CONF-8709217--4



# TUTORIAL ON COMPUTER CONTROL*

CONF-8709217--4

DE88 006185

R. C. Juras
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831

Thank you. The word tutorial in the title for this talk is probably a little ambitious. Ken asked me to talk about computer control and maybe stimulate some discussion.

I should mention my experience with computer control. I have worked with the Oak Ridge Holifield Tandem facility control system since 1976. As most of you know, that's a totally computer based system. Before that I worked at Sperry on a computer control system for an aircraft that was to mimic the characteristics of the space shuttle during landing. It is used for astronaut training.

What comes first? First you must decide if you want a computer-based control system. Tandem accelerators, particularly smaller ones, can run just fine without computer-based control systems. They've been doing it for years, successfully putting beams on target. However, computer based controls offer advantages, particularly for large tandem accelerators.

What are the disadvantages of computer control? There are several disadvantages. Complexity is a disadvantage. You have to write software and people have to understand how it works and how to keep it running.

---

*Research sponsored by the Division of Basic Energy Sciences, U.S. Department of Energy, under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc.

This manusacript is an edited version of the author's remarks. This format has been chosen to facilitate timely publication.

You have to write custom software because, although you can buy a wide variety of software these days, you can't buy much that's applicable to accelerator control. As the accelerator evolves, new controls must be integrated and software rewritten. So, you must spend time maintaining the accelerator control software.

You need to do system management. You have to do backups and worry about maintaining the operating system. When the computer vendor upgrades the operating system, for example, some of your programs may have to be changed because the details of system calls have changed. These sorts of things are a drain on manpower.

It can be difficult to integrate computer-based controls with some older equipment. Many of the older power supplies were not built with the intention of remote control or computer-based control. It takes some effort and expense to modify or replace those supplies.

Once you have a computer-based control system, accelerator operation is dependent on the hardware and software reliability. In fact, the computer system can even damage the accelerator if the software does crazy things as a result of bugs. And since the number of hours of beam on target is dependent on the reliability, you have to worry about making it work right and getting all the bugs out.

Given all those disadvantages, why have computer control? I believe the principal benefit is software aids-to-operation. Once the computer-based control system is in place, you're limited only by your imagination as to what you can do with it.

You can -- if you have the resources -- write completely automatic controls for accelerators. I believe that accelerators can be tuned automatically, but I don't think we'll see that soon because it's too expensive given the present state of software engineering.

But there are types of software which can be provided with much less cost that provide great benefit for accelerator operation --software, for example, to cycle analyzing magnets automatically (to avoid hysteresis), to set up the accelerator from scaled, recorded values from previous runs, to scan the ion source analyzing magnet and plot the results, or to keep a running log of accelerator parameters.

Controls can be easily multiplexed for operation at elevated potentials and this can be done at little extra expense because computer-based control systems multiplex data on serial data links anyway. Of course, you have to worry about spark protection of the electronics at elevated potential, but it has been demonstrated, in a number of applications, that this is a solvable problem.

Computer-control systems offer precise, repeatable controls so that you can go back to exactly where you were at some previous time. Elimination of long cables and ground loops is advantageous, particularly for larger tandems. It is expensive to pull many long cables throughout a building and the resulting ground loops can be a difficult battle to fight. Computer systems operate with just a few serial data link cables between equipment racks and these serial links are easily ground isolated.

Record keeping is another advantage. You can keep detailed records of all the beams you have run with little effort.

New controls are usually easy to add although this can sometimes be difficult. The wiring is always easy. There are no long cables to pull, just a cable to the nearest equipment rack. The software is easy if it's another device of the same type that you already have. The software is difficult if it's a brand new type of device so that you have to write a whole section of new software and update every existing program.

The remainder of this talk will be divided into three parts, the three important components of control systems: hardware, software and the operator interface.

## HARDWARE

What considerations should be kept in mind for hardware selection? Select computer hardware so as to reduce software costs. Why do I say that? Most control hardware can be purchased these days and the price performance ratio is improving rapidly. It doesn't take a lot of manpower to buy computer hardware. Just buy it and bring it in. It can be done quickly. But almost all the accelerator control software must be custom designed. So, I think it's necessary to keep software cost in mind when deciding on hardware. Spending a little bit more on hardware may reduce the total system cost.

You may buy a cheap computer only to find that you have to write tens of thousands of dollars of software to get the system to work right

because, for example, the software must be written in assembly language or some other low-level language to achieve the execution speed you require.

Select a computer system that can be upgraded without rewriting software. We all know that computers rapidly become obsolete. They're obsolete sometimes before you get them in. So, someday you're going to want to upgrade your computer. The problem is the software. If the software's very computer dependent, it may lock you into an obsolete computer. Assembly language software, for example, is computer-dependent. If the software is written in a high level language, it is more portable, but even then you're probably going to have to rewrite some of the I/O interface.

If your computer is a member of a compatible family of computers, even assembly language software can be portable. Examples of this are the VAX or IBM PC families. In such a family, there exists a range of computers from the low end to the high end. So you can start with an inexpensive, low end computer and move up to more powerful computers keeping the same software all along the line including, in many cases, the I/O software.

We are using Concurrent computers (Concurrent is a subsidiary of Perkin-Elmer). Concurrent turned out to be a good choice for us. We were able to replace the original computers and install more powerful computers with minimal software changes.

Another way to facilitate upgrades is to use computer independent

buses and I/O. CAMAC, for example, is independent of the computer that you use. When you upgrade your computer you don't have to change any of your A/D converters, D/A converters, or other input or output modules. The accelerator interface stays the same. Avoid A/D and D/A convertor systems that are tied to whatever computer bus you are using unless, of course, your computer bus is itself a standard, such as VME.

Another way to save on total system cost is to standardize the hardware interface. This is a little harder to define. I'll give an example shortly.

If you can standardize the hardware, the software will cost less to write in the first place because you will be able to make use of common subroutines and common procedures. In other words, if accelerator interfaces are identical, or can be made to look identical to the software, common software can be used.

Hardware upgrades that conform to the standard can be accommodated by simple database additions rather than new software. And the software is easier to maintain. There is less of it. It is easier to understand.

Here is an example (fig. 1). All of the devices, A, B and C, are the same type. They all require the same sort of conversion, maybe a linear conversion, by the software. However, the details of the interface to the three devices differ slightly from one to another. The three rectangular boxes represent the binary number read by the computer

from each of the three digital input registers (CAMAC modules) that are wired to the devices.

For devices A and C, logic 0 is a fault condition. Logic 1 is a fault for device B -- probably the interface module was simply wired to the other side of a form C relay contact. The other differences are simply due to differences in the wiring to the interface module.

What is the result of these small differences? The software shown on the left of the viewgraph is necessary to deal with the three devices wired as shown. If all the devices were wired like device B, the software on the right would result.

This, of course, is a trivial example. However, there are real-life examples that don't differ much. At the Workshop on Accelerator Controls, Michael Glass[1] talked about experiences at Fermilab where quite a bit of involved software was written to make up for the lack of an inexpensive integrated circuit in a CAMAC module.

Here are some rules of thumb for interfaces. Try to use common components whenever possible. Why? Because there will be fewer spare parts to keep in your inventory and there will be less special software to write. I'm talking here of A/D, D/A and other interface modules.

Provide true device status to the operator whenever possible. When the operator pushes a button on the console to turn on a power supply, for example, use a relay in the power supply to return the true status instead of just echoing his request to turn on the supply as the power supply "on" status.

Leave some room for growth. It doesn't cost much to leave spare wires in cables; you're probably going to need them later. Leave a little extra panel space, too.

Every interlocked device should have an indicator signal for each interlock instead of one indicator signal for several interlocks that are in series and perhaps scattered throughout the building. This implies that you should have a pair of wires coming to the interface from each interlock. Beam time is expensive. Without individual indicators, if you have a fault that is keeping beam off target you may have to rush through the facility to find which interlock is the culprit.

## SOFTWARE

We've already talked a little about software because the hardware and software are interrelated. Here are a few other ideas about software.

Additions and changes to the software are easier and faster if you use a database. For one thing, with a database there is no need to work in a cryptic format. Instead you work with easily understood text format with powerful text editors. That means that there is a smaller probability of error and that errors are easier to spot; you don't have to look at something encoded in hexadecimal, but can quickly scan through unencoded text.

Complex interrelations in the data buffer can be changed automatically by the database program. The cross-referencing is done

every time a run-time buffer is generated.

The documentation is improved. Listings can be made with the data sorted in various ways. Editors can be used to search the database, for example, for every occurrence of a particular type of device or every device turned on and off by a particular I/O module. Application programs can access the database for up-to-date information rather than relying on conversion factors and addresses buried within the application programs.

Figure 2 is an example of the value of a database. In this example it is seen that the database is easy to read. The database program is used to translate the accelerator database into the buffer image format dictated by the computer hardware. Imagine being faced with the need to make changes to the buffer image, perhaps because a power supply on the beam line has been replaced. Contrast the ease of changing the database with the difficulty of changing the buffer directly. In addition, the database program can make listings for documentation.

Figure 3 illustrates another value of a database. In this case, the value X is read from an accelerator component. It is typically a number read from an analog-to-digital converter. Several programs may access this number. Each program then must convert this number to a meaningful value perhaps by multiplying by a constant, A. The number X may be a reading from a rotating coil gaussmeter, for example. If the rotating coil gaussmeter is serviced and replaced, the constant, A, may change. Then you are faced with changing the constant in each of the

programs in which it occurs. First you must determine which programs contain the constant, then find the source code. If the programs are written in a compiled high-level language, such as FORTRAN, they will have to be recompiled. On the other hand, if the constant is stored in a database that the programs can access, making the change is simply a matter of replacing the constant in the one place it occurs.

Anytime you can, use a high-level language. The use of a high-level language increases programmer productivity. Because it is easier to understand and modify, it reduces maintenance costs. In addition, high-level languages are more transportable to new computers.

Not so long ago it was necessary to write time-critical software in assembly language or FORTRAN with in-line assembly code. The FORTRAN, and other high-level-language, compilers of ten years ago were not very efficient at generating good machine language code, a code that is compact and executes very quickly.

Compiler technology has come a long way. There are now 'optimizing' compilers available for FORTRAN and other high-level languages that produce machine code as good as that of a proficient assembly language programmer. Typically, you buy two compilers. One compiler is used for development of your programs. It compiles quickly but doesn't result in optimal machine code. You use this compiler until your program is debugged. Once the errors are out of your program, you compile the program with the optimizing compiler. It takes a much

longer time to compile, but it results in compact code that executes more quickly.

Provide a good programming environment. A good programming environment leads to better application programs. By "good programming environment" I mean a powerful operating system and transparent access to real time accelerator data. The people who design the computer hardware and system software should try to make it easy for those who are going to come in later to write application programs.

In our case, the scheme that is used to provide access to real-time accelerator data is a shared memory between the computer that runs the application programs and the computer that runs the accelerator. All parameters are kept in the shared memory by the accelerator control computer. The application programmers don't have to worry about scheduling I/O or requesting data from the control computer. Application programs simply read values from shared memory; the data is never more than 1/2 second old. Or the application program writes to shared memory; the data will be sent to the controlled device within 1/2 second.

It is also important to provide the application programmer access to the accelerator parameter database. Don't use the database for just your systems programs. Make it available for application programs.

A friendly operating system, a high level language and conceptually simple access to data make programming very straightforward. The result is that staff members, visitors on short-term assignment, graduate

students and co-op students can write application programs.

Having short-term people writing software can be a double-edged sword. They may produce good software, but when they leave you must maintain the software. This is another reason to use a high-level language.

## OPERATOR INTERFACE

The final topic to be discussed is the operator interface. This is the part that is likely to stimulate the most discussion. A computer-based control system represents an opportunity to create a better accelerator control environment -- or a worse one, if not done carefully. The operator interface can have a big effect on efficiency of operation.

The operator interface is important also because a significant portion of the control system software must be written for the operator interface.

Let me quickly describe our system. Then I'll try to tell you some things that we have found that are wrong with our system.

We have two identical control consoles. Each console has a color alphanumeric display with an associated page selector and trackball driven cursor. The cursor is moved to an item of interest, for example the on/off control for a device, and a button located next to the trackball is pressed to change the status of the device.

For those who may not know, a trackball is simply a mouse turned upside-down and mounted on a tabletop. If you buy an upside-down mouse for the McIntosh computer it is called a turbo mouse. So I guess we have a turbo mouse.

Each console has three 'assignable' shaft encoders. The cursor is moved to a CRT line, the 'assign' button next to a particular shaft encoder is pressed and the shaft encoder then controls the device pointed to by the cursor.

Three analog meters on each console are assigned in a similar manner. Analog meters are useful for trending information, for optimizing the beam on a faraday cup, for example. Eight 'analog jacks' can be assigned, also. The output of each analog jack is a zero to ten volt signal that is typically connected to a strip chart recorder and is used to monitor such things as vacuum in dead sections of the accelerator, the GVM and x-ray levels.

In addition to the assignable components, there are six dedicated meters for terminal potential stabilizer parameters. Oscilloscopes are used to display NMR resonances, beam profiles and the terminal capacitive pick-up. The oscilloscope signals do not go through the computers. The NMR is set by the computer, but the resonance condition is detected by the operator looking at the NMR oscilloscope. There are several emergency shutdown buttons. They are comfort buttons that are independent of the computer. The computer has been exceptionally

reliable so far and we haven't had much use for them, except during maintenance.

Now I will detail the shaft encoder operation. When the shaft encoder is assigned, the device label appears above the shaft encoder. This is a valuable means for the operator to keep track of which shaft encoder is assigned to which device.

'SAVE' and 'RESTORE' buttons are provided for each shaft encoder. Typically, when the operator assigns a shaft encoder, he presses the SAVE button. Then if the result of his tuning is a decrease in beam current, he presses the RESTORE and the beam current returns to its previous value. This really works and is used often. This is one benefit of the precise, repeatable nature of digital controls.

Each shaft encoder has a 12-position range switch for two to 4096 turns full scale. Thus the operator has a choice of the resolution of the shaft encoder. Percent of full scale output for the assigned device is displayed on the CRT.

About a year ago the method of shaft encoder assignment was changed slightly. Previously, if the assign button was pressed with the cursor on a CRT line with no legal assignment, the knob assignment was nulled. Now the previous shaft encoder assignment is retrieved. So it's possible to toggle back and forth quickly between two assignments without reference to the CRT page.

After five years of experience, we're still happy with the use of

the shaft encoders and the philosophy of multiplexed control, but agree with the operators that more knobs are needed. The consoles have three shaft encoders, but many controls naturally occur in pairs of pairs. An example is the x and y axis of a quadrupole triplet followed by the x and y axis of a steerer. So, the number of shaft encoders per console should be increased to at least four. Five is the number we're probably going to settle on. There doesn't appear to be a need for more than that.

The shaft encoders are now read and sent to the controlled device 50 times a second except for the bending magnets. The 20 bit digital-to-analog converter for the bending magnets is an electromechanical device and to avoid wear it is updated only twice per second. Our experience with bending magnet control leads us to believe that maybe twice a second is fast enough for all control devices. So, we may try changing all the shaft encoders to a 2 Hz update.

Analog meter assignment is just like the shaft encoder assignment. The device label appears above the meter, and the range and range units appear below the meter. Each meter has a times three switch that expands the lower one third of the meter to the full meter scale. The meters are updated 50 times per second.

Following the success of the change in the method of shaft encoder assignment to recall the previous assignment, the CRT page selection was modified. Pressing the Enter key on the key pad with no number entered

now recalls the previous page. It is possible to toggle between two CRT pages simply by pressing ENTER.

The reason I stress these modifications is that they illustrate a valuable point: In a computer control system, the ability to go back one step is beneficial. Even the save and restore buttons on the shaft encoders can be considered an implementation of this general philosophy. You should provide the operator with a means to go back one step wherever possible.

Another thing we have found since our control system was built is that alarm indications are necessary. In control systems with multiple CRT pages, only one CRT page is visible at a given time. The operator must be made aware of problems on other pages. Our present solution is a bank of alarm lights. Each alarm light has a legend indicating either a particular alarm condition (such as a serial highway error) or a CRT page number. If an alarm indicator with a CRT page number lights, the operator turns to that CRT page. On the page, the alarm is highlighted by a red background. The alarm may be acknowledged by moving the cursor to the alarm. The background for that alarm then turns yellow and the light on the panel extinguishes so that the operator will be aware of another alarm occurring on the page. When the alarm condition goes away, the alarm is rearmed.

We are looking at a new way to present alarms, perhaps a small CRT display, so the operator doesn't have to change pages. Alarms are an important part of a control system and require careful thought.

Software aids to operation are possibly the major benefit of a computer-based control system. I'd like to simply list here some of the generic things you can do to aid the operators.

You can write software to record beam parameters for successful beam tunings. Then, in the future, you can set up the accelerator for new beams based on scaling the recorded parameters.

Software can identify ion species. The computer can scan the mass analyzing magnet following the ion source to determine the ion source output, much like a mass spectrometer. Programs can also be written to look at the accelerator terminal GVM and magnet NMR's to verify which ion species is coming through the accelerator.

Software can log accelerator parameters both at timed intervals and on demand. The operator may then look back at the accelerator history. For example we recently discovered degraded vacuum in an external accelerator beam line. The question then was: How fast is the vacuum degrading? Must we shut down now or can we wait for a break in the schedule? In this case, the history log showed it had been getting worse slowly for quite a while, so repairs could wait.

Analyzing magnets can be cycled automatically to eliminate effects of hysteresis. This used to be a tedious chore for the operators. Programs can be written to calculate theoretical foil stripper lifetimes and to calculate parameters for manual setup (including charge state fractions after the strippers).

I now want to discuss a subject which is almost sure to stimulate discussion, alternative operator interfaces. As the technology changes, we should think about ways to utilize the new technology to improve the efficiency of accelerator operation.

An alternative to the trackball or mouse controls described previously is a touch panel display with menus. The menu systems are easier to learn in the first place, but the operators I've talked to complain that in many such systems, even experienced users must go through the tree structured menu for each operation. If the tree structure is several layers deep, the operator will soon become frustrated with the time required.

Our system, with almost 100 pages, for one example, is a little harder for the operators to learn in the first place, but after they become experienced they can remember which page almost every device is on. The capacity for recall in humans is great.

Which interface is best depends on the situation. Menus are great for inexperienced or infrequent users while full-time operators probably prefer a more direct interface.

Because of less expensive graphics along with software to help you do graphics, high resolution graphics displays might be the future. Expect to see more McIntosh-type displays with a mouse (or trackball) and pull-down or pop-up menus. The Vivitron designers are considering doing their operator interface with pull-down menus and I'm very interested to see how that turns out. It may be very good. Designers

at Fermilab are also working on an operator interface using that paradigm.

Up/down buttons versus shaft encoders are a continuing debate. We have control systems that use each. The cyclotron control system makes use of buttons while the tandem control system makes use of shaft encoders. When we began planning our upgrade, we asked the operators if they preferred up/down buttons or shaft encoders and they chose shaft encoders. One advantage of the shaft encoders, they said, was without looking you can go up two turns and down two turns and return to where you started.

Alternative analog displays are available. Bar-graph displays could be used in place of analog meters, particularly now that high-resolution displays are available. Another alternative is a meter or bar-graph display that is simulated on a graphics CRT display. I believe that the Vivitron designers are considering the use of meters simulated on a graphics CRT display.

## CONCLUSION

In conclusion, a good operator interface is important for efficient accelerator utilization. The operator interface is going to require a lot of software so it should be well thought out at the beginning.

The hardware for computer-based control is readily available now. It is becoming more affordable all the time. The software costs are not decreasing nearly as rapidly. Software will be expensive for the

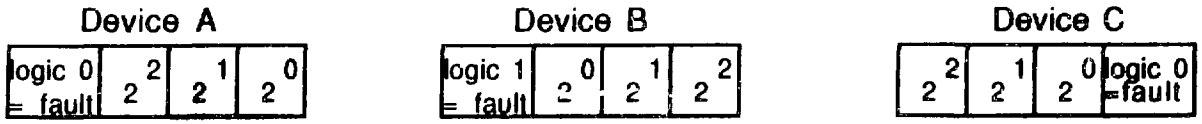
foreseeable future. So the hardware should be selected to minimize the total cost of the system.

## REFERENCES

1. M. Glass, "The Meeting of Two Realms: Lessons from the Tevatron Front End", Nucl. Instrum. and Meth. A247 (1986) 133-138.

List of Figures:

All Devices Type 1

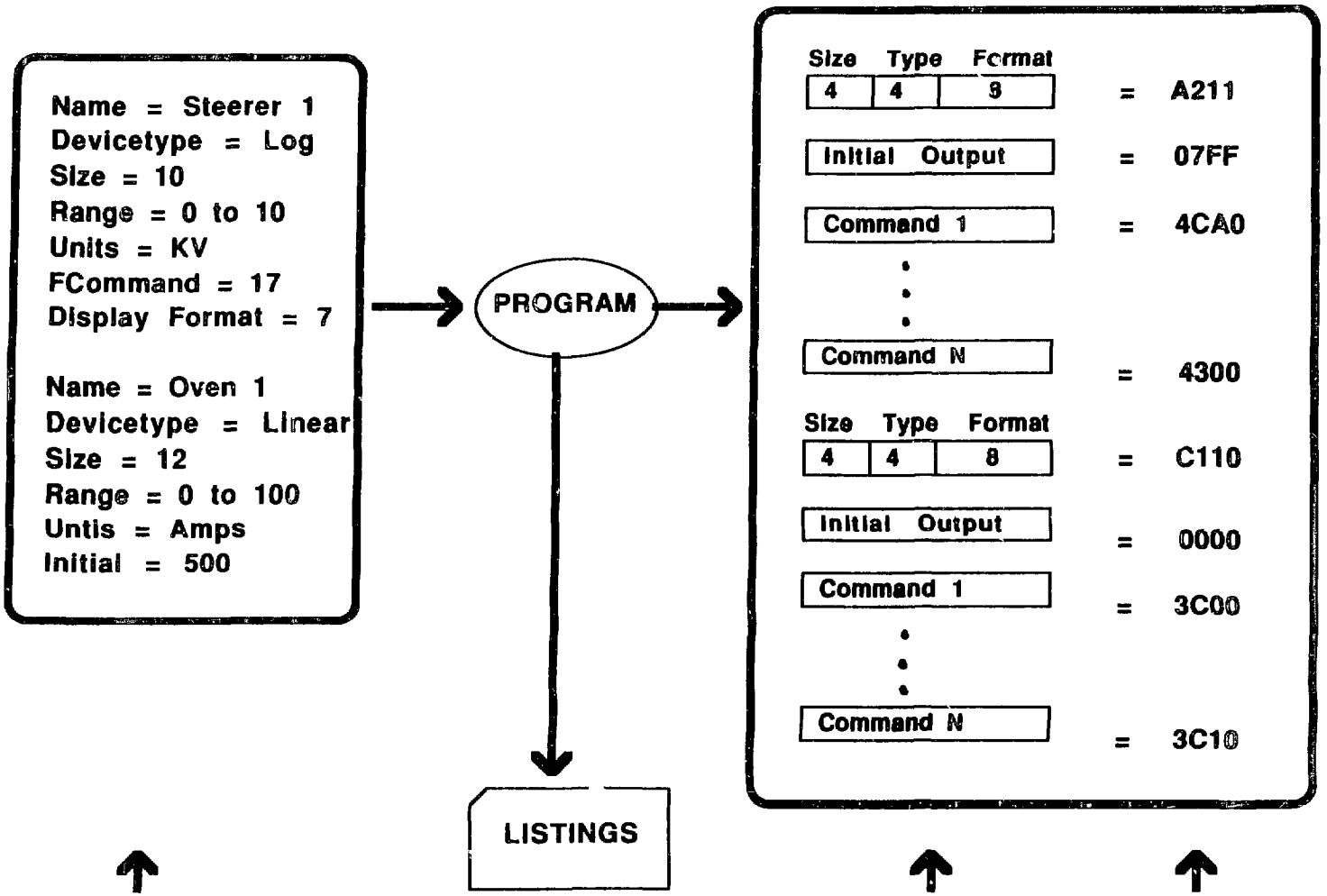


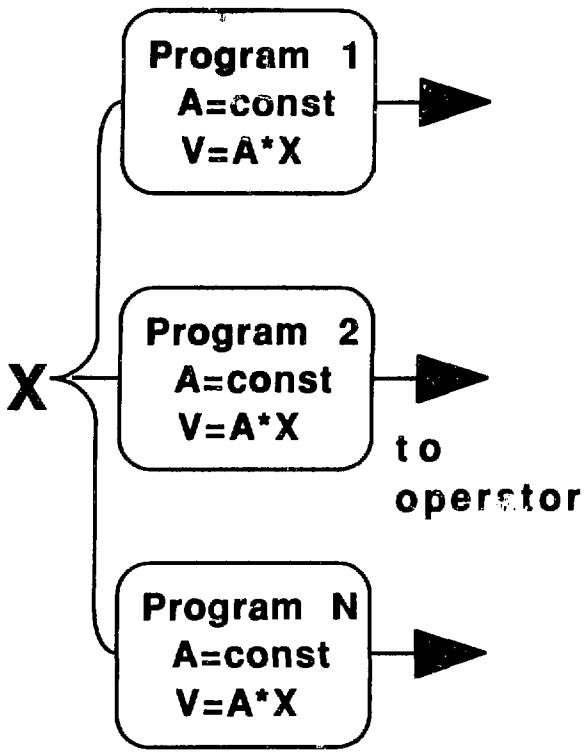

```
If Devicetype = 1 then
  If Device = A then
    If bit  x'8' = 0 then
       report error
    endif
    Mask input with x'7'
  If Device = B then
    If bit x'8' = 1 then
       report error
    endif
    mask input with x'7'
    goto subroutine to swap bits
  If Device = C then
    if bit x'1' = 1 then
       report error
    endif
    Shift input 1 bit right
    mask input with x'7'
  endif
endif
```

**Software is complex for non-standard interface shown above**

```
If Devicetype = 1 then
  If bit x'8' = 0 then
     report error
  endif
  mask input with x'7'
endif
```

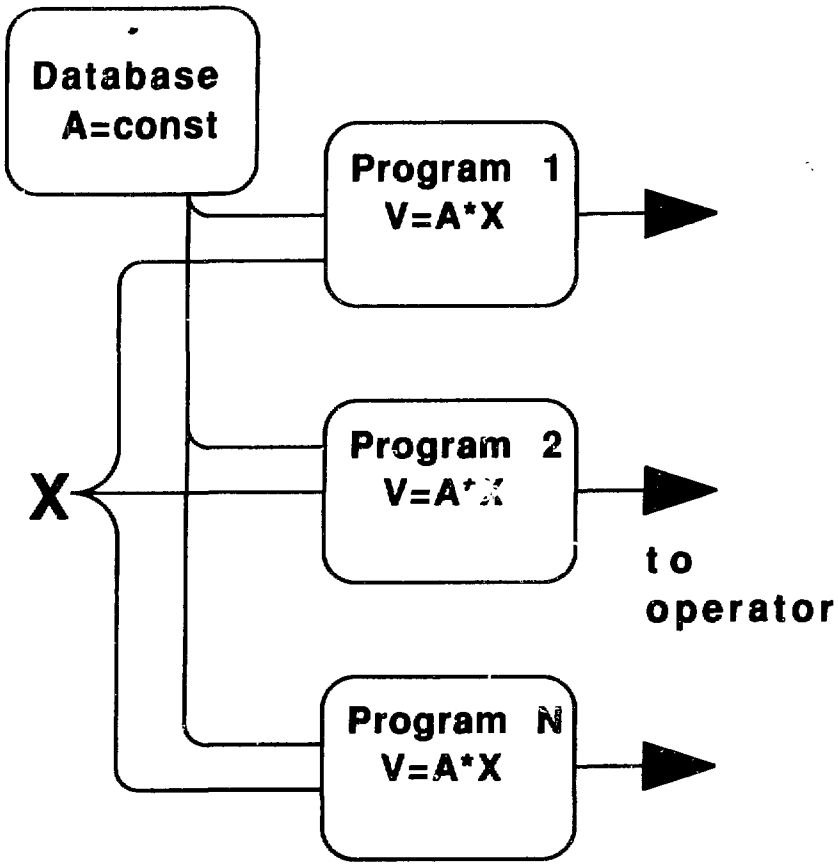
**Software is simplified if the interface for each device is exactly the same**


Name = Steerer 1
Devicetype = Log
Size = 10
Range = 0 to 10
Units = KV
FCommand = 17
Display Format = 7
Name = Oven 1
Devicetype = Linear
Size = 12
Range = 0 to 100
Untis = Amps
Initial = 500
PROGRAM
LISTINGS
Size Type Format
4 4 8
= A211
Initial Output
= 07FF
Command 1
= 4CA0
Command N
= 4300
Size Type Format
4 4 8
= C110
Initial Output
= 0000
Command 1
= 3C00
Command N
= 3C10
Accelerator Database
(Typical Entries)
Format Dictated
by Hardware
Buffer
Image




NO DATABASE

Constant stored in each program - diffucult to change




DATABASE

Constant stored in one place

fig. 3

## DISCLAIMER