Yes.

S. Before we leave the SSEC, was there anything else, any innovations that you got into in programming along the way?

B. I don't think so. Every problem on that machine was very interesting to do because you had the business of one other technique that made things interesting which was a problem in which there are so many choices to be made that you began to run out of tape stations for alternate sub-routines.

So that there came into use in one problem a technique whereby you had to make choices without ....leaving just a straight line in computing and that involved computing a function that was 0 or 1 according to whether the result was yes or no and then using that as a multiplier. You had the two calculations and you had to go through both of them but you added the two results at the end multiplied by 0 in one case and 1 in the other. There were lots of interesting applications of selectors and all of this great flexibility of the machine.

S. Do you recall your own sentiments about the computer?

B. Oh I liked it very much. Oh one of the things I should remark about the computer was that it had a number of features that later came to be regarded as advanced and mainly the fact that it was somewhat asynchronous, and ~~had in some sense,~~ the arithmetic section would ask for a transmission of a piece of data and before it indicated that it was ready to receive something, and the place where it was might not yet be ready to transmit it, so the signalling system was involved that would sort of coordinate these requests and abilities to deliver things. And since the machine was working on two rather involved instructions simultaneously and had two more instructions also in its decoding mechanism, it had quite a few instructions tha were in various states of completion involving getting the operand and doing the arithmetic and putting

the results back.

S. Were those the eight bussing channels?

B. Yes. I'm sure he described that feature of the machine better than I could. But I thought that was a very interesting feature of his machine.

S. At that time did you have a chance to look at other computers, things like the Princeton computer or the ENIAC?

B. I knew nothing about other computers really. Of course I heard about these others. In fact, when the 701 came along and having gotten used to this strange beast, I was appalled at how anyone could conceive of a computer like this operating. It was all operating silently and you couldn't see what was going on or hear what was going on and it went on at such an incredible speed by comparison that I figured, having gotten used to the kind of error rates that the SSEC made, with the electro-mechanical gear, it just seemed impossible that anyone could, where you were doing thousands of instructions a second instead of twenty or thirty, that you could possibly get a correct answer out of the machine.

S. Were you disturbed by the lack of audio in the 701?

B. Well it was mainly that the communication between the inside of the machine and the outside of the 701 was very difficult because it took rather elaborate programs to get just one number into the machine or get one number

out of the machine and when one thought of all of the work that had to go on just in order to get the number out, let alone the computation that might have been needed to generate it, why it was a pretty awe-inspiring thought.

S. But when you first saw the 701, was it completed, or did you look at it while it was being built?

B. No I didn't see it I don't think until it was completed.

S. Were you asked to do anything on it?

B. Well I was working on the program for it I guess, even before it arrived. The chronology is a little foggy there but I know that at least after it arrived I got right away involved with a programming system for it.

S. It didn't have any system to run it when.....

B. Oh no, heavens no. It had a printing program and a card reading program and it didn't even have an assembly program. Oh yes, it had what was called a relative assembly program which you wrote instructions ... you were allowed a few symbols to use for location and then you referred to that symbol, plus one plus two, plus fifteen and so on. You didn't generate new symbols.

S. This was for specifying storage locations?

B. Yes this was for specifying storage locations. So it had very primitive methods.....

S. What was the symbol plus 15, the current just counted through 15 storage locations and stopped?

B. It had these primitive facilities of accepting instructions in this form and you didn't ask it to assign a location to this symbol, but you wrote cards that said that A was to be stored in location such and such and then it said do this.....and you'd get whatever that storage cell number was and put that in actual machine instructions and of course it had to do the binary to decimal conversion and all that. That was another one of the principal topics of interest in those days, conversion programs and would you get exactly the conversion by such and such a conversion method or would you be off by one. And of course one of the big problems was getting input/output programs that would ..... in that machine it did input/output by sending out copy.... it worked with so-called card images and it sent them out a row at a time so that it would send out a bit essentially for each hole in that row and 0's for non-holes and then there was a good deal of or a good number of milliseconds that could elapse now before you had to send out the next row because this mechanical machine was cranking away at a speed that was very slow as far as the computer was concerned. In fact it was the only program I ever really wrote myself for almost any computer that was a printing program for the 701 that was part of

this programming system that my friends and I generated and it did all of the conversion of a bunch of floating point numbers from floating decimal or floating binary rather into some appropriate decimal notation and reconverted that all back to check and did this all between the copies of the .... that had to be sent out during the printing cycle.

S. How long did it take to write that program?

B. It took me a very long time because I used a lot of :... I used a different technique than was popular in those days. The technique that was being used then was they had sort of a column indicator... it was a word with a bit in the position corresponding to the column of the card and I used the technique of generating the entire ~~sort of~~ card image by logical operations. The machine had some very primtive logical operations, an and or an or or something and a not and using those, I generated the card image but it was very involved.

S. Just give me an idea of how you worked, kind of sitting down and figuring this out. Did you have an absolute knowledge of the paths in the machine or.....

B. Oh heavens no. I didn't know anything about how the machine worked physically. All I knew was the instruction set.

S. You mean some guy like Nat Rochester told you some of the.....

B. Well I simply had a manual of the instruction set and the timing operations were very important since you had these things that had to be synchronous and particularly in the work I was doing in the printing programs, you had to know how long it would take to do a piece of code because it had to be done in time to get out the next copy of instructions to the printer, or to receive the echo check copies coming back.

S. And how did you figure out the timing?

B. Oh just by reading the manual and seeing that .... I had written the instructions and you'd look up the time for each instruction and added it up.

S. So the game was simply not to exceed the time with the set of instructions.

B. Right. There was nothing much to it really.

B. Well yes, in this relative coding that we were talking about earlier. I was not writing in binary. I was writing in decimal for this primitive relative assembly program that was available, this system then available to the machine. The system that I was working on with my friends was one that would accept instead of the binary single address instructions of the 701, it accepted a peculiar code where each instruction had one three address part and one one add ress part and the three address part was referred to as floating decimal numbers and there were facilities for reading in floating decimal data and

printing out floating decimal data. It made the machine appear like a floating decimal, three address sort of odd ball three address computer with indexing operations which the machine didn't have. The further one address operations which came along with each instruction was for manipulating the index quantities and making tests and doing a lot of the housekeeping operations generally. This was a system that was called "speed-coding" and was used by very many of the 701 installations for scientific calculations, even though the rate of arithmetic operations was much slower but they were floating point and it saved people a lot of analysis time.

S. What was the relationship between the later introduction of indexing and this kind of effective indexing?

B. Well it showed people that it was much better to code with indexing than without it and it was a demonstration that people wanted. It was a demonstration of a number of desirable features that we later put into the 704, namely indexing was one and floating point another.

S. Well could this thing that you just described be considered a kind of pointing toward the whole concept of indexing?

B. Oh well no, we certainly didn't invent indexing. We simply heard about this guy Williams in England who invented what he called the B line and

it seemed like such a magnificent idea that we wanted to incorporate it and did into this system, since we had to go through this interpretation cycle and the typical operation of a computation was not part of the machine's repertoire and had to involve a number of instructions to do one floating point operation, why the added time it took to superimpose on that extra work this indexing business was very small and very worth while in speed. If you're going to take the time to interpret an instruction it was very worth while to do this extra ~~kutike but if wirj~~ little bit of work, so we did.

S. But it was based on what you observed from England?

B. Yes. Williams actually visited the computing center there on Madison Avenue and 57th Street and talked about that. Of course it was published. His ideas were published.

S. So that you got a kind of indexing through programming, whereas in the 704 it was built in.

B. Right. Of course obviously there were things that should be done but when the 701 was designed Williams hadn't had his idea yet. At least I don't think he had.

S. Well what was the next step you took in regard to computers?

B. Well let's see. I was then associated with that operation for a while.

In turning out that system.... do you want names of other people in this thing who were involved in such a project or what? Is that appropriate?

S. How appropriate do you think the names are?

B. Well I think it is very appropriate if you're going to mention the system that you should mention the other people who made it. They were principally Harlin Herrick, a fellow named Skillman who is no longer with IBM but is with RCA, and Don ~~Quarrels~~ Quarles, who is with IBM. I'm afraid I've left somebody out there. Boehm

S. Was Elaine Bowman working on that?

B. Oh no, she was working in Poughkeepsie on this assembly thing and what not. I'm not sure that she was there yet even at this point. So anyway after that, well really on leaving that thing I was starting already then to get into the business of getting FORTRAN going. I wrote to Hurd, Cuthbert Hurd sort of proposing .... wait a minute, no. During that business, working on speed coding and thereafter it must have been, because then came the business of working on the 704 because of course the idea for the FORTRAN very heavily depended on the idea that there would be a machine that had built in floating point and built in indexing. So I guess at that point in time there were a number of meetings in Poughkeepsie involving John McPherson, Nat Rochester, Gene Andall, Werner Buchholz I think. I'm not too sure about that. It was called

the 701 Improvement Program and the idea was for 25 cents or less to improve the 701 without doing very much to it. There was a lot of discussion. One of the main topics was how to get faster and bigger drums, magnetic drums hooked up to the machine. There was much discussion of that.

S. And what that would do for processing.

B. Right.

S. Did they ask you questions like if they hooked up so many bigger and better drums, what you could do with them?

B. Well this was something that it seemed was occuplying everybody else but it wasn't occupying me. I was pushing other things in that committee.

S. Like what?

B. Well like the desire for indexing. I think it was sort of generally agreed that there should be some form of indexing. I'm not sure just how that came, whether that was agreed upon at the outset or not because the initial objectives were so modest, that just where these various things were sort of admitted as being possible within the small expenditure that was implied we were to be allowed is not clear. But I know that my monomania at that point was to see that they got floating point into the machine. I kept saying, let's have floating point in the machine. But I didn't have anything more to say than, let's have floating point in the machine. The most important thing you can do is put

floating point in it. But everybody was so involved in the detailed discussions of these drum ideas that involved hours of description and technical discussion that every time the committee would meet, I'd say how about floating point. Let's put floating point in. They'd say yes, that's probably a good idea. Then it would go on and be forgotten. So finally after doing that about three or four times and getting no real response, nobody interested in doing it, I sat down and I worked out the most lugubrious design involving ten more registers, actually the clumsiest, worst, slowest, most expensive concdivable way of having floating point because I didn't understand many features of the machine that actually made it possible to do it very well. But I stood up at the blackboard and talked for half an hour or so. Here's how you can build in floating point. I wanted to keep their attention on floating point for half an hour or so instead of letting them forget it. It turned out that this was such a laughable proposal that Gene Andall and others took great delight and it was mainly Gene, in showing that you didn't need any extra registers and you could do it fifty times as fast and o/50th the cost. But that was what I had really wanted in the first place, was for somebody to do this. So I was very pleased that now Gene had demonstrated that we could have floating point in the machine and not at an exhorbitant cost. The fact that the majority I think of the 701 installations

used this greatly slowed down speed coding thing was enough evidence for the company to show them that if they could get floating point in they better had. I think I can modestly state that that was ... if there was any one thing that made the 704 a paying machine, it was that fact.

S. But it was done without any real modification of the machine.

B. Well you know it involved a fair bit of doing. The 704 ultimately turnedout to be quite a different machine than the 701 but it didn't start out on that basis. It cost I think very little more than the 701 to produce, but it was learned by that time that the 701 was not making money, and since it did so much more work in a unit time, that they were able to raise the price like 40 per cent or some very large price increase, and still sell a lot of them. People could see that they were going to get a whole lot more computing per dollar even with this great increase because it had built-in floating point and it had built-in indexing. So the 704 I understood turned out to be the first computer .... big computer that the company made money on. They really lost on the 701.

S. Talking about FORTRAN starting and the ideas for FORTRAN,

.....

B. Oh well then after it was clear that there was going to be a computer

with built-in floating point, then the whole philosophy of speed-coding, you know how to make a machine sort of doing.... sort of do some of the coding work for the user, made it clear that you cculdn't use this interpreter technique. The interpreter technique it was clear was as efficient as compiling when you ~~had to do~~ most of the operations you wanted to do were not in the machines repertoire. So that essentially ~~while a compiler would have to compile~~... what a compiler would have to compile would be just a string of calling sequences and sub-routines. Well it turns out that an interpretive program does the work of calling in sub-routines with about the same speed as having calling sequences. It gets the main program information in a more compcat form also. But if every operation that the programmer specifys is in the machine repertoire as it was now going to be in the 704, then the problem looked very different because you didn't want to have a calling sequence added to a single machine. operation because obviously it would be highly inefficient. So we had to obviously have a compiler to do the job. So I proposed that there ought to be an effort....oh I was also kicked off on this idea by visiting MIT and seeing a program that Lanning and ~~Dealer~~ Zierler had at MIT that would take algebraic expressions and turn them into code. That was partly a compiler, partly an interpreter I believe.

S. Was this the Whirlwind?

B. These two guys weren't even in the Whirlwind Project. They were in the Instrumentation Lab and they just did this for their own amusement and slight use and their own calculations. I guess it was for the Whirlwind computer. I'm not even sure which computer it was for they did it. But it seemed like a marvelous idea. They hadn't surrounded this with other features to make it into a really viable computing system, programming system but they had certainly had this thing running. It was very nice. So anyway I wrote a letter to Hurd asking him to let me and some others work on a system that would involve algebraic expressions and that would compile code, efficient code.

S. Why did you fix on algebraic expressions?

B. Well just because it was an idea that was available to me.

S. Was that because you saw that most of the work was done in mathematics.

B. Yes. My experience was with scientific computing and I wasn't at all oriented to the idea of bookkeeping on large computers. I wasn't interested in it.

S. Had any symbolic assembly programs come along?

B. Oh sure. Nat Rochester had generated symbolic computing on the 701 and had gotten that operating and it was clearly a good thing. So that got underway.

S. Was that after the machine came out that the symbolic assembly program was introduced or did that come with the machine?

B. The Symbolic Assembly Program was supplied with the 704, yes.

S. I mean with the 701.

B. No it was not supplied with the 701.

B. But it ....

S. But it came in later in the 701?

B. Later, yes. I guess Nat had a Symbolic Program running maybe when the machine was available but it wasn't the one that was generally used at the beginning. I know that it was not used, at least for a long time in the Computing Center 701 in New York. A lot of customers also did not use it at the beginning. When it started to be used I'm not clear. So then this was now the Summer of '54....starting with the Spring or Summer of '54 and working through until the late Fall, working on specifications for the FORTRAN language and working out methods for decoding various of the proposed statements and finally culminating in a tentative proposal of what the language would be like. I was involved in a lot of trips to visit customers and describe this proposal and hoped to get some suggestions but I didn't get. I don't think, one useful suggestion after traveling all over the country and talking to a lot of customers. They hadn't thought about it and we had been thinking about it hard for six months. But some of the customers liked it, they liked the idea and some liked it so much that they supplied people to help and with that backing, we were

able to continue for three years until we finished it. During that time I got involved in one machine design effort which was never built but it was a proposal to Livermore. Gene Andall was very much involved in that.

S. What kind of machine was this?

B. Well this was a machine that was to be the big super-duper kind of machine but at that time it was envisaged as being made of the same gear as the 704 and Gene Andall and a whole bunch of people in Poughkeepsie had designed this machine and it was well along in the design business, at which point John Sheldon and myself were asked to go up and look at it and comment on it and John Sheldon's comment was that it was an absolute impossibility to program, he pointed out because it was a machine that was designed as one very high speed central machine with very small memory. The bottleneck was how to make it fast enough in memory to keep this thing going. Then this machine was being fed by another machine that had its own instruction set and you had to keep these two things in synchronization. His comment was that it was an absolute abortion as far as programming was concerned. He then suggested that the way to do this would be to use a lot of different memory bits concurrently and this was an idea that appealed to me very much and I went off and thought about that for a while and tried to see how this could be done. I desinged what seemed to me was a charming little arrangement

It involved having instructions coming in, after a couple of steps having the indexing being done and then having that instruction take its place in the line and finally getting executed and while waiting in line, having the addresses being sent out to boxes. It was not an engineering design but it was that idea essentially. After a lot of back and forth between John Sheldon and myself on the one hand and Andall and the other designers of the machine on the other, they agreed even though we were under enormous time pressure to get this proposal done, to change the design to this kind of a thing. I'm not trying here to claim the design of the look ahead decoder because I didn't design it. I only proposed a sort of sketch of an idea that this was possible and that it would actually do the job correctly. It was Gene Andall who designed the actual thing.

S. Well he mentioned that he had a lot of help from you. But the machine was not accepted?

B. No. That was a very interesting story. That was where old shrewd Ralph Palmer entered the scene because he was then, from my viewpoint a very high official of IBM, and we were just some guys working on this proposal. If you've heard this story from another source, stop me, but it was very interesting to me. We finally got this design done and got these rather elaborate manuals and everything prepared and we were to go out to Livermore on Monday. This

must have been around 1955 and so we were all.... we went home exhausted. We were working night and day..... on Friday with our airline ticket reservations and everything, all ready to go on Monday. We got a hpone call on Saturday or Friday night, cancel your reservations. The whole thing is off. We're not going to go. We're not going to make the proposal at all. Then later in the weekend we were told, don't cancel your reservations if you haven't. If you have, get them back because we may go. Anyway we finally were told to come in to 590 Madison on Monday and see what happens.

S. With or without your reservations?

B. With our reservations. And so we sat around downstairs in 590 while Ralph Palmer and others were up in the executive suite discussing whether this proposal should be made or not. Finally it got around, he plane was to leave say two o'clock or something like that. One o'clock had arrived and they still had not decided. So they said, well you'd better go out to the airfield, go out to the airport even though we don't know yet because we still may go. So there were about eight of us who were going out to make this proposal. So we all went out to the airport and finally came fifteen minutes before flight time and we called up and asked, well have you made up your minds? No but get on the plane and go out. We still haven't decided and we may make the proposal. So all eight of us went out and after we arrived out in Livermore, it

became clear what decision they arrived at. It was a very brilliant decision although we all hated it because it just meant putting aside something we worked very hard on. But it was that we really would not make a serious proposal. That we would say we'll make you this proposal but the deadline was so long we knew it would be unacceptable. They said for another few million dollars we'll make you this other machine made at a much faster gear than this one. But it too had of course a longer delivery date than Livermore wanted.

S. And what was the competing machine?

B. The competing machine was the Lark, Sperry-Rand's Lark. So they got the contract and like Palmer apparently who was the chief arguer at this point, they locked up all their best guys working on a machine .... it was made with devices that were already known. It had a productive life of just about zero. They lost their shirt on it. They couldn't deliver it when they said they were going to deliver it and they just wasted a lot of time and effort, which we would have wasted if we had gotten the contract. He was smart enough to see that if we're going to get into something like that, we ought to tie it to the development of faster componentry. So we slipped out of that and let Sperry-Rand get socked with that enormous contract. But I thought that design was a nice one and had many of the features from which STRETCH took off.

S. STRETCH was the succeeding machine?

B. Yes it sort of started where this thing left off.

S. What was the name of this?

B. Oh I forget what it was called now, if it ever had a name. I guess it did.

S. Well by that time had FORTRAN been done or were you still working on this?

B. No, this was '55. FORTRAN was in the process of being done but the guys that I was working on it with really did all the work on FORTRAN. I had of the ideas but .....

S. Quarrels and some others?

B. No this was another group now. This started off with Irv Ziller and then got Harlan Herrick and then Bob Nelson and then Peter Sherridan, Dick Goodberg, Abe Sayers....Sheldon Best from MIT came down to work on it. Roy Nutt from United Aircraft. The names of these people ..... they're all listed elsewhere.

S. Well why don't you tell me a little, while they may have done the work, since you are known as the father of FORTRAN, if you can stand that phrase at this youthful point in your career, ...

B. Pretty soon I'm going to be known as the grandfather of FORTRAN.

S. Why don't you tell me what those ideas were and how they were implemented and what were the stumbling blocks and the successes in finally getting the thing out.

B. Well all right. Well one of the ideas that is still not recognized in many quarters even today and that is the principal problem is not that of decoding algebraic expressions. That is relatively easy, although it was new and interesting and a nice, orderly problem so it got a lot of attention. The main problem was how to deal with arrays of numbers and how to obtain their addresses in the way that programmers got them when they hand coded, which in almost all procedures was simply to save the address of the last reference and add a constant to it to get the address of the next element because almost always you're working in an orderly way through an array. If you do this though, you have to look at the entire context of the problem to see whenever a reference is being made to an element in an array. You have to see well when was the last reference made to an element in that array and how are the subscripts being buried between those two references.

S. By an array you mean what?

B. A rectangular array is numbers like AIJ, a three dimensional array A sub I sub J sub K where you are given two or three integers as the subscripts of a quantity and now you want to pick out that number that corresponds to those subscripts. It always used to make me furious after we got FORTRAN out... people like Alan Curtiss would say, "oh these idiots, they took three years to make the system and I had a graduate student write a compiler just this last summer. What took them so long? Well what took us so long was this problem. What Curtiss did and his graduate student did was when they

would see a reference to A subIJ, they'd multiply J by some constant of the length of a row and add I to it and then add . . . . place the address of . . . . .

End of Track #1.

Track #2.
#1.

S. This is Track #2 of Interview TC-46.

B. Well I was continuing my gripes about Perliss's ~~(?) Curliss's~~ comments about FORTRAN.

S. Perliss is a....

B. Peecisely, he is a guy who was then at Carnegie Tech and very interested in compilers and stuff like that. Well the reason he was able to say that his graduate student could construct a FORTRAN compiler in a summer whereas it had taken us twenty odd ~~and end~~ years to do so, was that the way he would handle references to arrays was to when a piece of code contained reference to AIJ, he would do this multiplication and addition and two additions required to get the address of AIJ. It was a very time-consuming operation. Every reference to this array would require this multiplication and addition just to get the address of the element in the array. This makes it possible to translate the source code item by item. You never have to know what's going on elsewhere in the problem. Whereas if we had a reference to AIJ in a loop, we would have analyzed the code to know that this reference was indeed in a loop and in fact that it might be in several loops. We would know that fact, and we would know that the I or J was varying most frequently and that the last time we went through this loop, I was varying by 1 or whatever it was

varying by in the code that was elsewhere. We would know just what constant to add to the address of AIJ the last time to get AIJ plus 1. So that instead of doing a number of additions and a multiplication to get the address of the thing, we did one addition and we did it very efficiently, usually with a maximum of possible efficiency. So that although we spent more time compiling, our code ran often two, three, four or five times faster than Perliss's generated code. Although for his purposes, his method of compiling was perfectly satisfactory because it was for students and students wanted to compile a few times and then run a problem for a couple of minutes and forget it. So that it was better for him to have short compiling time and inefficient object code. Whereas for us, we were designing a system for the aircraft users and aerospace users and people who had to have highly efficient code for many applications. And so we felt it necessary, in fact we did too much work to make things efficient. Anyway that is the major problem that FORTRAN faces, the problem of analyzing the context of the program and knowing its structure so it can understand how to make the references to arrays with the maximum efficiency.

S. You mention that you were designing it for the aerospace industry and that is what I would want to ask you about.

B. Well I mean those were the primary customers we had in mind.

S. You had to know their problems first, right?

B. Well no, I didn't really know their problems so I couldn't think of them first. All I knew was that it seemed to me that we wanted to produce a programming system that would do a lot of tasks of coding for the user and it seemed that we were up against the attitude in the computing industry that felt it was impossible to do this and do it and produce code that was sufficiently efficient that people would actually use the system. And so primed with this sort of having to beat a hostile world and prove that it really could produce efficient code and do it with sufficient reliability, that it would not be the kind of thing that would be efficient four times out of five and then on the fifth time be so disasterously inefficient that you couldn't use that program, so you would never know whether it was worth while to compile or not. But it had to almost always produce good code. We went at that so hard that we very often ran up the compiling time unnecessarily in causing the compiler to do a lot of analysis in the attempt to produce an efficient object program, but it really didn't pay off. The analysis wasn't worth a candle in many instances. But we felt if we didn't do it that some of that analysis just might take care of a situation that in the early tests of the thing would blast us if we hadn't done it.

S. In a way it was like designing a computer, that you were trying to cover a lot of possibilities and the users' needs.....

B. But here you see there was a price we were paying all the time and we weren't so keenly aware of that price as we ought to have been. That is that in attempting to produce efficient code, we were costing the customer in compiling time. One of the things, after FORTRAN was distributed and sort of out of our hands, I kept trying to convince the authorities then in programming that we should produce a quick and dirty FORTRAN compiler that would produce the most inefficient code conceivable a la Perliss. But do it as fast as possible. It is only recently that that is done. People kept arguing that they could, through extreme cleverness, sort of have the best of both possible worlds. That they could have very fast compiling and very efficient object code too, with only a slight loss in object code efficiency. It never really worked out satisfactorily.

S. Did the subsequent FORTRAN, FORTRAN II, III and IV, did they make those kind of changes?

B. No. We produced FORTRAN II. My group did that and from then on FORTRAN IV made some improvements in the language and they did improve the speed of the compiler. But their arguments in designing, which I disputed when the question came up, was that this was going to be the answer to both of these questions. It was going to be so fast in compiling that they wouldn't have to produce a quick and dirty compiler. That it could compile quickly

and still produce good code. In other words, they could do it with one compiler instead of two. They turned out to be wrong. It wasn't that much faster. In fact, when it first started out it was slower. But they worked up the speed to a very nice speed.

S. How do you work up speeds in that?

B. Just cutting analysis and going and doing it quick and dirty. You can produce an object program which does what is specified very quickly if you're not at pains to make it run fast. Because you then just don't do any analysis at all. You just translate things bit by bit. And you very definitely avoid any kind of translating procedure that makes it necessary to look at the context.

S. How did you get it to look at the context?

B. Well it ran through the source program and did what little translation it could, namely the translation of algebraic expressions into code, which is a relatively simple task and it stored away all the rest of the information that it was getting as it deciphered the input code into a lot of tables. Then the second pass of the system started looking at this business of context very heavily. It analyzed the entire structure of the do statements that caused these repetitions that we were determined to treat efficiently. If people had to have other loops that they didn't specify in the form of do loops, why we let them know in advance that these would not be treated in an efficient way.

S. Well when you first started conceiving the idea of FORTRAN, did you first think of a set of simple instructions that you would want people only to have to write or did they come afterwards?

B. No, the language came first. We didn't know how we were going to decode it. We tried to think of it but for the first six months we were sort of fiddling around with various ideas as to how you could express procedures. And we had a lot of other ideas in there that did have to do with things you would need to be able to express that in order that the machine could do the job efficiently that we later cut out. One idea was a way of making it possible to do input/output operations efficiently in overlying data when it was working on a large relaxation problem where you had three rows of a large array in at one time. This could sort of overwrite rows successively and still not have to rearrange all of the coding. It's a little too difficult to describe in words. But this is one kind of thing that we felt we would need and later it would still be nice to have. But we never put it in. But most of the features that we put in that initial description wound up in the system with only a few exceptions. Does that kind of answer your question?

S. I guess it is undoubtedly a fairly abstract subject because again you were thinking of... tell me if you were or you weren't... you were thinking

of specific mathematical problems that had to be done and various arrays....

B. Well it was just from our collective coding and programming experience we knew that certain kinds of operations were done a lot. Certain typical situations. The problem itself was immaterial to us. But we wanted a language that would allow you to express these things that we knew were the basic ingredients of telling a machine about a process that you wanted it to perform.

S. In this analysis business, did you have a series of passes that would continually scan what might have been put in .....

B. Well no, you see all this source program was completely put away in the form we wanted it after one pass. Now it was all in the form of tables and some compiled codes, little bits and pieces of compiled codes. The basic structure of the system was to do this first pass. Compile the code for algebraic expressions and put essentials in and whatever could be translated then and there was done and the rest was all put away in tables, in many different kinds of tables. The second pass then analyzed the structure of do loops and the structure of subscripts that they controlled and provided essentially the basic indexing instructions that would handle these things. But it wrote a program and I guess the idea to do this was one of the ideas that I did contribute in this thing and that was that instead of trying to get this program that was written

at this point, one that worked for a machine with three index registers which the 704 had, was not to try to get into that problem at this point but to assume .... to write a program that assumed that the machine had an unlimited number of index registers. Because that was another kind of a problem I felt and that's what was done. So that then there was a third pass which sort of put together all the bits and pieces and there were lots of little readjustments and things that had to be done, things that had been overlooked and couldn't be done until this time. It was done in Section III. So at the end of Section III we had a program essentially for a machine which was almost the same as the 704 but it had this unlimited number of index registers. The next two sections then ... Section IV did a sort of Monte Carlo analysis of how frequently the different blocks of the program would be executed. It was like sort of playing a Monte Carlo game with it. Then having decided now which were the most frequent ...the most frequently executed blocks of the program, then this very, very elaborate analysis that ~~Sherwin~~ Sheldon Best did came into play to transform this program that had used maybe seventeen index registers into one that used three index registers and minimized the number of ..... and minimized the number of transfers of index quantities between the real index registers and various storage cells in which they were kept when they weren't being used. That program now assumedly was assembled by an assembly program which was

special to the system and that was that. Now one interesting story in connection with this analysis that Sheldon wrote is that we had one of our first big sample programs. It was a very elaborate calculation. I forget just what it was. But in the compilation of that it took about half an hour and of that half hour, about twenty minutes was involved in those last few sections. It sort of generated these larger and larger regions of the program that were then to be treated by this indexing analysis. It was a very complex business. It turned out after the whole thing was over that that region generation was entirely unnecessary because there wasn't any subscripts used at all in the whole program. That twenty minutes of work in the compilation was absolutely worthless and that's the kind of thing that we had not foreseen too well.

S. Well when you sat down originally to start figuring out what could be done, did you kind of map out in flow charts or something, what you thought might work and then try and originate or implement it?

B. No it really was not this sort of master plan business. The people who did these various sections were all very, very competent people and we sort of generally knew that and we got started right away on Section I and on Section II and from there on and people were sort of thinking about Sections IV and V. But as all of these people progressed along they saw that one day

somebody would arise holding their head and saying, good Lord I need such and such information from you George in your section, and I didn't know it until now. But would you please create a table of this stuff.

S. Well this was almost like designing a computer, wasn't it?

B. It was as involved, yes. I suppose that the amount of complexity and the degrees of freedom of design in designing a system involving twenty or thirty thousand instructions is as complex.....

S. Is that how many compiler instructions there were?

B. Just about.

S. And everybody had his own section to work out. He was writing programs so that the machine would be programmed.

B. Yes.

S. Did you use the computer frequently in the course of this or was this done on paper?

B. Well this whole program was of necessity done on paper.

S. But I mean were you trying these things out on the machine every few days?

B. Well essentially people would debug. Everybody had his own philosophy of coding. Some people would write little sections and debug that right away.

I guess that's the more popular method because otherwise you'd forget the technical details of the problem before the program was written if you don't debug it pretty quickly.

S. Was it a debugging for convenience or did they need to run it on the machine to see how it was working?

B. Well no, because we never knew how it was going to work until we got the whole thing going. One of the essential qualities of programming is if you don't know how it is going to work when you design the program, it's not going to work. You can rest assured that if you don't know how it's going to work when you've got the program designed, that it won't work. This has been demonstrated time and time again in programming systems. With the exception of one or two people who are temporarily involved in the project, these people really knew what they were doing and they had to because they were doing very involved and very complex things. After a while it got to the point where most of the stuff was all written and the programs were individually debugged and it came time to fire up the whole thing. Of course nothing worked at all and things were getting clobbered right and left and then we were on the machine night and day. We had a suite of rooms in the Landon... or the Langdon Hotel which is now torn down. It was on 56th Street and Fifth Avenue and we used to sleep there and we would have the entire machine to ourselves practically all night. S omething that people would never think of doing nowadays. I can remember Roy Metz sitting at the console thinking for fifteen minutes while the machine just sat there. Then he'd key in something and see what happened.

S. Well this was after three years of work, is that right?

B. Yes. It took us about a year to debug the thing before we distributed it I guess. It was the last six months of the time before we distributed it. It was solid just debugging or more.

S. When you went through this project you must have been pretty sure that it was going to work, right?

B. Oh yes, we knew it was going to work but we didn't know by the time you get through having worked out the details, you know it is going to work. But it was sufficiently complex so that when it first started actually compiling bits of code, you didn't know in advance what the code was going to be like. You could tell how it got it. Everybody understood .... not everybody but the people in his section knew how this section had generated that piece of code. But it was very fascinating to look at a piece of code that was not constructed by human hands and find all kinds of elaborate and unexpected changes that the compiler had made in the original order of things. That was a very exciting and fascinating period.

S. So it printed out the compiled code and then you checked it.

B. Yes to see .....

S. Visually.

B. Well then we ran the code to see that it did what it was supposed to do. We often found that it dndi't and then had to find what the compiler had done wrong.

S. Do you recall any instances where very difficult problems arose where it wasn't just fifteen minutes figuring out what to do?

B. We had one case which we used to laugh about later ..... there was one guy who really didn't know much about coding. He was sort of thrown in to help speed things up a la IBM philosophy of the more the faster. And this guy wrote a few little pieces of code which we later realized ... or if we had realized he had never done so we would have been far better off. He would do things in the most strange fashion. We discovered much later than one debugging business, that somethkng kept going wrong in the most mysterious way. And after about two or three days we discovered that this piece of code that this guy had written due to some peculiarity and certain circumstances would start putting data all over memory in a very selective and disasterous way. It did it so cleverly that it took us a very long time to discover what was going on. Because his errors weren't showing themselves during the execution of his code but .... somebody else that should have had nothing whatsoever to do with his code was running theirs and all kinds of mysterious bugs would occur and they would look at their code and the code was correct and yet it wasn't doing what the code said. It's the kind of thing that happens all the time in a large system.

S. How did you extract it once you found out?

B. Well just jerked that code out and rewrote it or corrected it. So it didn't do these strange things any more.

S. How did you figure out that it was that code?

B. Finally by printing out enough memory, somebody noticed finally what was coming out maybe.... what was coming out of memory that was supposed to contain their program was changed in some slight way when it was put in and we went through a lot of tests to find out what was causing the changes and we finally discovered some obscure error.

S. It was a tracing problem.

B. Well I don't recall exactly. Usually tracing was not employed. Tracing in those days meant this business of sort of executing each instruction under the control of a tracing program and seeing what it did. That we didn't usually do. We usually just printed out parts of memory.

S. And analyzed it on paper.

B. Yes. How one debugged was a very mysterious thing in those days. It was up to each guy who knew his own code how he was going to do it. I guess there still aren't very good methods for doing it. Because you are essentially the owner of the unforeseen.

S. What was your position with FORTRAN? You were the manager or the overseer of getting the whole thing out?

B. My formal status was unclear much of the time that it was going on,

It was somehow understood that I was more or less in charge of this activity. Rather late in the game I guess I was made a manager or something.

S. You had mentioned that these other people did a lot of the writing. You kind of coordinatee their efforts.

B. Yes. I was general sort of handyman around the place. I took care of getting supplies and things like that. We had a lot of discussions about technical problems with people.

S. Well is there anything more that we should say about FORTRAN that I can't find out from reading all the write-ups of FORTRAN?

B. I believe we've said more than enough about FORTRAN. I can't think of anything more.

S. Did you get involved with anything else simultaneously or ....

B. Well yes, I think it was during that period that I got involved with STRETCH trying to combat what I thought were stupid errors in STRETCH Programs.

S. This was when the machine was being built or being designed?

B. Well it was being designed. When it was being built it was much too late. I always seemed to get into these things too late and always on the wrong foot. A group of people who have been working together designing something, never receive an outsider who comes saying, you're all wrong and I'm right, with any great welcome mat. That's what happened in this case. I was

S. Well how did you enter into the STRETCH picture?

B. I don't remember exactly how I got involved in it, whether I was asked to or whether I just knew people who were working in it and since I had been connected with this earlier effort .... I was interested in what was happening with what appeared to be its successor. That is this Livermore proposal. When I looked I was dismayed and always enjoyed trying to figure out how machine design should be done. I sort of tinkered around with various ideas and tried to show that if it was done this way it would be faster than the kind of thing that they have proposed. One of the things that my tinkering did do was to up the number of index registers very much in the machine that before I started hounding them they had very few index registers in the machine as compared to what they finally wound up with.

S. Well what was your approach? When you took a look at the machine and it needed more index registers, what was your approach?

B. It was not my approach to say it needed more index registers. In fact I didn't really feel that because I wanted an entirely different philosophy. I wanted no index registers essentialy.... that is I wanted registers but that would be used in a somewhat different way than index registers were used. I was sort of pushing for very little, tiny instructions that had a different character

than these complex instructions that they had in the machine. My approach was always to come at them with a different machine and say, look here, my machine is better than your machine.

S. What was your reason for wanting smaller instructions?

B. Well primarily because I felt that big, complex instructions existed because there was a certain word size in the computer and that if you sort of broke it up in various ways, you could get that word made into one instruction and then chop it up into various fields. You could get this complex package of action that in the hands of a human coder could be used with great efficiency. But that for automatic programming, these packages, these lugubrious packages of actions which had index quantities and little register addresses and little bits that said yes or no to certain things, that all of this made far too difficult a combinatorial problem to get an efficient program from some sort of theoretical approach. It seemed to me that if there was ever going to be a way of mechanically devising an efficient program, you'd have to have some sort of theory about it and it would have to be concerned with sort of elementary actions that the machine could take, like getting a number from a storage cell or adding two numbers, or putting a number back into a storage cell, very simple elementary actions. And that if you had to deal with arbitrary packages of these things

that came together that you might be able to figure out an efficient sequence of elementary actions that would do what you wanted with the utmost efficiency. But now if you couldn't have those actions in that sequence and you had to take the packages with these big bags of actions that were represented by machine instructions, big complex machine instructions, that a quite different sequence might be the most efficient way of doing it. Because if you figured out the sequence of elementary actions that did the job most efficiently, you would find then that you'd only be able to effect one of those elementary actions in each successive instruction and you would then be wasting all of the other opportunities that each of these instructions gave you, and so that in an entirely different way it involved lots more action and would be a more efficient way to do it on this machine. I was trying to see how you could design a mchine that would behave efficiently with small instructions because I thought that you would then get the possibility of more efficient automatic programmed operations than you could otherwise get. And of course, STRETCH was going to the just horrid extremes of complexity in that it had this huge word size and I felt no one had gotten this idea that you shouldn't package things in such big packages and I wanted to show that you didn't have to. I mean it's not easy to make a machine with little tiny instructions that will beat the conventional

kind of machine.

S. Well what was the reaction to your proposal?

B. I've kind of forgotten now what all the details were. But one of the main ingredients was that there be a number of little registers that would hold various little quantities that you needed so that you wouldn't have to be sending this back to the memory all the time and that these be used in .... that instead of having a machine that had an operation and then a large address, since you're going to have lots of memory and so a very large address, and then having this constant address modified by lots of quantities you got from these little cells, or index registers, as they kept referred..... they kept referring to them, that it would be feasible and perhaps even preferable to have an operation and only the addresses of cells coming at you as instructions, having none of these large constant addresses of main memory and then using these instructions to generate the addresses of main memory that you wanted in a way and this would mean that I was able to show that in a number of circumstances that you could make an instruction stream that had higher density of information in it and that it wasn't so wasteful of instruction bits. Many things could get done faster. Instead of trying to make a machine that had small instructions, what they did was to improve the performance of their design by adding or saying well your machine works lovely because it has got all these registers. If we put

them in our machine, our machine will work better and it did. So it was in that sense that I had the impression at least that this argument had some effect in determining the registers they would use.

S. Was this while Andall was there or after he had left?

B. This was all so damn long ago and I've been interested in so many things since, I can't remember the time scale. I think Andall had left by that time, but I'm not sure. When did he leave the STRETCH effort, while it was still being designed?

S. Yes.

B. He may well have left then.

S. Did you get involved in other things with STRETCH? Were you involved with it all along the way?

B. No. This was one of the reasons of course. The complaints of these people about my appearances with these proposals wws always that if you really want to have an effect on this design, you should join it and get really involved in the gorey details, which I didn't really want to do. It's hard enough to try and come up with a decent design without having to argue with a large committee of people about design. I was fatalistic enough about the possibility of ever really influencing the design anyway and to have it really be a test

of your own theories, like this example of this look ahead business getting blasted by interrupt. I think the look ahead decoder idea is still a good one. You'd never find it out through STRETCH. I think this is typical of machine design. There are so many people competing to effect the design that it turns out to be the union of everybody's ideas instead of the idea of any one person.

S. Aside from this session with STRETCH, did you have anything to do with the final programming?

B. No. I was very much involved with FORTRAN anyway all this time. These were just little forrays into this other field.

S. What do you think the effect of your suggestion would have been, to speed the machine up?

B. Well you know I can't really say. It's not fair of me to go claiming that if they had done what I told them.... machine design is a hard business and you may have an idea that looks elegant and as if it has great possibilities, but some little cross effect with some other thing may screw it up. I've got another idea now that I've never checked out at all, again because I've never worked it out to the point where I can even convince myself that it's a good idea. Because these things depend on so damn many factors that are hard to get all in hand unless you actually go build a machine or desing it in detail.

S. Well did machines at that point have, or even up to the 360 were programmers called in early enough to avoid problems or has that never occurred?

B. Well you know, programmers are always called in. They all say different things and I'm not sure it's a good idea even to call them in because again the more people you call in, the more illfitting ideas get thrust together into the final machine. Where the problem it seems to me, in machine design, is to drop ideas rather than to get them. It's cutting them out that's the decision. There are just so many good ideas that you want to put into a machine and the real hard thing is to say we won't put that one in, because it doesn't make a good total thing. It may speed up some operations, but the cost that it adds to the machine is going to make a worse machine in the long run.

S. After FORTRAN, is that when you went to Research?

B. Yes. I went into Research in '59 I believe.