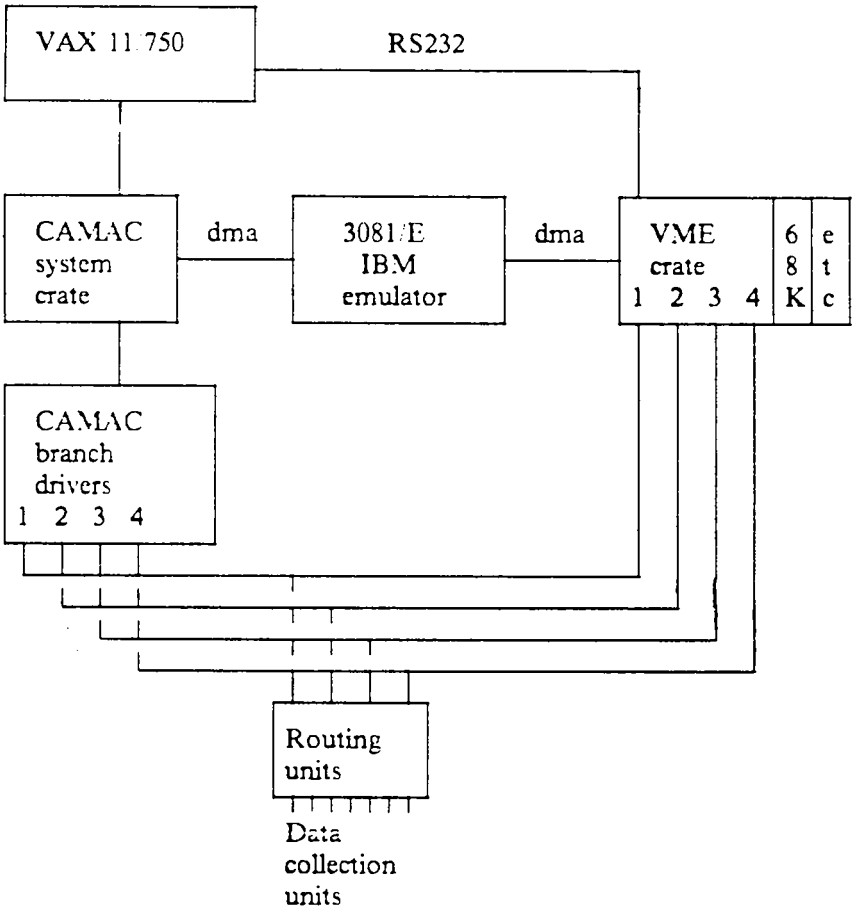
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

Data Collection in Elementary Particle Physics with 32 – bit VAX/68K Forth.

Ralph Haglund[1)]

Geneva, 28 August 1985

Abstract: Two 32 – bit Forth – 83 systems have been developed, running identical Forth code, with identical development environment.



A standard data collection structure of a big experiment at CERN is along the leftmost branch in the figure, with all user written software in Fortran.

The volume of data in a modern experiment in elementary particle physics is continuously growing. In this medium – sized CERN experiment, by the group HELIOS (NA34), the 'data collection units' in the figure fill two walls of a 40 meters long control room, and car loads of magnetic tape are produced in a year.

---

1) CERN, EP Division, Geneva, Switzerland and Lund University, Lund, Sweden.

To compress and filter data before writing on tape, a limited on−line[1] data analysis must be made. The 3081/E[2] is a joint effort by Stanford University and CERN to add the number crunching power of an IBM mainframe, running standard Fortran off−line software (to begin with) on−line. For the biggest experiments 'farms' of emulators are planned to run in parallel, each being fed one 'event' at a time.

To control and test the hardware in the VME crate and emulator, two Forth systems have been developed in parallel. Both can be used for development work. The 68K VME system is much faster but just for a single user.

The two systems have been developed in parallel, and they have the same structure as far is possible.

---

[1] On−line = directly connected to the electronics in the experiment while off−line is analysis of data stored on tape.

[2] Hardware emulating the CPU of an IBM 3081 mainframe, running at about 50% of its speed, priced at less than 50K US$. IBM machine code is translated to micro code by software on the IBM and downloaded via VAX to the emulator.

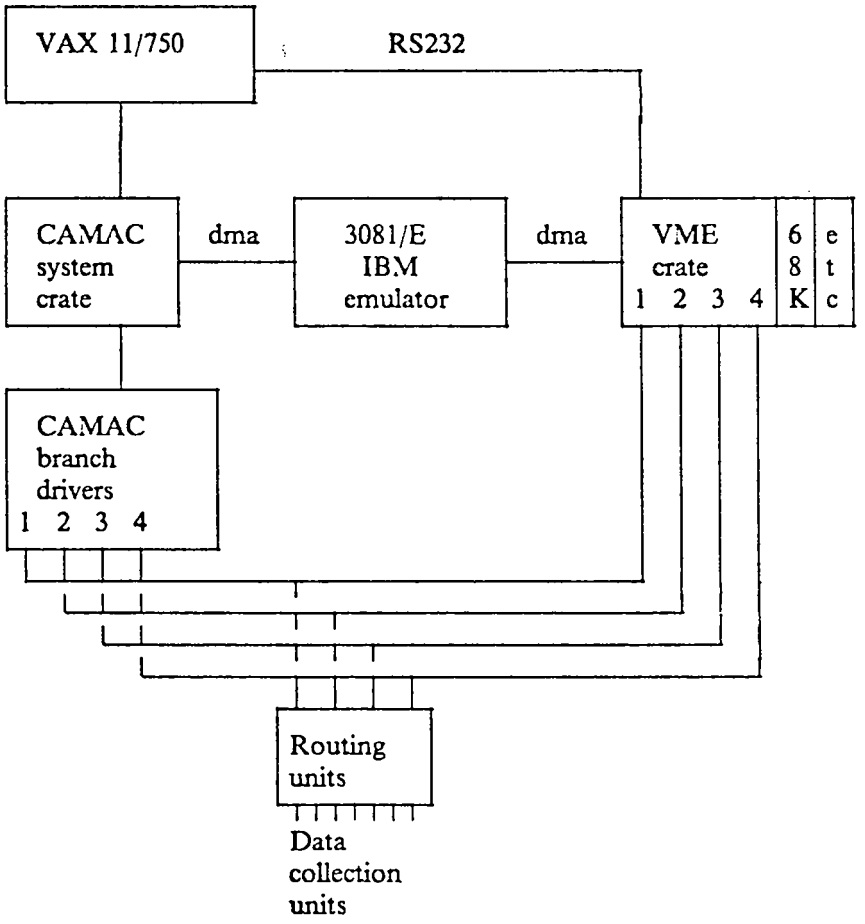
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

Data Collection in Elementary Particle Physics with 32 – bit VAX/68K Forth.

Ralph Haglund[1)]

Geneva, 23 October 1985

Abstract: Two 32 – bit Forth – 83 systems have been developed, running identical Forth code, with identical development environment.



A data collection structure of a big experiment at CERN is indicated in the figure, with all user software normally written in Fortran or assembler.

The volume of data in modern experiments in elementary particle physics is continuously growing. In our CERN experiment, HELIOS (NA34), the 'data collection units' in the figure fill two walls of a 40 meters long control room, and car loads of magnetic tape are produced in a year.

---

1) CERN, EP Division, Geneva, Switzerland and Lund University, Lund, Sweden.

To compress and filter data before writing on tape, a limited on−line[1] data analysis must be made using a reasonable amount of computing power, in our case the 3081/E.[2]

To control and test the hardware in the VME crate and emulator, two Forth systems have been developed in parallel for the 68K and for the VAX. Both can be used for development work intended for both targets. The 68K VME system is much faster to use than the VAX 11/750 under the load of a few users, but is just a single−user system.

The two systems have been developed in parallel, and they have the same structure as far as possible.

Differences include:
For the VAX:
floating point is included
most significant half of double precision numbers in higher address
more inherent 64−bit instructions, reflected in the arithmetic functions.
For the 68K:
assembler and disassembler are available
most significant half of double precision numbers in lower address (according to the Forth standard).

Both systems use the same screen editor, and other systems programs, like decompiler, line editor and tracing. The screen editor is trivially reconfigurable for any modern terminal. Observe that these programs are loaded from symbol code, so one is free to modify them according to personal taste.

A modern orthogonal control structure is implemented. Both routines "jump if stack true" and "jump if stack false" are available, so instead of the usual
0= IF ... THEN
we can write
NIF ... THEN
and in general we have:

(N)IF ... (ELSE) ... (flag (N)THENIF) ... (ELSE) ... THEN
BEGIN ... (flag (N)WHILE/LEAVE) ... flag (N)UNTIL/REPEAT
DO ... (flag (N)WHILE/LEAVE) ... (n +)LOOP

The ONLY concept, with vocabulary stack, is implemented.

Shadow screens on a separate file offer one screen for comments for every screen with symbol code "forcing" the user to comment properly.

Also in the biggest experiments at CERN, the parts of the online software specific to that experiment are mainly written in assembler, to attain top speed and flexibility − a high−level language is inefficient when it comes to using special features in the machine code of a particular processor. At this level compatibility is not relevant − who will ever read the assembler lists once the system is functioning... On the other hand, already as documentation, a Forth description is very useful − and later on implementable. The criticism Forth has got as being a "write−only" language might be justified at the lowest level with intense stack manipulation, but not at the higher levels normally seen by the user.

---

[1] On−line = directly connected to the electronics in the experiment while off−line is analysis of data stored on tape.

[2] Hardware emulating the CPU of an IBM 3081 mainframe, running at about 50% of its speed, priced at approximately 50K Swiss Francs with two Mbytes of memory (maximum 7 Mbytes). IBM machine code is translated to micro code by software on the IBM and downloaded via VAX to the emulator. This project is a joint effort by Stanford University and CERN to add the number crunching power of an IBM mainframe, running standard Fortran off−line software (to begin with) on−line. For the biggest experiments 'farms' of emulators are planned to run in parallel, each being fed one 'event' at a time.

In HELIOS, Forth is mainly used in the VME system to control hardware, data transfers etc. This task would be a fairly large project in assembler. It will be used in other parts of the data collection structure, once it has shown its feasibility.

The reason why Forth is suitable for a number of applications in physics can be expressed in these keywords:

- Speed of developing and testing.
  Editor and compiler is integrated in the system. As Forth is interpreted (with an interpreter consisting of three machine instructions), one can either write, compile and execute a Forth word immediately, or edit symbol code, save this and later load (= compile).
  There is never any time consuming edit – compile – link – load – execute cycle. Once a Forth word is compiled, it is simply a new element in the 'dictionary'.
  As parameter transfers regularly are via stack, one doesn't have to write any main program to test every subroutine.
- Execution speed.
  One way of looking at Forth is as an "infinitely powerful" macro assembler. There is a continuous path from writing Forth words in machine code, to your own very – high – level language. Lower level Forth can be sealed off for any level of information hiding.
- Total speed.
  One can optimize both development and execution speed to the most suitable level.
- Modularity, readability.
  Symbol code is edited on small 1024 – byte screens, and accompanied by a shadow screen of equal size for comments, forcing modularity that way. It is harder to write 20 – page subroutines than in conventional languages, to the dismay of Fortran addicts. The GOTOs are concealed, and not explicit, at least for the novice.
  Readability is enhanced because of the small size. There is also a convention for how to write the screens to make them more readable, and this should be adhered to.

Reference:

Ralph Haglund: FORTH. Implementation on the VAX, 68000 and 99000.
In preparation, to be published as a Data Division report at CERN.